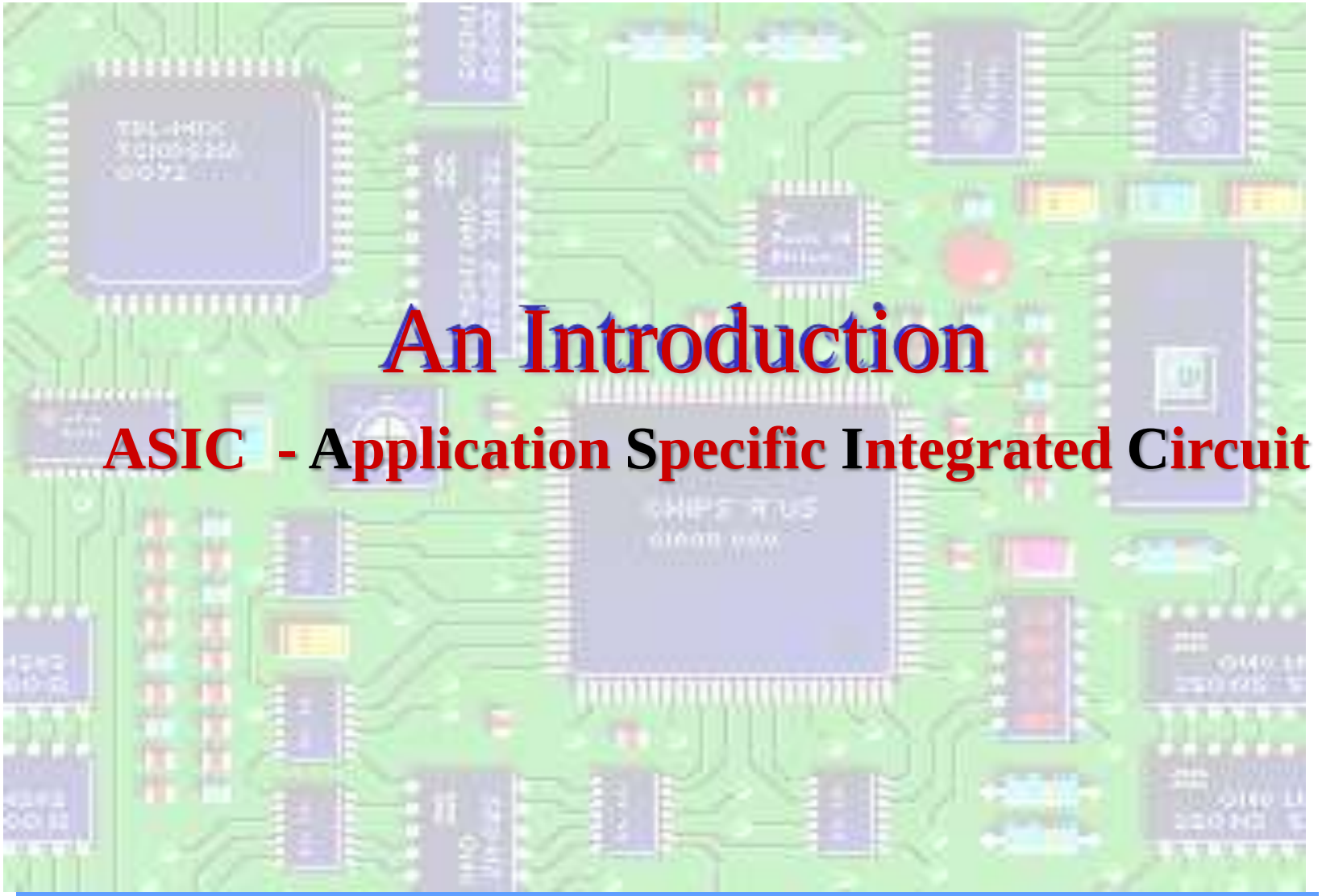
14EC770 ASIC DESIGN

Dr.(Mrs).D.Gracia Nirmala Rani
Assistant Professor
ECE Department
Thiagarajar College of Engineering
Madurai-15
Email : gracia@tce.edu



An Introduction

ASIC - Application Specific Integrated Circuit

14EC770 : ASIC DESIGN

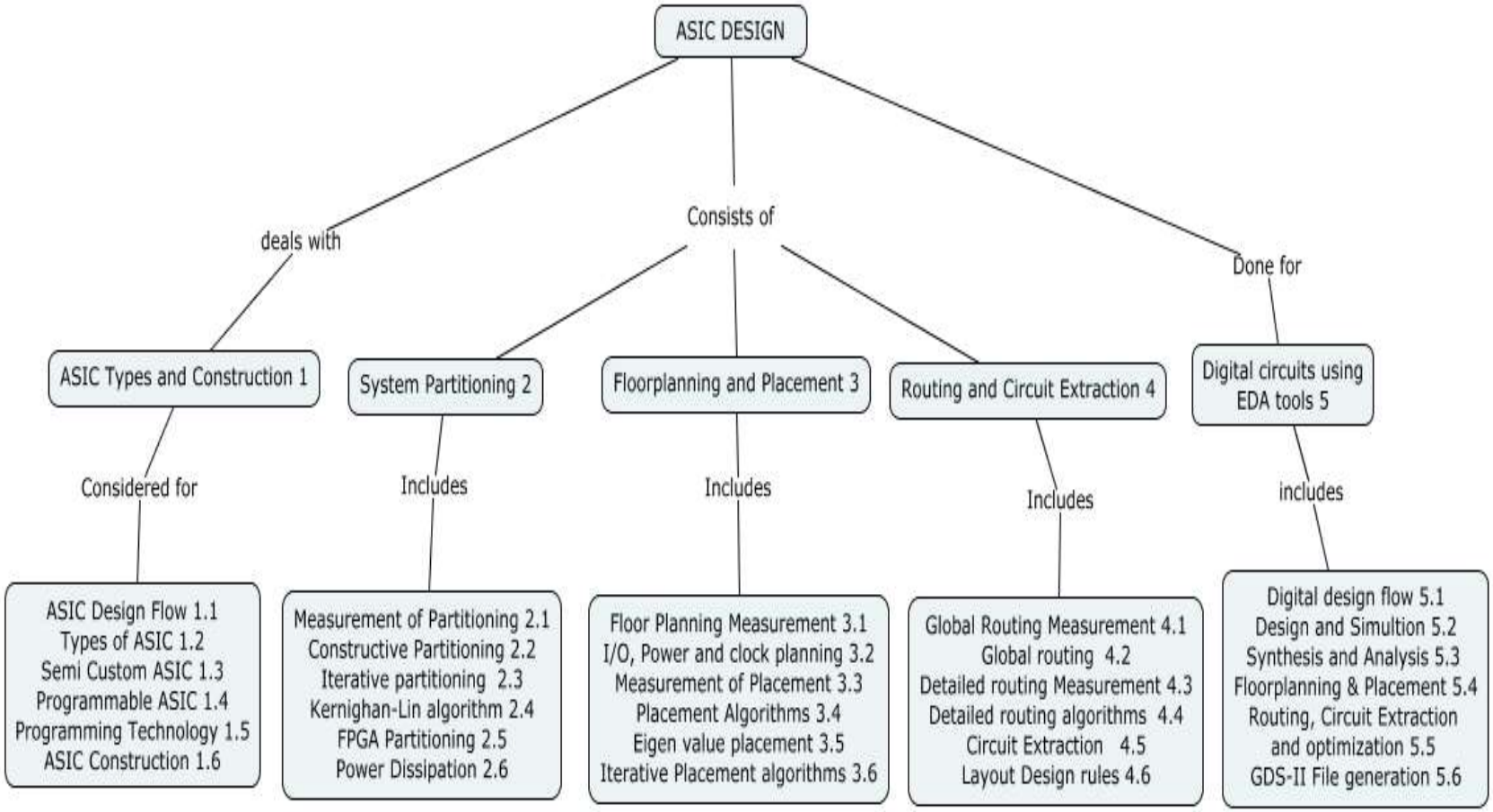
- **Preamble**
- 14EC270 : Digital Logic Circuit Design
- 14EC520 : Digital CMOS Systems

- **Objective**

This course provide the students, the knowledge about

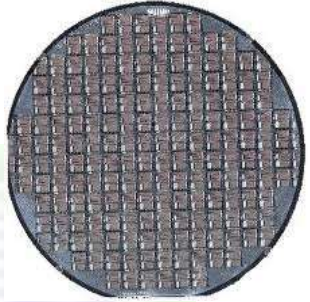
- Physical design flow
  - Logic synthesis, Floor-planning, Placement and Routing
- Experiments explore complete digital design flow of programmable ASIC through VLSI EDA tools.
- Students work from design entry using verilog code to GDSII file generation of an ASIC.

Concept MAP

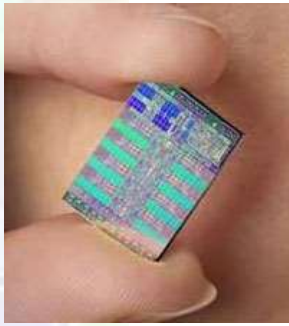

ASIC DESIGN
deals with
Consists of
Done for
ASIC Types and Construction 1
System Partitioning 2
Floorplanning and Placement 3
Routing and Circuit Extraction 4
Digital circuits using EDA tools 5
Considered for
Includes
Includes
Includes
includes
ASIC Design Flow 1.1
Types of ASIC 1.2
Semi Custom ASIC 1.3
Programmable ASIC 1.4
Programming Technology 1.5
ASIC Construction 1.6
Measurement of Partitioning 2.1
Constructive Partitioning 2.2
Iterative partitioning 2.3
Kernighan-Lin algorithm 2.4
FPGA Partitioning 2.5
Power Dissipation 2.6
Floor Planning Measurement 3.1
I/O, Power and clock planning 3.2
Measurement of Placement 3.3
Placement Algorithms 3.4
Eigen value placement 3.5
Iterative Placement algorithms 3.6
Global Routing Measurement 4.1
Global routing 4.2
Detailed routing Measurement 4.3
Detailed routing algorithms 4.4
Circuit Extraction 4.5
Layout Design rules 4.6
Digital design flow 5.1
Design and Simultion 5.2
Synthesis and Analysis 5.3
Floorplanning & Placement 5.4
Routing, Circuit Extraction and optimization 5.5
GDS-II File generation 5.6

Course Outcomes

CO1	Describe the design flow, types and the programming technologies of an ASIC and its construction.	Understand
CO2	Describe the goals, objectives, measurements and algorithms of partitioning then apply those algorithms to partition the network to meet the objectives.	Apply
CO3	Describe the goals, objectives, measurements and algorithms of floorplanning & placement then apply those algorithms to place the logic cells inside the flexible blocks of an ASIC to meet the objectives.	Apply
CO4	Describe the goals, objectives, measurements and algorithms of routing then apply those algorithms to route the channels then describing various circuit extraction formats and Investigate the issues and discover solutions in each step of physical design flow of an ASIC.	Analyze
CO5	Design an ASIC for digital circuits with ASIC design flow steps consists of simulation, synthesis, floorplanning, placement, routing, circuit extraction and generate GDSII File for fabrication of an ASIC, then analyze the ASIC to meet the performance in terms of area, speed and power using EDA tools.	Analyze

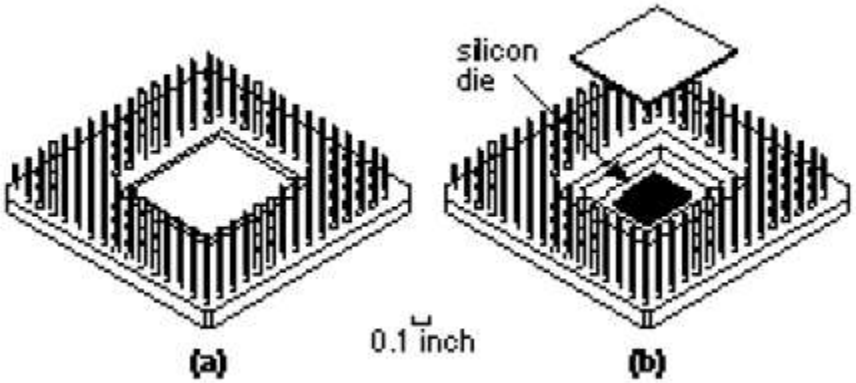


Integrated Circuit



- **Wafer :** A circular piece of pure silicon (10-15 cm in dia, wafers of 30 cm dia are expected soon)
- **Wafer Lot:** 5 ~ 30 wafers, each containing hundreds of chips(dies) depending upon size of the die
- **Die:** A rectangular piece of silicon that contains one IC design
- **Mask Layers:** Each IC is manufactured with successive mask layers(10 – 15 layers)
  - First half-dozen or so layers define transistors
  - Other half-dozen define Interconnect

Integrated Circuit (IC) in a package



(a) A pin-grid array (PGA) package.

(b) The silicon die or chip is under the package lid.

Evolution of IC

- SSI (**Small-Scale Integration)-(1962)**
  - **Tens of Transistors**
  - **NAND, NOR**

- MSI (**Medium-Scale Integration)-(late 1960)**
  - **Hundreds of Transistors**
  - **Counters**

- LSI (**Large-Scale Integration)-(mid 1970)**
  - **Tens of Thousands of Transistors**
  - **First Microprocessor**

- VLSI (**Very Large-Scale Integration)-(1980)**
  - **started Hundreds of Thousands of Transistors-several billion transistors in 2009**
  - **64 bit Microprocessor with cache memory and floating-point arithmetic units**

- ULSI (**Ultra Large-Scale Integration)-(late 1980)**
  - **More than about one million circuit elements on a single chip.**
  - **The Intel 486 and Pentium microprocessors, use ULSI technology**

IC technologies

- Bipolar
  - More accuracy
- MOS
  - Gate-Aluminium
  - Low power consumption
  - Low cost
- CMOS
  - Gate-Poly-Silicon
  - Low power consumption
  - Low cost
- BiCMOS

Types of IC

- Standard ICs
- Glue Logic-Microelectronic system design then becomes a matter of defining the functions that you can implement using standard ICs and then implementing the remaining logic functions (sometimes called glue logic) with one or more custom ICs.
- ASIC
- ASSP **(Application-Specific Standard Products)**

ASIC and Non ASIC

- Examples of **ICs that are *not ASICs*** *include standard parts such as:*
  - *memory chips* sold as a commodity item—ROMs, DRAM, and SRAM; microprocessors;
  - TTL or TTL-equivalent ICs at SSI, MSI, and LSI levels.

- Examples of **ICs that *are ASICs*** *include:*
  - *a chip for a toy bear that talks;*
  - *a chip for a* satellite;
  - a chip designed to handle the interface between memory and a microprocessor for a workstation CPU;
  - a chip containing a microprocessor as a cell together with other logic.

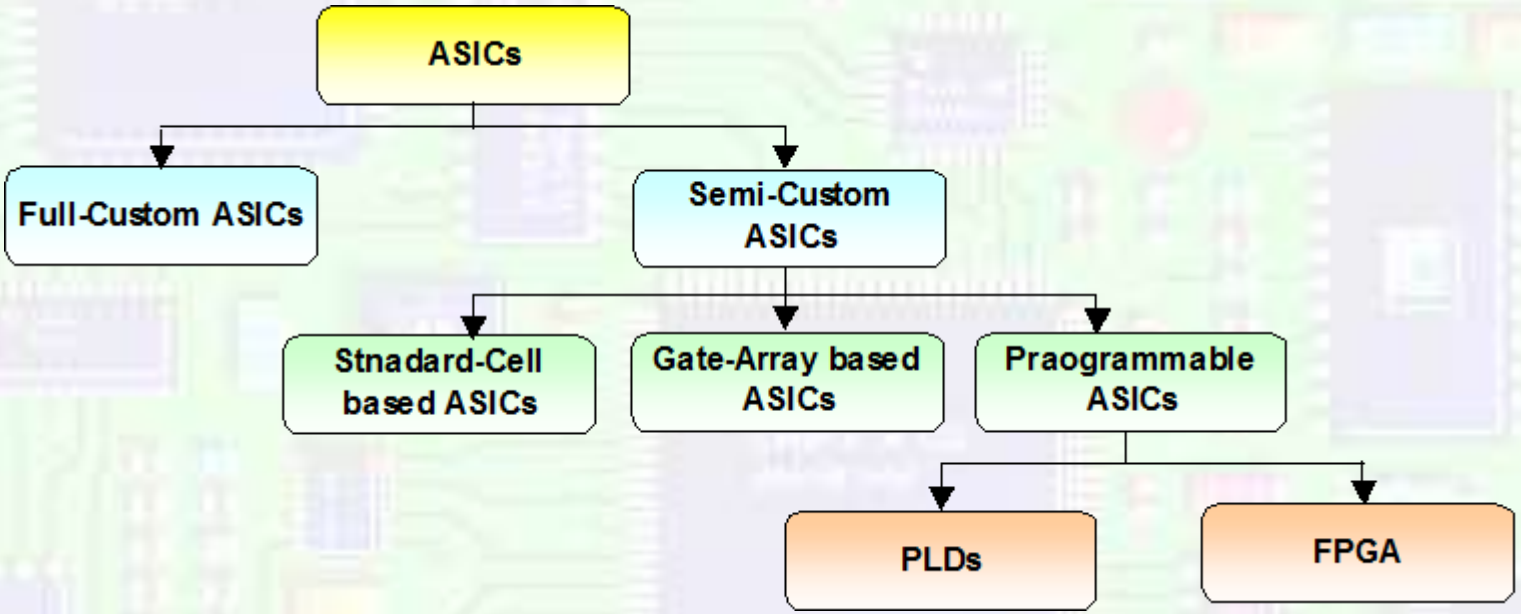
- **ASSP** (two ICs that might or might not be considered ASICs)
  - controller chip for a PC and a chip for a modem.
  - Both of these examples are specific to an application (shades of an ASIC) but are sold to many different system vendors (shades of a standard part). ASICs such as these are sometimes called **application-specific standard products (ASSPs).**

Measurement of IC

- Gate Equivalent
  - Number of gates or transistors
  - Gate refer to two input NAND Gate
  - In CMOS, each NAND gate consist of 4 transistors
  - Example : 10k gate IC
  - (10,000 two-input NAND gates or 40,000 transistors in CMOS)

- Feature Size (smallest feature size = λ)
  - Half of smallest transistor length
  - Example: 0.5µm IC
  - Feature size, λ = 0.25µm

Types of ASICs – Cont'd



- Full-Custom ASICs: Possibly all logic cells and all mask layers customized
- Semi-Custom ASICs: all logic cells are pre-designed and some (possibly all) mask layers customized

Types of ASICs – Cont'd

❑Full-Custom ASICs

- ❖ Include some (possibly all) customized logic cells
- ❖ Have all their mask layers customized
- ❖Manufacturing lead time is typically 8 weeks (time taken to make the IC does not include design time)
- ❖ Full-custom ASIC design makes sense only
  - ✓ *When no suitable existing libraries exist or*
  - ✓ *Existing library cells are not fast enough or*
  - ✓ *The available pre-designed/pre-tested cells consume too much power that a design can allow or*
  - ✓ *The available logic cells are not compact enough to fit or*
  - ✓ *ASIC technology is new or/and so special that no cell library exits.*

Types of ASICs – Cont'd

❑Full-Custom ASICs

- ❖ *Advantages:*
  - ❖Offer highest performance
  - ❖lowest cost (smallest die size)

- ❖Disadvantages
  - ❖Increased design time
  - ❖Increased Complexity
  - ❖ Higher design cost
  - ❖ Higher risk.

- ❖ **Some Examples:**
  - ❖Microporcessor,
  - ❖High-Voltage Automobile Control Chips
  - ❖ Ana-Digi Communication Chips
  - ❖ Sensors and Actuators

Types of ASICs – Cont'd

❑ Semi-Custom ASICs

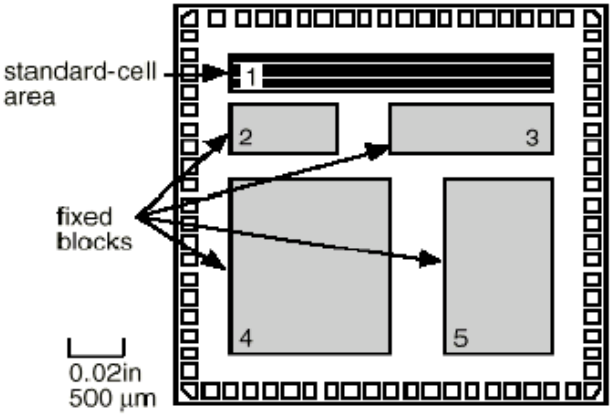
❖ Standard-Cell based ASICs (CBIC- "sea-bick")

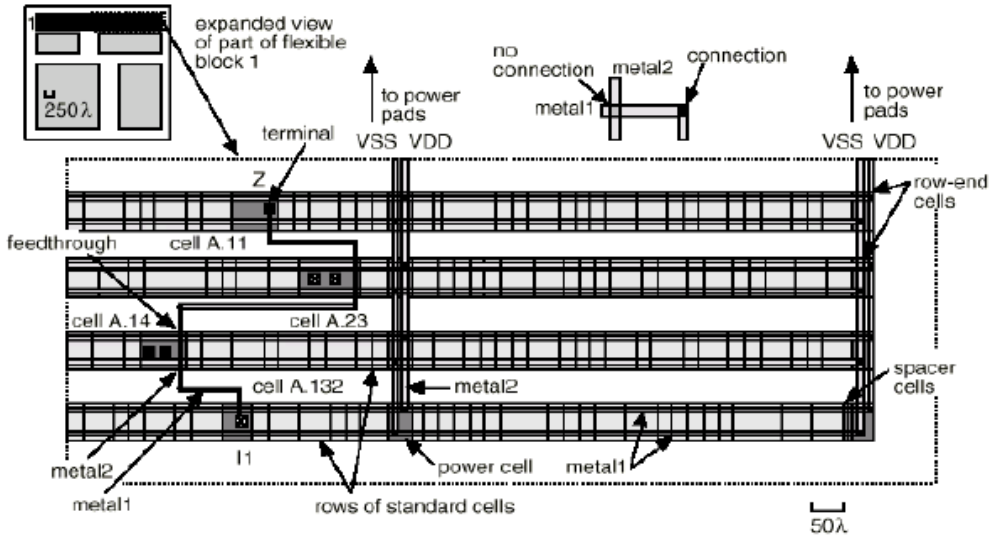
- ✓ Use predesigned logic cells (Called standard cells) from
  - ✓standard cell libraries
  - ✓other mega-cells (Microcontroller or Microprocessors)
  - ✓full-custom blocks
  - ✓System-Level Macros(SLMs)
  - ✓ Functional Standard Blocks (FSBs)
  - ✓ cores etc
- ✓ Get all mask layers customized- transistors and interconnect
- ✓ Manufacturing lead time is about 8 weeks
- ✓Custom blocks can be embedded

Types of ASICs – Cont'd

- **Semi-Custom ASICs** – Cont'd
  - **Standard-Cell based ASICs**

(CBIC- "sea-bick") – Cont'd





- 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**

Types of ASICs – Cont'd

❑ Standard Cell in Flexible block of CBIC

- Std cell in library is constructed using full-custom design methodology-
  - Same performance and flexibility but reduce time and risk.
- ASIC designer defines only placement of standard cells
  - It can be placed anywhere on silicon.

❑ Flexible blocks in CBIC

- Standard cells are designed like bricks in a wall.
- Groups of standard cells fit horizontally to form rows.
- The rows stack vertically to form flexible blocks- reshape during design
- Flexible blocks connected with other std cell blocks or full custom block

Types of ASICs – Cont'd

❑ Wiring cells in Standard Cell based ASICs

•Feedthrough cell:

•Piece of metal that is used to pass a signal through a cell or to a space in a cell waiting to be used as a feedthrough

•Spacer cells

•The width of each row of standard cells is adjusted so that they may be aligned using **spacer cells .**

•Row end cells

•**The power buses, or rails, are then connected to** additional vertical power rails using **row-end cells at the aligned ends of each** standard-cell block.

• Power cells

•If the rows of standard cells are long, then vertical power rails can also be run in metal2 through the cell rows using special **power cells that just connect** to VDD and GND.

•Usually the designer manually controls the number and width of the vertical power rails connected to the standard-cell blocks during physical design.

Types of ASICs – Cont'd

- **Advantages of CBIC**
  - Save time, money, reduce risk
  - Standard cell optimized individually for speed or area
- **Disadvantages of CBIC:**
  - Time to design standard cell library
  - Expenses of designing std cell library
  - Time needed to fabricate all layers of the ASIC for new design

Types of ASICs – Cont'd

❑ Semi-Custom ASICs – Cont'd

❖ Gate Array based ASICs

- ❖**Transistors are predefined** on the silicon wafer
- ❖Predefined pattern of transistors on a gate array is **base array.**
- ❖Smallest element repeated to form base array is **base cell.**
- ❖Only the top few layers of metal, which define the interconnect between transistors, are defined by the designer using custom masks.It is often called a **masked gate array (MGA).**
- ❖**Less turnaroundtime:** fewdays or couple of weeks.

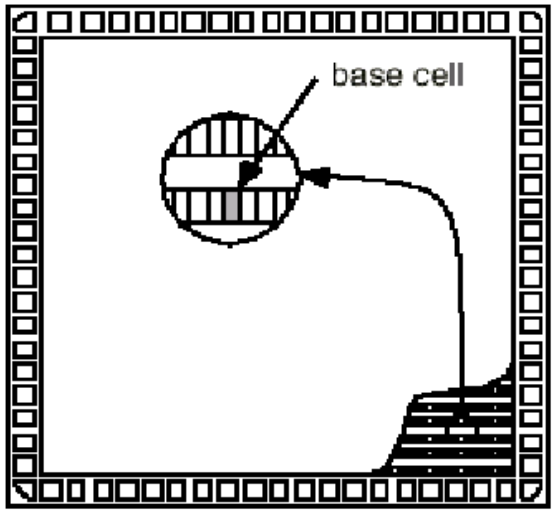
A **gate array**, **masked gate array**, **MGA**, or **prediffused array** uses **macros** (**books**) to reduce **turnaround time** and comprises a **base array** made from a **base cell** or **primitive cell**. There are three types:

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

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

❖Similar to CBIC –but here space is fixed

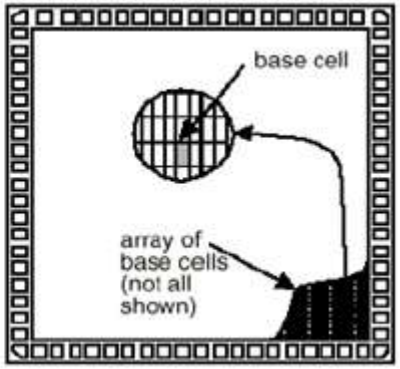


Types of ASICs – Cont'd

❑ Semi-Custom ASICs – Cont'd

❖ Chanelless Gate Array ASIC

- 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.



•The key difference between a channelless gate array and channeled gate array

•there are **no predefined areas set aside for routing between cells on a channelless gate array**.

•Use an **area of transistors for routing in a channelless array**, we do not make any contacts to the devices lying underneath; we simply leave **the transistors unused**.

•The logic density—the amount of logic that can be implemented in a given silicon area is higher for channelless gate array

Types of ASICs – Cont'd

- **Semi-Custom ASICs –** Cont'd
  - **Difference between Channeled and Chanelless Gate Array ASIC**

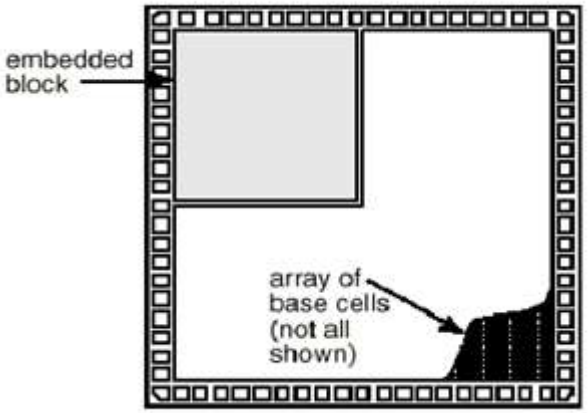
•This is usually attributed to the difference in structure between the two types of array. In fact, the difference occurs because the **contact mask is customized in a channelless gate array,** but is not usually customized in a channeled gate array. This leads to **denser cells in the channelless architectures.**

•Customizing the contact layer in a channelless gate array allows us to increase the density of gate-array cells because we can route over the top of unused contact sites.

Types of ASICs – Cont'd

❑ Semi-Custom ASICs – Cont'd

❖ Structured Gate Array based ASICs



- 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.

•An embedded gate array or structured gate array (also known as masterslice or masterimage) combines some of the features of CBICs and MGAs.

•One of the **disadvantages of the MGA** is the **fixed gate-array base cell**. This makes the **implementation of memory, for example, difficult and inefficient.**

• In an embedded gate array we set aside **some of the IC area and dedicate it to a specific function.**

• This embedded area either can contain **a different base cell that is more suitable for building memory cells, or it can contain a complete circuit block, such as a microcontroller.**

Channelled gate array

Adv: Specific space for interconnection

Disadv: compared to CBIC space is not adjustable

Channelless gate array

Adv :

- Logic density is higher for channelless gate array
- Contact layers are customized

Disadv:

- No specific area for routing
- Rows of transistors used for routing are not used for other purpose.

Structured Gate Array

Adv:

- Embedded gate array set in some of IC area and dedicate to specific function-customized.
- Increase area efficiency, performance of CBIC
- low cost and fast turn around of MGA

Disadv:

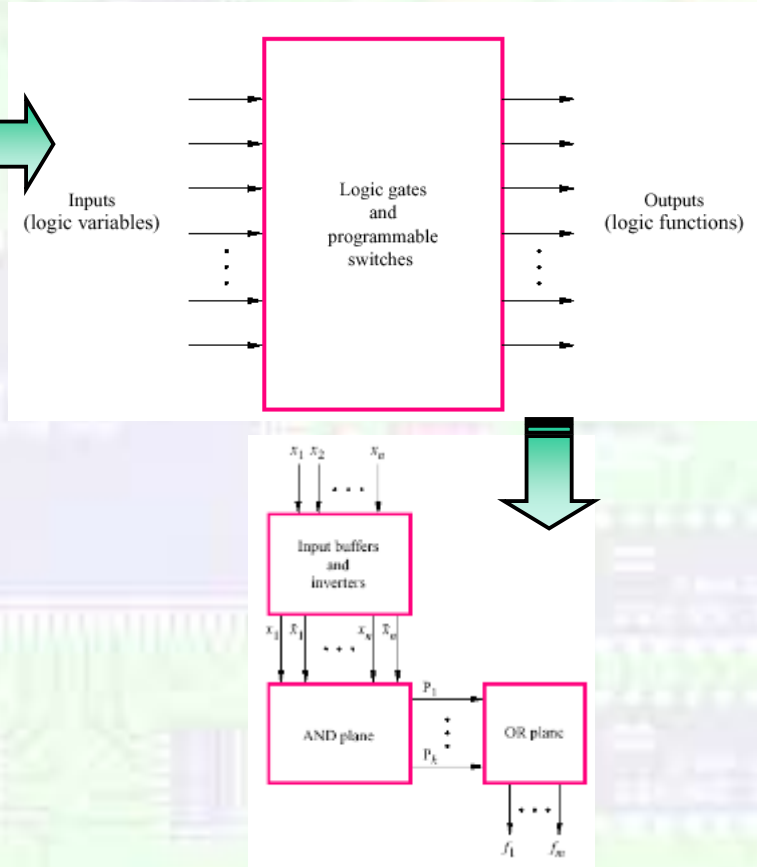
Embedded function is fixed

Types of ASICs – Cont'd

❑ Semi-Custom ASICs – Cont'd

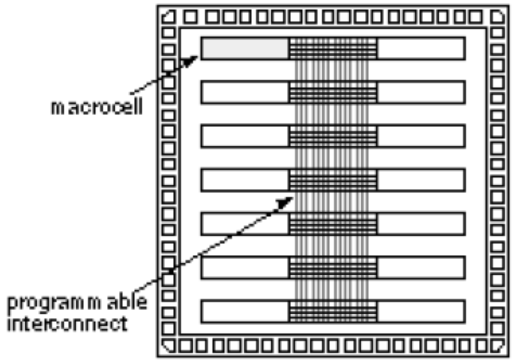
❖ Programmable ASICs

- ✓ **PLDs -** PLDs are low-density devices which contain 1k – 10 k gates and are available both in bipolar and CMOS technologies **[PLA, PAL or GAL]**
- ✓ **CPLDs or FPLDs or FPGAs -** FPGAs combine architecture of gate arrays with programmability of PLDs.
- ✓**User Configurable**
- ✓ **Contain Regular Structures** - circuit elements such as AND, OR, NAND/NOR gates, FFs, Mux, RAMs,
- ✓**Allow Different Programming Technologies**
- ✓ **Allow both Matrix and Row-based Architectures**



Programmable Logic Devices

- Programmable logic devices (PLDs) are standard ICs
  - Available in standard configurations
  - Sold in very high volume to many different customers.
  - PLDs may be configured or programmed to create a part customized to a specific application
  - PLDs use different technologies to allow programming of the device.



- The important features of PLDs:
  - No customized mask layers or logic cells
  - Fast design turnaround
- Structure of programmable logic device (PLD)
  - 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

Examples of PLD

- The simplest type of programmable IC is a **read-only memory (ROM).**

- The most common types of ROM use a metal fuse that can be blown permanently

  (a **programmable ROM or PROM).**

- **An electrically programmable ROM (EPROM) ,** uses programmable MOS transistors whose characteristics are altered by applying a high voltage.

- **Erasable PROM**
  - Erase an EPROM either by using another high voltage (an **electrically erasable PROM , or EEPROM)**
  - **Exposing the device** to ultraviolet light (**UV-erasable PROM , or UVPROM).**

- There is another type of ROM that can be placed on any ASIC—a **maskprogrammable ROM (mask-programmed ROM or masked ROM).**
  - **A masked ROM** is a regular array of transistors permanently programmed using custom mask patterns.

- An embedded masked ROM is thus a large, specialized, logic cell.

Type of PLDs-PLA and PAL

- Place a logic array as a cell on a custom ASIC. This type of logic array is called a **programmable logic array** (PLA).

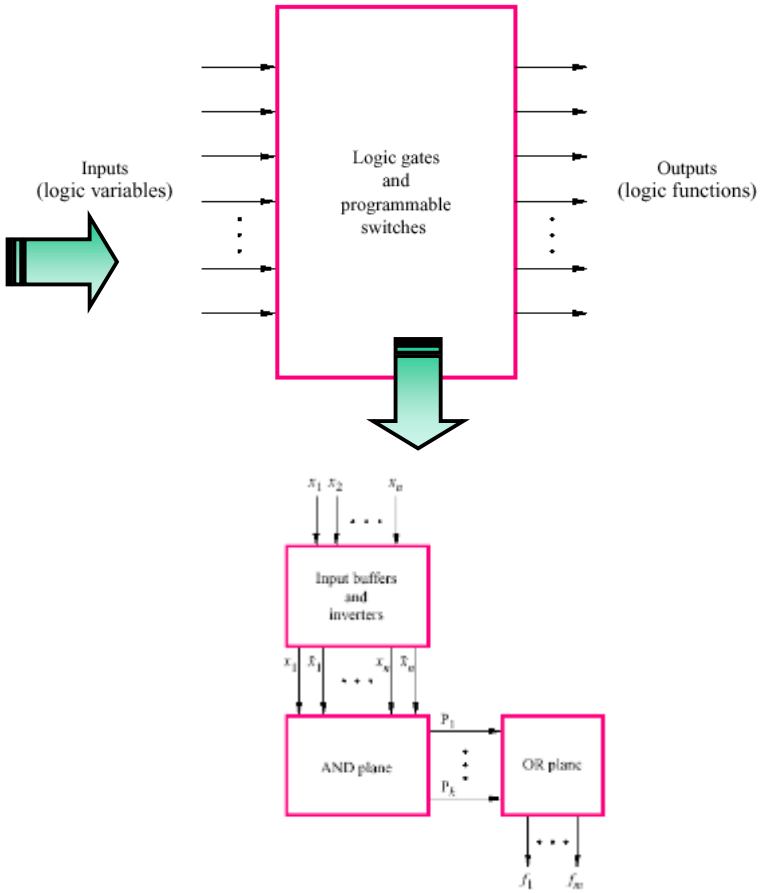
- A **PLA** has a programmable AND logic array, or **AND plane , followed by a programmable OR logic array, or OR plane**

- **A PAL has a programmable AND plane and, in contrast to a PLA, a fixed** OR plane.

- Depending on how the PLD is programmed, we can have an
  - **Erasable PLD (EPLD),**
  - **Mask-programmed PLD (called as masked PLD but usually just PLD).**

- The first bipolar based PALs, PLAs, and PLDs used p**rogrammable fuses or links**.

- CMOS PLDs usually employ floating-gate transistors

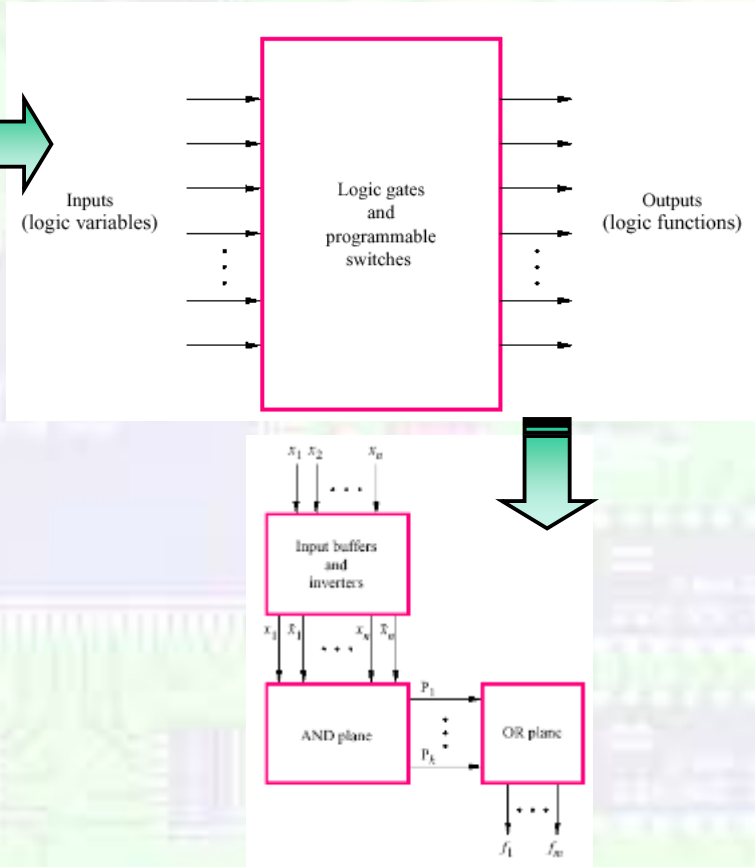


Types of ASICs – Cont'd

❑ Semi-Custom ASICs – Cont'd

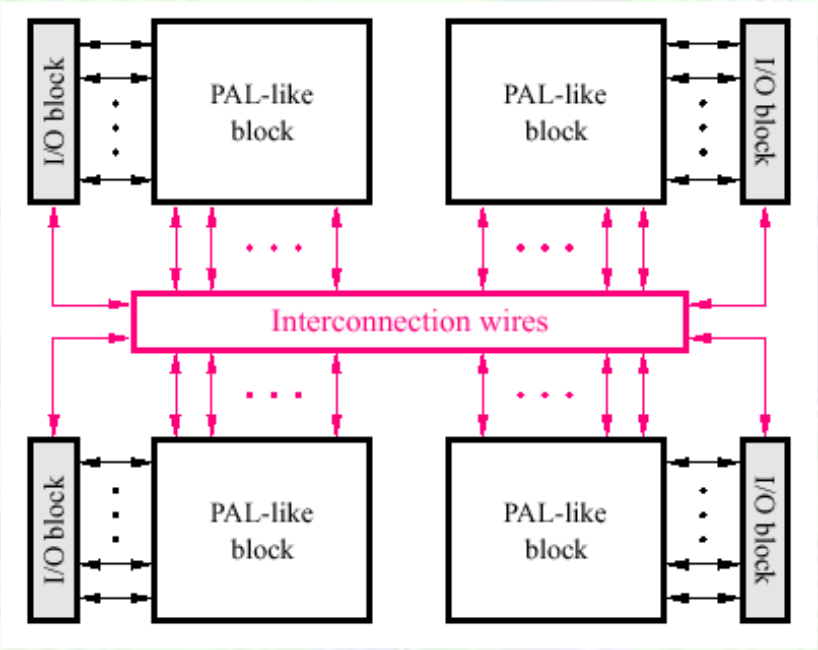
❖ Programmable ASICs

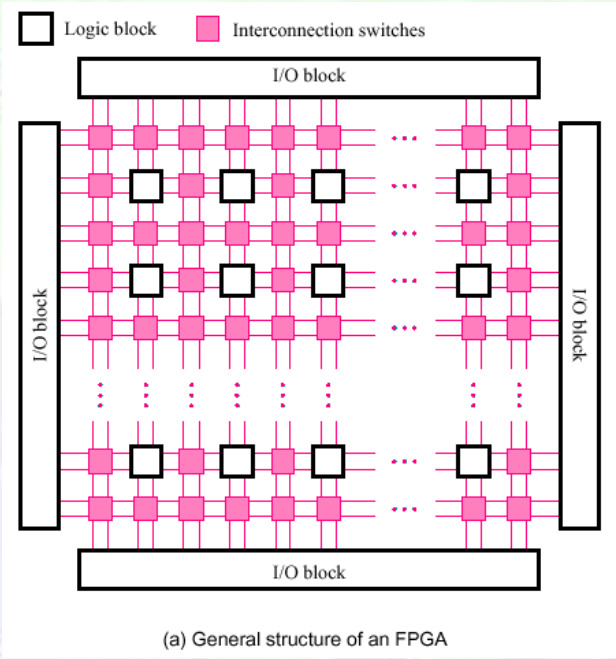
- ✓ **PLDs -** PLDs are low-density devices which contain 1k – 10 k gates and are available both in bipolar and CMOS technologies **[PLA, PAL or GAL]**
- ✓ **CPLDs or FPLDs or FPGAs -** FPGAs combine architecture of gate arrays with programmability of PLDs.
- ✓**User Configurable**
- ✓ **Contain Regular Structures** - circuit elements such as AND, OR, NAND/NOR gates, FFs, Mux, RAMs,
- ✓**Allow Different Programming Technologies**
- ✓ **Allow both Matrix and Row-based Architectures**



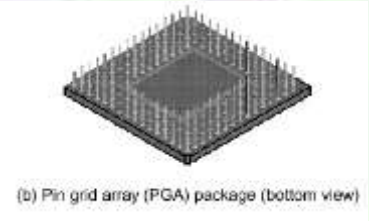
Types of ASICs – Cont'd

- **Semi-Custom ASICs** – Cont'd
  - **Programmable ASICs** - Cont'd
    - **Structure of a CPLD / FPGA**





(a) General structure of an FPGA



(b) Pin grid array (PGA) package (bottom view)

Essential characteristics of FPGA

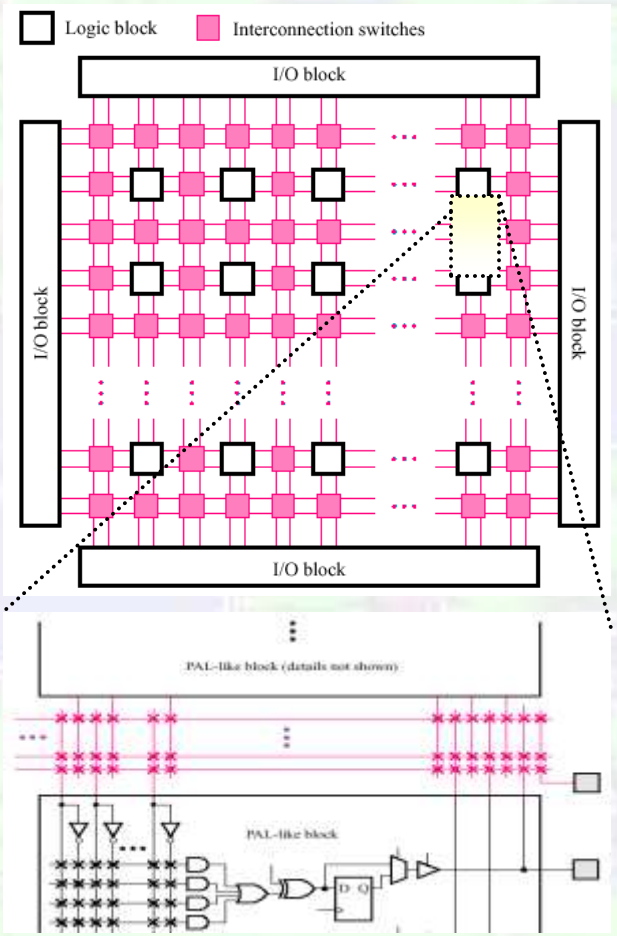
- None of the mask layers are customized
- A method of programming the basic logic cells and interconnect
- Core-regular array of Programmable basic logic cells implement combinational or sequential logic
- Matrix of programmable interconnects surround the basic logic cells
- Programmable I/O cells surround the core
- **Design turnaround is few hours.**
- **Difference between PLD and FPGA:**
  - FPGA are larger and more complex than PLD

Why FPGA-based ASIC Design?

- **Choice is based on Many Factors ;**
  - **Speed**
  - **Gate Density**
  - **Development Time**
  - **Prototyping and Simulation Time**
  - **Manufacturing Lead Time**
  - **Future Modifications**
  - **Inventory Risk**
  - **Cost**

Different Categorizations of FPGAs

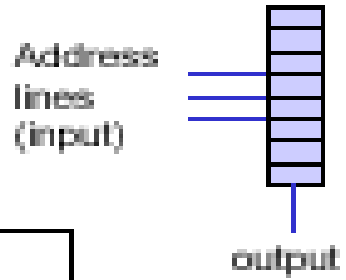
- **Based on Functional Unit/Logic Cell Structure**
  - **Transistor Pairs**
  - **Basic Logic Gates: NAND/NOR**
  - **MUX**
  - **Look –up Tables (LUT)**
  - **Wide-Fan-In AND-OR Gates**
- **Programming Technology**
  - **Anti-Fuse Technology**
  - **SRAM Technology**
  - **EPROM Technology**
- **Gate Density**
- **Chip Architecture** **(Routing Style)**



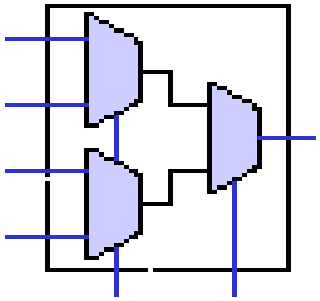
Different Types of Logic Cells

FPGA Architecture: Functional Units

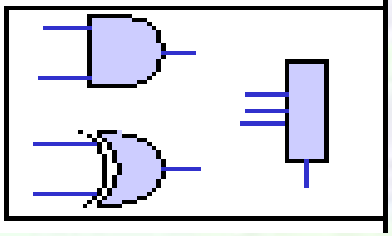
- Functional units
  - RAM blocks (Xilinx): implement function truth table



  - Multiplexers (Actel): build Boolean functions using muxes



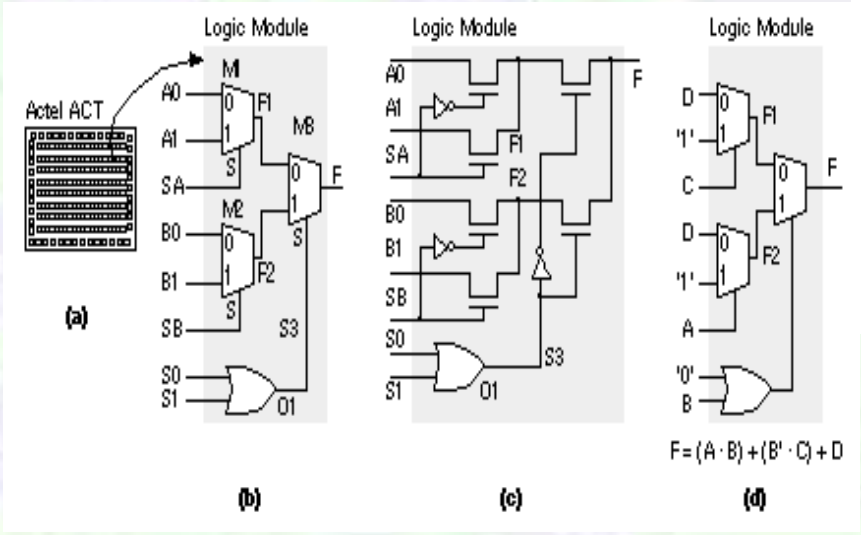
  - Logic gates, flip-flops: Such as carry chains. Used for high-performance computations

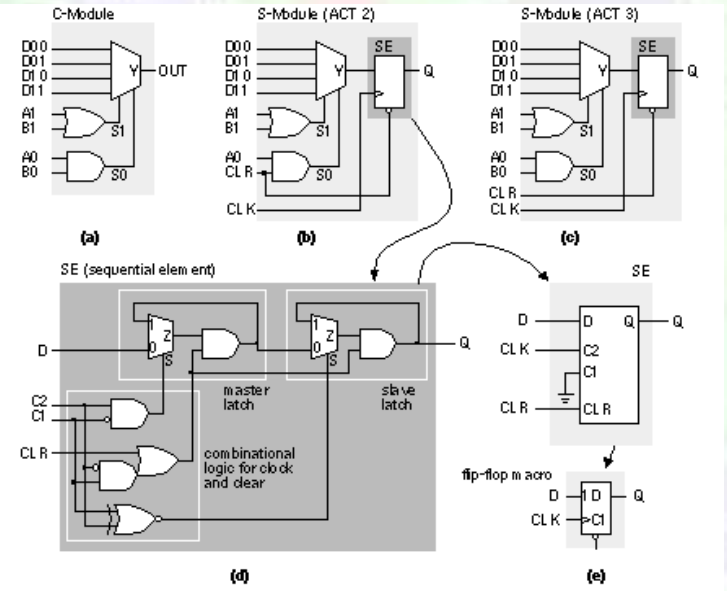


Different Types of Logic Cells – Cont'd

❑ Actel Act Logic Module Structure

- ❖ Use Antifuse Programming Tech.
- ❖ Based on Channeled GA Architecture
- ❖ Logic Cell is MUX which can be configured as multi-input logic gates

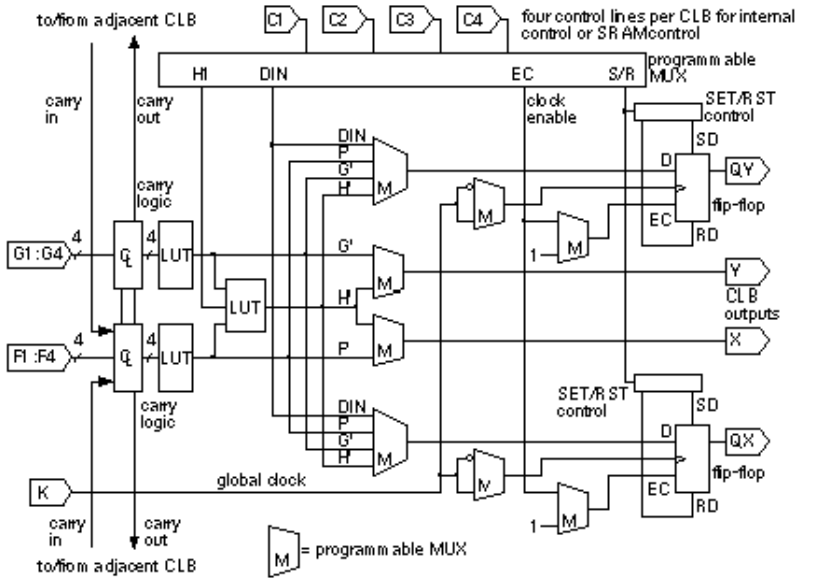


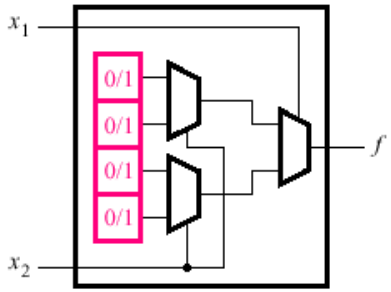


The Actel ACT 2 and ACT 3 Logic Modules. (a) The C-Module for combinational logic. (b) The ACT 2 S-Module. (c) The ACT 3 S-Module. (d) The equivalent circuit (without buffering) of the SE (sequential element). (e) The sequential element configured as a positive-edge–triggered D flip-flop. (Source: Actel.)

Different Types of Logic Cells – Cont'd

- **Xilinx XC4000 CLB Structure**

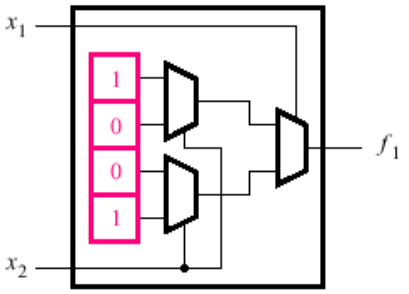




(a) Circuit for a two-input LUT

x_1	x_2	f_1
0	0	1
0	1	0
1	0	0
1	1	1

(b) $f_1 = \bar{x}_1\bar{x}_2 + x_1x_2$

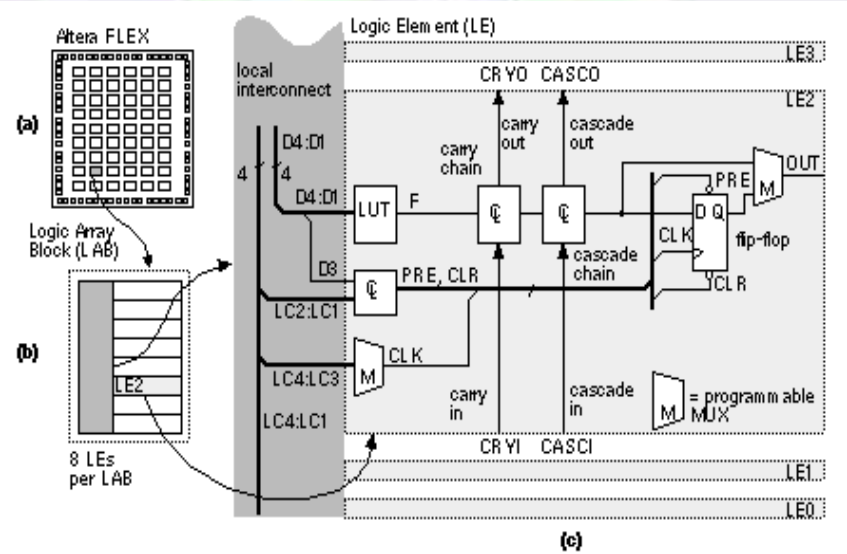


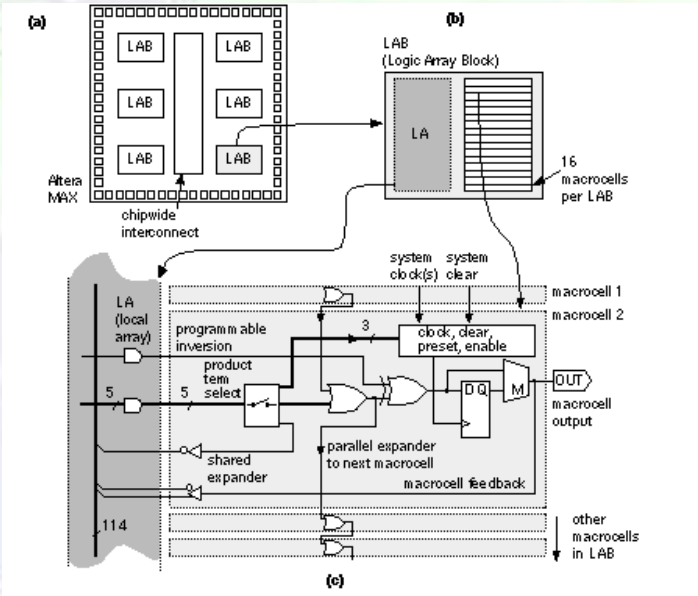
(c) Storage cell contents in the LUT

Different Types of Logic Cells – Cont'd

❑ Altera Flex / Max Logic Element Structure

❖**Flex 8k/10k Devices – SRAM Based LUTs, Logic Elements (LEs) are similar to those used in XC5200 FPGA**





The Altera MAX architecture. (a) Organization of logic and interconnect. (b) A MAX family LAB (Logic Array Block). (c) A MAX family macrocell. The macrocell details vary between the MAX families—the functions shown here are closest to those of the MAX 9000 family

Different Types of Logic Cells – Cont'd

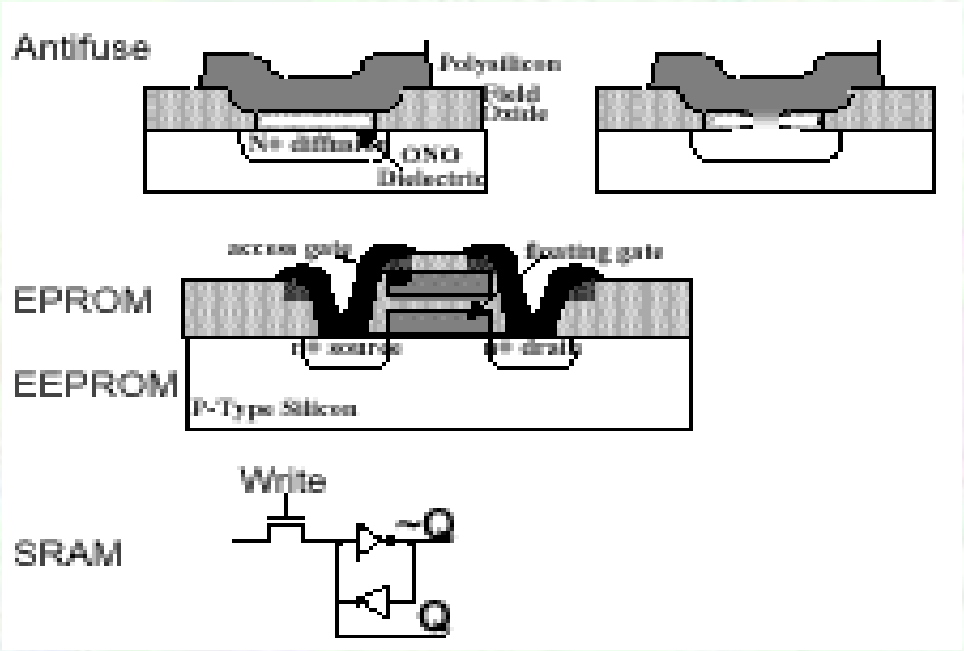
To SUMMARIZE, FPGAs from various vendors differ in their

- **Architecture (Row Based or Matrix Based Routing Mechanism)**
- **Gate Density (Cap. In Equiv. 2- Input NAND Gates)**
- **Basic Cell Structure**
- **Programming Technology**

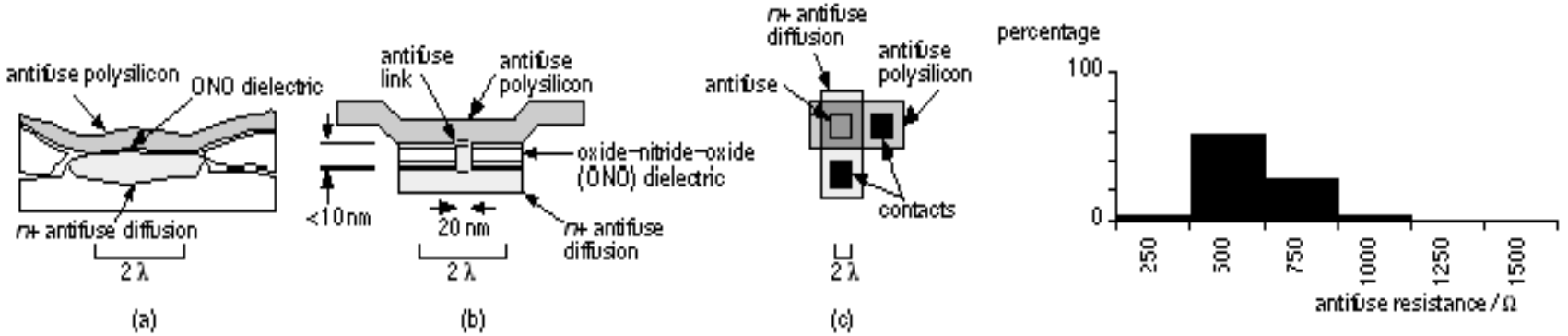
Vendor/ Product	Architechture	Capacity	Basic Cell	Programming Technology
Actel	Gate Array	2-8 k	MUX	Antifuse
QuickLogic	Matrix	1.2-1.8 k	MUX	Antifuse
Xilinx	Matrix	2-10 k	RAM Block	SRAM
Altera	Extended PLA	1- 5 k	PLA	EPROM
Concurrent	Matrix	3-5 k	XOR, AND	SRAM
Plessy	SOG	2-40 k	NAND	SRAM

Programming Technologies

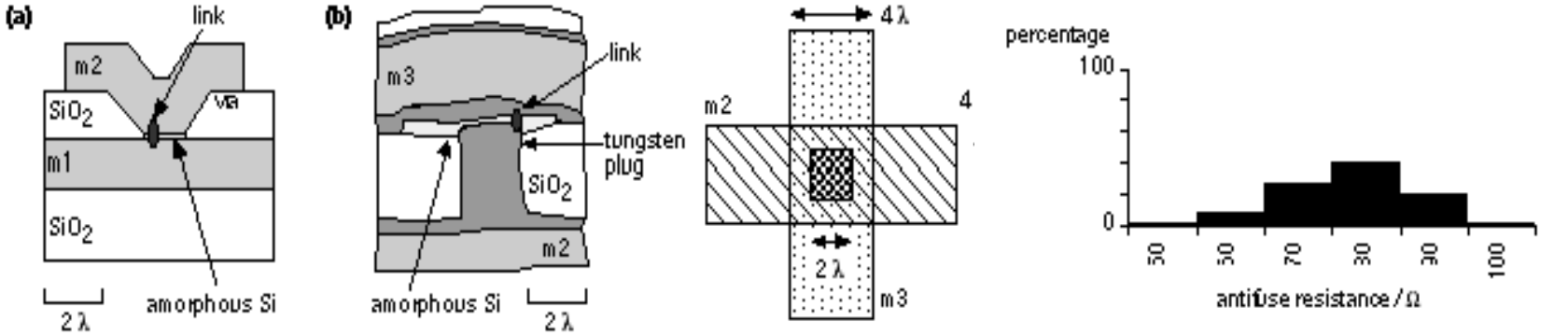
- Three Programming Technologies
  - The Antifuse Technology
  - Static RAM Technology
  - EPROM and EEPROM Technology



Antifuse



[a] [b]

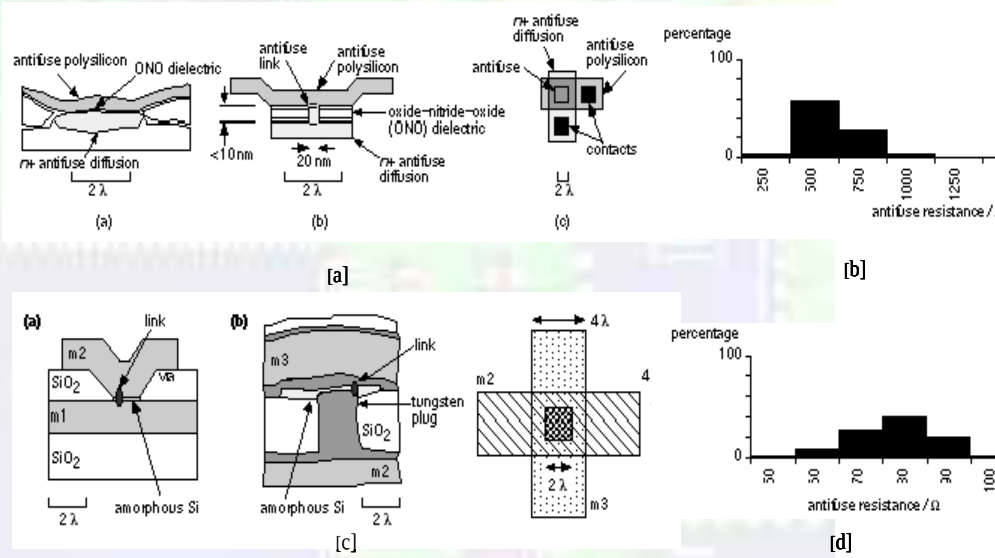


[c] [d]

Programming Technologies – Cont'd

❑ The Antifuse Technology

- ❖Invented at Stanford and developed by **Actel**
- ❖Opposite to regular fuse Technology
- ❖Normally an open circuit until a programming current (about 5 mA) is forced through it
- ❖Two Types:
  - ❖**Actel's PLICE** [Programmable Low-Impedance Circuit Element]- A High-Resistance Poly-Diffusion Antifuse
  - ❖**QuickLogic's** Low-Resistance metal-metal antifuse [**ViaLink**] technology
    - ❖Direct metal-2-metal connections
    - ❖Higher programming currents reduce antifuse resistance
- ❖Disadvantages:
  - ❖Unwanted Long Delay
  - ❖**OTP** Technology



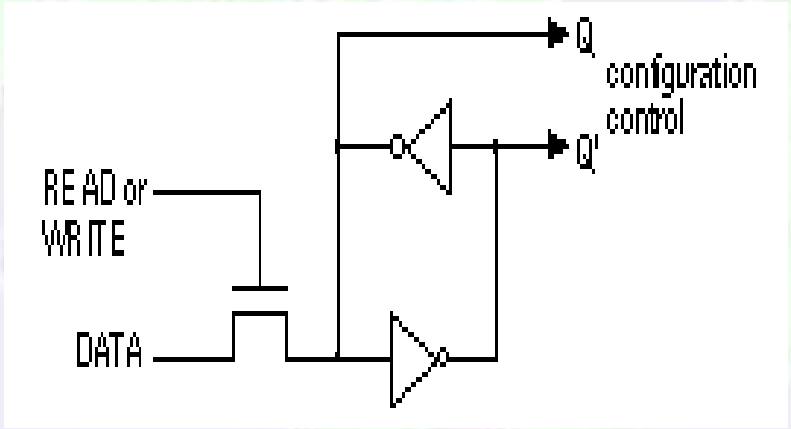
Actel Antifuse [b] Actel Antifuse Resistance [c] QuickLogic Antifuse [d] QL Antifuse Resistance

Programming Technologies – Cont'd

❑ Static RAM Technology

❖ SRAM cells are used for

- ✓As Look-Up Tables (LUT) to implement logic (as Truth Tables)
- ✓As embedded RAM blocks (for buffer storage etc.)
- ✓As control to routing and configuration switches



❖ Advantages

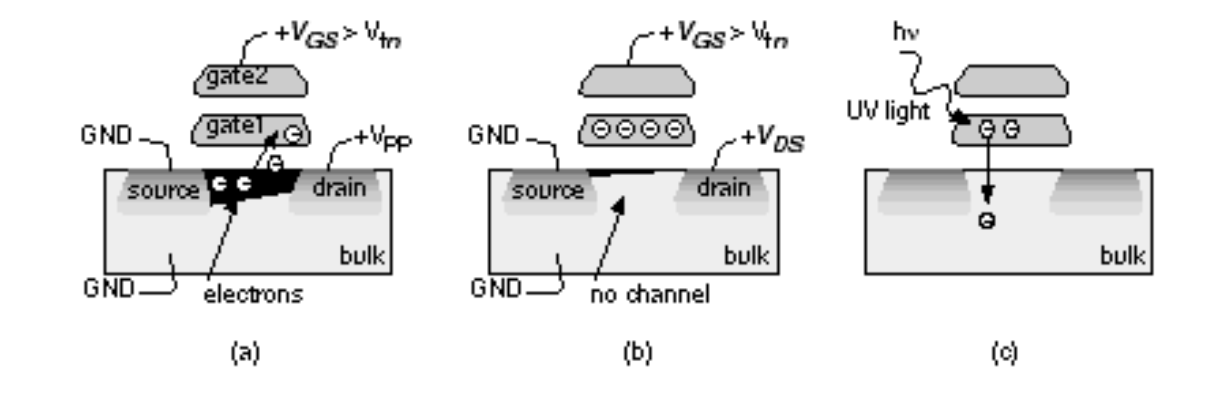
- ✓Allows In-System Programming (ISP)
- ✓Suitable for Reconfigurable HW

❖ Disadvantages

- ✓Volatile – needs power all the time / use PROM to download configuration data

Programming Technologies – Cont'd

❑ EPROM and EEPROM Technology-



- ❖ **EPROM Cell is almost as small as Antifuse**
- ❖ **Floating-Gate Avalanche MOS (FAMOS) Tech.**
  - ✓Under normal voltage, transistor is on
  - ✓With Programming Voltage applied, we can turn it off (configuration) to implement our logic
  - ✓Exposure to UV lamp (one hour) we can erase the programming
  - ✓Use EEPROM for quick reconfiguration, also, ISP possible

Programming Technologies – Cont'd

- **Summary Sheet**

Programmable ASIC technologies

	Actel	Xilinx LCA[1]	Altera EPLD	Xilinx EPLD
Programming technology	Poly–diffusion antifuse, PLICE	Erasable SRAM ISP	UV-erasable EPROM (MAX 5k) EEPROM (MAX 7/9k)	UV-erasable EPROM
Size of programming element	Small but requires contacts to metal	Two inverters plus pass and switch devices. Largest.	One n-channel EPROM device. Medium.	One n-channel EPROM device. Medium.
Process	Special: CMOS plus three extra masks.	Standard CMOS	Standard EPROM and EEPROM	Standard EPROM
Program-ming method	Special hardware	PC card, PROM, or serial port	ISP (MAX 9k) or EPROM program-mer	EPROM program-mer

	QuickLogic	Crosspoint	Atmel	Altera FLEX
Programming technology	Metal–metal antifuse, ViaLink	Metal–polysilicon antifuse	Erasable SRAM. ISP.	Erasable SRAM. ISP.
Size of programming element	Smallest	Small	Two inverters plus pass and switch devices. Largest.	Two inverters plus pass and switch devices. Largest.
Process	Special, CMOS plus ViaLink	Special, CMOS plus antifuse	Standard CMOS	Standard CMOS
Program-ming method	Special hardware	Special hardware	PC card, PROM, or serial port	PC card, PROM, or serial port

[1]Lucent (formerly AT&T) FPGAs have almost identical properties to the Xilinx LCA family

ASIC Design Process

S-1 Design Entry: Schematic entry or HDL description

S-2: Logic Synthesis: Using Verilog HDL or VHDL and Synthesis tool, produce ***a netlist***-logic cells and their interconnect detail

S-3 System Partitioning: Divide a large system into ASIC sized pieces

S-4 Pre-Layout Simulation: Check design functionality

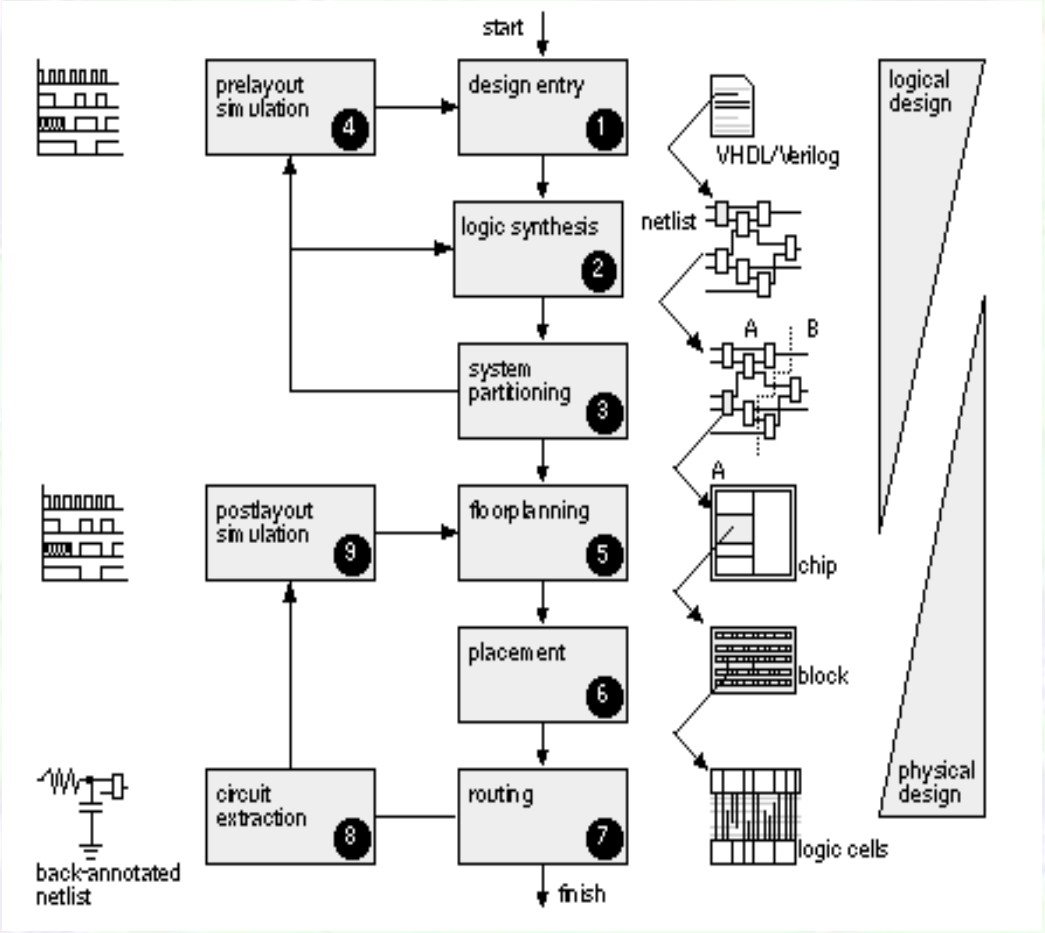
S-5 Floorplanning: Arrange netlist blocks on the chip

S-6 Placement: Fix cell locations in a block

S-7 Routing: Make the cell and block interconnections

S-8 Extraction: Measure the interconnect R/C cost

S-9 Post-Layout Simulation



ASIC Design Flow

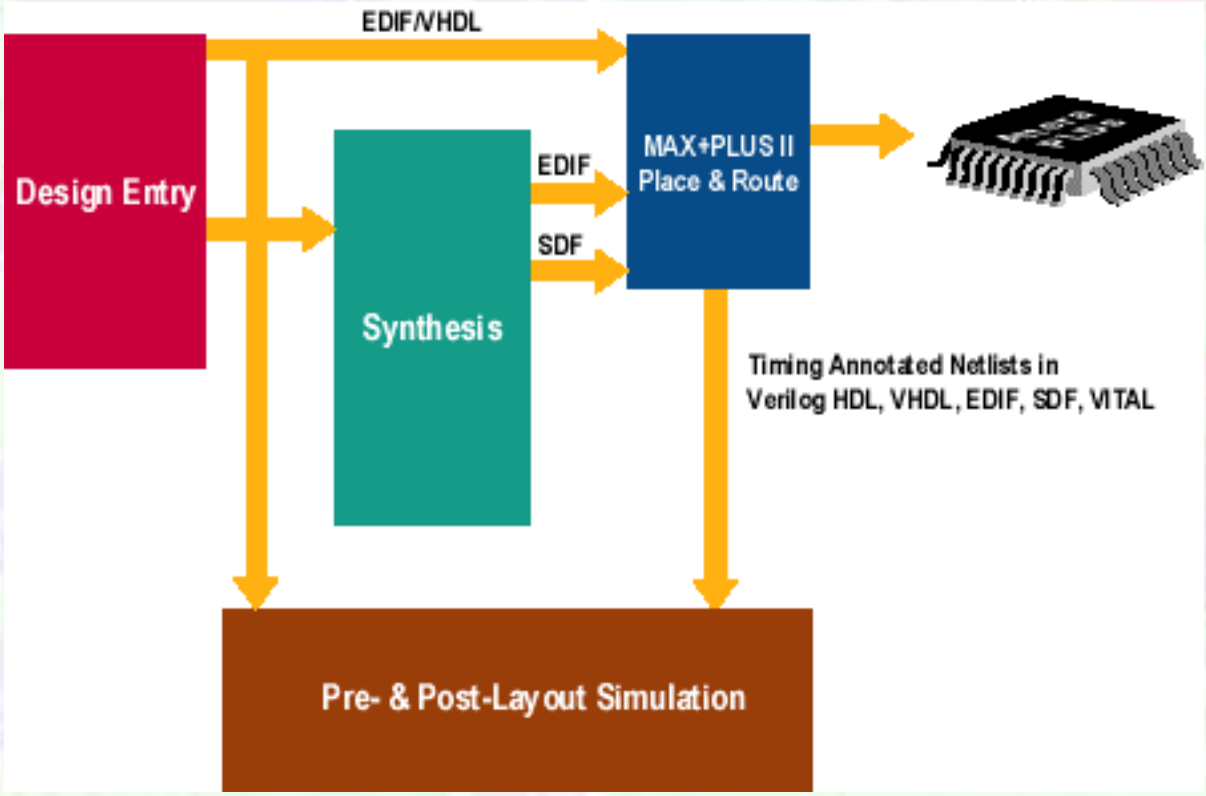
1.*Design entry.*	*Enter the design into an ASIC design system, either using a* ***hardware description language (HDL) or schematic entry .***
2. *Logic synthesis.*	*Use an HDL (VHDL or Verilog) and a logic synthesis tool to p*roduce a **netlist —a description of the logic cells and their connections.**
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.

Steps 1–4 are part of **logical design ,** and steps 5–9 are part of physical design **.**

- There is some overlap. For example, s**ystem partitioning might be considered as either logical or physical design.** To put it another way, when we are performing system partitioning we have to consider both logical and physical factors.

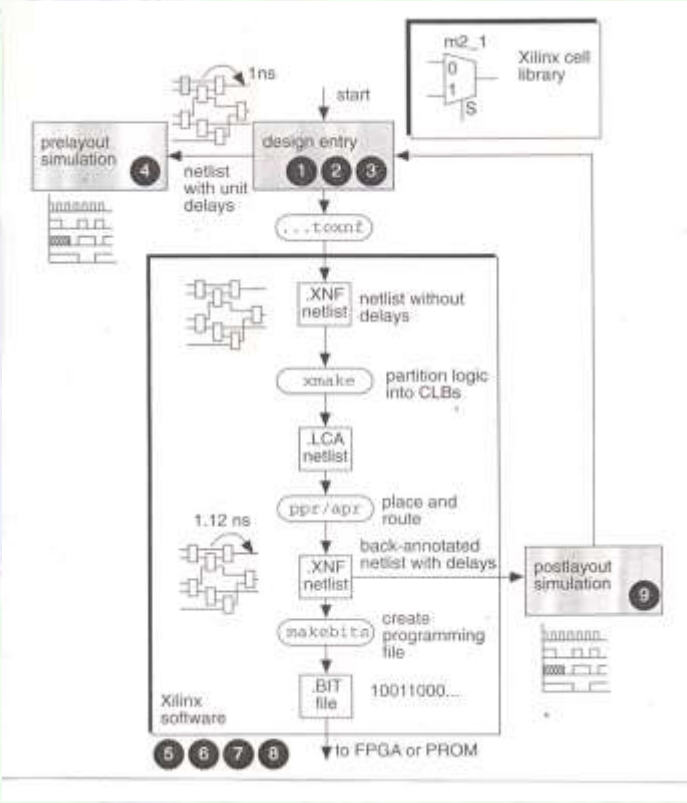
ASIC Design Process – Cont'd

❑ **Altera FPGA Design Flow – A Self-Contained System that does all from Design Entry, Simulation, Synthesis, and Programming of Altera Devices**



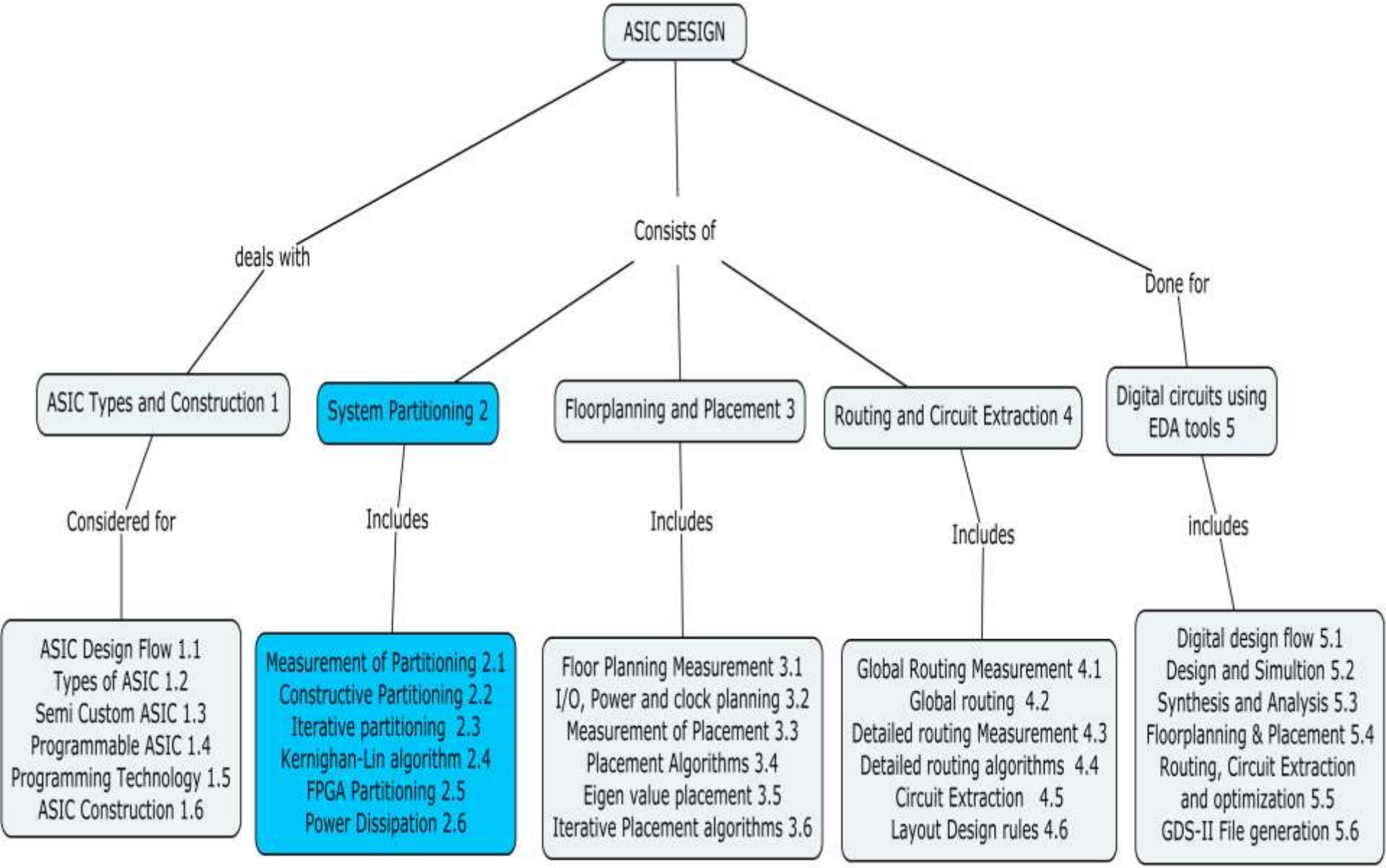
ASIC Design Process – Cont'd

- **Xilinx FPGA Design Flow – Allows Third-Party Design Entry SW, Accepts their generated netlist file as an input**
  - **Use Pin2xnf and wir2xnf SW to convert the netlist file to .XNF**
  - **xnfmap and xnfmerge programs convert .xnf files to create a unified netlist file (Nand/Nor Gates) .MAP file are generated**
  - **map2lca program does fitters job, produces un-routed .LCA file**
  - **apr or ppr SW does the routing job, post-layout netlist generated**
  - **makebits SW generates BIT files**



Module II
System Partitioning

Dr.(Mrs).D.Gracia Nirmala Rani
Assistant Professor
ECE Department
Thiagarajar College of Engineering
Madurai-15
Email : gracia@tce.edu


ASIC DESIGN
deals with
Consists of
Done for
ASIC Types and Construction 1
System Partitioning 2
Floorplanning and Placement 3
Routing and Circuit Extraction 4
Digital circuits using EDA tools 5
Considered for
Includes
Includes
Includes
includes
ASIC Design Flow 1.1
Types of ASIC 1.2
Semi Custom ASIC 1.3
Programmable ASIC 1.4
Programming Technology 1.5
ASIC Construction 1.6
Measurement of Partitioning 2.1
Constructive Partitioning 2.2
Iterative partitioning 2.3
Kernighan-Lin algorithm 2.4
FPGA Partitioning 2.5
Power Dissipation 2.6
Floor Planning Measurement 3.1
I/O, Power and clock planning 3.2
Measurement of Placement 3.3
Placement Algorithms 3.4
Eigen value placement 3.5
Iterative Placement algorithms 3.6
Global Routing Measurement 4.1
Global routing 4.2
Detailed routing Measurement 4.3
Detailed routing algorithms 4.4
Circuit Extraction 4.5
Layout Design rules 4.6
Digital design flow 5.1
Design and Simultion 5.2
Synthesis and Analysis 5.3
Floorplanning & Placement 5.4
Routing, Circuit Extraction and optimization 5.5
GDS-II File generation 5.6

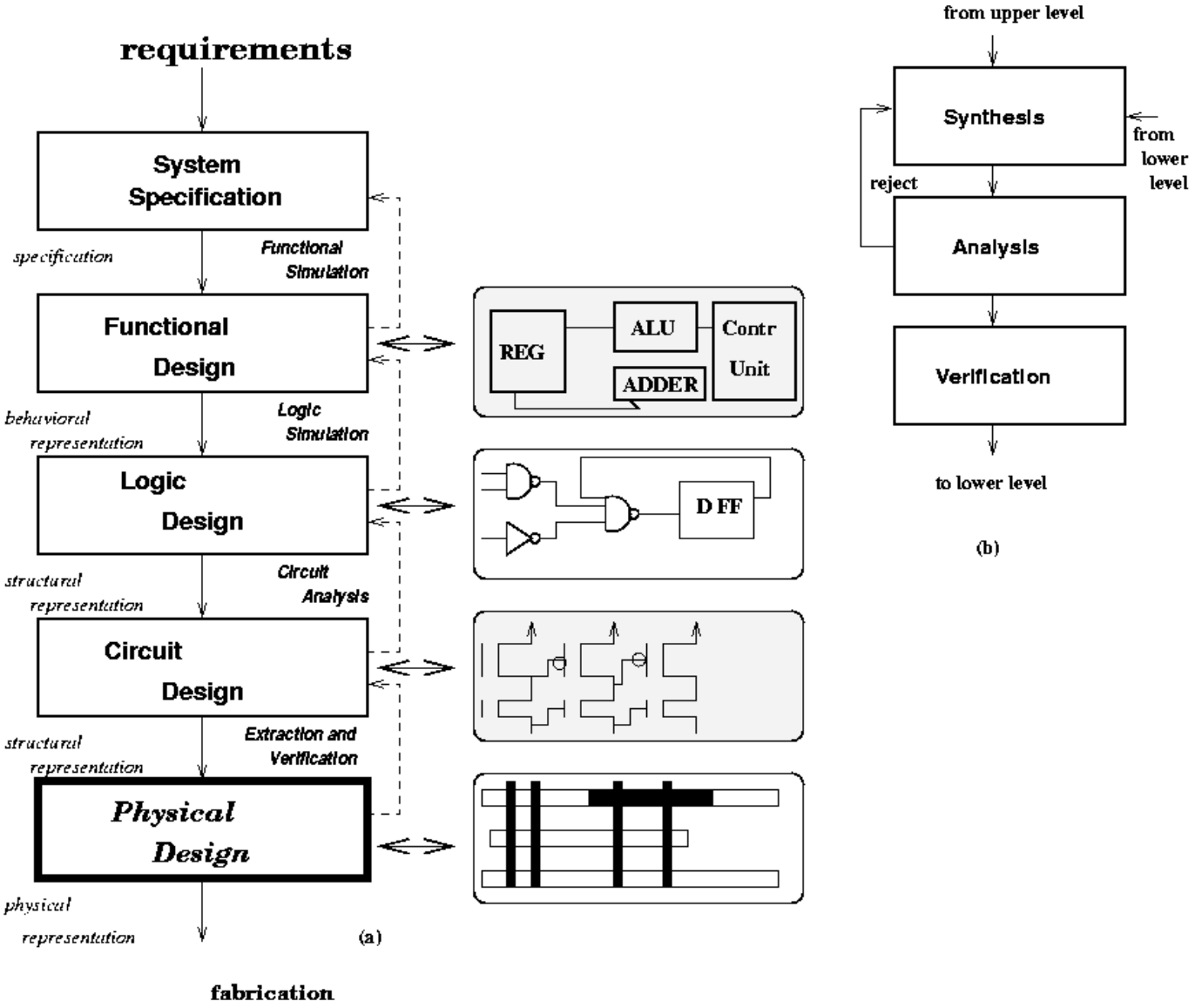
INTRO

- A Town planner works out the number,types, and sizes of buildings in a development.
- An architect designs each building, including the arrangement of the rooms in each building.
- A builder carries out the construction according to the architect's drawings.
- Electrical wiring is one of the laststeps in the construction of each building.
- The physical design of ASICs is normally divided into

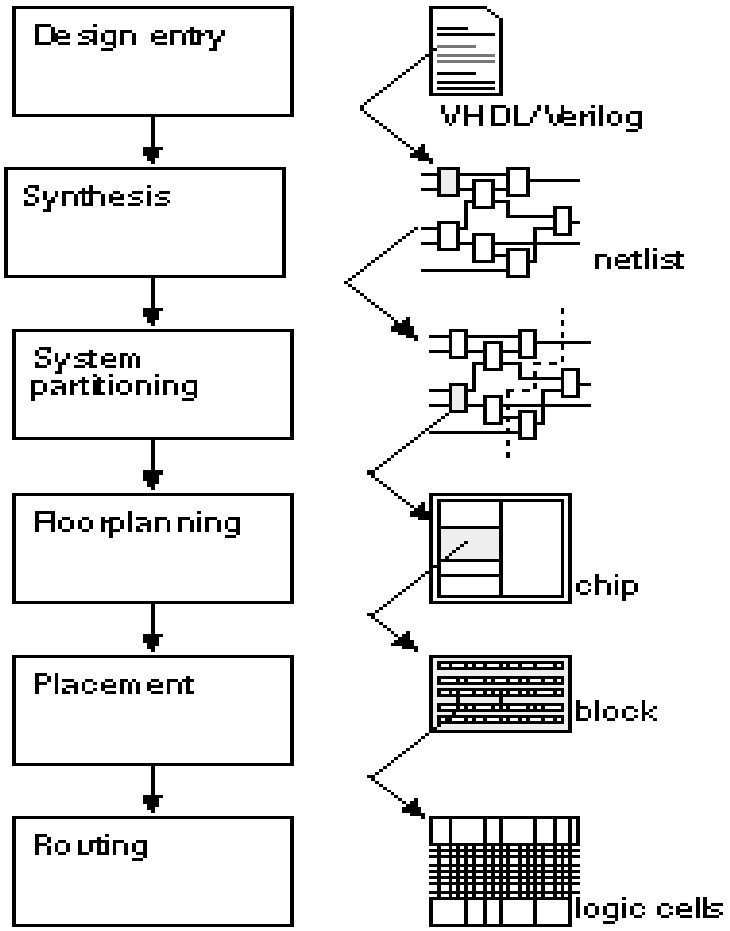
  System partitioning,Floor Planning, placement and routing

- Analogy - ASIC Design
- Town - Microelectronic system
- Town planner - System partitioning
- architect – Floor planning
- builder - Placement
- Electrical wiring - Routing

VLSI Design Flow



Physical Design Steps



- **Part of an ASIC design flow showing the system partitioning, floorplanning, placement, and routing steps.**
- **Performed in a slightly different order, iterated or omitted depending on the type and size of the system and its ASICs.**
- **Floorplanning assumes an increasingly important role.**
- **Sequential-Each of the steps shown in the figure must be performed and each depends on the previous step.**
- **Parallel- However, the trend is toward completing these steps in a parallel fashion and iterating, rather than in a sequential manner.**

CAD Tools

It is necessary to convert each of the physical design steps to a problem with well defined goals and objectives

The Goal for each physical design step are the things we must achieve.

The Objectives for each step are things we would like to meet on the way to achieving the goals.

Each step of ASIC physical design steps are solved by:

- A set of goals and objectives
- A way to measure the goals and objectives
- Algorithm or method to find a solution that meets the goals and objectives.

CAD Tools

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.

Floor planning:

- **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.

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.

Methods and Algorithm

- A CAD tool needs methods or algorithms to generate a solution to each problem using a reasonable amount of computer time.
- There is no best solution for particular problem – heuristic Algorithm
- complexity of an algorithm is O (f (n))
- there are constants k and n 0 so that the running time of the algorithm T (n) is less than k f (n) for all n > n 0 [Sedgewick, 1988]. Here n is a measure of the size of the problem (number of transistors, number of wires, and so on)
- The function f (n) is usually one of the following kinds:
  - f(n) = constant. The algm is constant in time.
    (Steps of the algorithm are repeated once or twice).
  - f (n) = log n. The algm is logarithmic in time. It happen when a Big problem is transferred into small.
  - f (n) = n (linear in time).
  - f (n) = n log n (Large prblm divided into small and solved independently) .
  - f (n) = n^2 (Quadratic in time).only practical for ASIC problems.

Methods and Algorithm(Contd.,)

- **Polynomial:** If the time it takes to solve a problem increases with the size of the problem at a rate that is polynomial but faster than quadratic (or worse in an exponential fashion).

- Each of the ASIC physical design steps, in general, belongs to a class of mathematical problems known as **NP-complete problems**.

- **Definition** : This means that it is unlikely we can find an algorithm to solve the problem exactly in polynomial time.

- Measurement or objective function: We need to make a quantitative measurement of the quality of the solution that we are able to find. Often we combine several parameters or metrics that measure our goals and objectives into a measurement function or objective function

Methods and Algorithm (Contd.,)

Cost Function :

If we are minimizing the measurement function, it is a cost function.

Gain function :

If we are maximizing the measurement function, we call the function a gain function (sometimes just gain).

Estimating ASIC size

- Estimate the die size of a 40 k-gate ASIC in a 0.35 µm gate array, three-level metal process with 166 I/O pads.

- Die size includes core size and I/O size.

- Core size(logic and routing)=(gates/gate density) ×routing factor
  ×(1/gate array utilization)
  - Gate density=standard cell density ×gate array utilization

- I/O size = a^2 where a is the one side of die.
  - One side of die= No of I/O pads in a side × I/O pad pitch

1µm(micron)=0.0393701mil

(1mil= Thousands of inch)

Estimating ASIC size

- For this ASIC the minimum feature size is 0.35 μm.

gate density = 0.35 μ m standard-cell density ∞ (0.8 to 0.9)

= $4 \infty 10^{-4}$ to $4.5 \infty 10^{-4}$ gate/ λ^2 .

This gives the core size (logic and routing only) as

(4 ∞ 10^4 gates/gate density) ∞ routing factor ∞ (1/gate-array utilization)

= $4 \infty 10^4 / (4 \infty 10^{-4}$ to $4.5 \infty 10^{-4}) \infty (1$ to $2) \infty 1/(0.8$ to $0.9) = 10^8$ to $2.5 \infty 10^8 \lambda^2$

= 4840 to 11,900 mil^2 .

- No of I/O pads=166.
- One side=166/4=42 I/O pads per side.
- If a I/O pad pitch=5 mil then One side of die=5×42=210mil
- Minimum requirement of die size=210×210=$4.4\times10^4 mil^2$ to fit 166 I/O pads
- Die area utilized by core logic=$1.19\times10^4/4.4\times10^4 mil^2$=27% by core logic

Estimating ASIC size

Some useful numbers for ASIC estimates, normalized to a 1 μm technology			
Parameter	**Typical value**	**Comment**	**Scaling**
Lambda, λ	0.5 μm=0.5 (minimum feature size)	In a 1 μm technology, λ≈0.5 μm.	NA
Effective gate length	0.25 to 1.0 μm	Less than drawn gate length, usually by about 10 percent.	λ
I/O-pad width (pitch)	5 to 10mil =125 to 250 μm	For a 1 μm technology, 2LM (λ=0.5 μm). Scales less than linearly with λ.	λ
I/O-pad height	15 to 20mil =375 to 500 μm	For a 1 μm technology, 2LM (λ=0.5 μm). Scales approximately linearly with λ.	λ
Large die	1000 mil/side, 10^6 mil^2	Approximately constant	1
Small die	100 mil/side, 10^4 mil^2	Approximately constant	1
Standard-cell density	1.5×10^{-3} gate/μm^2 =1.0gate/mil^2	For 1μm, 2LM, library = 4×10^{-4} gate/λ^2 (independent of scaling).	$1/\lambda^2$
Standard-cell density	8×10^{-3} gate/μm^2 = 5.0gate/mil^2	For 0.5 μm, 3LM, library = 5×10^{-4} gate/λ^2 (independent of scaling).	$1/\lambda^2$
Gate-array utilization	60 to 80% 80 to 90%	For 2LM, approximately constant For 3LM, approximately constant	1 1
Gate-array density	(0.8 to 0.9) × standard cell density	For the same process as standard cells	1
Standard-cell routing factor=(cell area+route area)/cell area	1.5 to 2.5 (2LM) 1.0 to 2.0 (3LM)	Approximately constant	1
Package cost	$0.01/pin, "penny per pin"	Varies widely, figure is for low-cost plastic package, approximately con-	1

Estimating ASIC Size

Qn.: Estimate the die size

40K gate ASIC

0.35 μm Gate Array

Three level metal process.

166 I/O Pads.

Minimum Feature Size 0.35μm

$\lambda = 0.35\mu m/2 = 0.175\ \mu m$.

From the table, 0.35μm Std-cell density = 5×10^{-4} gate/λ^2

Gate density = (0.8 to 0.9) × Std cell density

= (0.8 to 0.9) × 0.35μm Std cell density

= (0.8 to 0.9) × 5×10^{-4} gate/λ^2

Gate density = 4×10^{-4} to 4.5×10^{-4} gate/λ^2

Then Calculate Core Size

(logic + routing)

$= (4\times10^4 \text{ gates} / \text{gate density}) \times \text{routing factor} \times \frac{1}{\text{gate array utilization}}$

$= (4\times10^4 / (4\times10^{-4} \text{ to } 4.5\times10^{-4})) \times (1 \text{ to } 2) \times \frac{1}{(0.8 \text{ to } 0.9)}$

$= \frac{4\times10^4}{4\times10^{-4}} \times 1 \times \frac{1}{0.8}$ to $\frac{4\times10^4}{4.5\times10^{-4}} \times 2 \times \frac{1}{0.9}$

$= 1.25\times10^8\ \lambda^2$ to $1.975\times10^8\ \lambda^2$

1μm = 0.0393 mil.

166 I/O pads

4 sides = 166/4 = 41.5 ≈ 42 I/O

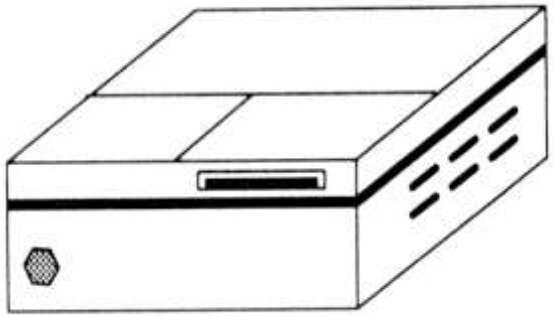
1 pad pitch = 5 mil.

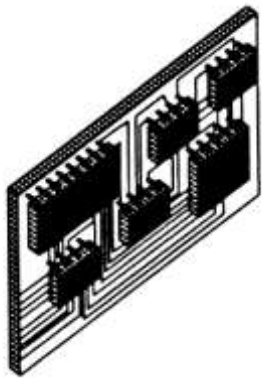
42 × 5 mil = 210 mil on a side

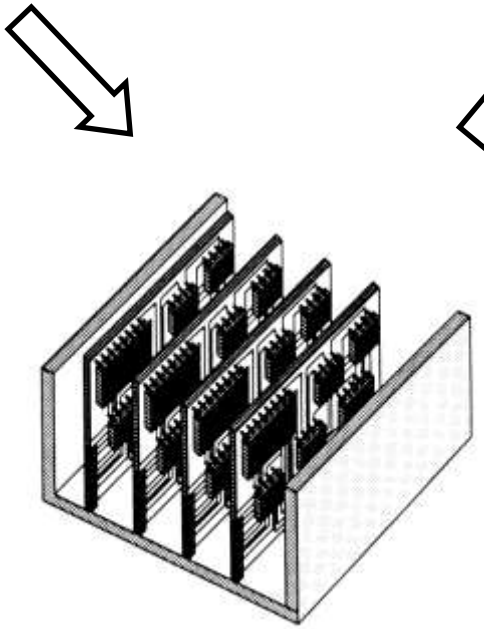
210 × 210 mil = 4.4×10^4 mil^2 to fit I/O's

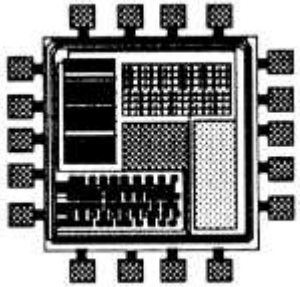
In die area, only 27% occupies I/O pad area.

System Hierarchy



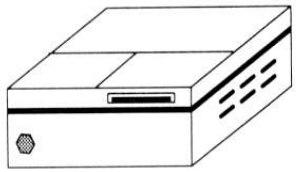




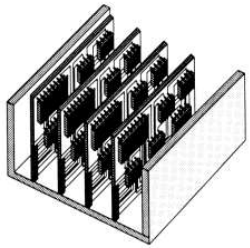


Levels of Partiti

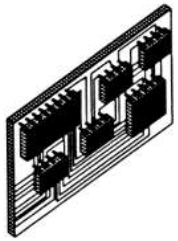
- System Level Partitioning
- Board Level Partitioning
- Chip Level Partitioning



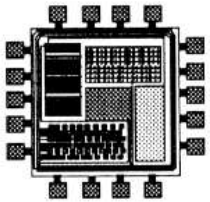
- System



- PCBs

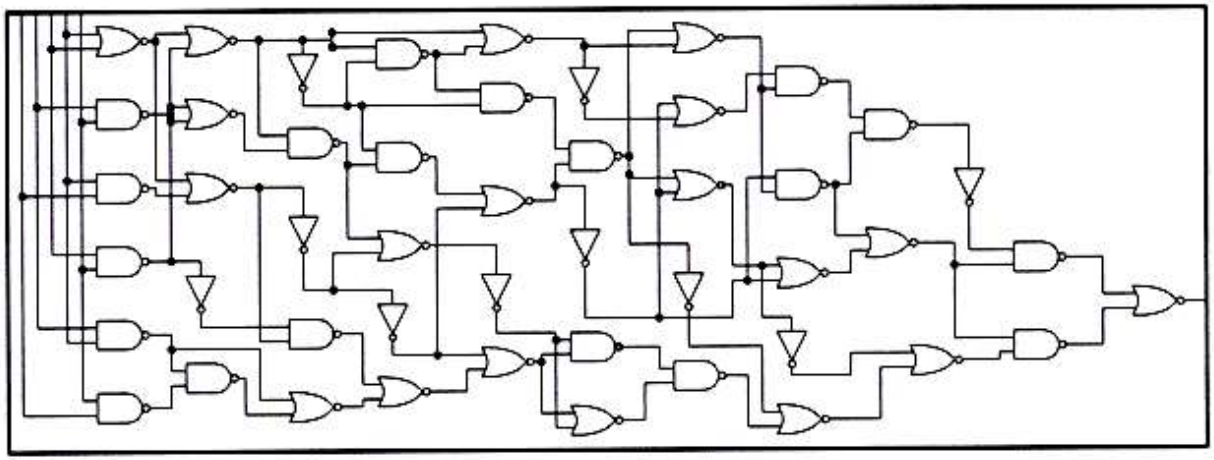


- Chips

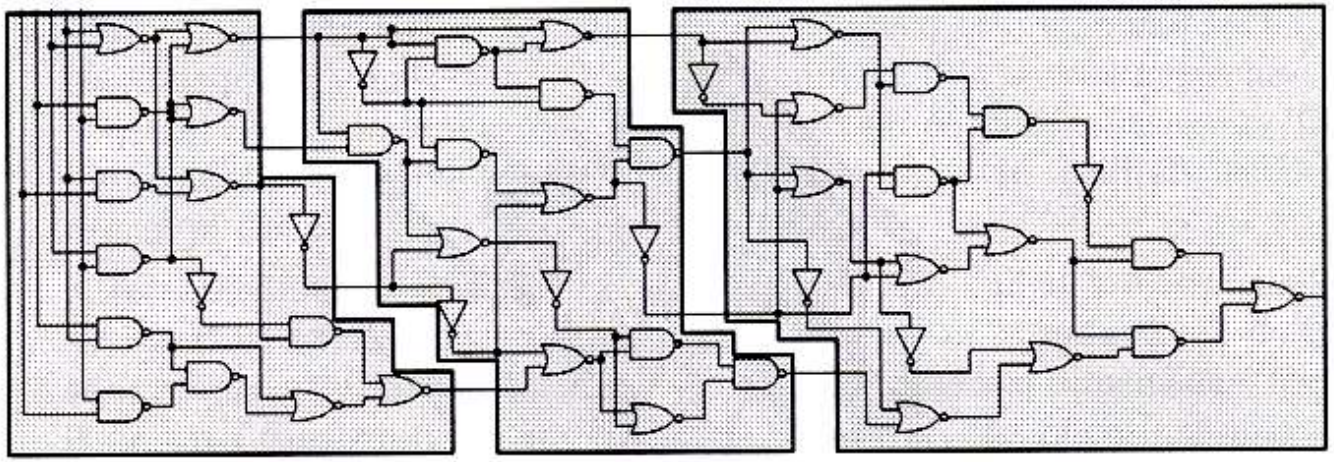


- Subcircuits
- / Blocks

Partitioning of a Circuit



(a)



(b)

VLSI Physical Design :- Partitioning

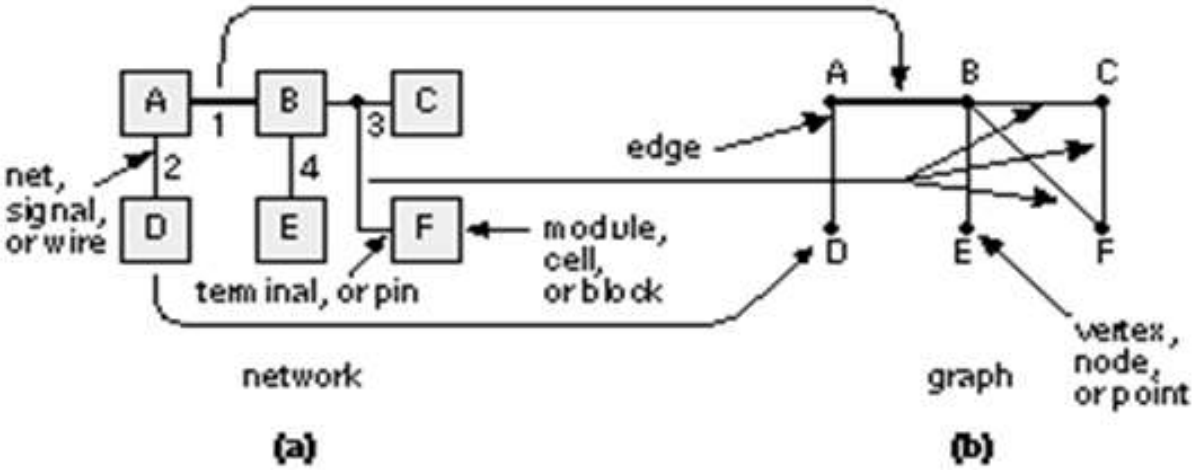
- **System partitioning requires**
  - Goals and Objectives
  - Methods and algorithms to find solutions
  - Ways to evaluate these solutions.

- **Goal of partitioning**
  - Divide the system into number of small systems.

- **Objectives of Partitioning**

  we may need to take into account any or all of the following objectives:
  - 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

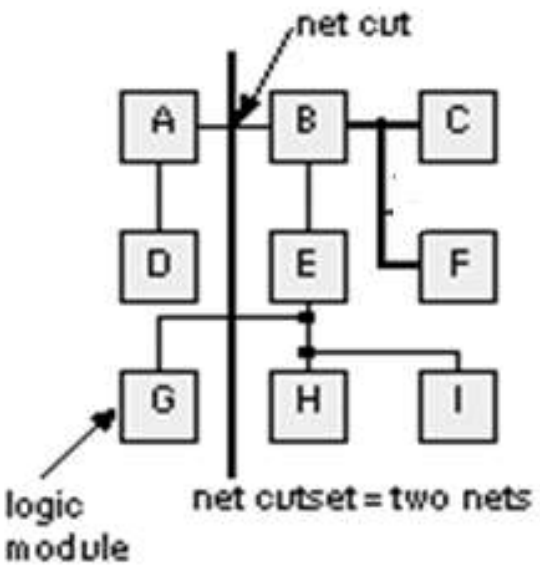
Measuring Connectivity

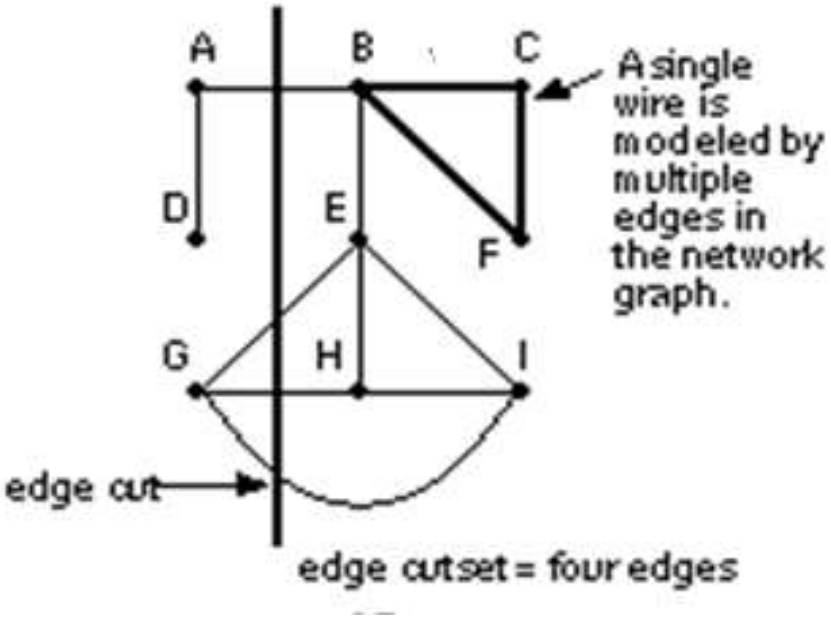


Measuring Connectivity

- Figure (a) shows a **circuit schematic, netlist, or network.**
- The **network** consists of **circuit modules A–F**. Equivalent terms for a **circuit module - cell, logic cell, macro, or a block**.
- A **cell or logic cell -a small logic gate (NAND etc.),collection of other cells**;
- **Macro - gate-array cells**;
- **Block - a collection of gates or cells**.
- Each logic cell has **Electrical connections between the terminals- connectors or pins**.
- The **network** can be represented as the **mathematical graph** shown in Figure (b).
- A **graph** is like a **spider's web**:
  - it contains **vertexes (or vertices)** A–F -**graph nodes or points**) that are connected by edges.
  - A **graph vertex** corresponds to a **logic cell**.
  - An **electrical connection (a net or a signal) between two logic cells** corresponds to a **graph edge**.

Measuring Connectivity

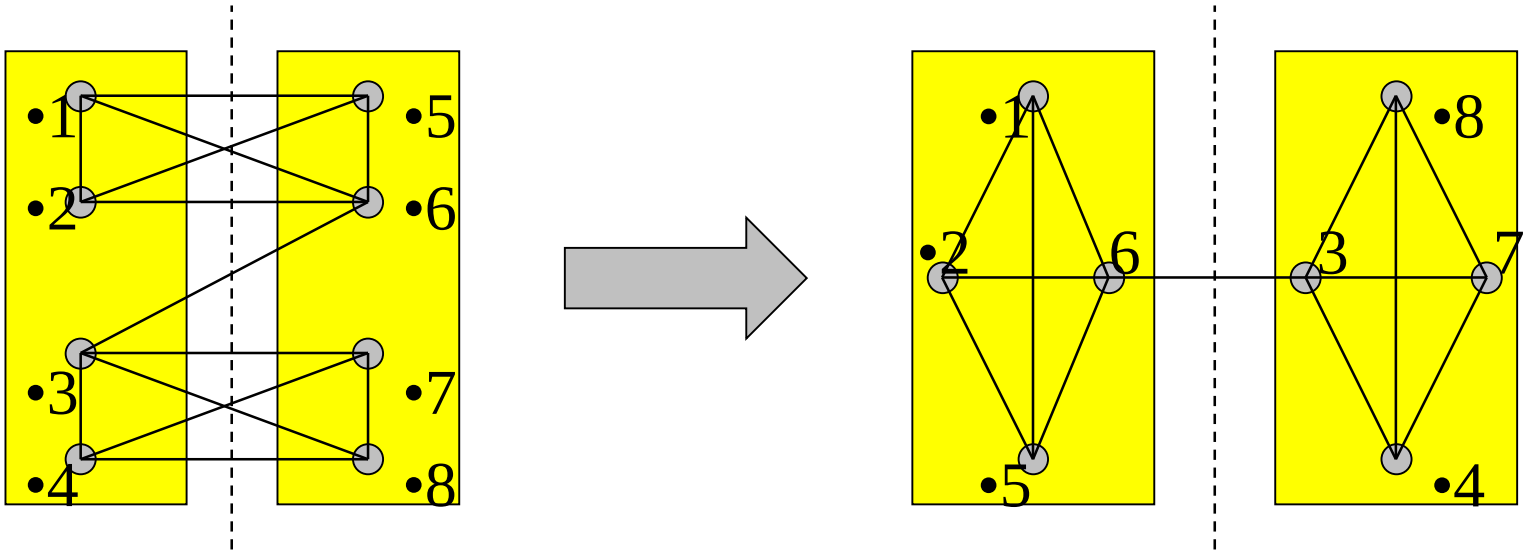




Measuring Connectivity

- Net Cutset
  - Divide the network into two by drawing a line across connections, make net cuts. The resulting set of net cuts is the net cutset.
  - Number of net cuts - the number of external connections between the two partitions in a network.
- Edge cutset.
  - When we divide the network graph into the same partitions we make edge cuts and we create the edge cutset.
  - Number of edge cuts – the number of external connections between the two partitions in a graph
  - **Number of edge cuts in a graph is not necessarily equal to the number of net cuts in the network.**

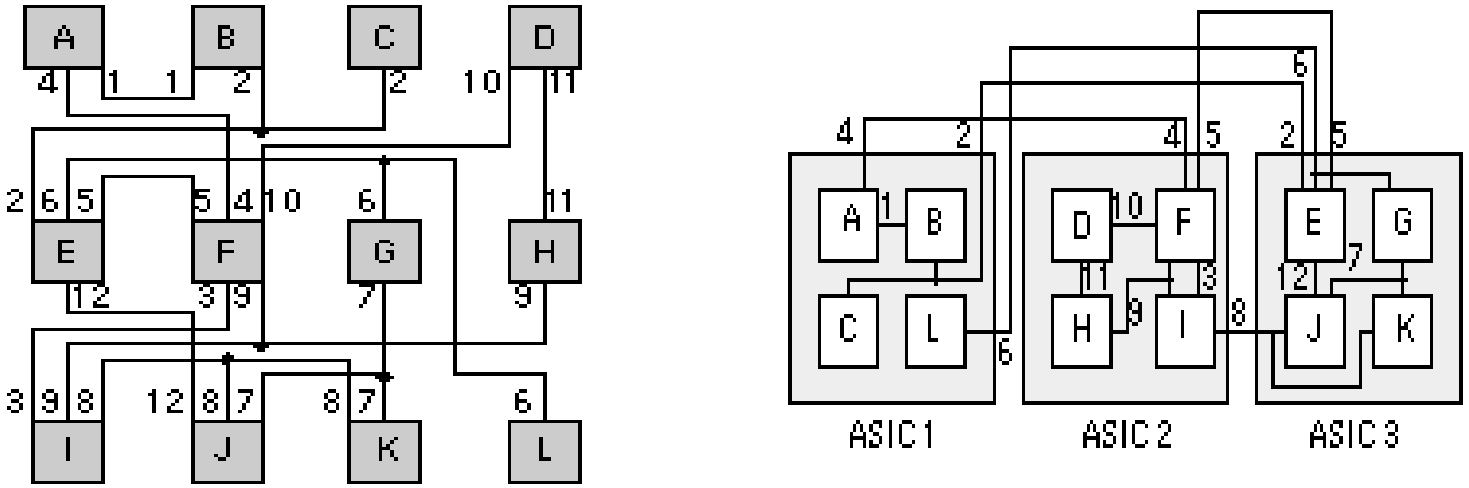
Partitioning



- Initial Bisections
- Cutsize = 9

- Final Bisections
- Cutsize = 1

A Simple Partitioning Example



• FIGURE 15.7 Partitioning example.

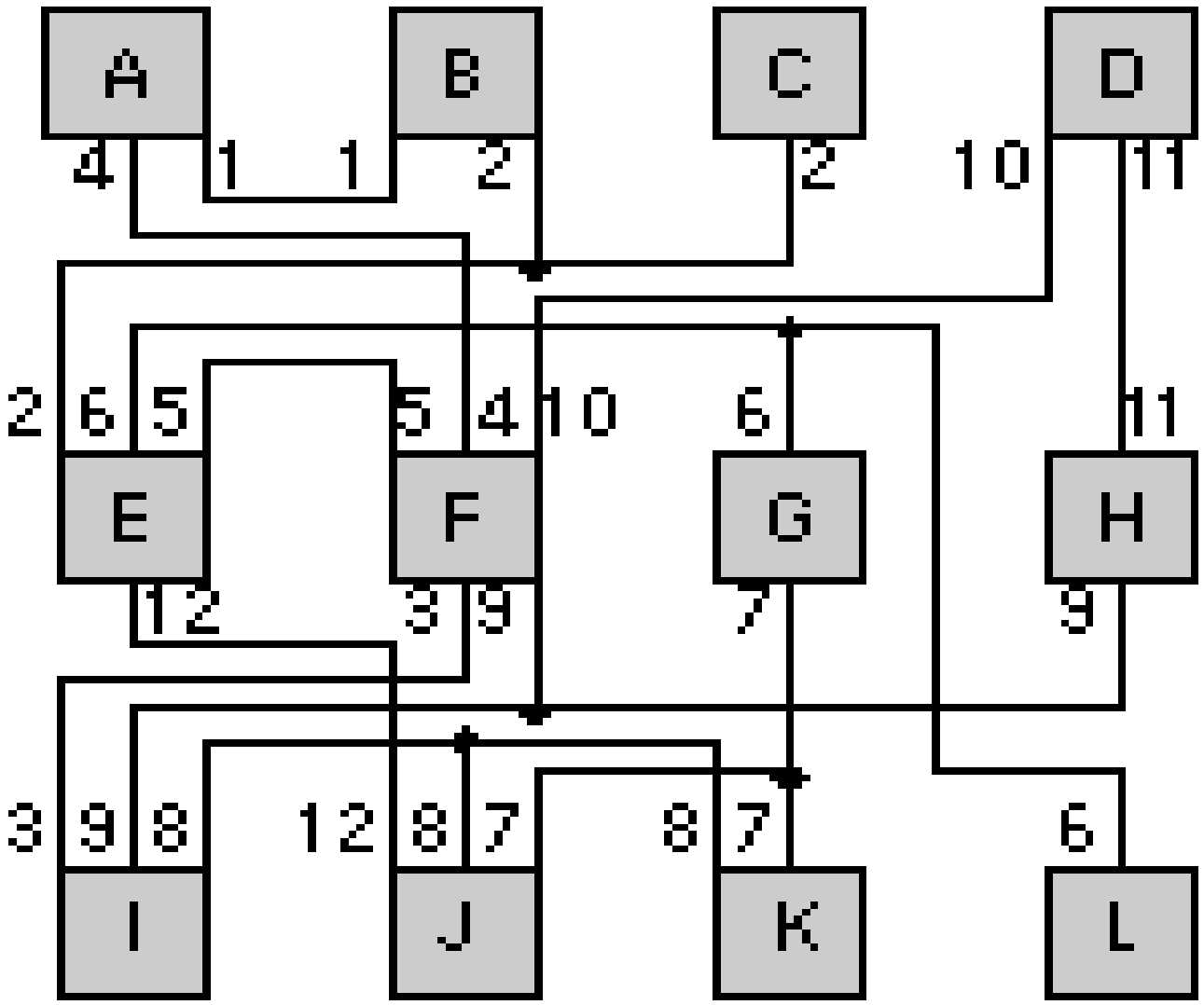
• (a) We wish to partition this network into three ASICs with no more than four logic cells per ASIC.

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

•Goal : to partition our simple network into ASICs.

•Objectives are the following:

- -Use no more than three ASICs.
- -l Each ASIC is to contain no more than four logic cells.
- -l Use the minimum number of external connections for each ASIC.


A
B
C
D
E
F
G
H
I
J
K
L
4
1
1
2
2
10
11
2
6
5
5
4
10
6
11
12
3
9
7
9
3
9
8
12
8
7
8
7
6

•Types of Partitioning

- Splitting a network into several pieces - network partitioning problem.
- Two types of algorithms used in system partitioning are
  - **Constructive partitioning** - uses a set of rules to find a solution.
  - **Iterative partitioning improvement** (or iterative partitioning refinement - takes an existing solution and tries to improve it.

Constructive Partitioning

- The most common constructive partitioning algorithms - **seed growth or cluster growth.**

- The **steps** of a simple seed-growth algorithm for constructive partitioning:

1. ***Start a new partition with a seed logic cell.***

2. ***Consider all the logic cells that are not yet in a partition. Select each of these logic cells in turn.***

3. ***Calculate a gain function, g(m) , that measures the benefit of adding logic cell m to the current partition. One measure of gain is the number of connections between logic cell m and the current partition.***

4. ***Add the logic cell with the highest gain g(m) to the current partition.***

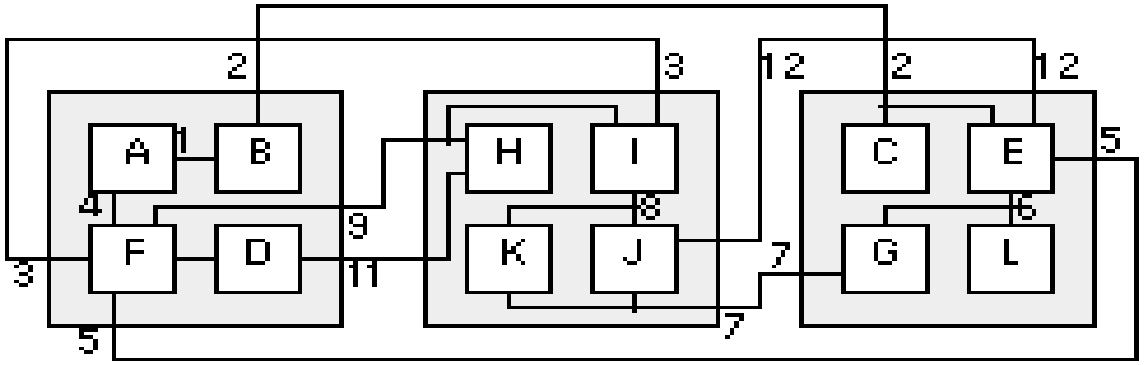
5. ***Repeat the process from step 2. If you reach the limit of logic cells in a partition, start again at step 1.***

Cluster Growth

m : size of each cluster, V : set of nodes

```
n  =  |V| / m ;
  for  (i=1; i<=n; i++)
    {
        seed = vertex in V with maximum degree;
        Vi = {seed};
        V = V – {seed};
        for  (j=1; j<m; j++)
          {
              t = vertex in V maximally connected to Vi;
              Vi = Vi U {t};
              V = V – {t};
           }
       }
```

Constructive Partitioning



•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.

Improvement in Partitioning

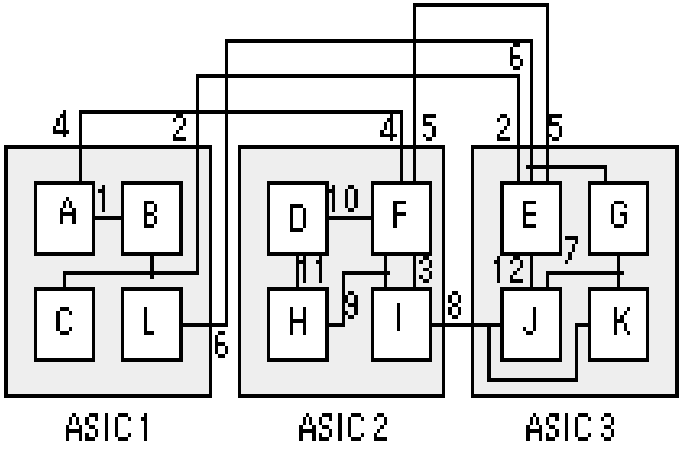


Fig 1 with 5 external connections

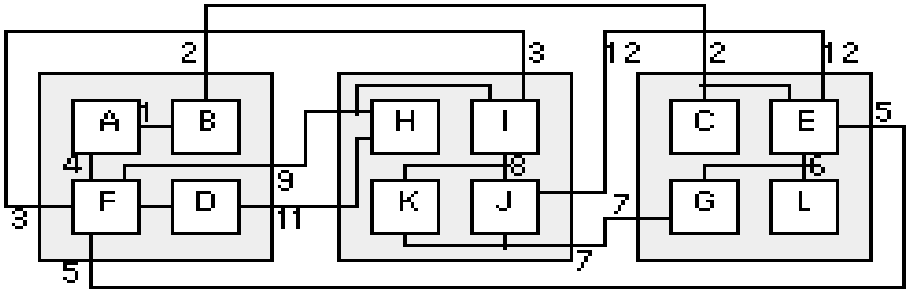


Fig 2 with 7 external connections

- To get from the solution shown in Fig 2 to the solution of Fig 1, which has a minimum number of external connections, requires a complicated swap.
- The three pairs: D and F, J and K, C and L need to be swapped—all at the same time. It would take a very long time to consider all possible swaps of this complexity.

Iterative Partitioning Improvement

Algorithm based on Interchange method and group migration method

Interchange method (swapping a single logic cell):

If the swap improves the partition, accept the trail interchange otherwise select a new set of logic cells to swap.

Example: Greedy Algorithm –

It considers only one change

-Rejects it immediately if it is not an improvement.

-Accept the move only if it provides immediate benefit.

It is known as local minimum.

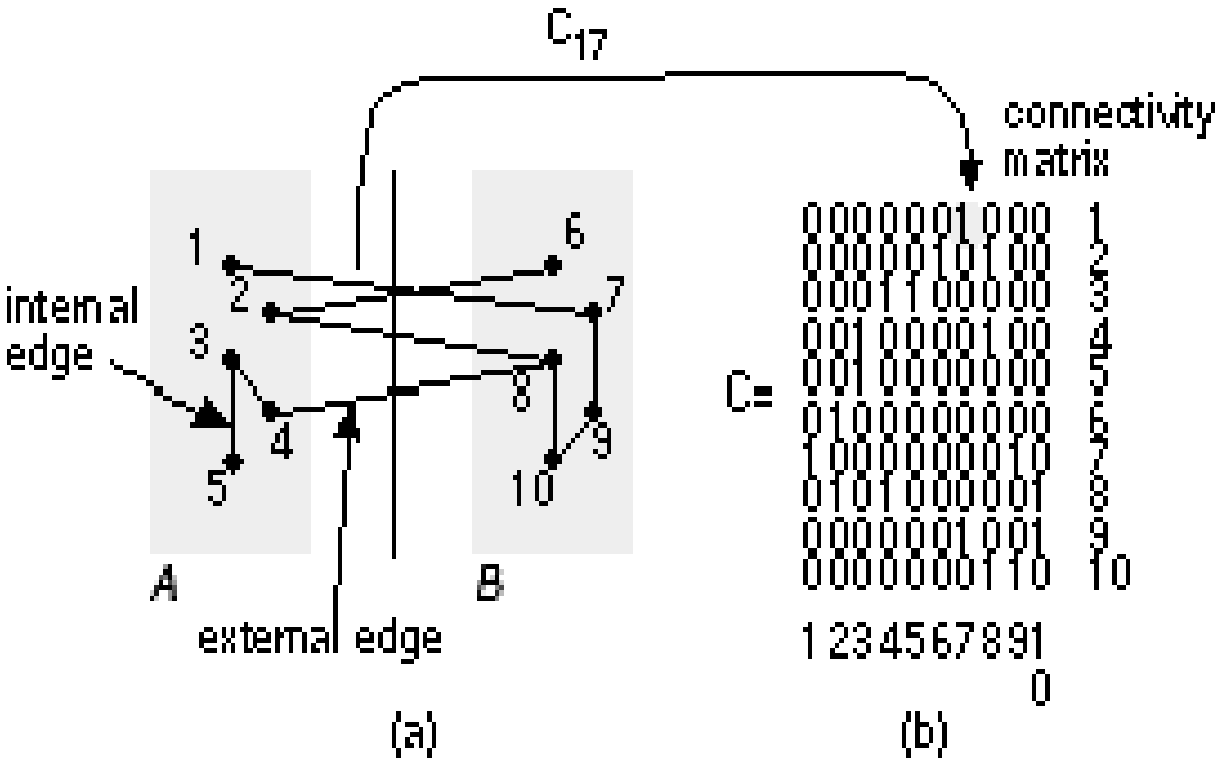
Group Migration (swapping a group of logic cell):

- Group migration consists of swapping groups of logic cells between partitions.
- The group migration algorithms –
  - better than simple interchange methods at improving a solution
  - but are more complex.

Example: Kernighan – Lin Algorithm (K-L)

- Min cut Problem : Dividing a graph into two pieces, minimizing the nets that are cut

The Kernighan–Lin Algorithm



The Kernighan–Lin Algorithm (contd.,)

- Total external cost, cut cost, cut weight

$$W = \sum_{a \in A, b \in B} c_{ab}$$

- External edge cost

$$E_a = \sum_{y \in B} c_{ay}$$

- Internal edge cost

$$I_a = \sum_{z \in A} c_{az}$$

- Gain

$$g = D_a + D_b - 2C_{ab}$$

where

$$D_a = E_a - I_a$$

The Kernighan–Lin Algorithm (contd.,)

- The K–L algorithm finds a group of node pairs to swap that increases the gain even though swapping individual node pairs from that group might decrease the gain.

The steps of K-L algorithm are:

1. Find two nodes, a_i from A, and b_i from B, so that the gain from swapping them is a maximum. The gain g_i is

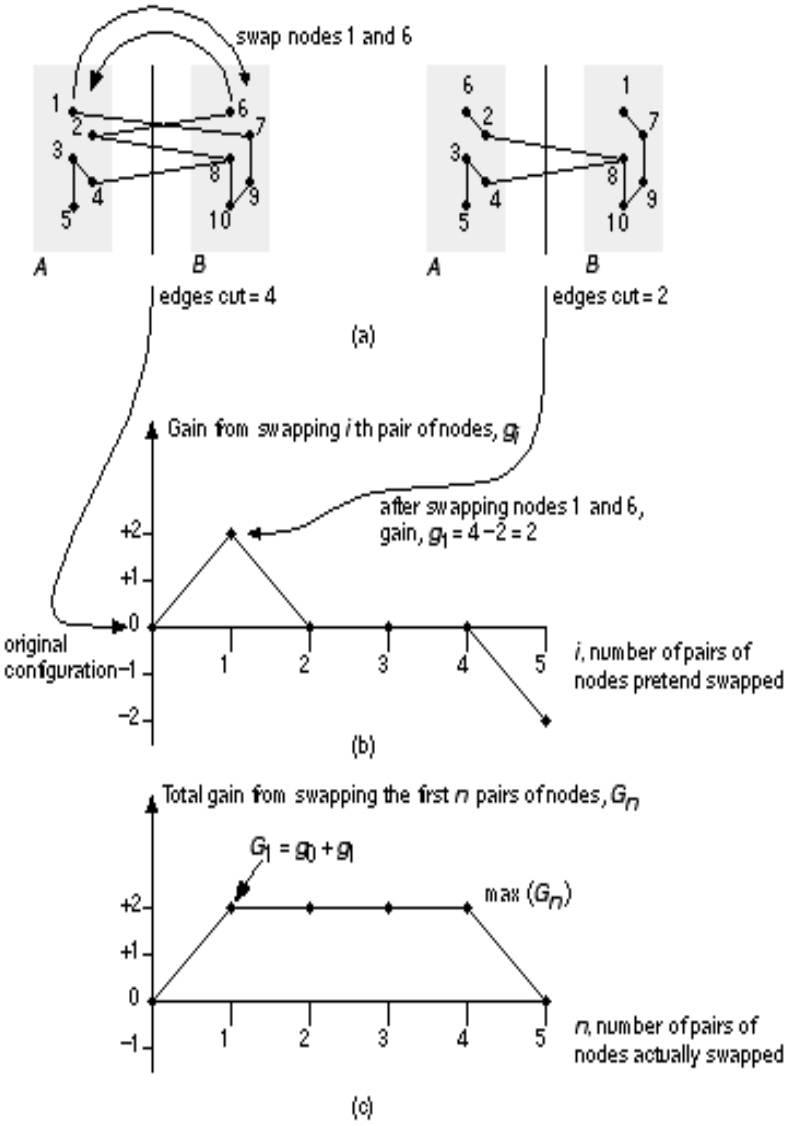
$$g_i = D_{a_i} + D_{b_i} - 2C_{a_i b_i}$$

2. Next pretend swap ai and bi even if the gain g_i is zero or negative, and do not consider ai and bi eligible for being swapped again.
3. Repeat steps 1 and 2 a total of m times until all the nodes of A and B have been pretend swapped. We are back where we started, but we have ordered pairs of nodes in A and B according to the gain from interchanging those pairs.

The Kernighan–Lin Algorithm (contd.,)

4. Now we can choose which nodes we shall actually swap. Suppose we only swap the first n pairs of nodes that we found in the preceding process. In other words we swap nodes X = a1, a2, &…., an from A with nodes Y = b1, b2,&…..,bn from B. The total gain would be, $G_n = \sum_{i=1}^{n} g_i$

5. We now choose n corresponding to the maximum value of G_n
   - If the maximum value of $G_n > 0$, *then swap the sets of nodes X and Y and thus* reduce the cut weight by G_n .
   - *Use this new partitioning to start the process again* at the first step.
   - If the maximum value of $G_n = 0$, *then we cannot improve the current* partitioning and we stop.
   - We have found a locally optimum solution.



•Partitioning a graph using the Kernighan–Lin algorithm.

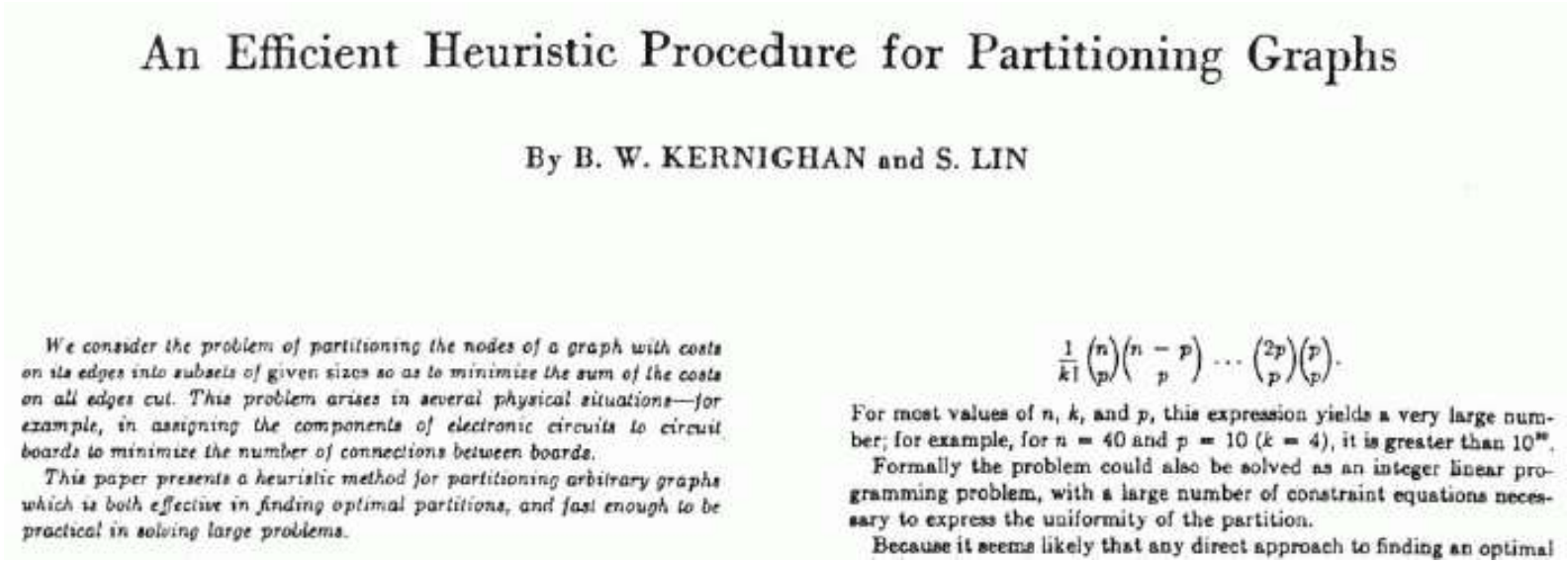
•(a) Shows how swapping node 1 of partition A with node 6 of partition B results in a gain of g = 2.

•(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.

Kernighan and Lin heuristic

"An Efficient Heuristic Procedure for Partitioning Graphs" B. W. Kernighan and S. Lin, The Bell



An Efficient Heuristic Procedure for Partitioning Graphs

By B. W. KERNIGHAN and S. LIN

We consider the problem of partitioning the nodes of a graph with costs on its edges into subsets of given sizes so as to minimize the sum of the costs on all edges cut. This problem arises in several physical situations—for example, in assigning the components of electronic circuits to circuit boards to minimize the number of connections between boards.

This paper presents a heuristic method for partitioning arbitrary graphs which is both effective in finding optimal partitions, and fast enough to be practical in solving large problems.

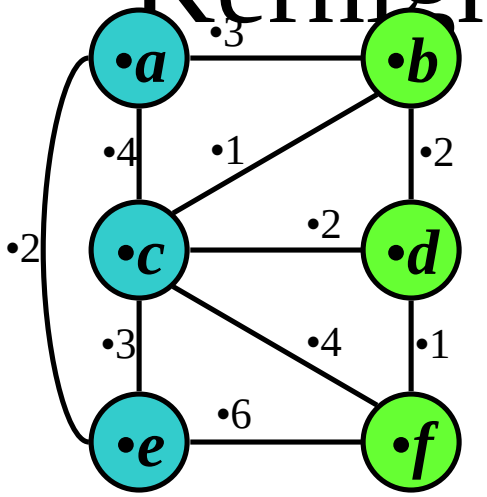
$$\frac{1}{k!}\binom{n}{p}\binom{n-p}{p}\cdots\binom{2p}{p}\binom{p}{p}.$$

For most values of n, k, and p, this expression yields a very large number; for example, for $n = 40$ and $p = 10$ ($k = 4$), it is greater than 10^{30}.

Formally the problem could also be solved as an integer linear programming problem, with a large number of constraint equations necessary to express the uniformity of the partition.

Because it seems likely that any direct approach to finding an optimal

Kernighan-Lin Algorithm (1)



- **Given:**

  Initial weighted graph *G* with

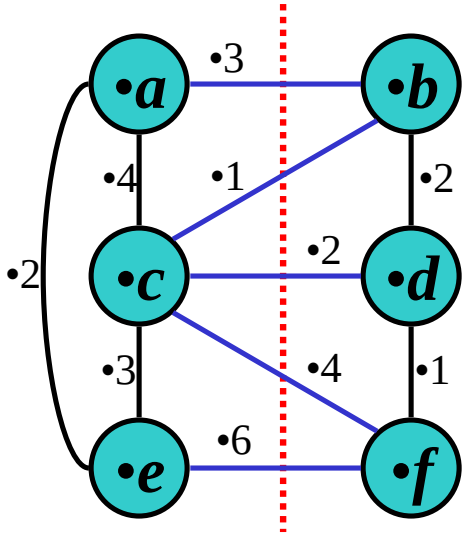
  $V(G) = \{ a, b, c, d, e, f \}$

- **Start with any partition of V(G) into X and Y, say**

- $X = \{ a, c, e \}$

- $Y = \{ b, d, f \}$

KL algorithm (2a)



•cut-size = 3+1+2+4+6 = 16

•$X = \{ a, c, e \}$
$Y = \{ b, d, f \}$

•**Compute the gain values of moving node x to the others set:**

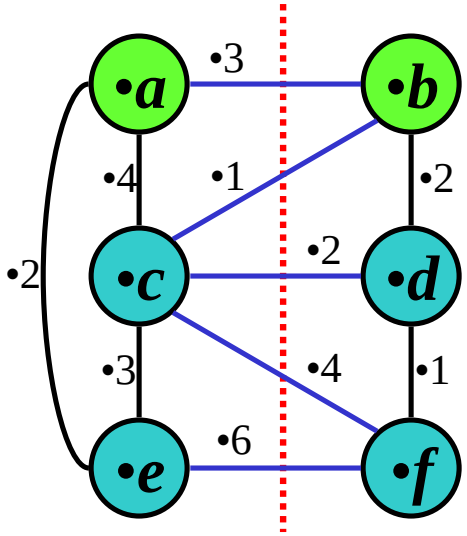
•$G_x = E_x - I_x$

E_x = **cost of edges connecting node x with the other group (extra)**

I_x = **cost of edges connecting node x within its own group (intra)**

- $G_a = E_a - I_a = -3$ (= 3 – 4 – 2)
- $G_c = E_c - I_c = 0$ (= 1 + 2 + 4 – 4 – 3)
- $G_e = E_e - I_e = +1$ (= 6 – 2 – 3)
- $G_b = E_b - I_b = +2$ (= 3 + 1 –2)
- $G_d = E_d - I_d = -1$ (= 2 – 2 – 1)
- $G_f = E_f - I_f = +9$ (= 4 + 6 – 1)

KL algorithm (2b)

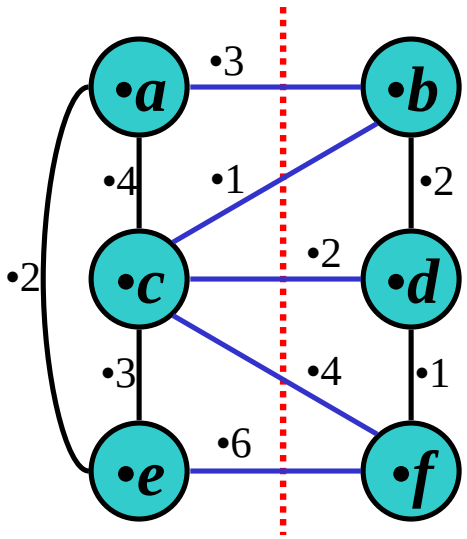


- **Cost saving when exchanging a and b is essentially $G_a + G_b$**
- **However, the cost saving 3 of the direct edge was counted twice. But this edge still connects the two groups**
- **Hence, the real "gain" (i.e. cost saving) of this exchange is $g_{ab} = G_a + G_b - 2c_{ab}$**

- $X = \{ a, c, e \}$
  $Y = \{ b, d, f \}$

- $G_a = E_a - I_a = -3$ $(= 3 - 4 - 2)$
- $G_b = E_b - I_b = +2$ $(= 3 + 1 - 2)$
- $g_{ab} = G_a + G_b - 2c_{ab} = -7$ $(= -3 + 2 - 2 \cdot 3)$

KL algorithm (3)



- $G_a = -3$ $G_b = +2$
- $G_c = 0$ $G_d = -1$
- $G_e = +1$ $G_f = +9$

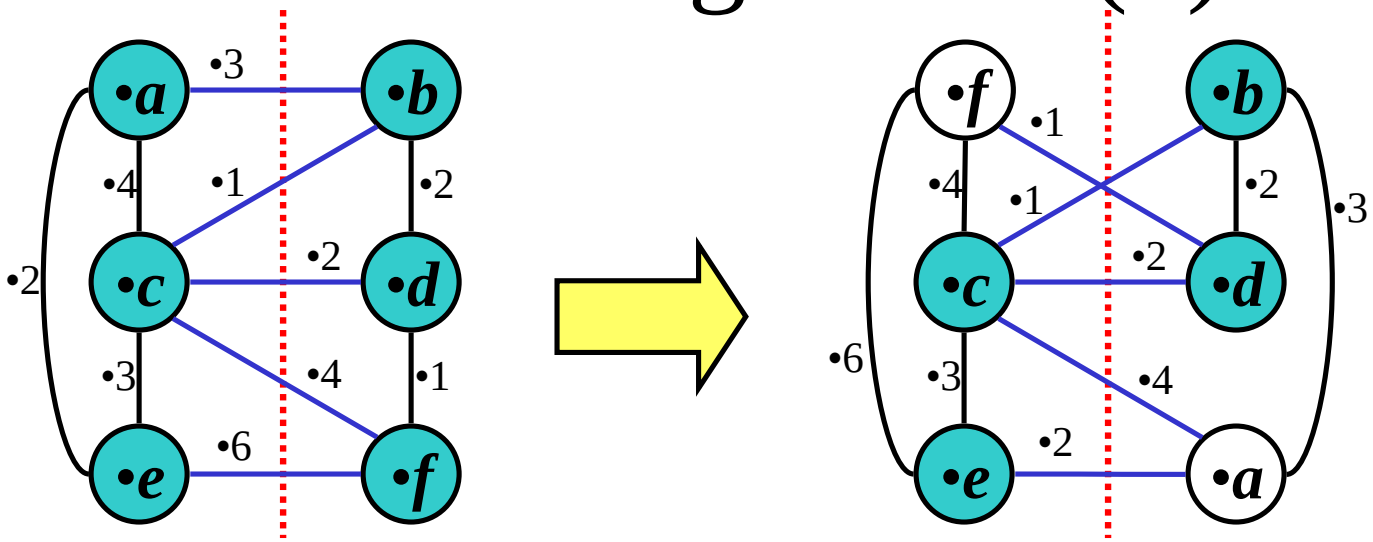
- Compute all the gains

- $g_{ab} = G_a + G_b - 2w_{ab} = -3 + 2 - 2\cdot 3 = -7$
- $g_{ad} = G_a + G_d - 2w_{ad} = -3 - 1 - 2\cdot 0 = -4$
- $g_{af} = G_a + G_f - 2w_{af} = -3 + 9 - 2\cdot 0 = +6$
- $g_{cb} = G_c + G_b - 2w_{cb} = 0 + 2 - 2\cdot 1 = 0$
- $g_{cd} = G_c + G_d - 2w_{cd} = 0 - 1 - 2\cdot 2 = -5$
- $g_{cf} = G_c + G_f - 2w_{cf} = 0 + 9 - 2\cdot 4 = +1$
- $g_{eb} = G_e + G_b - 2w_{eb} = +1 + 2 - 2\cdot 0 = +3$
- $g_{ed} = G_e + G_d - 2w_{ed} = +1 - 1 - 2\cdot 0 = 0$
- $g_{ef} = G_e + G_f - 2w_{ef} = +1 + 9 - 2\cdot 6 = -2$

- cut-size = 16

- Pair with maximum gain

KL algorithm (4)



•cut-size = 16

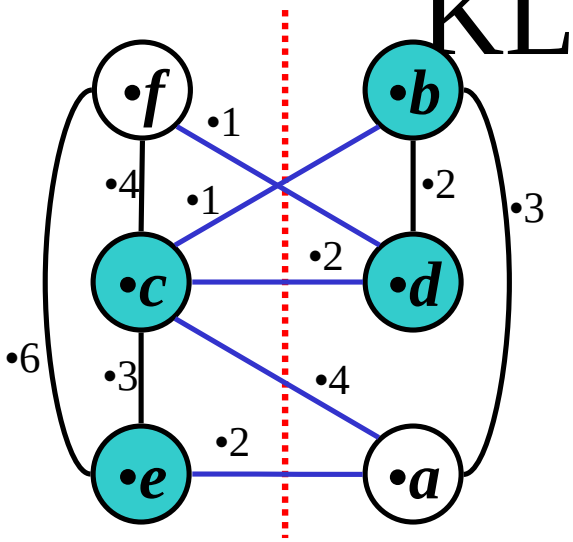
•cut-size = 16 – 6 = 10

•Exchange nodes *a* and *f*

•Then lock up nodes *a* and *f*

•$g_{af} = G_a + G_f - 2c_{af} = -3 + 9 - 2 \cdot 0 = +6$

KL algorithm (5)



•cut-size = 10

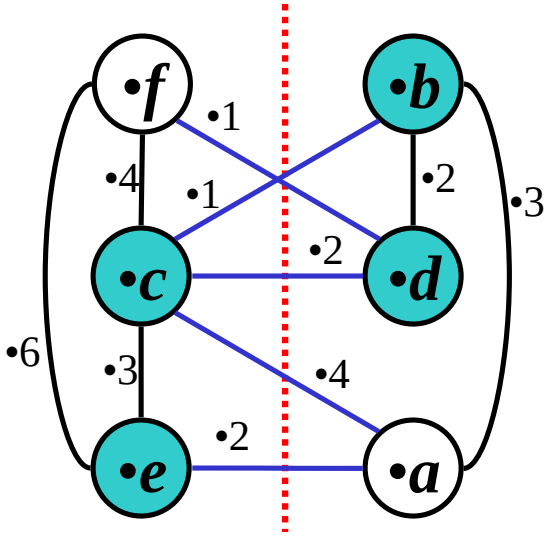
•$X' = \{ c, e \}$
$Y' = \{ b, d \}$

- $G_a = -3$
- $G_b = +2$
- $G_c = 0$
- $G_d = -1$
- $G_e = +1$
- $G_f = +9$

•**Update the G-values of unlocked nodes**

- $G'_c = G_c + 2c_{ca} - 2c_{cf} = 0 + 2(4 - 4) = 0$
- $G'_e = G_e + 2c_{ea} - 2c_{ef} = 1 + 2(2 - 6) = -7$
- $G'_b = G_b + 2c_{bf} - 2c_{ba} = 2 + 2(0 - 3) = -4$
- $G'_d = G_d + 2c_{df} - 2c_{da} = -1 + 2(1 - 0) = 1$

KL algorithm (6)



•cut-size = 10

•$X' = \{ c, e \}$
$Y' = \{ b, d \}$

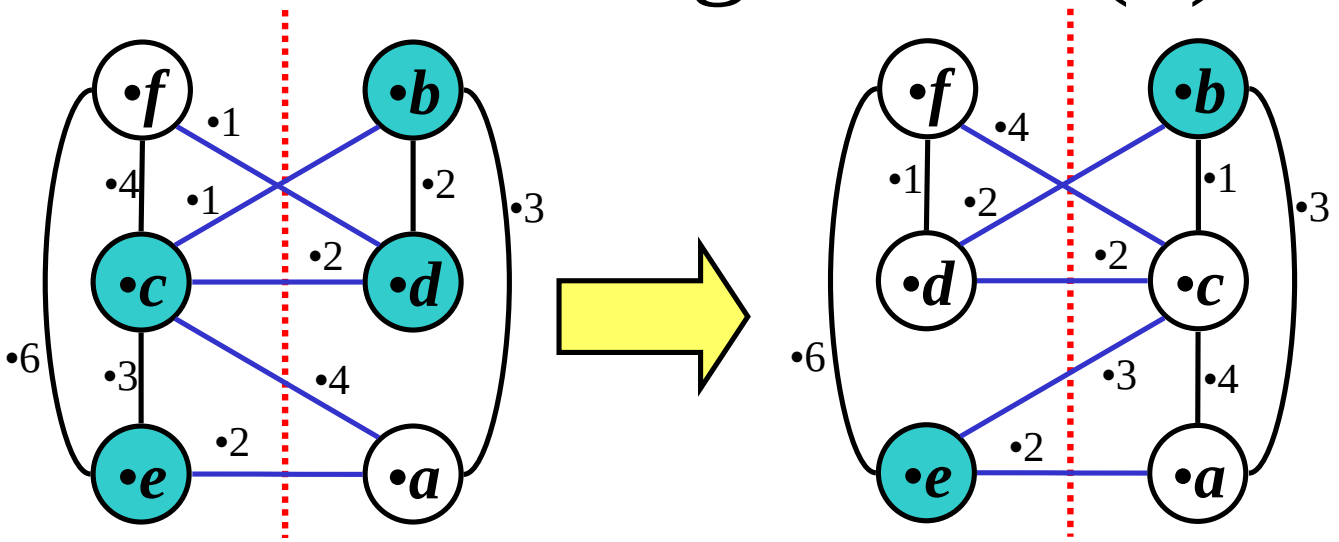
- $G'_c = 0$ $G'_b = -4$
- $G'_e = -7$ $G'_d = +1$

•**Compute the gains**

- $g'_{cb} = G'_c + G'_b - 2c_{cb} = 0 - 4 - 2 \cdot 1 = -6$
- $g'_{cd} = G'_c + G'_d - 2c_{cd} = 0 + 1 - 2 \cdot 2 = -3$
- $g'_{eb} = G'_e + G'_b - 2c_{eb} = -7 - 4 - 2 \cdot 0 = -11$
- $= G'_e + G'_d - 2c_{ed} = -7 + 1 - 2 \cdot 0 = -6$

•**Pair with maximum gain (can also be neative)**

KL algorithm (7)



•cut-size = 10

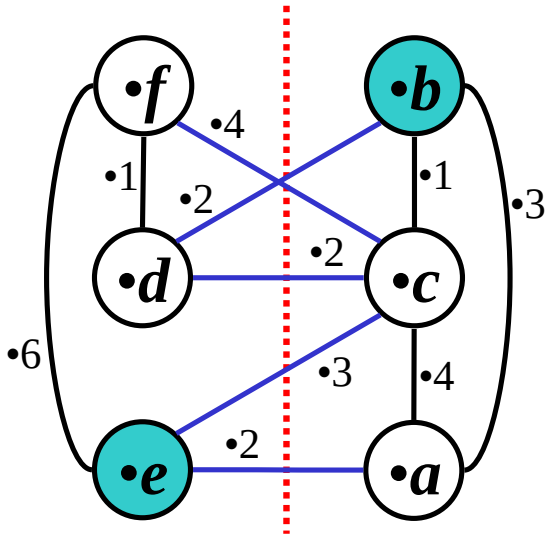
•cut-size = 10 – (–3) = 13

•Exchange nodes *c* and *d*

•Then lock up nodes *c* and *d*

$$\bullet g'_{cd} = G'_c + G'_d - 2c_{cd} = 0 + 1 - 2 \cdot 2 = -3$$

KL algorithm (8)



- $G'_c = 0$ $G'_b = -4$
- $G'_e = -7$ $G'_d = +1$

- $X'' = \{ e \}$
- $Y'' = \{ b \}$

- cut-size = 13

- Update the G-values of unlocked nodes

- $G''_e = G'_e + 2c_{ed} - 2c_{ec} = -7 + 2(0 - 3) = -1$
- $G''_b = G'_b + 2c_{bd} - 2c_{bc} = -4 + 2(2 - 1) = -2$

- Compute the gains

- Pair with max. gain is (e, b)

- $g''_{eb} = G''_e + G''_b - 2c_{eb} = -1 - 2 - 2 \cdot 0 = -3$

KL algorithm (9)

- Summary of the Gains…
  - $g = +6$
  - $g + g' = +6 - 3 = +3$
  - $g + g' + g'' = +6 - 3 - 3 = 0$
- Maximum Gain = $g = +6$
- Exchange only nodes *a* and *f*.
- End of 1 pass.

❑ ***Repeat the Kernighan-Lin.***

Demerits of Kernighan–Lin Algorithm

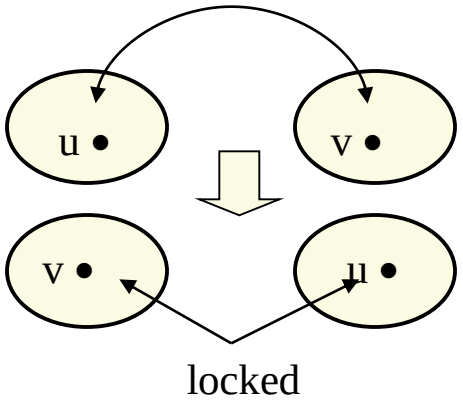
- Minimizes the number of **edges cut**, not the number of nets cut.
- Does not directly allow for more than **two partitions**.
- Does not allow logic cells to be **different sizes**.
- Does not allow partitions to be **unequal** or find the optimum partition size.
- Does not allow for **selected logic cells to be fixed in place**.
- K-L Finding local optimum solution in random fashion
  - **Random starting partition**
  - **Choice of nodes to swap may have equal gain**
- **Expensive in computation time.**
  - An amount of computation time that grows as $n^2 \log n$ for 2n nodes.

Solution:

To implement a net-cut partitioning rather than an edge-cut partitioning, keep track of the nets rather than the edges – FM algorithm

Recap of Kernighan-Lin's Algorithm

- Pair-wise exchange of nodes to reduce cut size
- Allow cut size to increase temporarily within a pass
  - Compute the gain of a swap
  - Repeat
    - Perform a feasible swap of max gain
    - Mark swapped nodes "locked";
    - Update swap gains;
- Until no feasible swap;
- Find max prefix partial sum in gain sequence $g_1, g_2, \ldots, g_m$
- Make corresponding swaps permanent.



- Start another pass if current pass reduces the cut size
- (usually converge after a few passes)

Fiduccia-Mattheyses Algorithm

- "A Linear-time Heuristics
- for Improving Network Partitions"
- 19th DAC, pages 175-181, 1982.

Fiduccia-Mattheyses (F-M) Algorithm

- Addresses the difference between nets and edges.
- Reduce the computational time.

Key Features of F-M:

- **Base logic cell -** Only one logic cell moves at a time.
  - Base logic cell is chosen to maintain **balance** between partitions in order to stop the algorithm from moving all the logic cells to one large partition
  - **Balance -** the ratio of total logic cell size in one partition to the total logic cell size in the other. Altering the balance allows us to vary the sizes of the partitions.

- **Critical nets** - used to simplify the gain calculations.
  - A net is a **critical net** if it has an attached logic cell that, when swapped, changes the number of nets cut.
  - It is only necessary to recalculate the gains of logic cells on critical nets that are attached to the base logic cell.

- The logic cells that are free to move are stored in a doubly linked list. The lists are sorted according to gain. This allows the logic cells with maximum gain to be found quickly.

- **Reduce the computation time** - increases only slightly more than linearly with the number of logic cells in the network.

Overcome problems in K-L using F-M algorithm

- To generate unequal partitioning
  - **Dummy logic cells** with no connections introduced in K-L algorithm
  - **Adjust partition size according to balance parameter** in F-M algorithm
- To fix logic cells in place during partitioning
  - That logic cells **should not be considered as base logic cells** in F-M algorithm.

Features of FM Algorithm

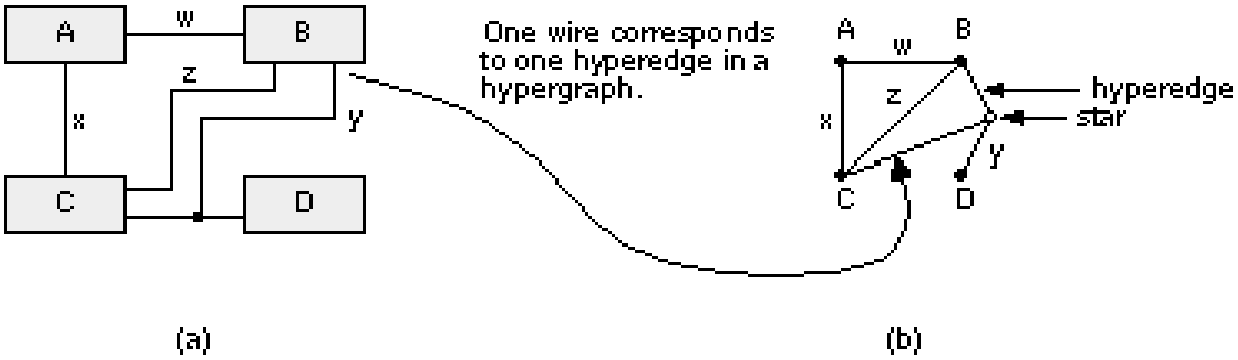
- Modification of KL Algorithm:
  - Can handle non-uniform vertex weights (areas)
  - Allow unbalanced partitions
  - Extended to handle hypergraphs
  - Clever way to select vertices to move, run much faster.

Hypergraph

•Hypergraph- To represent nets with multiple terminals in a network accurately

•A hypergraph consist of –

- a star- a special type of vertex
- a hyperedge- represents a net with more than two terminals in a network.



•FIGURE - 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

Problem Formulation

- Input: A hypergraph with
  - Set vertices V. (|V| = n)
  - Set of hyperedges E. (total # pins in netlist = p)
  - Area a_u for each vertex u in V.
  - Cost c_e for each hyperedge in e.
  - An area ratio r.
- Output: 2 partitions X & Y such that
  - Total cost of hyperedges cut is minimized.
  - area(X) / (area(X) + area(Y)) is about r.

- This problem is NP-Complete!!!

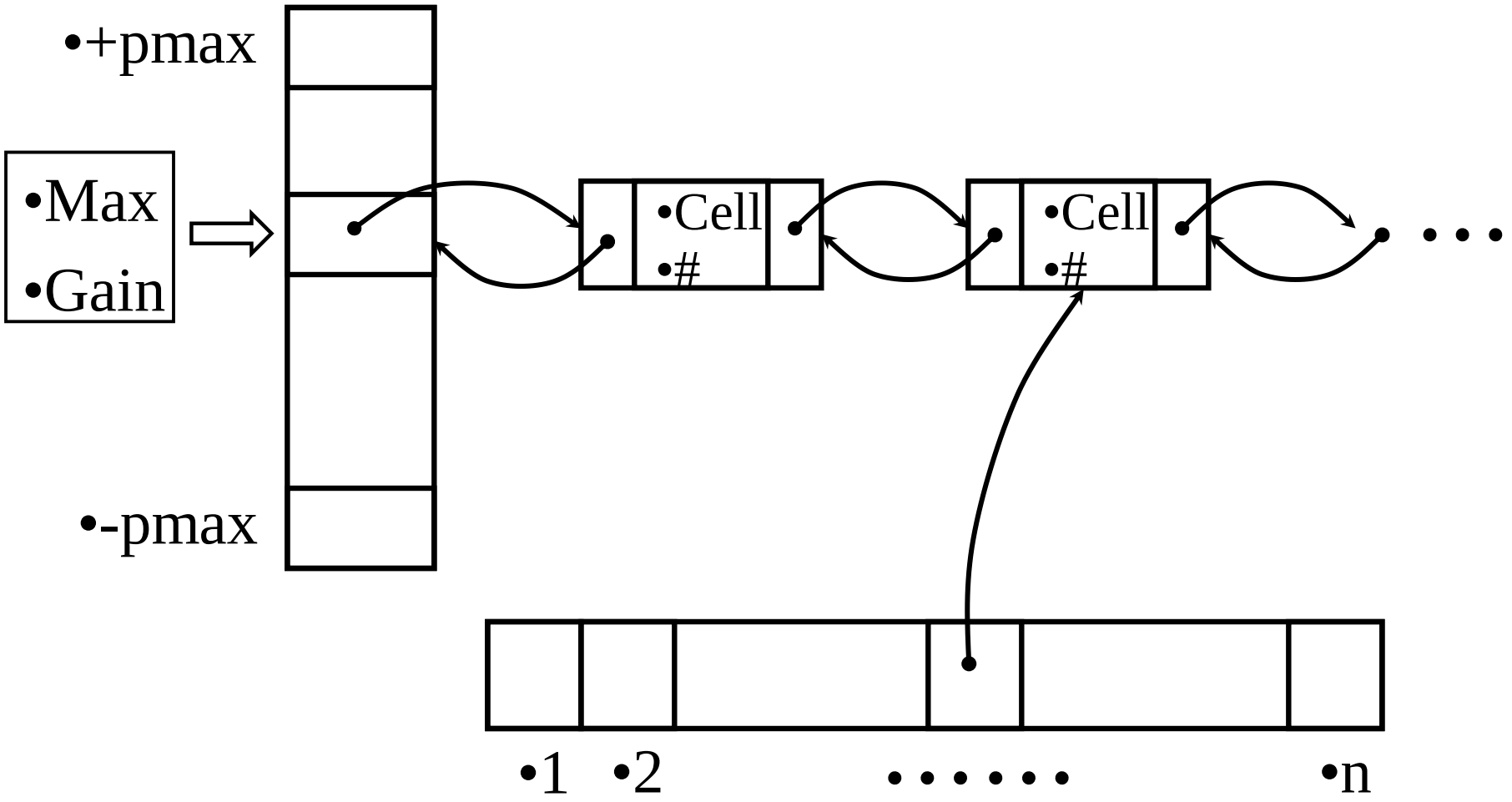
Ideas of FM Algorithm

- Similar to KL:
  - Work in passes.
  - Lock vertices after moved.
  - Actually, only move those vertices up to the maximum partial sum of gain.

- Difference from KL:
  - Not exchanging pairs of vertices.

    Move only one vertex at each time.
  - The use of gain bucket data structure.

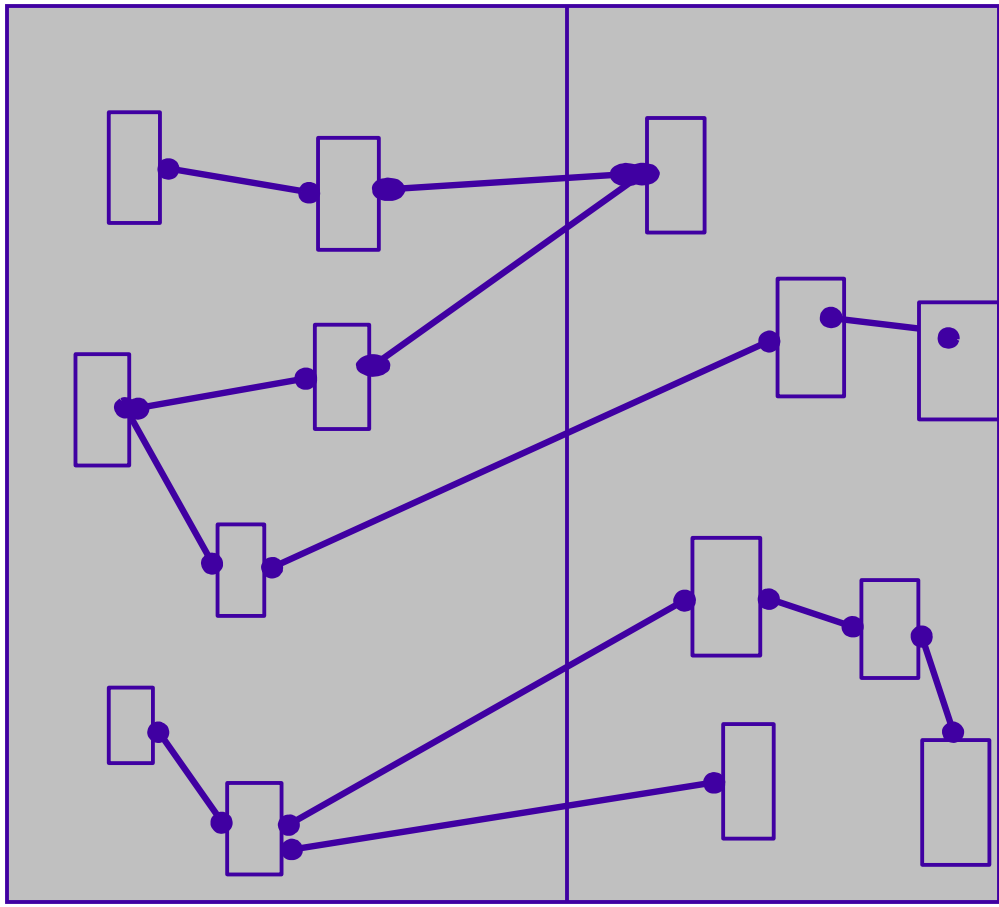
Gain Bucket Data Structure



FM Partitioning:

Moves are made based on object gain.

Object Gain: The amount of change in cut crossings
that will occur if an object is moved from
its current partition into the other partition



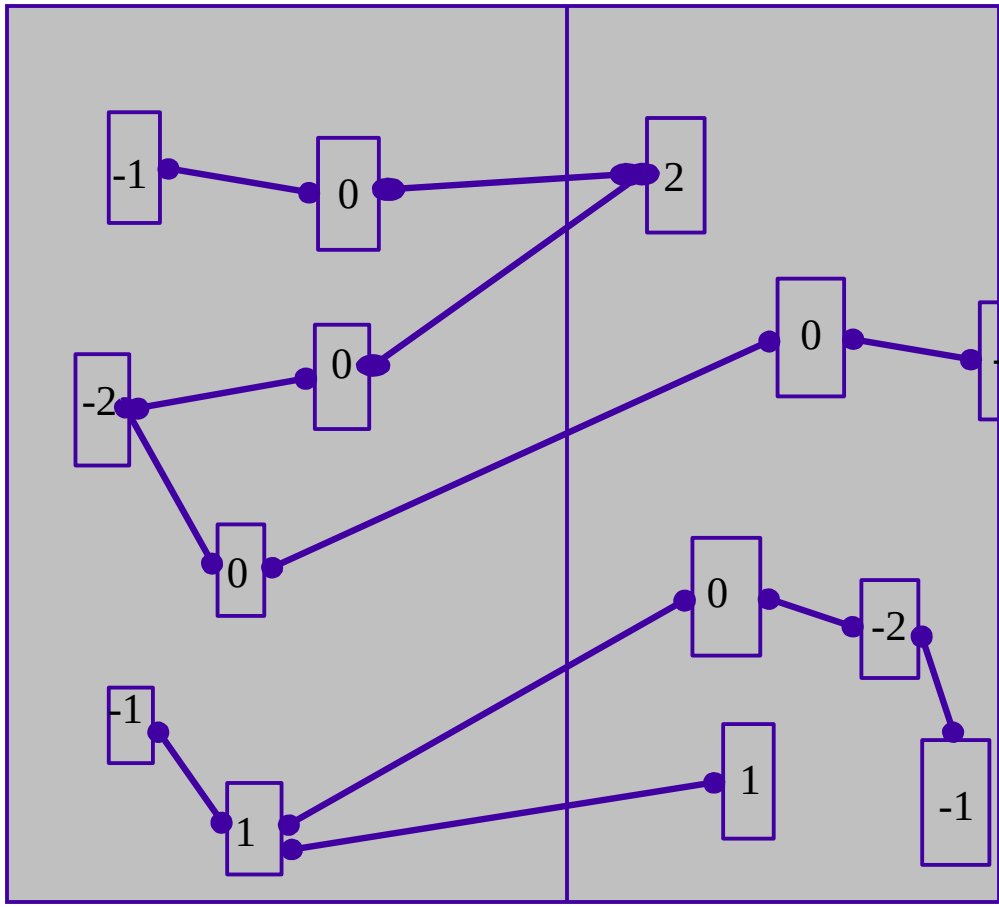
- each object is assigned a gain
- objects are put into a sorted gain list
- the object with the highest gain from the larger of the two sides is selected and moved.
- the moved object is "locked"
- gains of "touched" objects are recomputed
- gain lists are resorted

FM Partitioning:

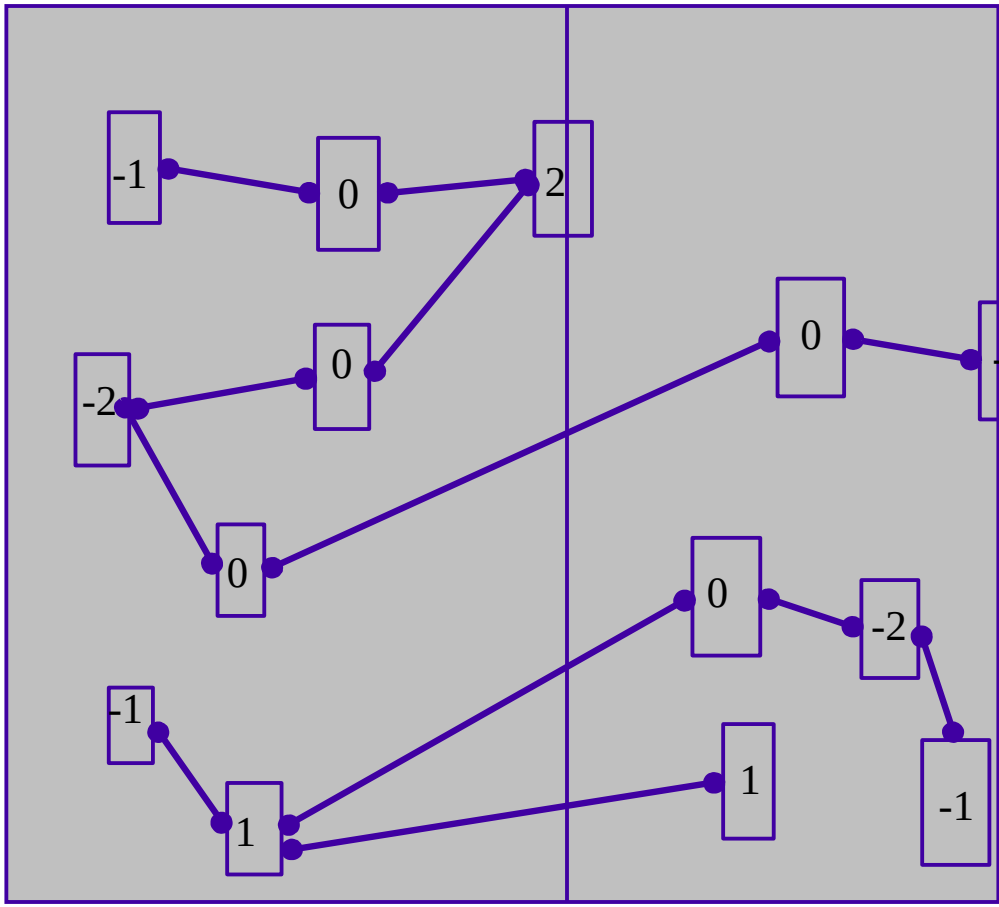
Moves are made based on object gain.

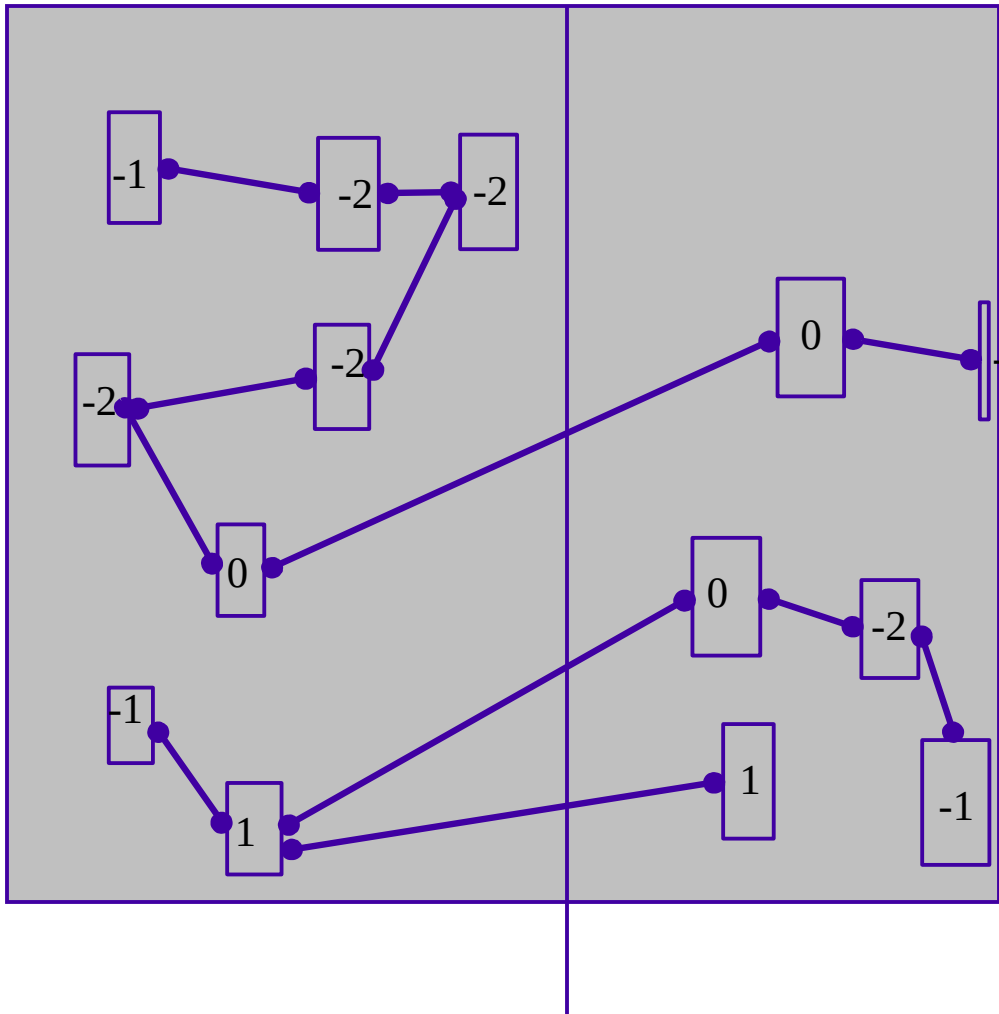
Object Gain: The amount of change in cut crossings
that will occur if an object is moved from
its current partition into the other partition

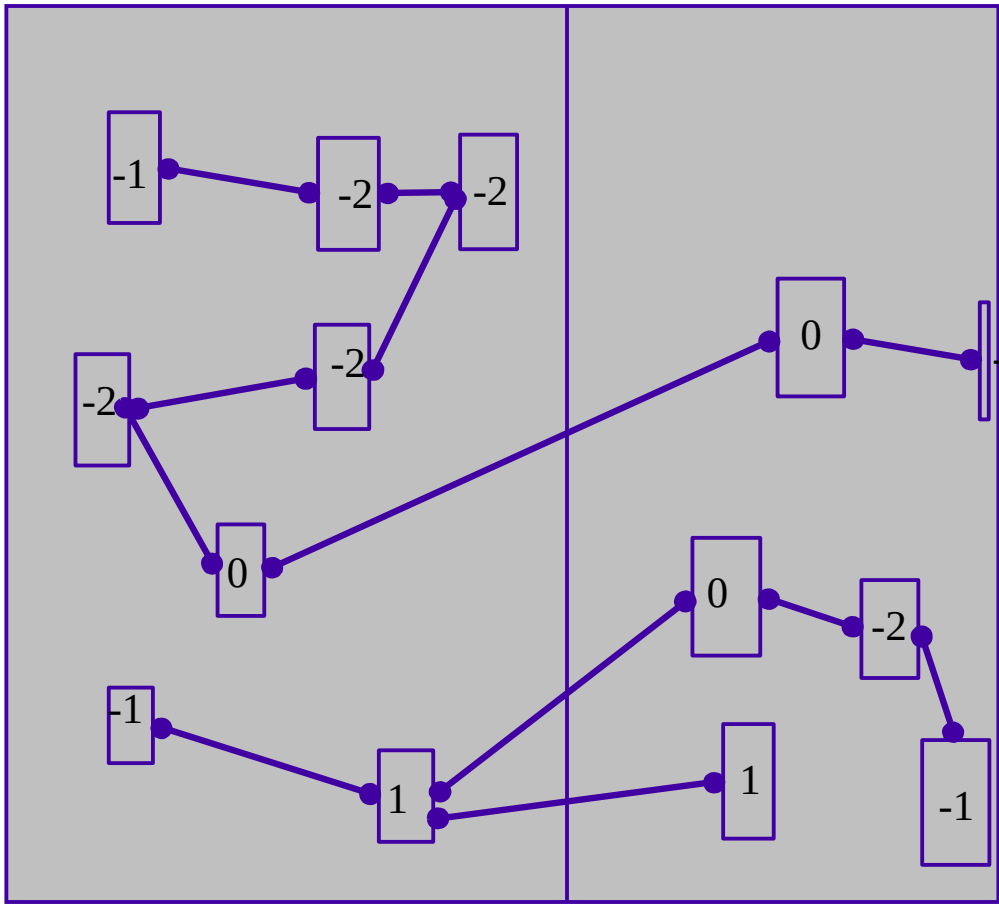
- each object is assigned a gain
- objects are put into a sorted gain list
- the object with the highest gain from the larger of the two sides is selected and moved.
- the moved object is "locked"
- gains of "touched" objects are recomputed
- gain lists are resorted

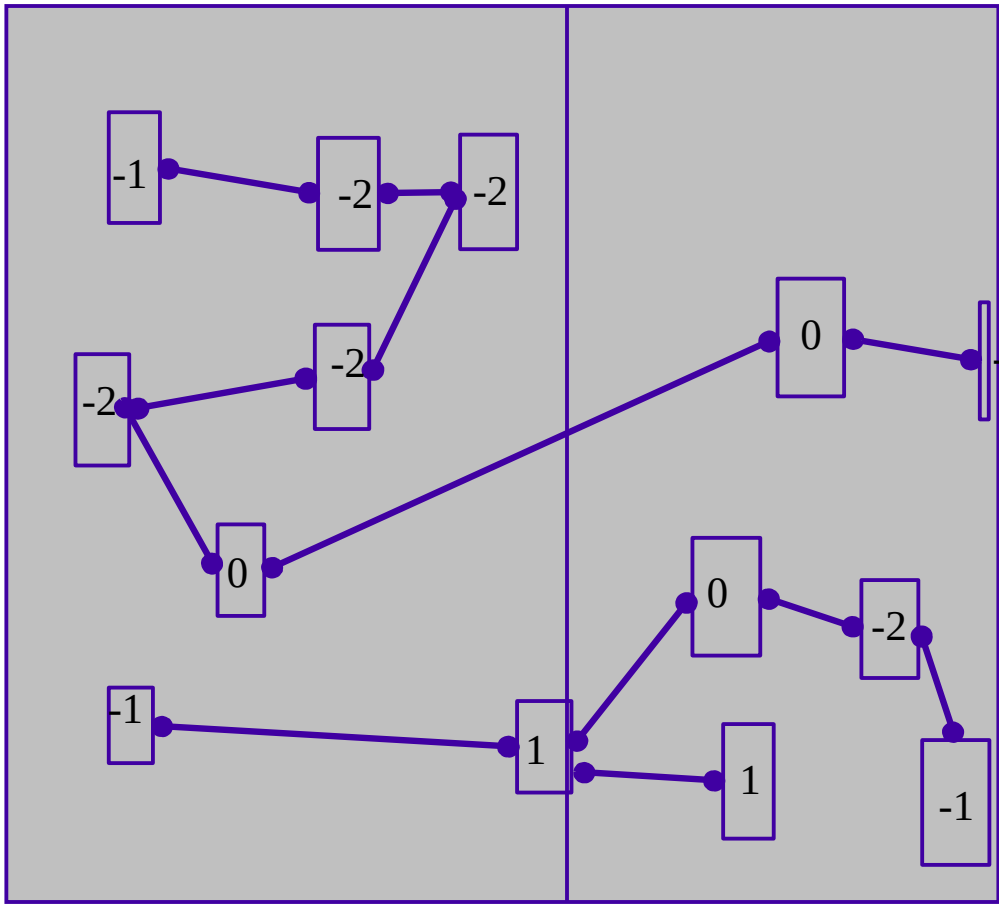


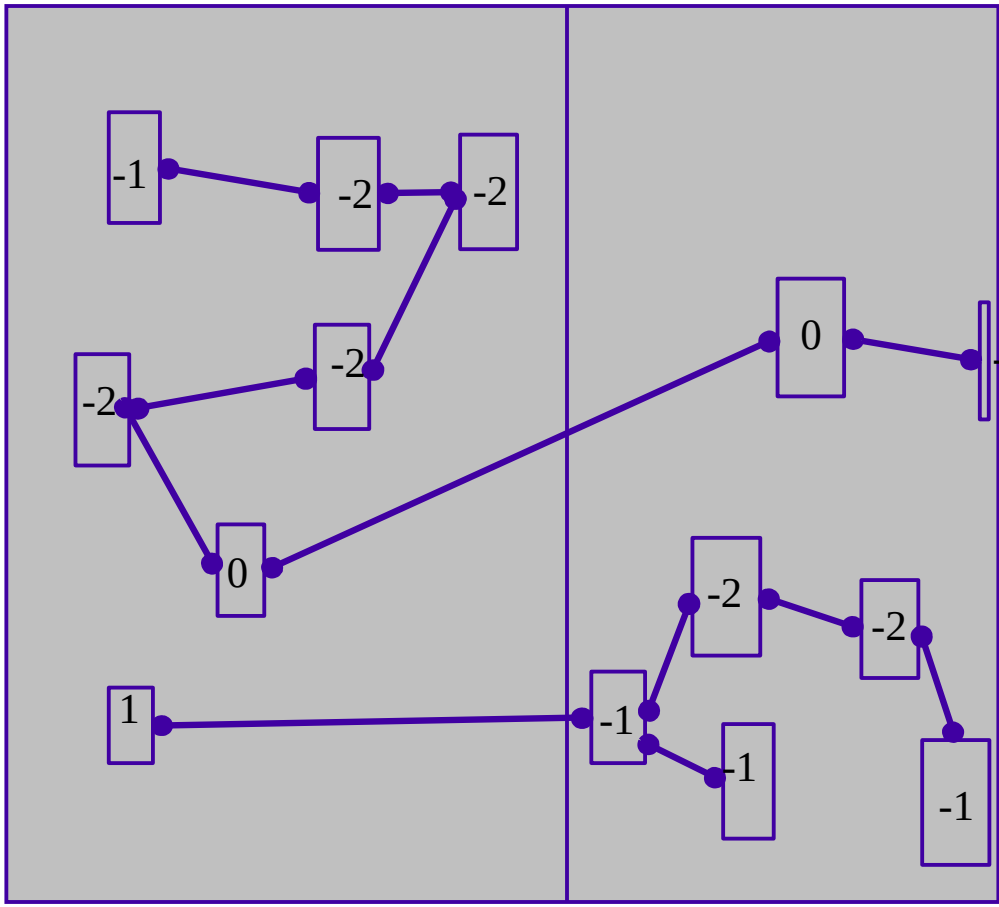
FM Partitioning:

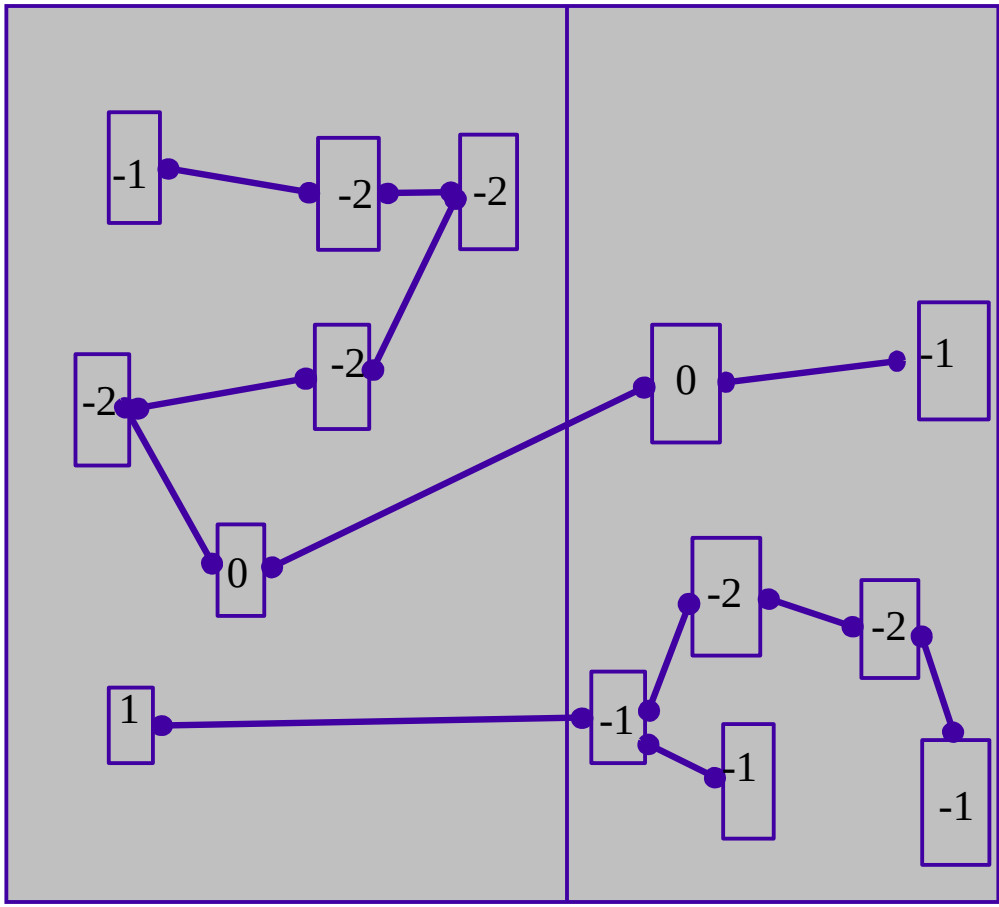


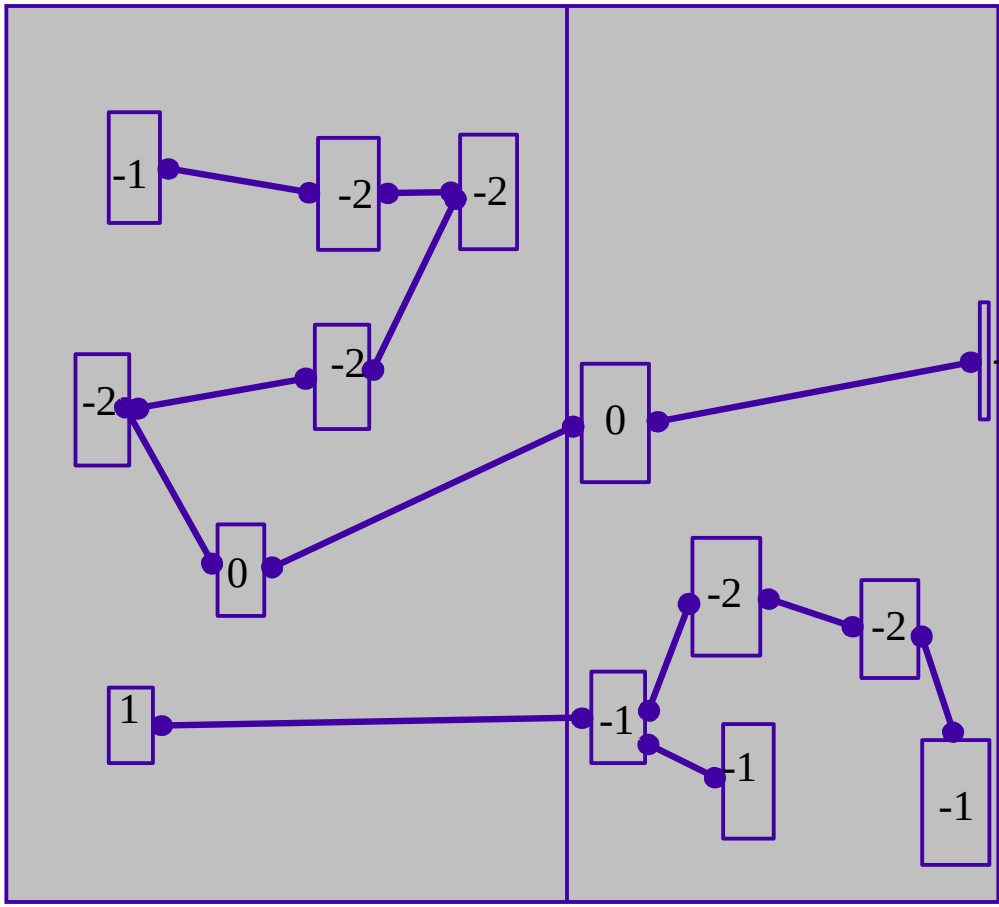

-1
-2
-2
-2
-2
0
0
0
-2
-1
1
1
-1

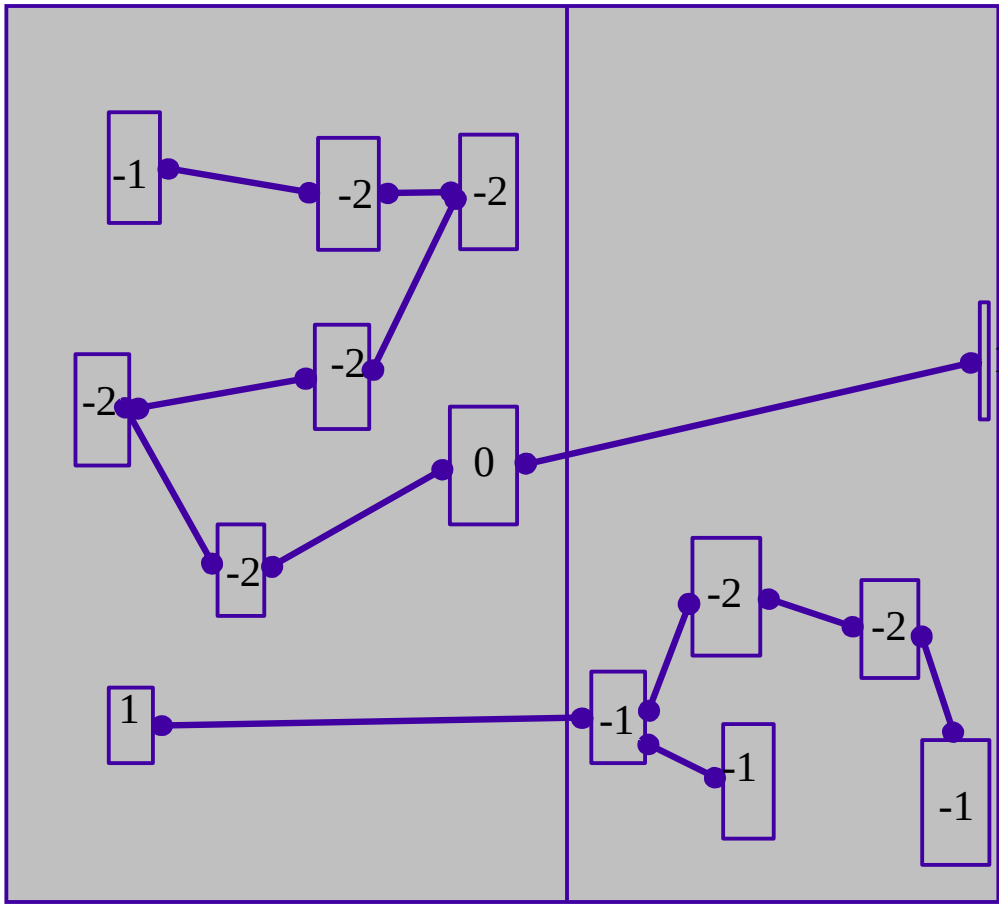

-1
-2
-2
-2
-2
0
0
0
-2
-1
1
1
-1

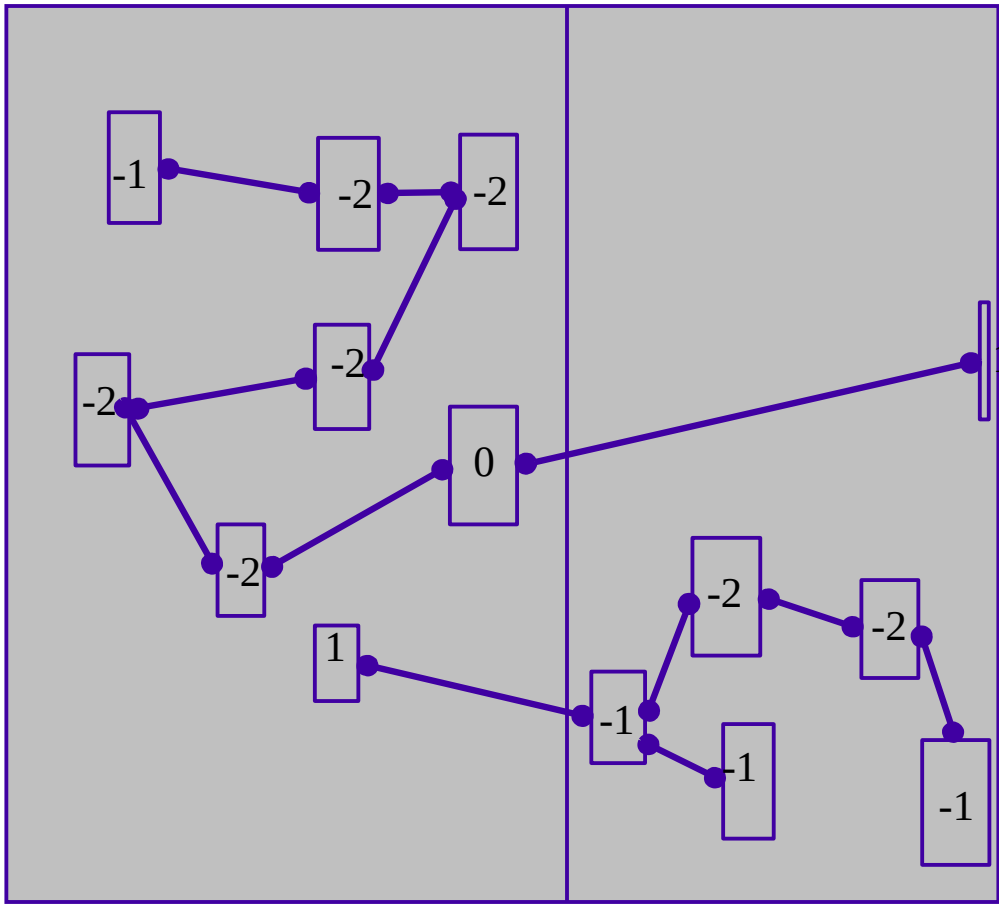

-1
-2
-2
-2
-2
0
0
0
-2
-1
1
1
-1

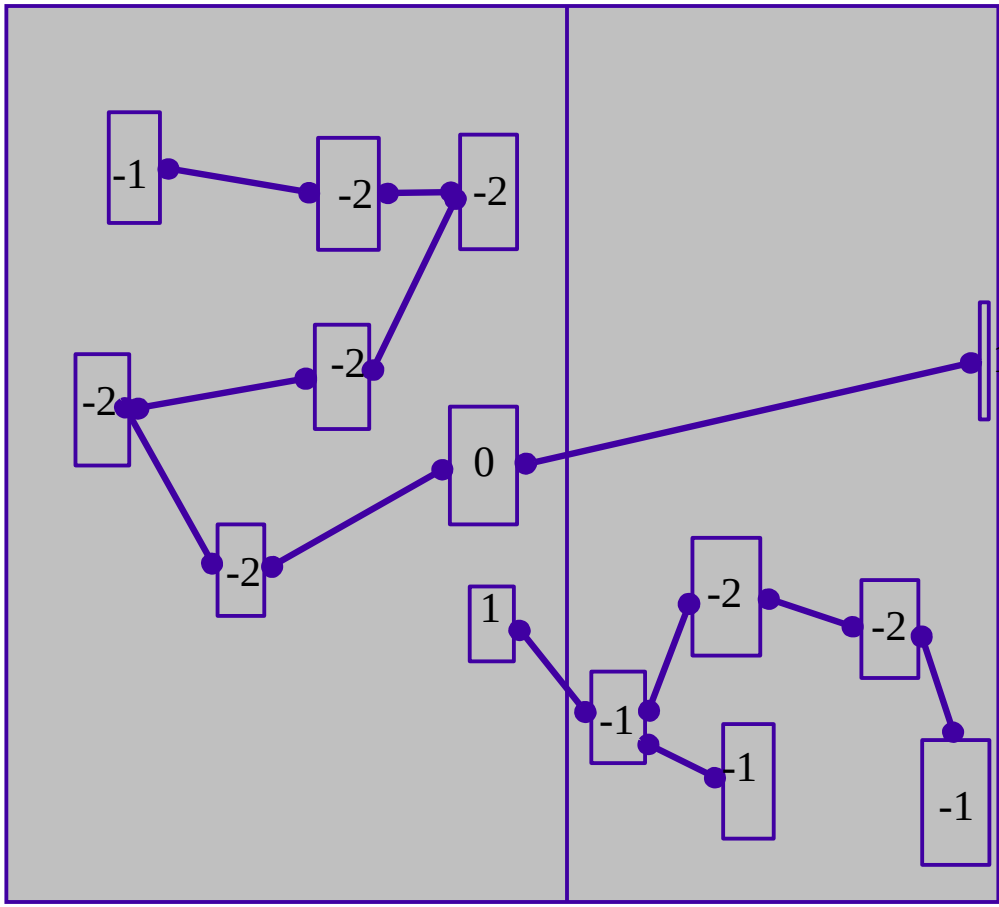

-1
-2
-2
-2
-2
0
0
-2
-2
1
-1
-1
-1

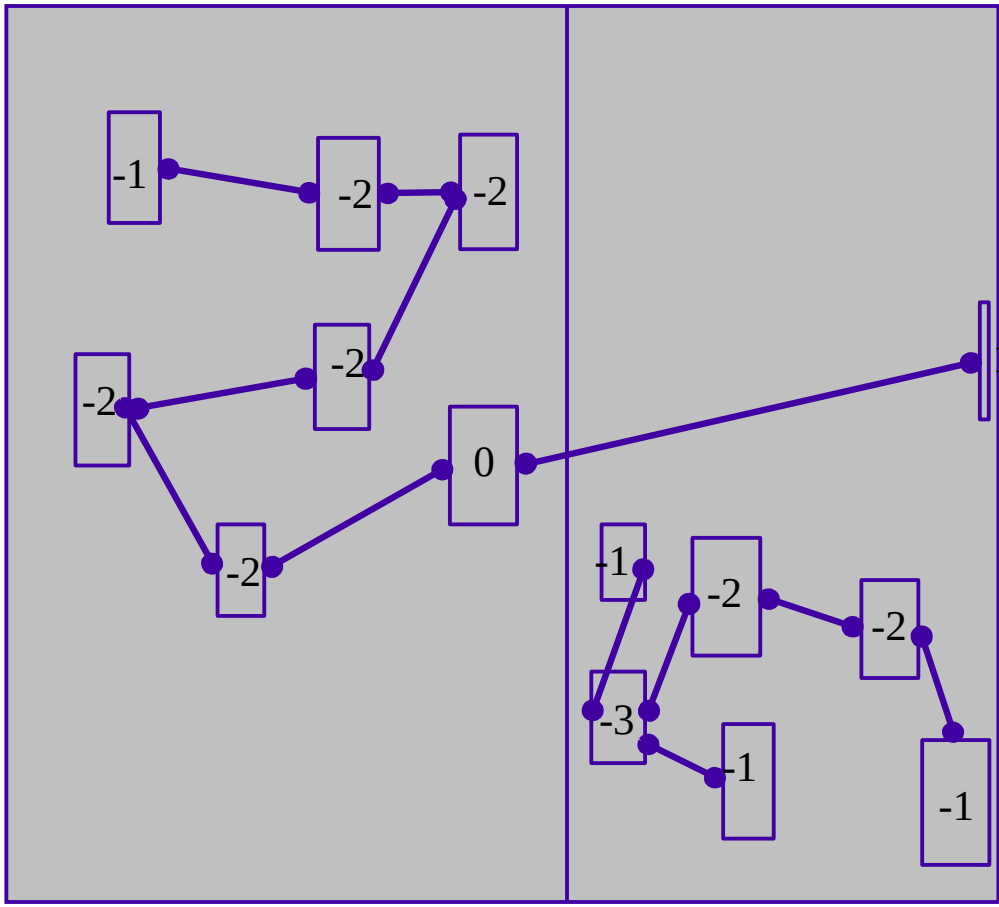

-1
-2
-2
-2
-2
0
-1
0
-2
-2
1
-1
-1
-1

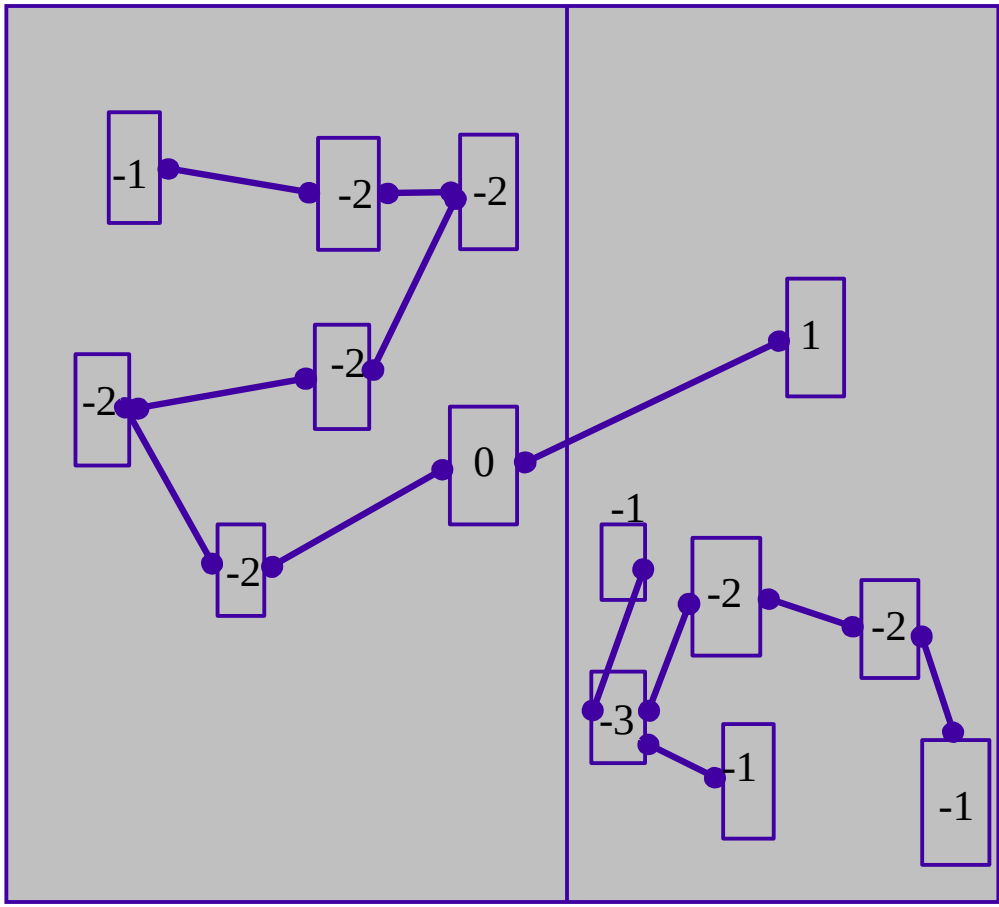

-1
-2
-2
-2
-2
0
0
1
-1
-2
-2
-1
-1

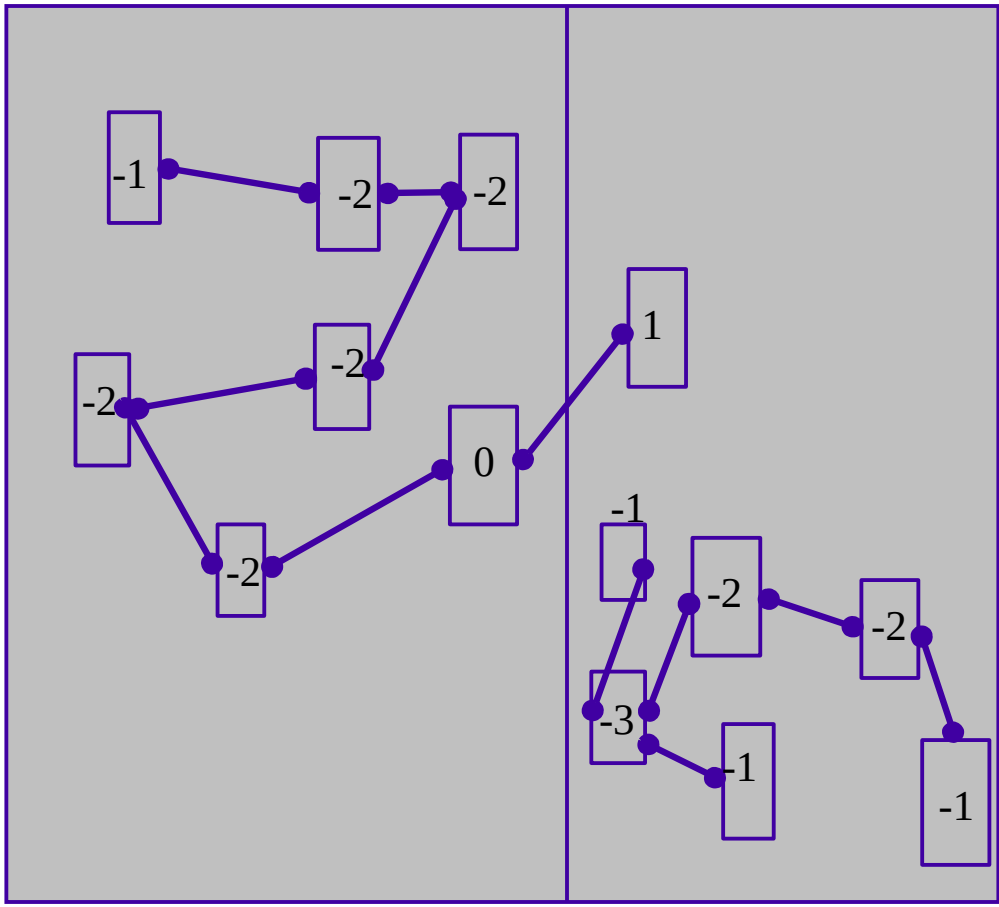

-1
-2
-2
-2
-2
0
-2
1
-2
-2
-1
-1
-1

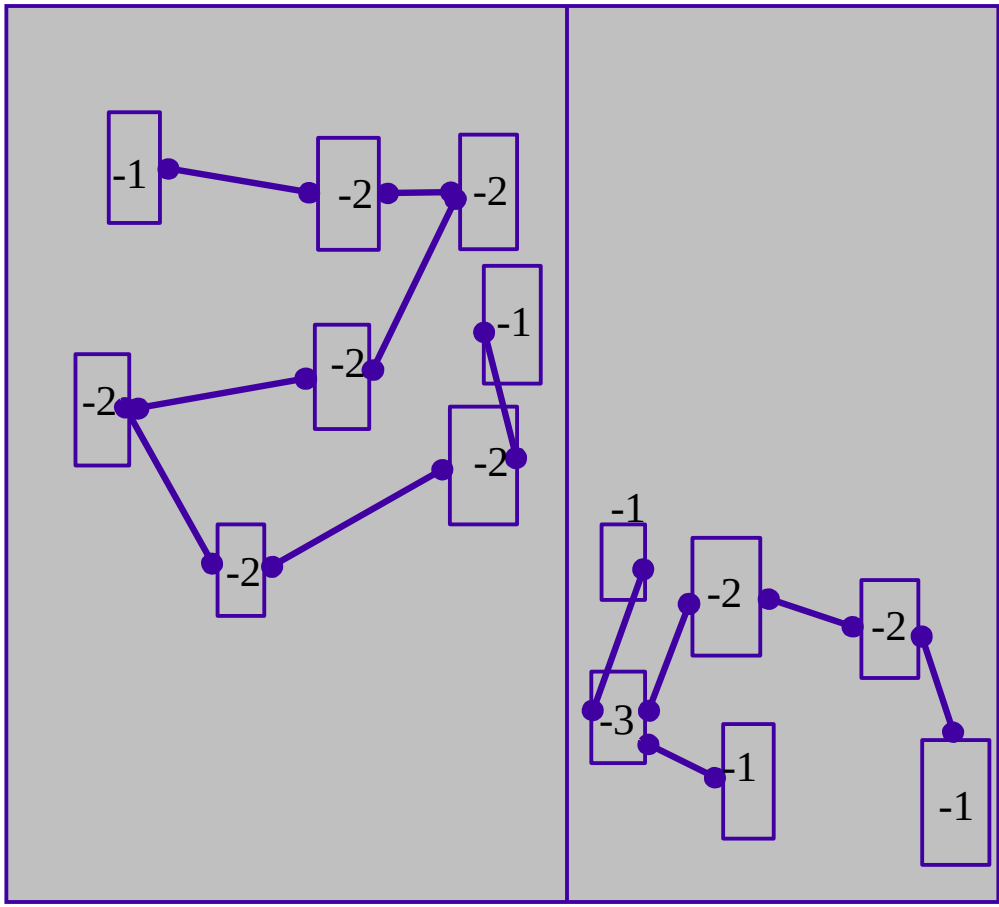

-1
-2
-2
-2
-2
0
-2
1
-2
-2
-1
-1
-1


-1
-2
-2
-2
-2
0
-2
1
-2
-2
-1
-1
-1


-1
-2
-2
-2
-2
0
-2
-1
-2
-2
-3
-1
-1


-1
-2
-2
-2
-2
1
0
-2
-1
-2
-2
-3
-1
-1


-1
-2
-2
-2
-2
1
0
-2
-1
-2
-2
-3
-1
-1


-1
-2
-2
-1
-2
-2
-2
-2
-1
-2
-2
-3
-1
-1

Ratio-Cut Algorithm

- **Removes the restriction of constant partition sizes.**

- The **cut weight W** for a cut that divides a network into two partitions, A and B , is given by ,

$$W = \sum_{a \in A, b \in B} c_{ab}$$

- The **ratio of a cut** is defined as

$$\mathbf{R = W / (|A| \, |B|)}$$

The |A| and |B| are size of a partition is equal to the number of nodes it contains (also known as the set cardinality). The cut that minimizes R is called the ratio cut.

Ratio-Cut Algorithm (contd.,)

- A network is partitioned into small, highly connected groups using ratio cuts.
- A reduced network is formed from these groups.
  —Each small group of logic cells forms a node in the reduced network.
- Finally, apply the F–M algorithm to improve the reduced network

Advantage of Ratio-cut than K-L

The K–L algorithm minimizes W while keeping partitions A and B the same size.

Look-ahead Algorithm

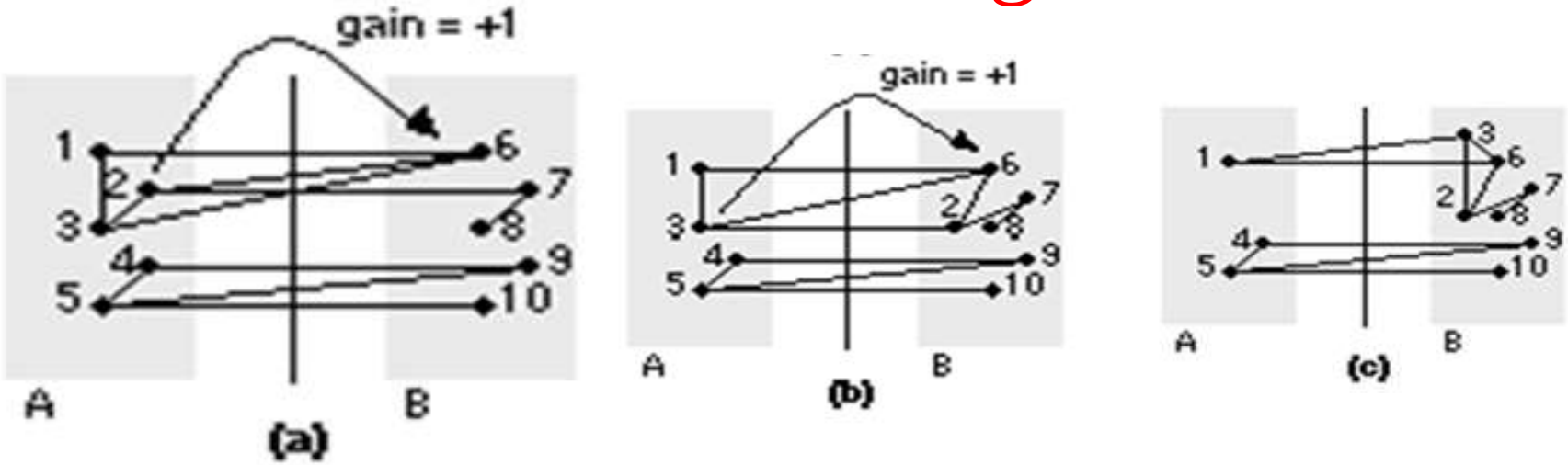
Why Look-ahead?

- **K–L and F–M algorithms consider only the immediate gain** to be made by moving a node.
- When **there is a tie between nodes with equal gain** (as often happens), there is no mechanism to make the best choice.

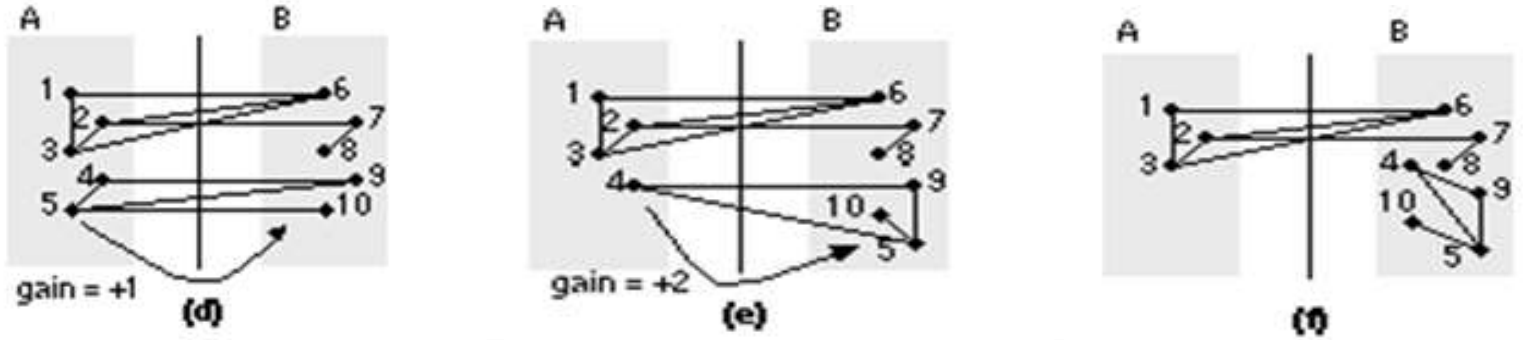
Algorithm

- The gain for the initial move is called as the **first-level gain.**
- Gains from subsequent moves are then **second-level and higher gains.**
- Define a **gain vector** that contains these gains.
- The **choice of nodes to be swapped are found** Using the **gain vector** .
- This **reduces both the mean and variation** in the number of cuts in the resulting partitions.

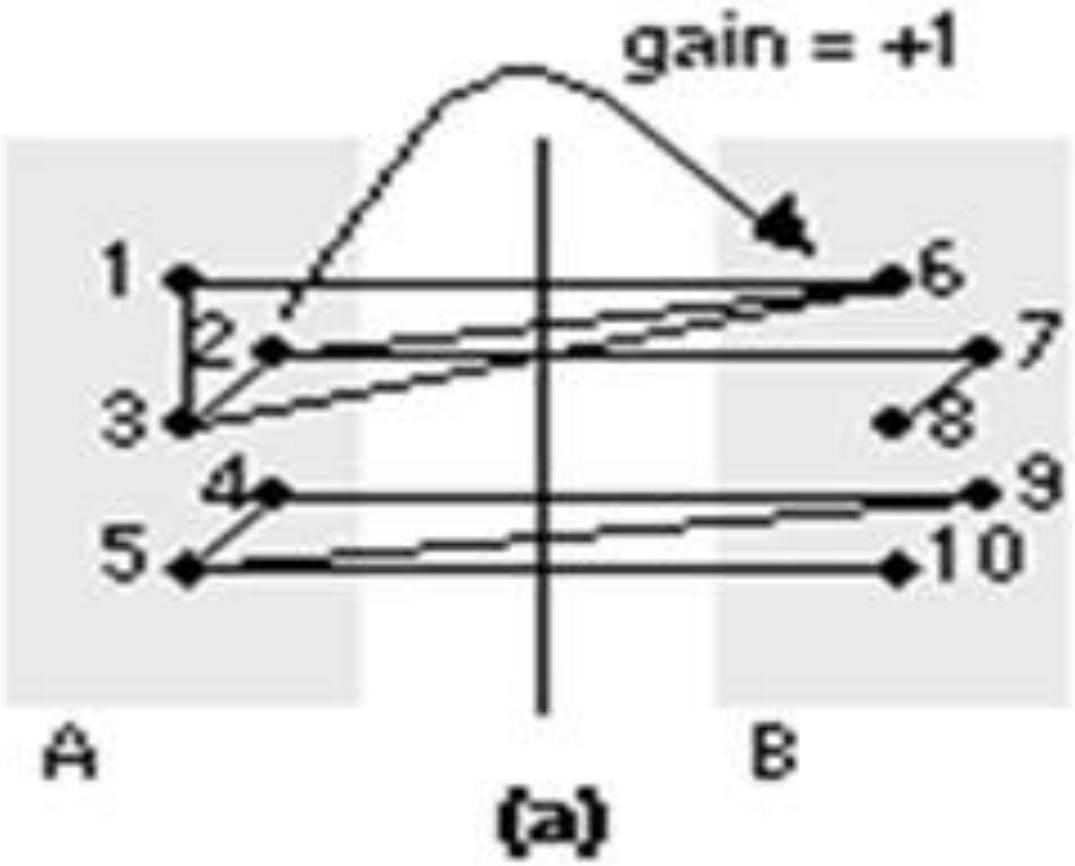
Look-ahead Algorithm



- Gain vector: Move 2 to B=+1, Move 3 to B=+1



- Gain vector: Move 5 to B=+1, Move 4 to B=+2



(a)

Look-ahead Algorithm (contd.,)

•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 – Partition I

•Partitionings (d), (e), and (f) show a second sequence – Partition II

•Partition I:

•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.

•Partition II:

•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.

Partitioning: Simulated Annealing

State Space Search Problem

- Combinatorial optimization problems (like partitioning) can be thought as a State Space Search Problem.
- A State is just a configuration of the combinatorial objects involved.
- The State Space is the set of all possible states (configurations).
- A Neighbourhood Structure is also defined (which states can one go in one step).
- There is a cost corresponding to each state.
- Search for the min (or max) cost state.

Greedy Algorithm

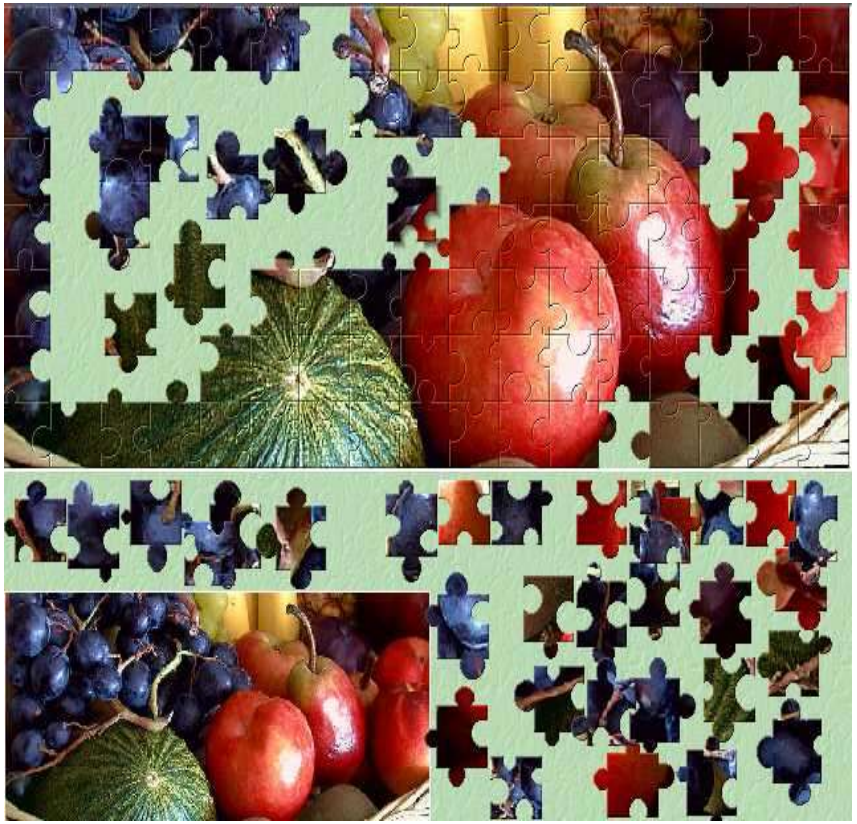
- A very simple technique for State Space Search Problem.
- Start from any state.
- Always move to a neighbor with the min cost (assume minimization problem).
- Stop when all neighbors have a higher cost than the current state.

Simulated Annealing

- Very general search technique.
- Try to avoid being trapped in local minimum by making probabilistic moves.
- Popularize as a heuristic for optimization by:
  - Kirkpatrick, Gelatt and Vecchi, “Optimization by Simulated Annealing”, Science, 220(4598):498-516, May 1983.

•Jigsaw puzzles – Intuitive usage of Simulated Annealing

- Given a jigsaw puzzle such that one has to obtain the final shape using all pieces together.
- Starting with a random configuration, the human brain unconditionally chooses certain moves that tend to the solution.
- However, certain moves that may or may not lead to the solution are accepted or rejected with a certain small probability.
- The final shape is obtained as a result of a large number of iterations.



Basic Idea of Simulated Annealing

- Inspired by the *Annealing Process*:
  - The process of carefully cooling molten metals in order to obtain a good crystal structure.
  - First, metal is heated to a very high temperature.
  - Then slowly cooled.
  - By cooling at a proper rate, atoms will have an increased chance to regain proper crystal structure.
- Attaining a min cost state in simulated annealing is analogous to attaining a good crystal structure in annealing.

The Simulated Annealing Procedure

Let t be the initial temperature.

Repeat

Repeat

- Pick a neighbor of the current state randomly.
- Let c = cost of current state.

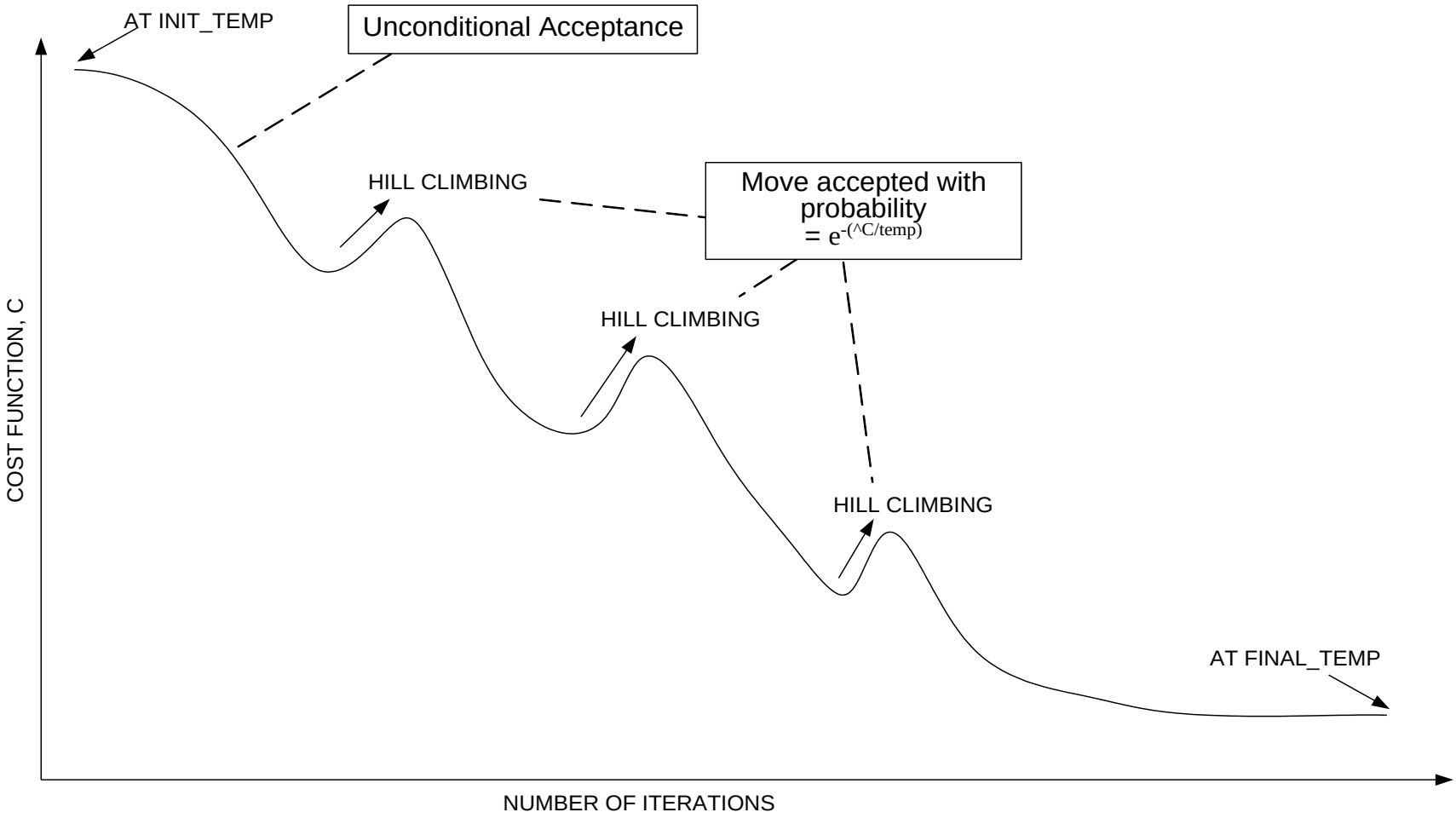
  Let c' = cost of the neighbour picked.
- If c' < c, then move to the neighbour (downhill move).
- If c' > c, then move to the neighbour with probablility $e^{-(c'-c)/t}$ (uphill move).

Until equilibrium is reached.

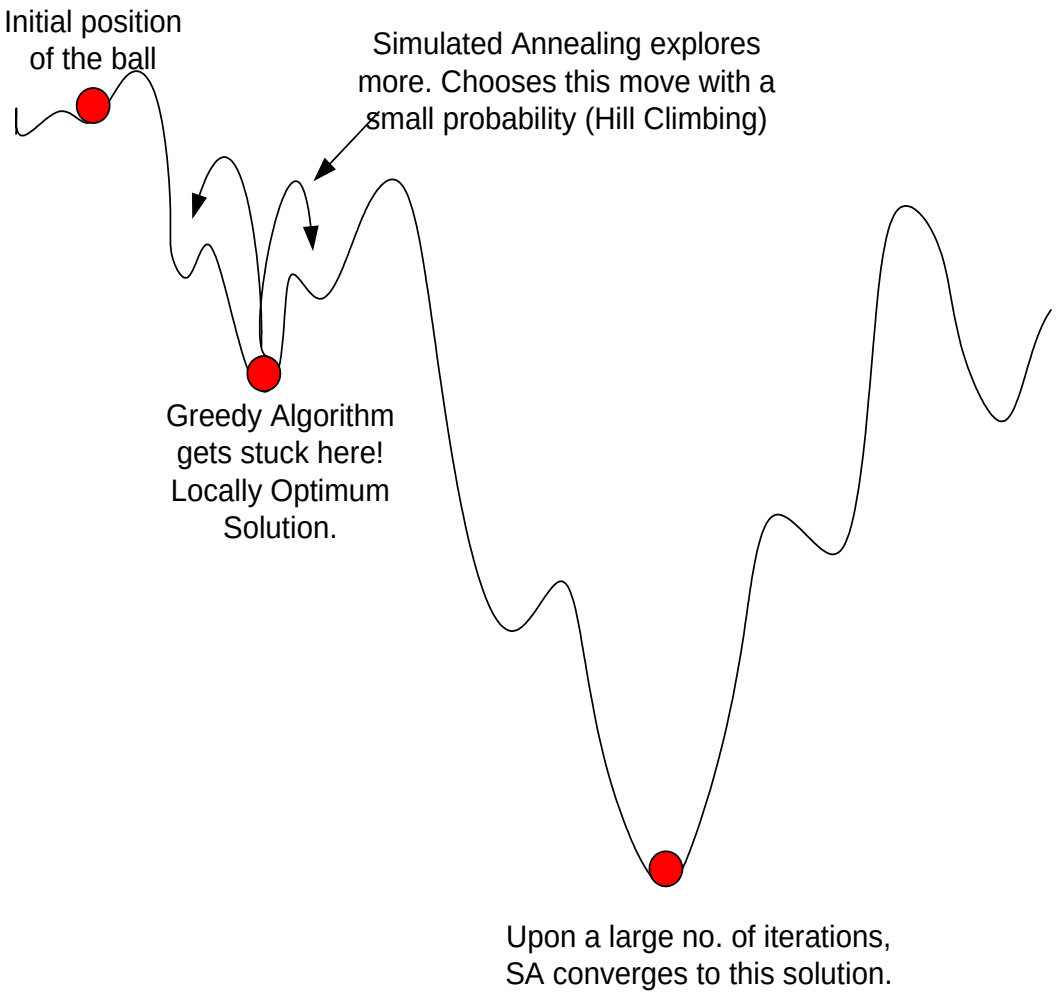
Reduce t according to cooling schedule.

Until Freezing point is reached.

•Convergence of simulated annealing



•Ball on terrain example – SA vs Greedy Algorithms



Simulated Annealing

- **Takes an existing solution** and then makes successive changes in a **series of random moves**.
- Each move is accepted or rejected based on an **energy function**.
- In the **Interchange method**,
  - **Accept the new trial configuration** only if the **energy function decreases**,

    which means the new configuration is an improvement
- But in the **simulated Annealing,**
  - **Accept the new configuration** even if the **energy function increases** for the new configuration—which means things are getting worse.
  - The probability of **accepting a worse configuration** is controlled by the **exponential expression exp(–ΔE / T)**,

    where, **Δ E** - the resulting increase in the energy function.

    T - a variable that can be controlled and corresponds to the temperature in the annealing of a metal cooling (this is why the process is called simulated annealing).

Simulated Annealing

- A parameter that relates the temperatures, T_i and T_{i+1}, at the i th and i + 1 th iteration:

$$T_{i+1} = \alpha T_i .$$

- As the temperature is slowly decreased, the probability of making moves that increase the energy function gets decreased.
- **Cooling schedule –** The critical parameter of the simulated-annealing algorithm is the rate at which the temperature T is reduced.
- Finally, as the **temperature approaches zero**, refuse to make any moves that increase the energy of the system and the system falls and comes to rest at the **nearest local minimum**.
- The **minimums of the energy function** correspond to possible solutions.
- The best solution is the **global minimum**.

Simulated Annealing

- **Requirement of Simulated Annealing:**
- To find a good solution, a local minimum close to the global minimum, requires a **high initial temperature and a slow cooling schedule**.
- **Disadvantage:**
- This results in many trial moves and very long computer run time.(it gives optimum)
- **Advantage:**
- To solve **large graph problems**
- **Hill climbing-** Accept moves that seemingly take us away from a desirable solution to allow the system to **escape from a local minimum** and find other, better, solutions.

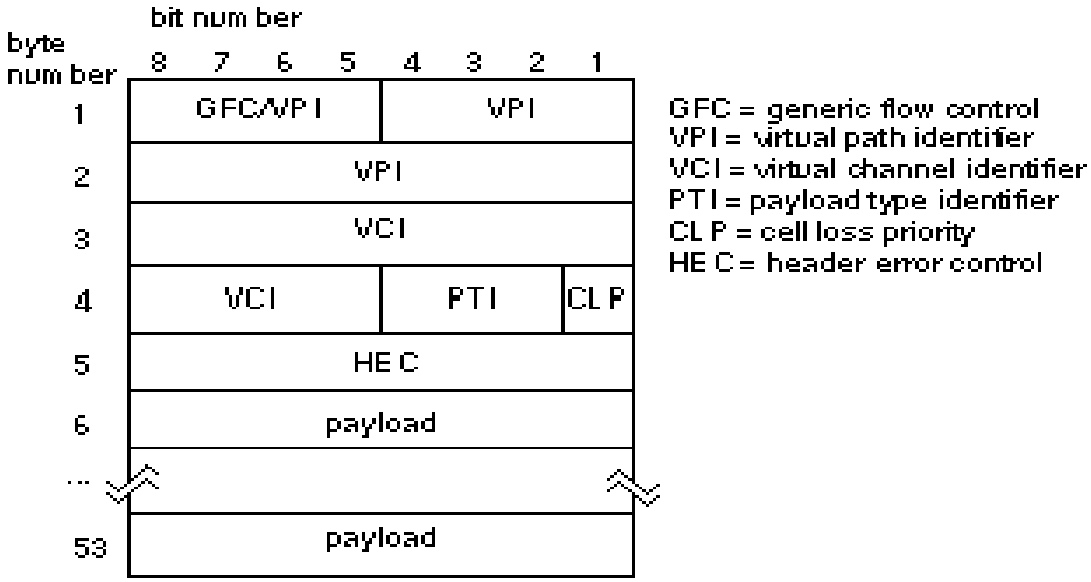
Other Partitioning Objectives

Constraints or Objectivies	Purpose	Implemented
Timing Constraints	certain logic cells in a system may need to be located on the same ASIC in order to avoid adding the delay of any external interconnections	Adding weights to nets to make them more important than others.
Power Constraints	Some logic cells may consume more power than others	To assign more than rough estimates of power consumption for each logic cell at the system planning stage, before any simulation has been completed.
Technology Constraints	To include memory on an ASIC	It will keep logic cell together requiring similar technology
Cost Constraints	To use low cost package	To keep ASICs below a certain size.
Test Constraints	Maintain Observability and Controllability	It require that we force certain connection to external

FPGA Partitioning
- An asynchronous transfer mode (ATM) connection simulator

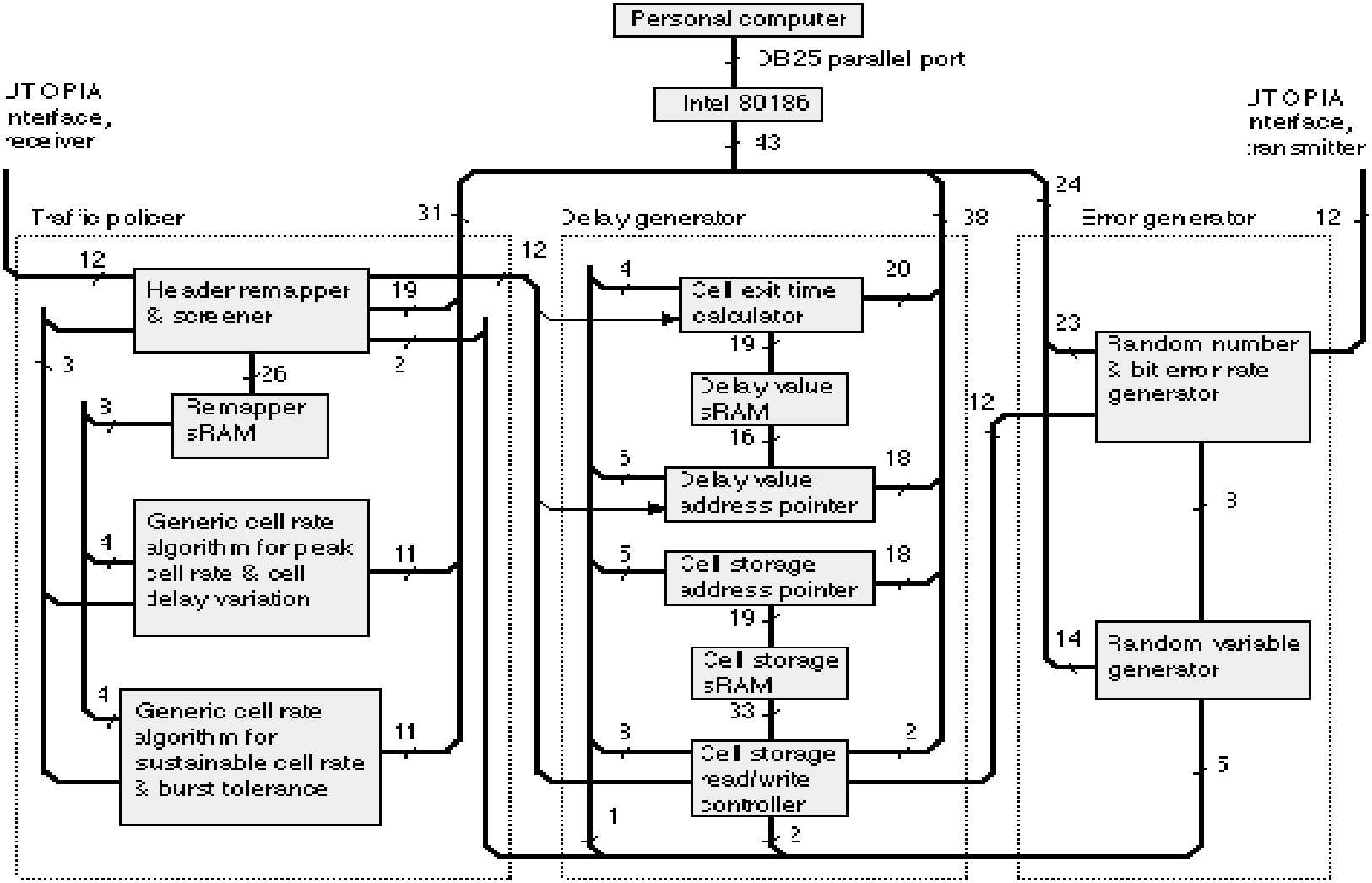
- ATM is a signaling protocol for many different types of traffic including constant bit rates (voice signals) as well as variable bit rates (compressed video).
- The **ATM Connection Simulator is a card** that is connected to a computer.
- Under computer control the card monitors and corrupts the ATM signals to simulate the effects of real networks.
- An example would be to test **different video compression algorithms. Compressed video is very bursty (brief periods of very high activity), has very strict delay constraints, and is susceptible to errors.**

•Asynchronous transfer mode (ATM) cell format



•FIGURE 15.4 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



FPGA Partitioning

- The **simulator** is partitioned into the three major blocks
  - ATM traffic policer - which regulates the input to the simulator
  - ATM cell delays generator – which delays ATM cell, reorders ATM cells and inserts ATM cells with valid ATM cell headers.
  - ATM cell error generator – which produce bit errors and four random variables that are needed by the other two blocks.
- The **Traffic Policer** performs the following operations:
  - Performs header screening and remapping.
  - Checks ATM cell Conformance.
  - Delete selected ATM Cells.
- The **delay generator** delays, misinserts and reorders the target ATM cells.
- The **error generator** performs the following operations:
  - Payload bit error ratio generation: The user specifies the Bernoulli probability P_{BER} of the payload bit error ratio.
  - Random variable generation for ATM cell loss, misinsertion, reordering and deletion.

Find the connectivity matrix for the ATM Connection Simulator shown in Figure. Use the following scheme to number the blocks and ordering of the matrix rows and columns: 1 = Personal Computer, 2 = Intel 80186, 3 = UTOPIA receiver, 4 = UTOPIA transmitter, 5 = Header remapper and screener, 6 = Remapper SRAM, . . . 15 = Random-number and bit error rate generator, 16 = Random-variable generator. All buses are labeled with their width except for two single connections (the arrows).

Automatic Partitioning with FPGAs

- Altera hardware design language (AHDL)

  - To direct the partitioner to automatically partition logic into chips within the same family, using AUTO keyword,

```
DEVICE top_level  IS AUTO ; % the partitioner assign logic
```

- CLIQUE keyword to keep logic together

```
CLIQUE fast_logic
  BEGIN
        | shift_register : MACRO;% keep this in one device
END;
```

Power Dissipation

- **Dynamic Power Dissipation**
  - **Switching current** from charging and discharging parasitic capacitance.
  - **Short-circuit current** when both n -channel and p -channel transistors are momentarily on at the same time.

- **Static Power Dissipation**
  - **Subthresold current**
  - **Leakage current**

Switching current

- **From charging and discharging of parasitic capacitance**
- When the **p -channel transistor in an inverter is charging a capacitance**, ***C***, *at a* frequency, *f* ,
  - *the current through the transistor is I=C (d V /d t).*
  - *The power dissipation* is P=VI=*CV (d V /d t) for one-half the period of the input, t = 1/(2 f).*
  - *The power* dissipated in the p -channel transistor is thus

$$\int_0^{\frac{1}{2f}} P\,dt = \int_0^{V_{DD}} CV\,dV = \frac{1}{2}CV_{DD}^2$$

- When the **n -channel transistor discharges the capacitor**, the power dissipation is equal.(ie., $\frac{1}{2}CV_{DD}^2$)
- Then **total power dissipation**,

$$P_1 = fCV_{DD}^2$$

- **Most of the power dissipation in a CMOS ASIC** arises from this source—the switching current.

- The best way **to reduce power is to reduce V_{DD} and to reduce *C*** , *the amount of capacitance we have* to switch.

Short circuit current

- **Both n-channel and p-channel transistors momentarily on at the same time**

- The **short-circuit current or crowbar current** can be particularly important for **output drivers** **and** **large clock buffers.**

- For a CMOS inverter, the **power dissipation** due to the crowbar current is

$$P_2 = \frac{\beta f t_{rf}}{12}\left(V_{DD} - 2V_{tn}\right)^3$$, Transistor gain factor $$\beta = \frac{I_{DS}}{\left[\left(V_{GS} - V_{tn}\right) - \frac{1}{2}V_{DS}\right]V_{DS}}$$

  - Where $\beta = (W/L)\mu C_{OX}$ is the same for both p - and n -channel transistors.
  - The **threshold voltages V_{tn}** *are assumed equal for both transistor types.*
  - ***t_{rf} is the rise and fall time*** *(assumed equal) of the input signal [Veendrick, 1984].*

Problem on Power Dissipation

- consider an output buffer that is capable of sinking 12mA at an output voltage of 0.5 V., Derive the transistor gain factor (Assume $V_{GS}=V_{DD}=3.3V$; $V_{th}=0.65V$)
- If the output buffer is switching at 100 MHz and the input rise time to the buffer is 2ns, Calculate the power dissipation due to short-circuit current.
- If the output load is 10 pF, Calculate the dissipation due to switching current.
- What do you infer from this?

- Inference:

  ($\beta=0.01AV^{-1}$, short-circuit current, $P_2=0.00133W$ or 1mW, switching current, $P_1=0.01089W$ or 10mW)

  - **short-circuit current is typically less than 10 percent of the switching current.**

Subthreshold current

- **CMOS transistor is never completely off**
- **When the gate-to-source voltage, V_{GS} , *of an MOS transistor is less than the* threshold voltage, V_t** , the transistor conducts a very small subthreshold current in the subthreshold region

$$I_{DS} = I_0 \exp\left(\frac{qV_{GS}}{nKT} - 1\right)$$

  - where I_0 *is a constant, and the constant, n, is normally between 1 and 2.*

- The **slope, S, of the transistor current in the subthreshold region** is

$$S = \frac{nkT}{q} \ln 10$$

- **Find the slope of transistor current S at a junction temperature, T = 125 °C (400 K) and assuming n = 1.5** (assume q = $1.6 * 10^{-19}$ Fm $^{-1}$, k = $1.38 * 10^{-23}$ JK $^{-1}$). What do you infer from the result of slope of transistor current.

Inference:

S = 120 mV/decade which does not scale.

The constant value of S = 120 mV/decade means it takes 120 mV to reduce the subthreshold current by a factor of 10 in any process.

Leakage Current

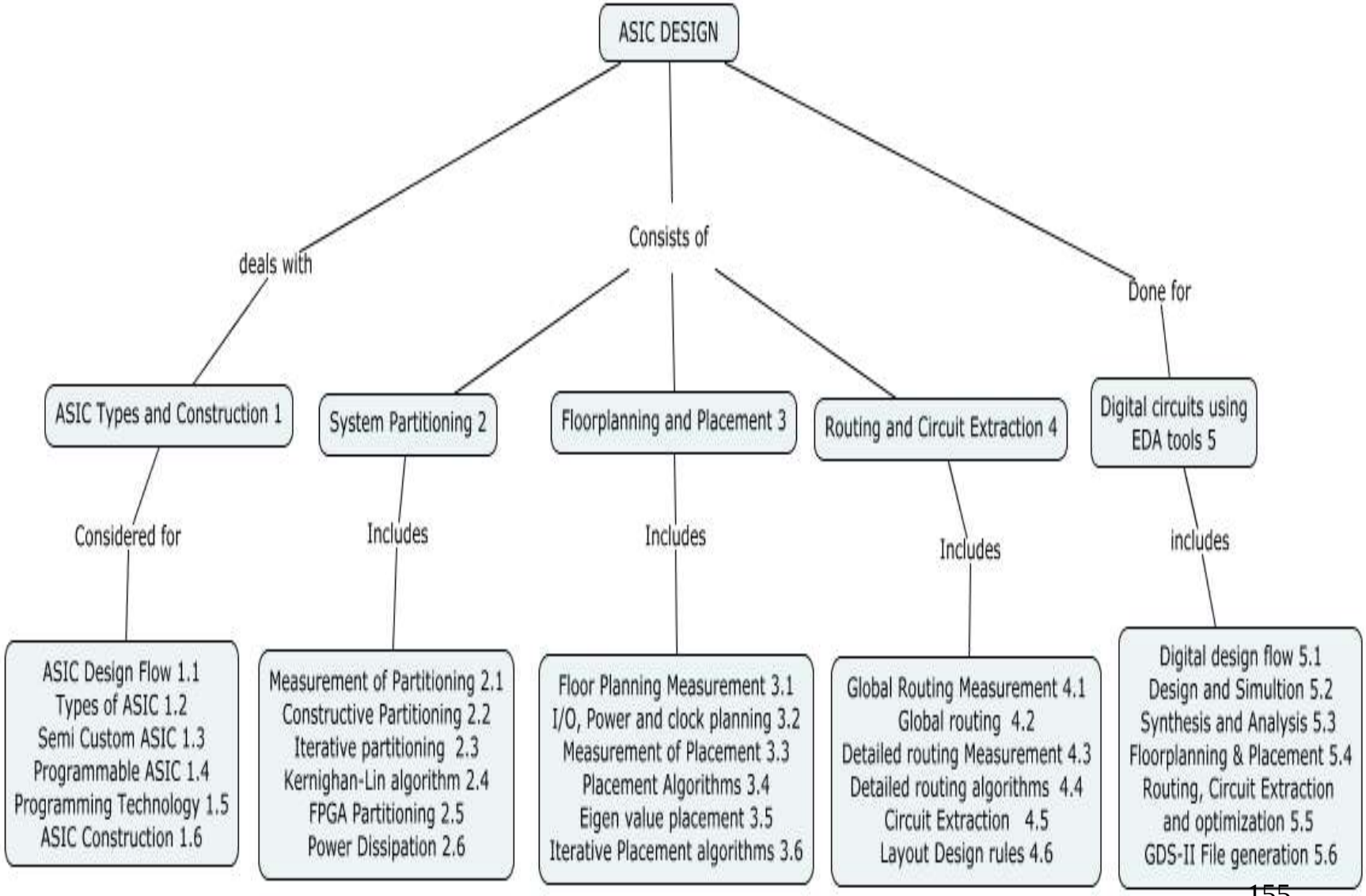
- Transistor leakage is caused by the fact that a **reverse-biased diode conducts a very small leakage current.**
- The **sources and drains of every transistor**, as well as the **junctions between the wells and substrate**, form **parasitic diodes.**
- The parasitic-diode leakage currents are **strongly dependent** on the
  - **type and quality of the process**
  - **temperature.**
- The parasitic diodes have two components in parallel: **an area diode and a perimeter diode.**
- **The leakage current due to perimeter diode is larger than area diode.**
- **The ideal parasitic diode currents** are given by the following equation:

$$I = I_s \exp\left(\frac{qV_D}{nKT} - 1\right)$$

Module III
Floorplanning

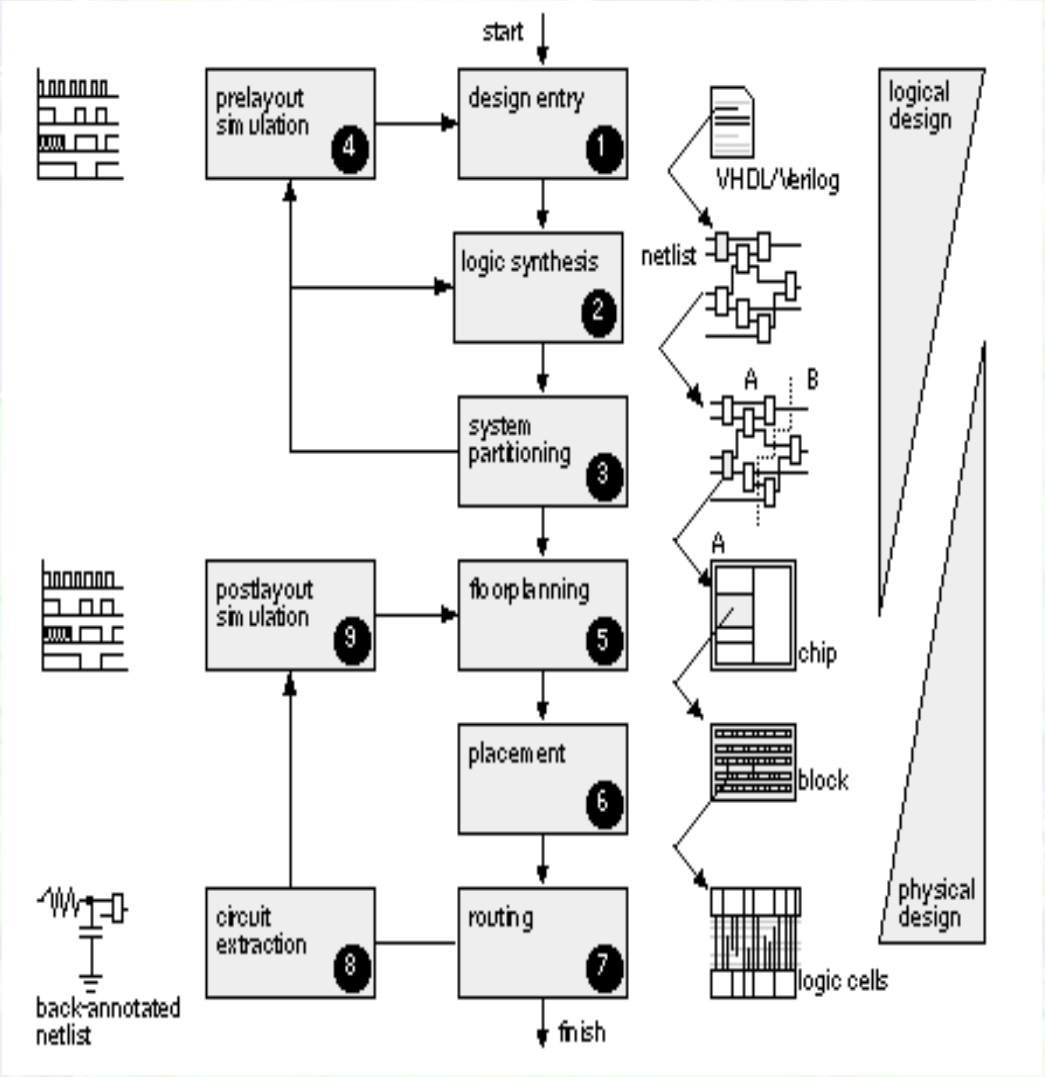
Dr.(Mrs).D.Gracia Nirmala Rani
Assistant Professor
ECE Department
Thiagarajar College of Engineering
Madurai-15
Email : gracia@tce.edu

Concept MAP


ASIC DESIGN
deals with
Consists of
Done for
ASIC Types and Construction 1
System Partitioning 2
Floorplanning and Placement 3
Routing and Circuit Extraction 4
Digital circuits using EDA tools 5
Considered for
Includes
Includes
Includes
includes
ASIC Design Flow 1.1
Types of ASIC 1.2
Semi Custom ASIC 1.3
Programmable ASIC 1.4
Programming Technology 1.5
ASIC Construction 1.6
Measurement of Partitioning 2.1
Constructive Partitioning 2.2
Iterative partitioning 2.3
Kernighan-Lin algorithm 2.4
FPGA Partitioning 2.5
Power Dissipation 2.6
Floor Planning Measurement 3.1
I/O, Power and clock planning 3.2
Measurement of Placement 3.3
Placement Algorithms 3.4
Eigen value placement 3.5
Iterative Placement algorithms 3.6
Global Routing Measurement 4.1
Global routing 4.2
Detailed routing Measurement 4.3
Detailed routing algorithms 4.4
Circuit Extraction 4.5
Layout Design rules 4.6
Digital design flow 5.1
Design and Simultion 5.2
Synthesis and Analysis 5.3
Floorplanning & Placement 5.4
Routing, Circuit Extraction and optimization 5.5
GDS-II File generation 5.6

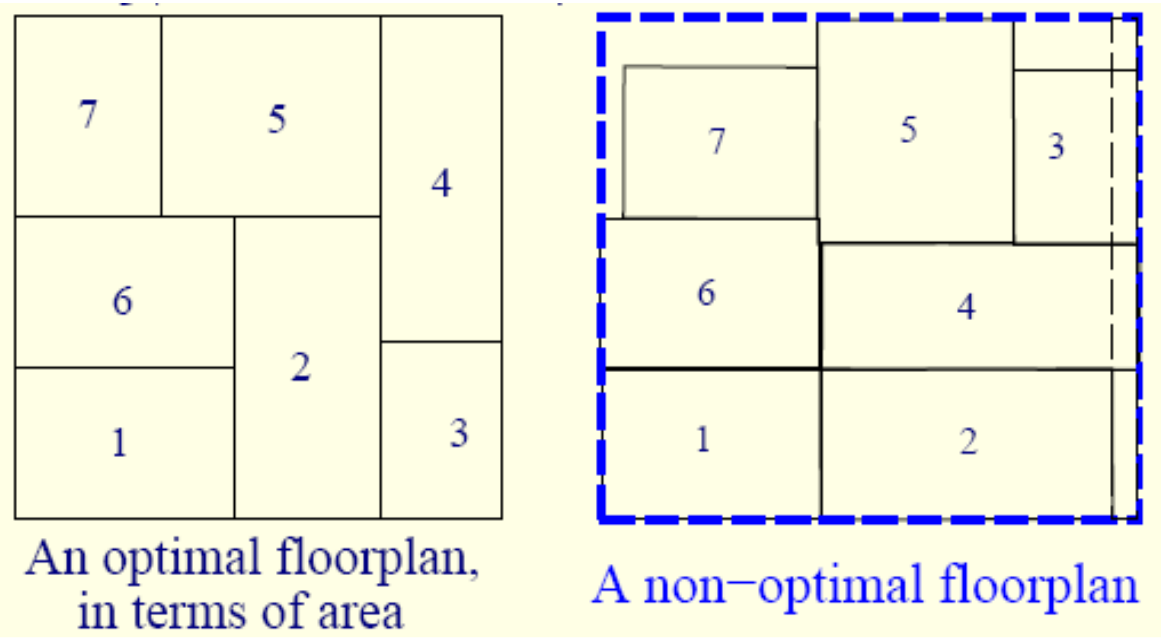
ASIC Design Process

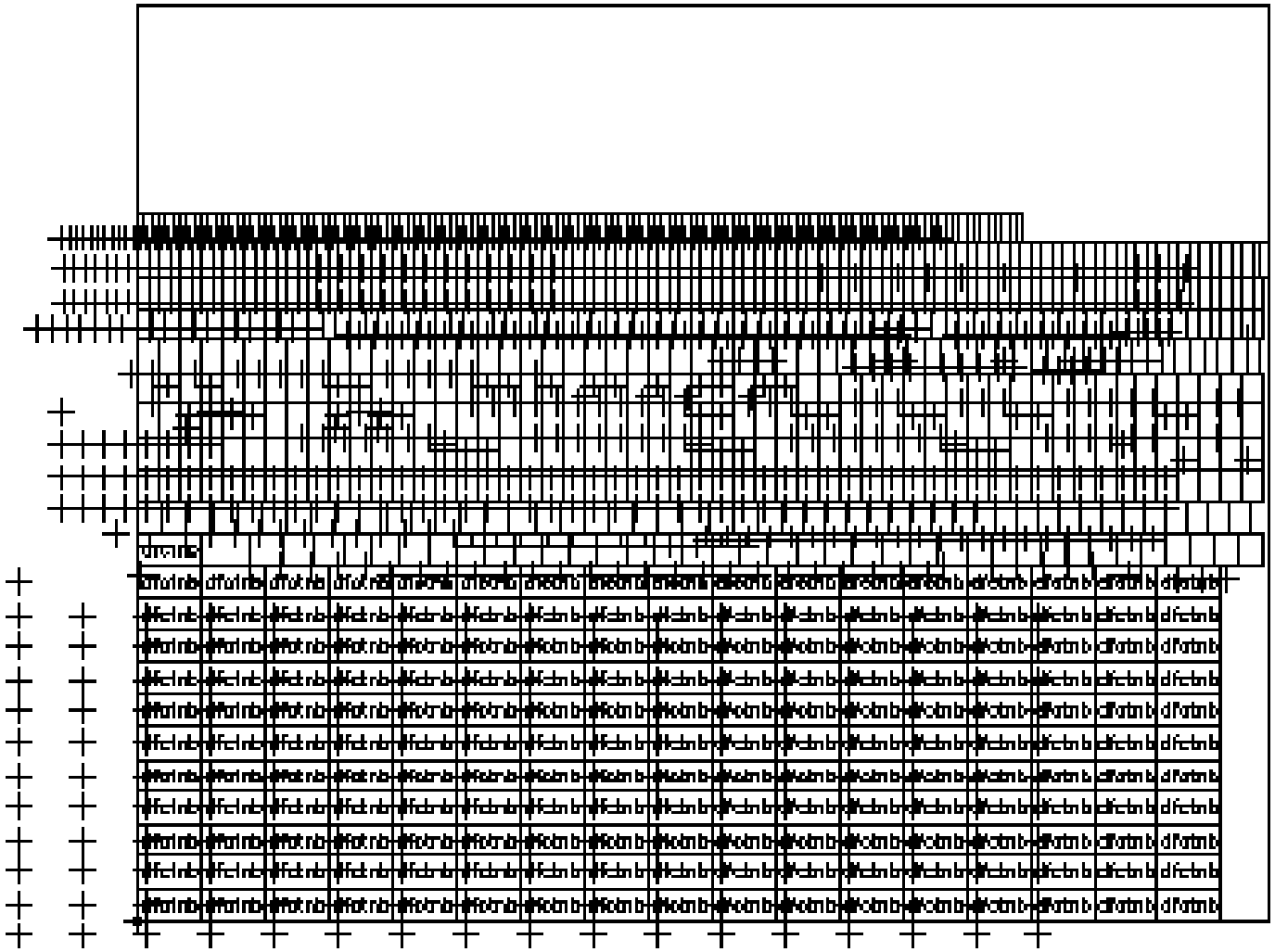
- **S-1 Design Entry:** Schematic entry or HDL description
- **S-2: Logic Synthesis:** Using Verilog HDL or VHDL and Synthesis tool, produce ***a netlist***-logic cells and their interconnect detail
- **S-3 System Partitioning:** Divide a large system into ASIC sized pieces
- **S-4 Pre-Layout Simulation:** Check design functionality
- **S-5 Floorplanning:** Arrange netlist blocks on the chip
- **S-6 Placement:** Fix cell locations in a block
- **S-7 Routing:** Make the cell and block interconnections
- **S-8 Extraction:** Measure the interconnect R/C cost
- **S-9 Post-Layout Simulation**



Introduction

- The input to the floorplanning step - **output of system partitioning and design entry—a netlist.**
- Netlist - describing **circuit blocks, the logic cells within the blocks, and their connections.**



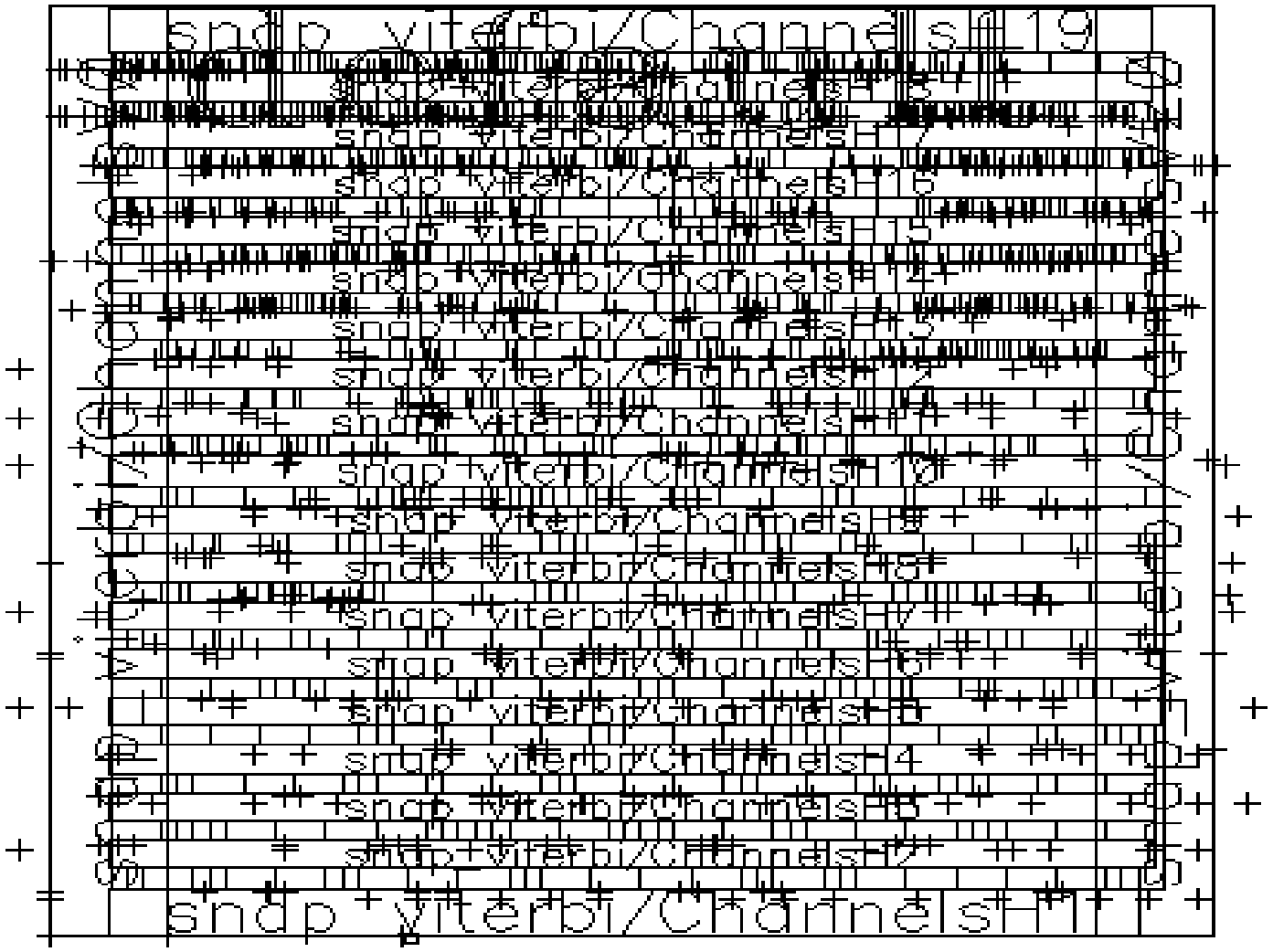


- The starting point of floorplaning and placement steps for the viterbi decoder

- **-collection of standard cells with no room set aside yet for routing.**

The starting point of floorplaning and placement steps for the viterbi decoder

- **Small boxes that look like bricks** - outlines of the standard cells.
- **Largest standard cells, at the bottom of the display** (labeled dfctnb) - 188 D flipflops.
- **'+' symbols** -drawing origins of the standard cells—for the D flip-flops they are shifted to the left and below the logic cell bottom left-hand corner.
- **Large box surrounding all the logic cells** - estimated chip size.
- (This is a screen shot from Cadence Cell Ensemble.)



The viterbi decoder after floorplanning and placement

The viterbi decoder after floorplanning and placement

- **8 rows of standard cells separated by 17 horizontal channels** (labeled 2–18).

- **Channels are routed as numbered**.

- In this example, the I/O pads are omitted to show the cell placement more clearly.

Floorplanning Goals and Objectives

- The input to a floorplanning tool is a hierarchical **netlist** that describes
  - the interconnection of the blocks (RAM, ROM, ALU, cache controller, and so on)
  - the logic cells (NAND, NOR, D flip-flop, and so on) within the blocks
  - the logic cell connectors (terminals , pins , or ports)

- The netlist is a **logical description** of the ASIC;
- The floorplan is a physical **description** of an ASIC.
- Floorplanning is a **mapping between the logical description (the netlist) and the physical description (the floorplan).**

The **Goals of Floorplanning** are to:

- 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 –

To minimize the chip area

To minimize delay.

Measuring area is straightforward, but measuring delay is more difficult

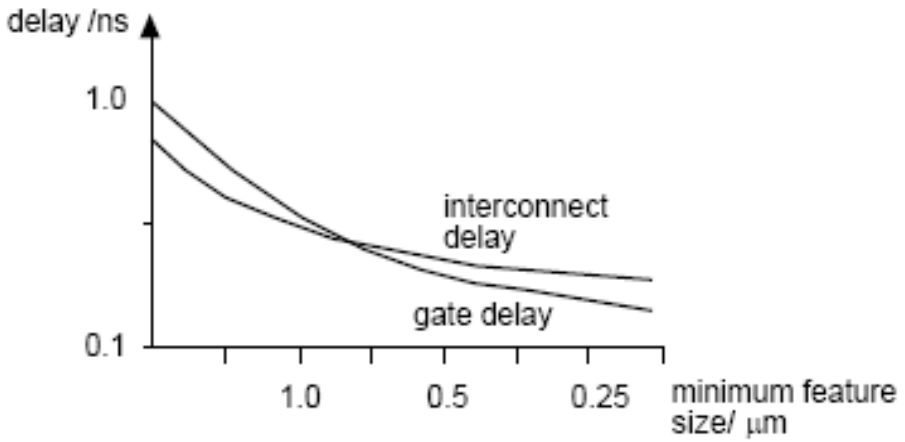
Measurement of Delay in Floor planning

Interconnect and gate delays.

As feature sizes decrease, both average interconnect delay and average gate delay decrease—but at different rates.

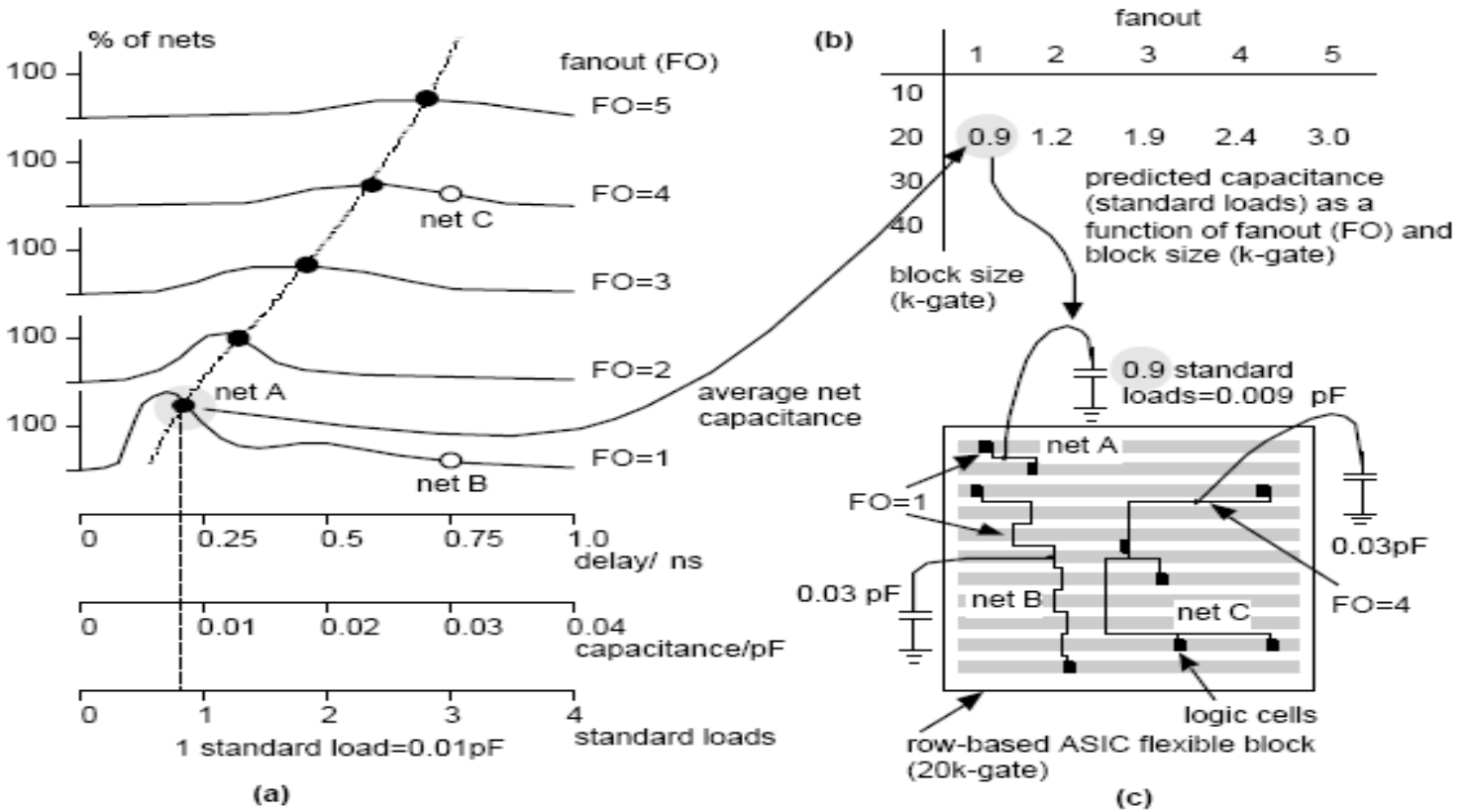
This is because interconnect capacitance tends to a limit that is independent of scaling.

Interconnect delay now dominates gate delay.



•Floor planning - To predict interconnect delay by estimating interconnect length.

Measurement of Delay in Floor planning



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).

Measurement of Delay in Floor planning (contd.,)

- A floorplanning tool can use **predicted-capacitance tables (also known as interconnect-load tables or wire-load tables)**.
- Typically between **60 and 70 percent of nets have a FO = 1.**
- **The distribution for a FO = 1 has a very long tail**, stretching to interconnects that run from corner to corner of the chip.
- **The distribution for a FO = 1 often has two peaks**, corresponding to a distribution for close neighbors in subgroups within a block, superimposed on a distribution corresponding to routing between subgroups.

Measurement of Delay in Floor planning (contd.,)

- We often see a **twin-peaked distribution** at the chip level also, corresponding to separate distributions for **interblock routing (inside blocks) and intrablock routing (between blocks)**.
- The distributions for **FO > 1 are more symmetrical and flatter** than for FO = 1.
- **The wire-load tables can only contain one number**, for example the **average net capacitance**, for any one distribution.
- Many tools take a worst-case approach and use the 80- or 90-percentile point instead of the average. Thus a tool may **use a predicted capacitance** for which we know **90 percent of the nets will have less than the estimated capacitance.**

Measurement of Delay in Floor planning (contd.,)

- **Repeat the statistical analysis for blocks with different sizes.**

  For example, a net with a FO = 1 in a 25 k-gate block will have a different (larger) average length than if the net were in a 5 k-gate block**.**

- **The statistics depend on the shape (aspect ratio) of the block** (usually the statistics are only calculated for **square blocks**).

- The statistics will also **depend on the type of netlist.**

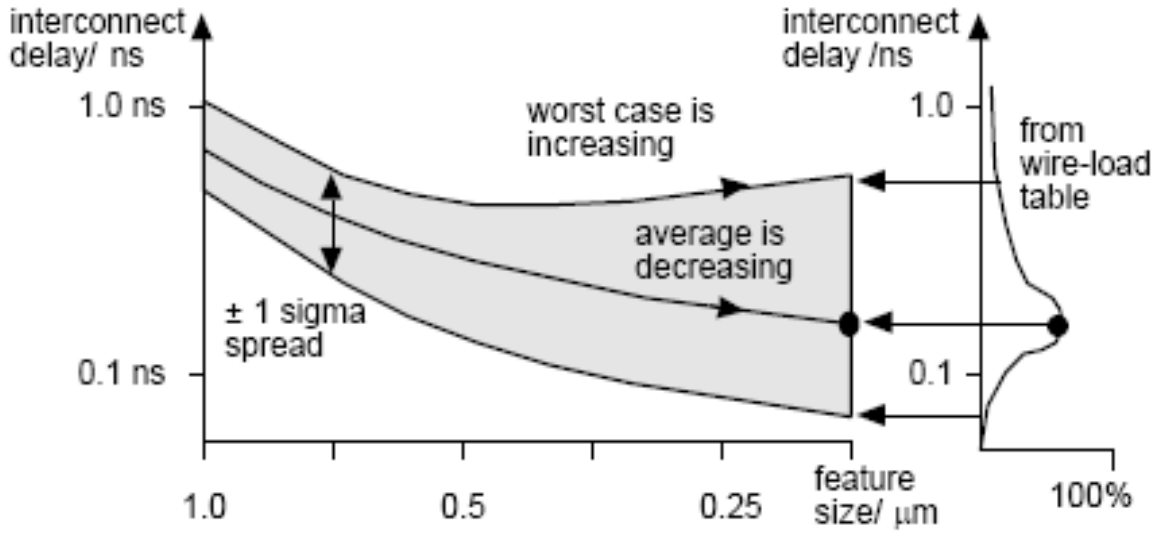
  For example, the distributions will be different for a netlist generated by setting a constraint for **minimum logic delay during synthesis**—which tends to generate large numbers of two-input NAND gates—than for netlists generated using **minimum-area constraints**.

A wire-load table showing average interconnect lengths (mm).

Array (available gates)	Chip size (mm)	Fanout 1	Fanout 2	Fanout 4
3k	3.45	0.56	0.85	1.46
11k	5.11	0.84	1.34	2.25
105k	12.50	1.75	2.70	4.92

Worst-case interconnect delay.

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



Floorplanning - Optimization

Optimize Performance

- Chip area.
- Total wire length.
- Critical path delay.
- Routability.
- Others, e.g. noise, heat dissipation.

Cost = $\alpha A + \beta L$,

Where

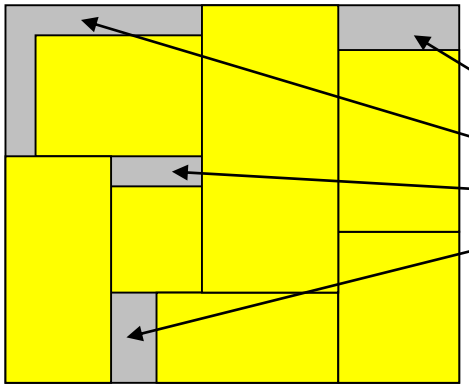
A = total area,

L = total wire length,

α and β constants.

Floorplanning

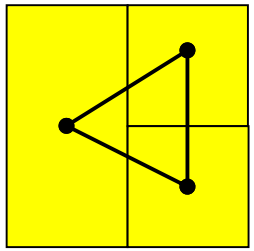
Area

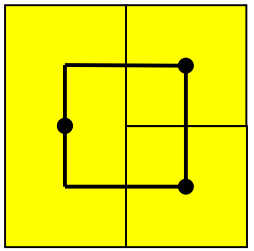


•Deadspace

•Minimizing area = Minimizing deadspace

- Wire length estimation
- Exact wire length not known until after routing.
- Pin position not known.
- How to estimate?
  - Center to center estimation.





Floorplanning Tools

- **Flexible blocks (or variable blocks)** :
  - Their total area is fixed,
  - Their shape (aspect ratio) and connector locations may be adjusted during the placement.

- **Fixed blocks:**
  - The dimensions and connector locations of the other fixed blocks (perhaps RAM, ROM, compiled cells, or megacells) can only be modified when they are created.

- **Seeding:**
  - Force logic cells to be in selected flexible blocks by **seeding** . We choose seed cells by name.
  - Seeding may be **hard or soft**.
- **Hard seed** - fixed and not allowed to move during the remaining floor planning and placement steps.
- **Soft seed** - an initial suggestion only and can be altered if necessary by the floor planner.

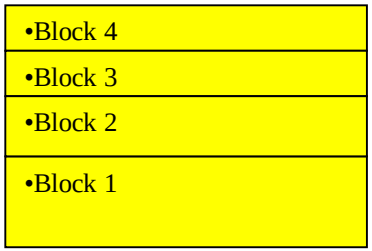
- **Seed connectors** within flexible blocks—forcing certain nets to appear in a specified order, or location at the boundary of a flexible block.

- **Rat's nest**:-display the connection between the blocks

- Connections are shown as **bundles** between the centers of blocks or as **flight lines** between connectors.

Aspect Ratio Bounds

No Bounds

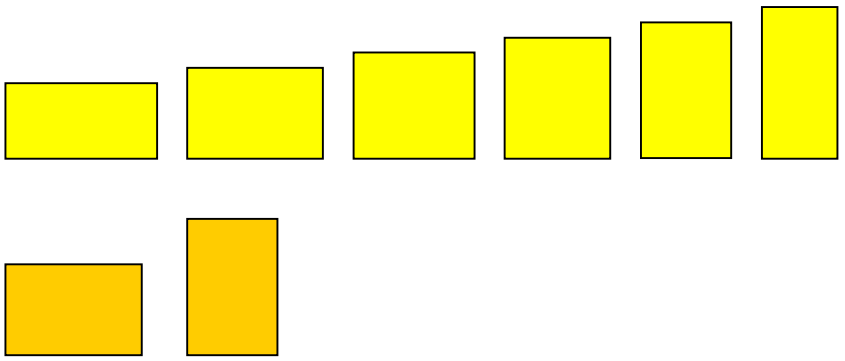


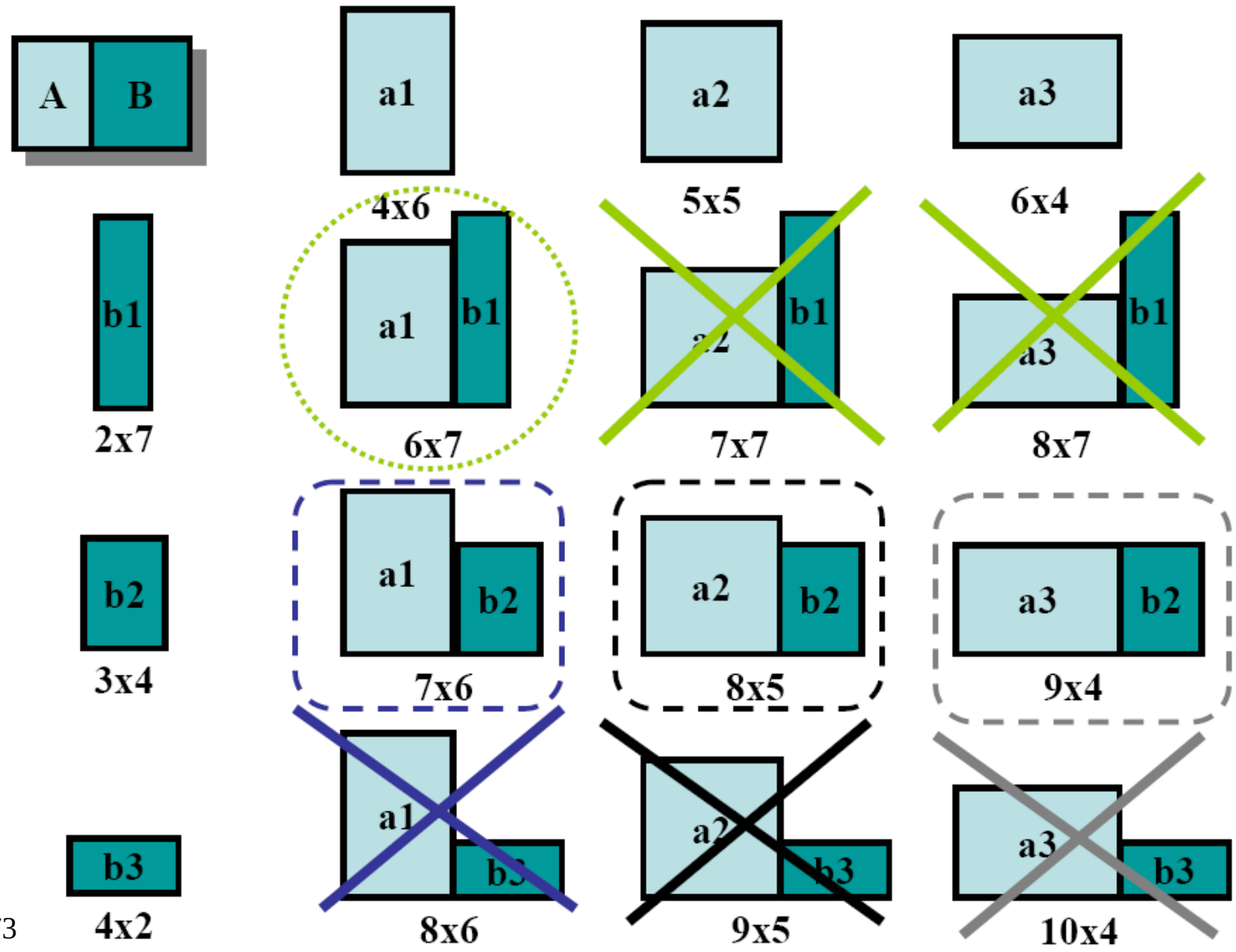
•NOT GOOD!!

With Bounds

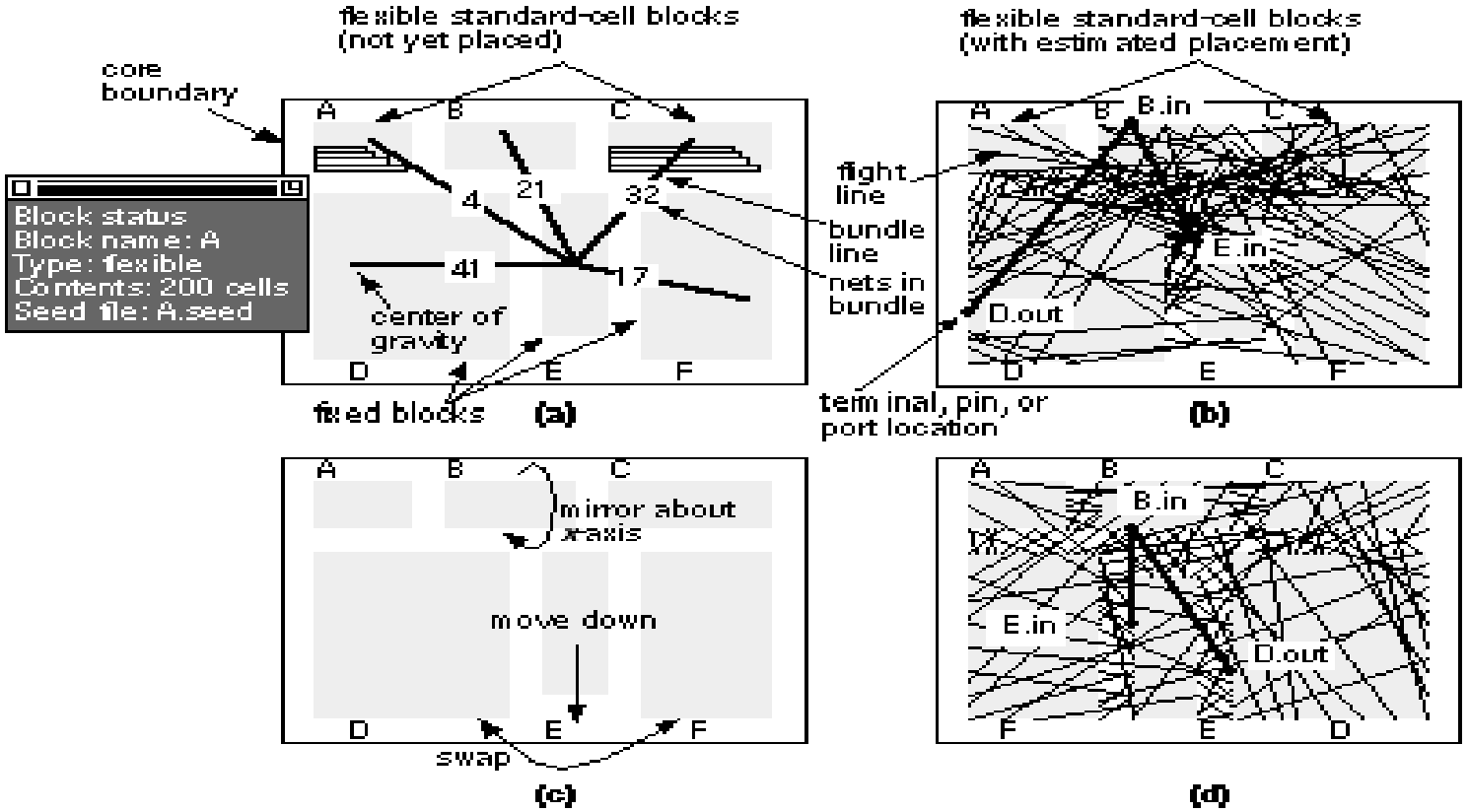
lower bound ≤ height/width ≤ upper bound

- Soft Blocks
  - Flexible shape
  - I/O positions not yet determined
- Hard Blocks
  - Fixed shape
  - Fixed I/O pin positions




A
B
a1
a2
a3
4x6
5x5
6x4
b1
a1
b1
a2
b1
a3
b1
2x7
6x7
7x7
8x7
b2
a1
b2
a2
b2
a3
b2
3x4
7x6
8x5
9x4
b3
a1
b3
a2
b3
a3
b3
4x2
8x6
9x5
10x4

Floorplanning Tools



Floorplanning a cell-based ASIC.

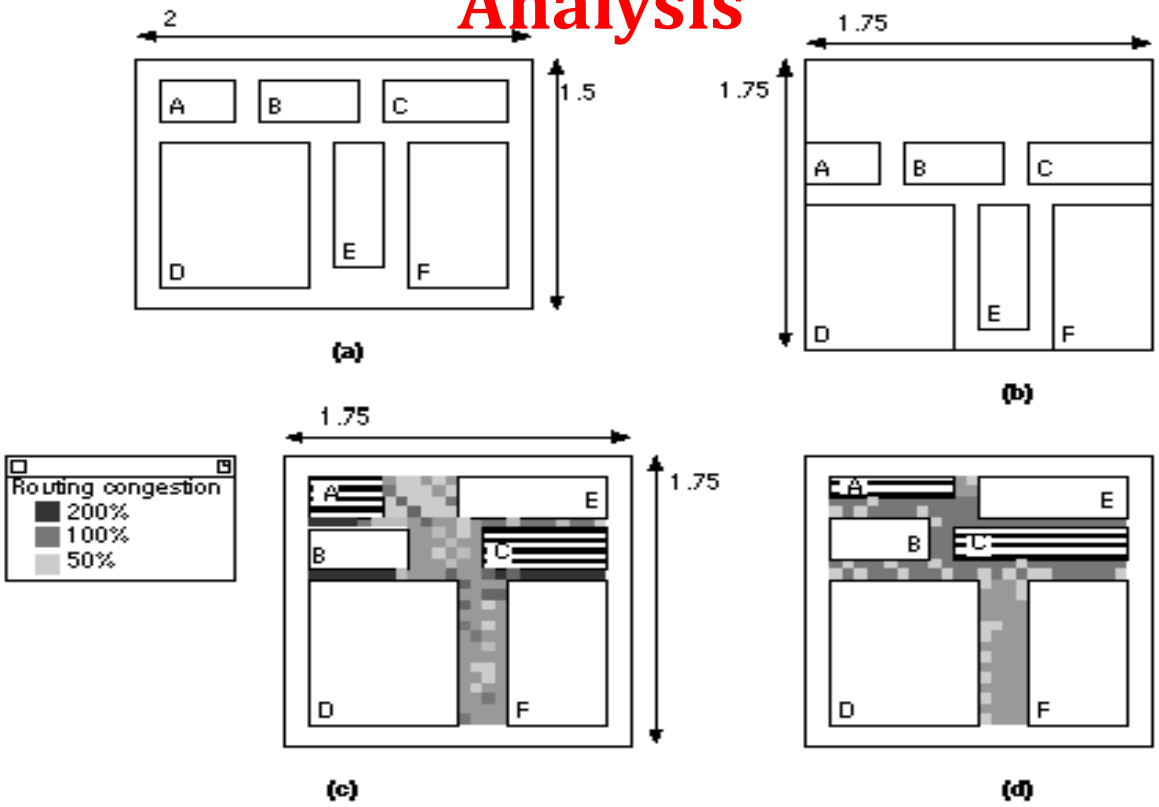
(a) **Initial floorplan generated by the floorplanning tool.** Two of the blocks are flexible (A and C) and contain rows of standard cells (unplaced). A pop-up window shows the status of block A.

(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.

•Aspect ratio and Congestion Analysis



(a) The initial floorplan with a 2:1.5 die aspect ratio.

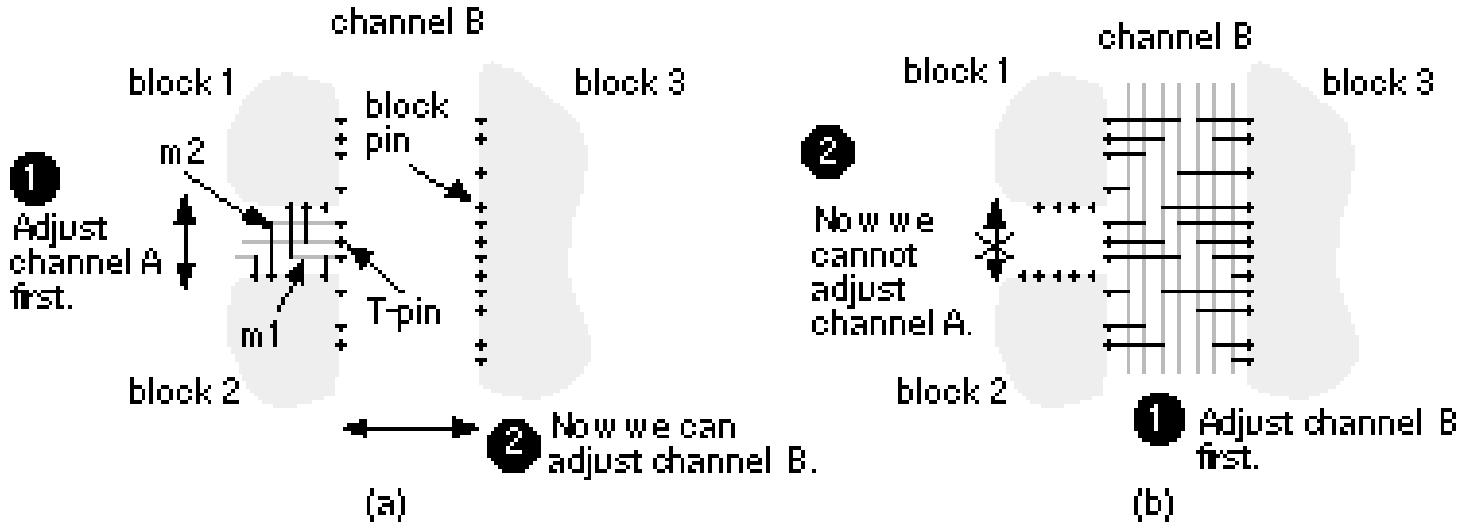
(b) Altering the floorplan to give a 1:1 chip aspect ratio.

Congestion analysis-One measure of congestion is the difference between the number of interconnects that we actually need, called the channel density , and the channel capacity

(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.

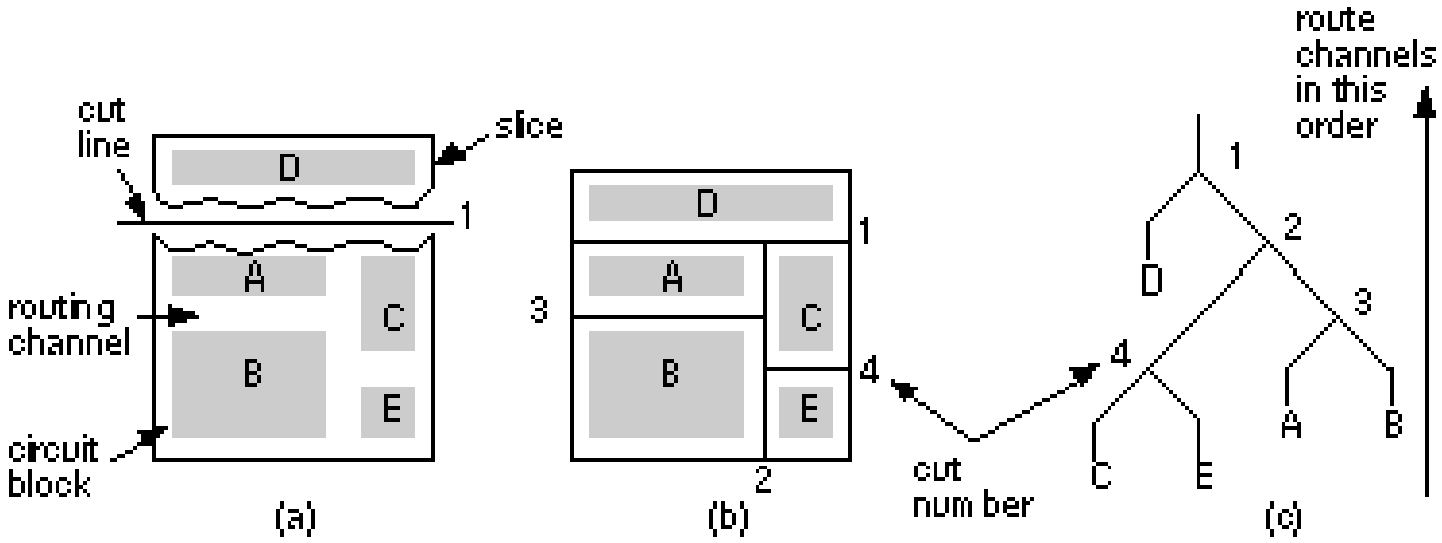
(d) **Resizing flexible blocks A and C alleviates congestion.**

Channel Definition



- **Channel definition or channel allocation**
- **During the floorplanning step, assign the areas between blocks that are to be used for interconnect.**
- **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.
- **Route the stem of a T-junction before route the top**.

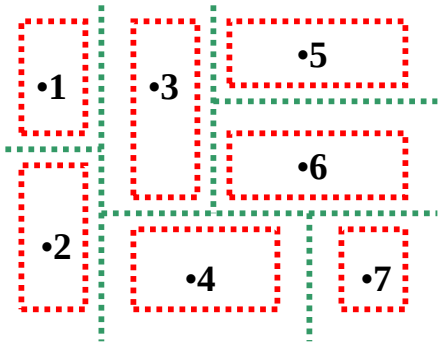
Channel Routing

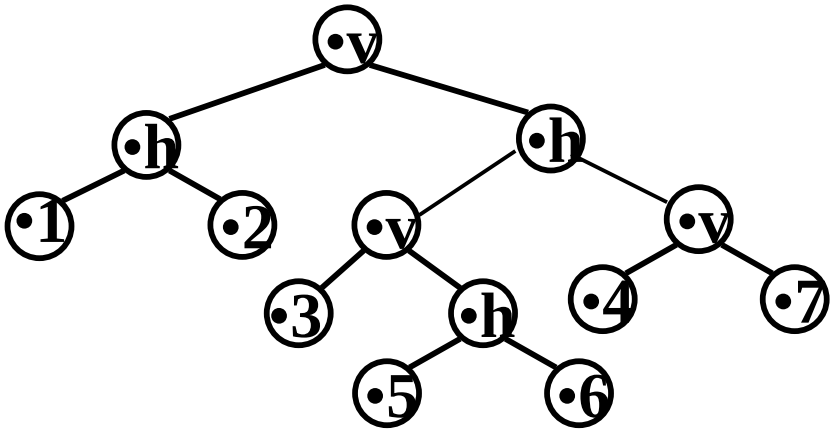


- **Defining the channel routing order for a slicing floorplan using a slicing tree.**
- (a) Make a cut all the way across the chip between circuit blocks. Continue slicing until each piece contains just one circuit block. Each cut divides a piece into two without cutting through a circuit block.
- (b) 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.

Slicing Floorplan and General Floorplan

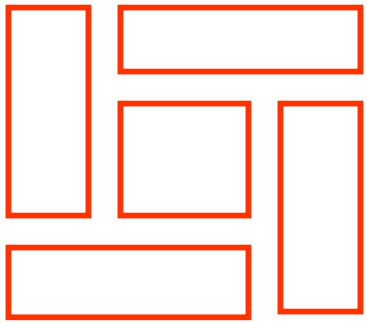
- **Slicing floorplan**





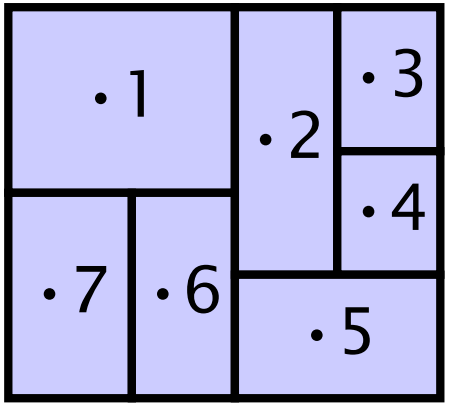
- **Slicing Tree**

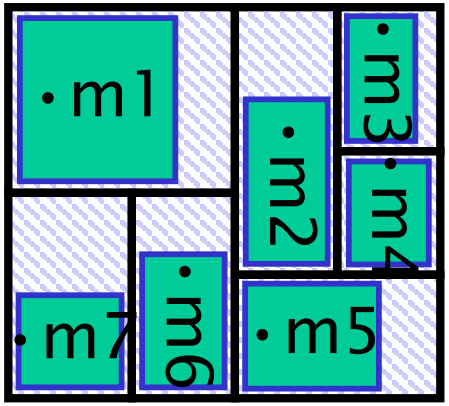
- **non-slicing floorplan**

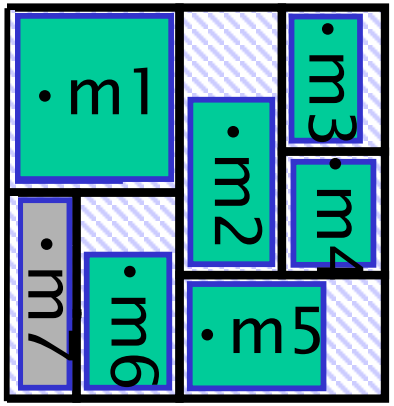


Area Utilization

- Area utilization
  - Depends on how nicely the rigid modules' shapes are matched
  - Soft modules can take different shapes to "fill in" empty slots
  - Floorplan sizing



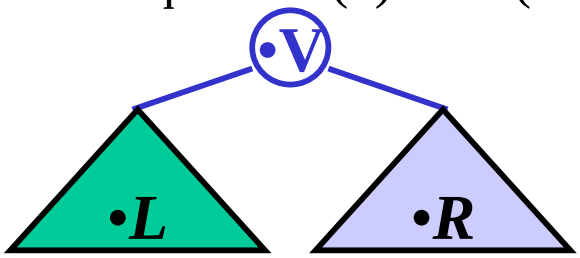


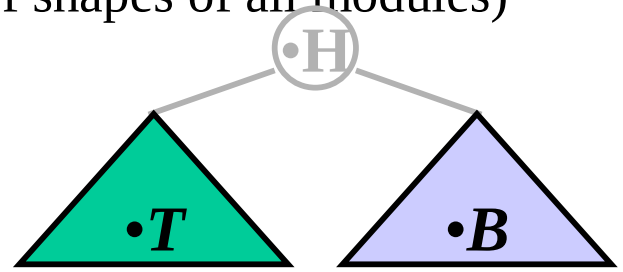


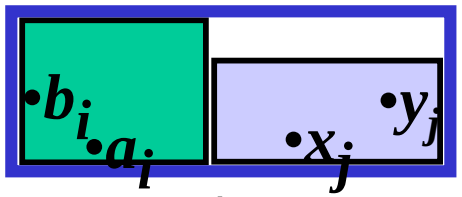
Area = 20x22 = 440 Area = 20x19 = 38

Slicing Floorplan Sizing

- Bottom-up process
  - Has to be done per floorplan perturbation
  - Requires O(n) time (N is the # of shapes of all modules)

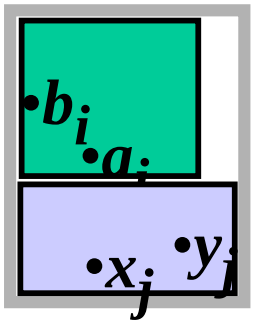






•$max(b_i, y_j)$

•$a_i + x_j$

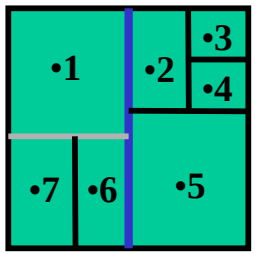


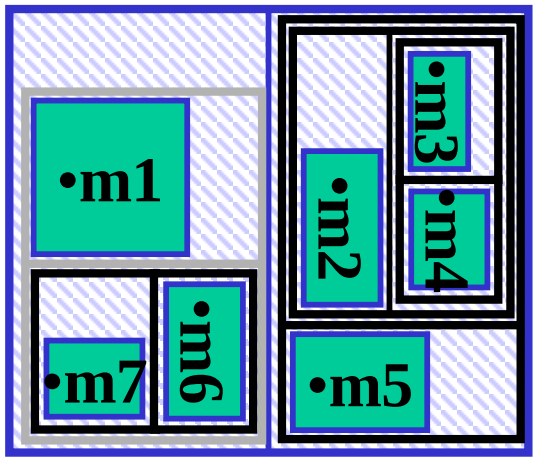
•$b_i + y_j$

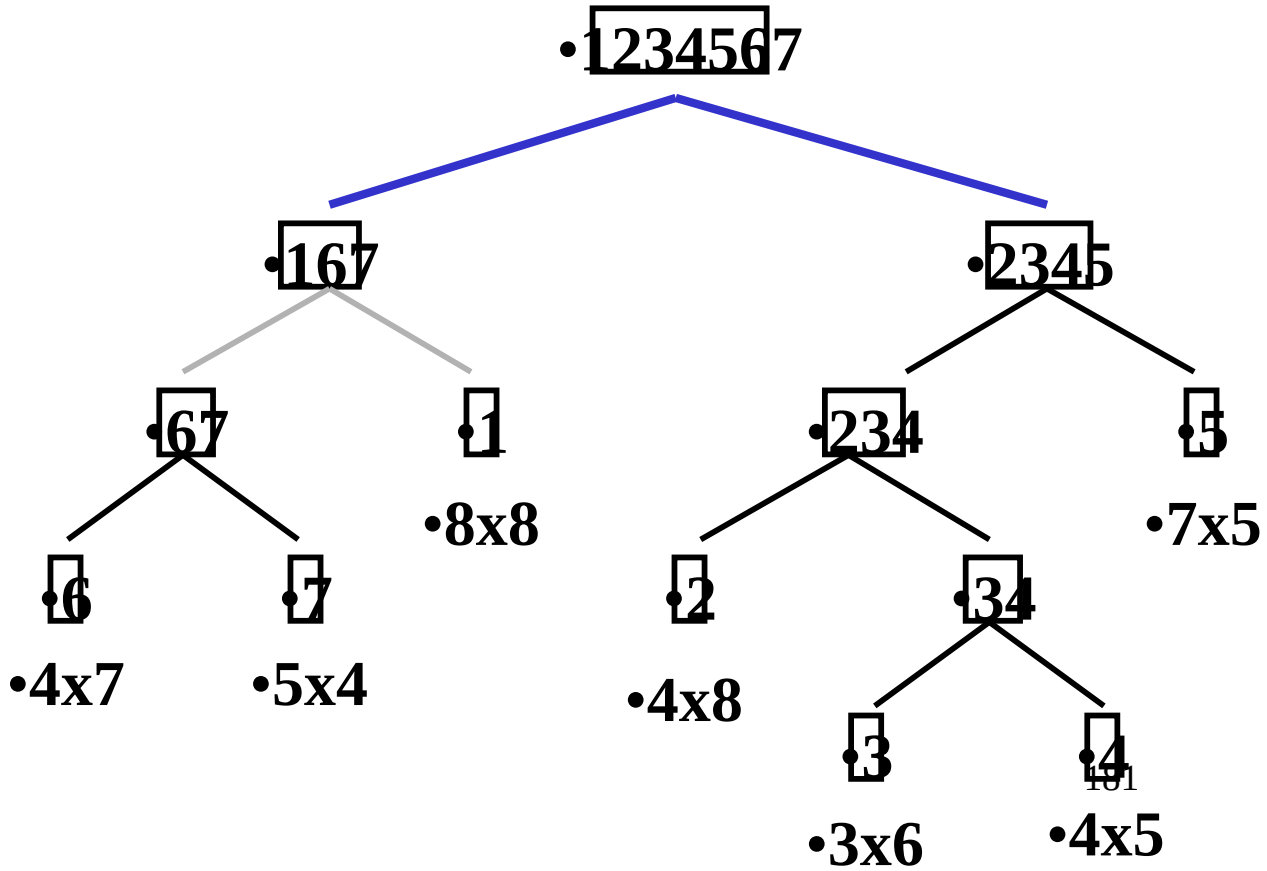
•$max(a_i, x_j)$

Slicing Floorplan Sizing

- Simple case: all modules are hard macros
  - No rotation allowed, one shape only



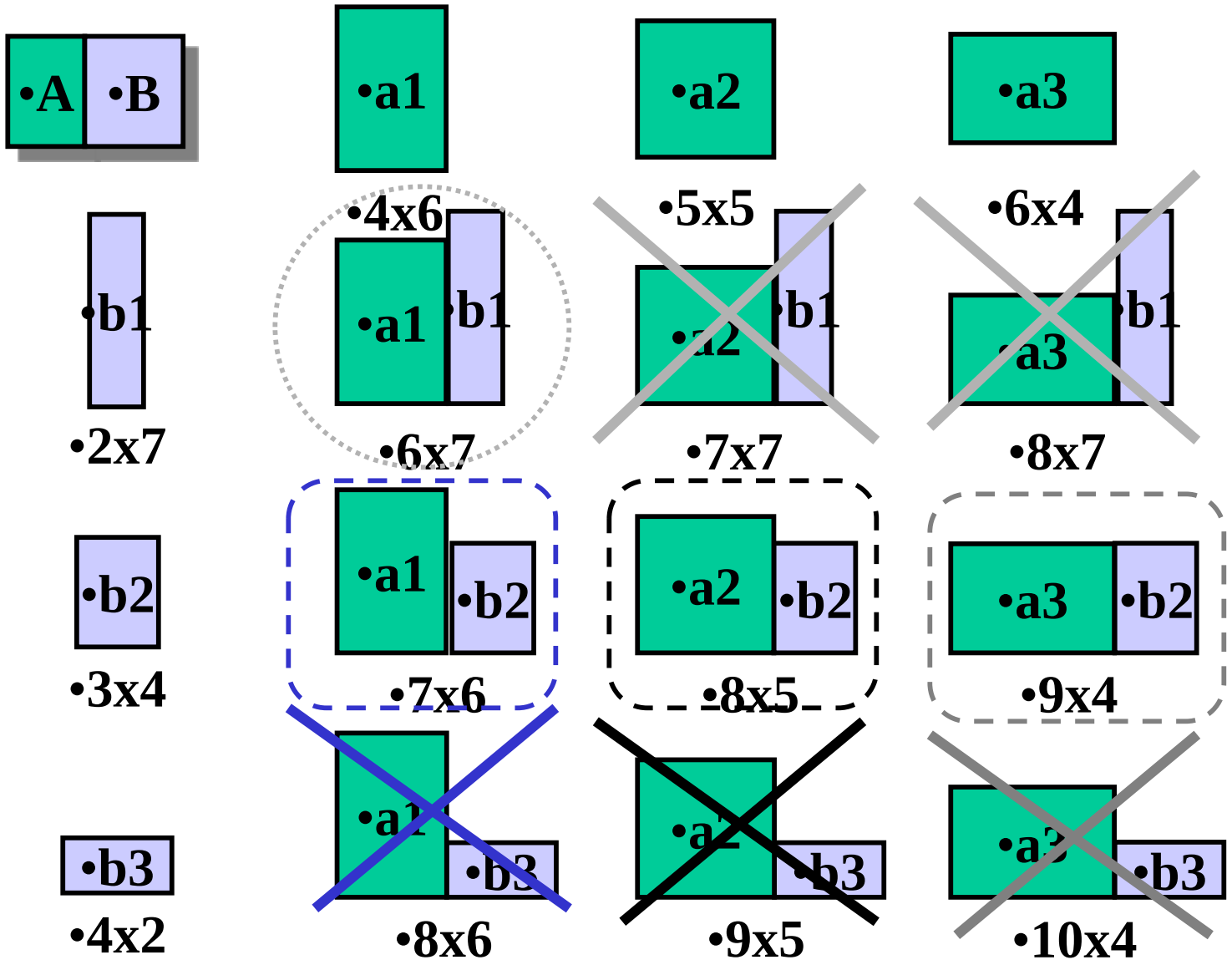




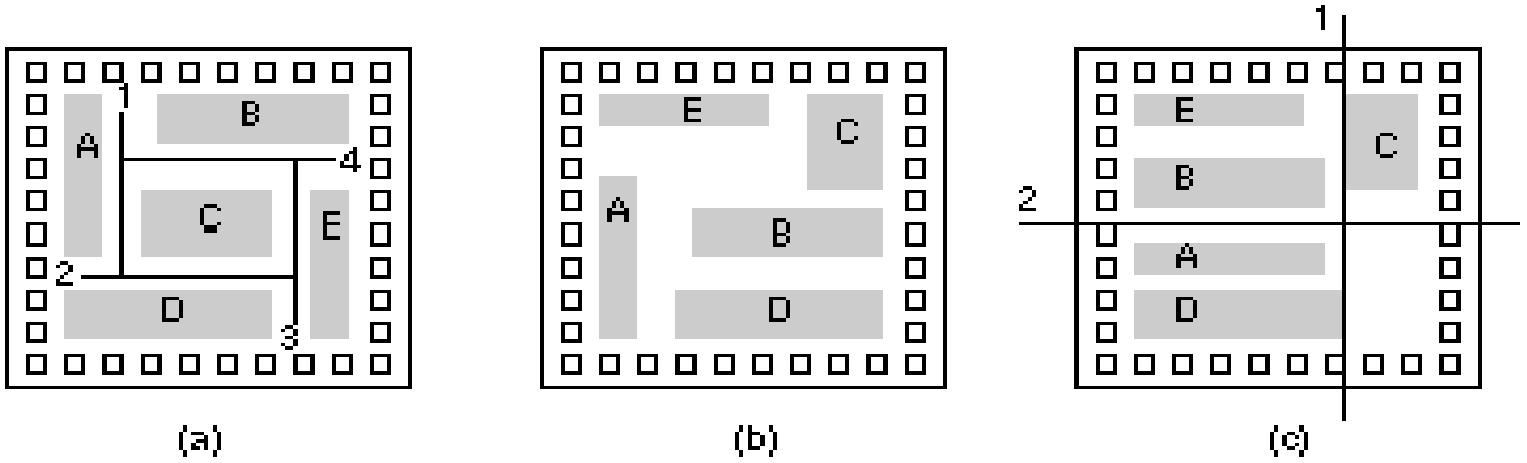
•Slicing Floorplan Sizing

- **General case: all modules are soft macros**
  - **Stockmeyer's work (1983) for optimal module orientation**
  - **Non-slicing = NP complete**
  - **Slicing = polynomial time solvable with dynamic programming**
- **Phase 1: bottom-up**
  - **Input: floorplan tree, modules shapes**
  - **Start with sorted shapes lists of modules**
  - **Perform Vertical_Node_Sizing & Horizontal_Node_Sizing**
  - **When get to the root node, we have a list of shapes. Select the one that is best in terms of area**
- **Phase 2: top-down**
  - **Traverse the floorplan tree and set module locations**

Sizing Example



Cyclic Constraints

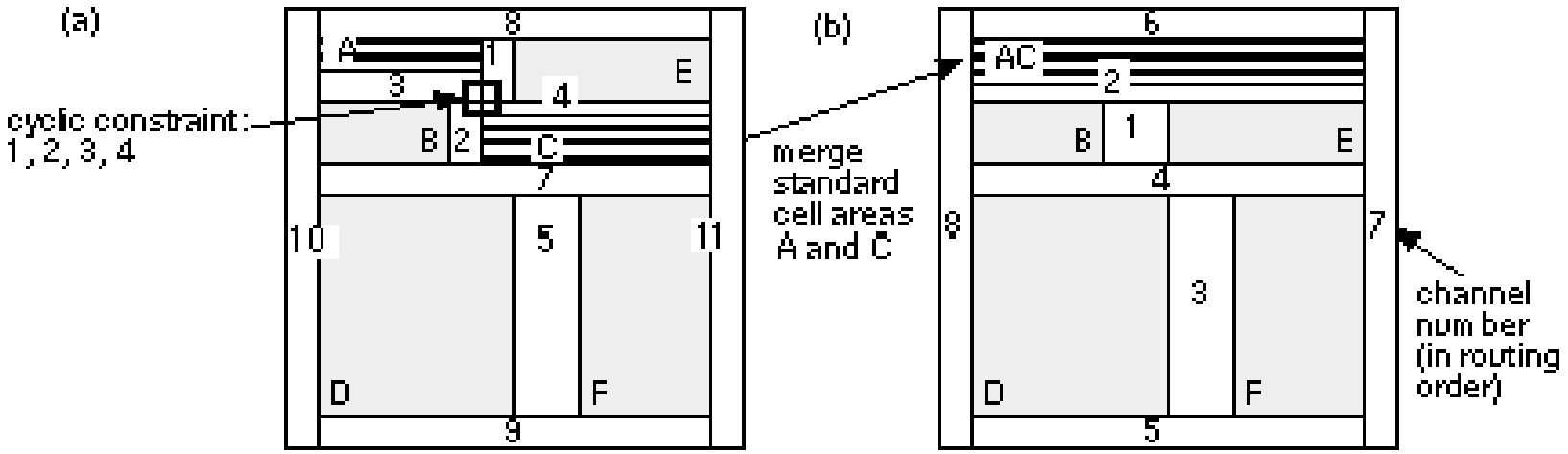


•Cyclic constraints.

•(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.

Cyclic Constraints



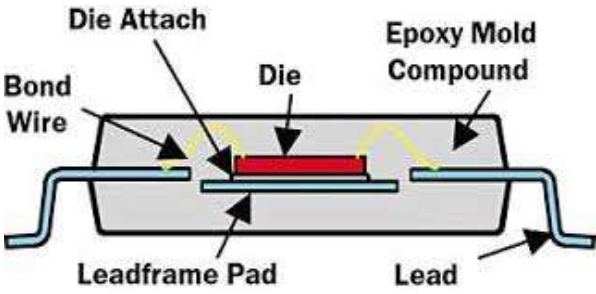
•

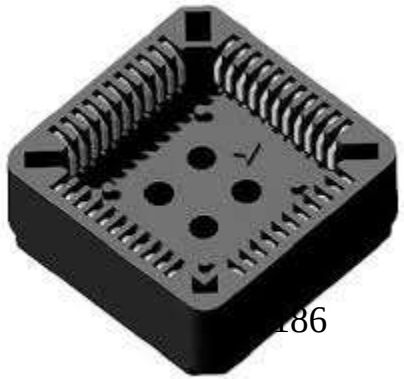
•(a) We can eliminate the cyclic constraint by merging the blocks A and C.

•(b) A slicing structure.

I/O and Power Planning (contd.,)

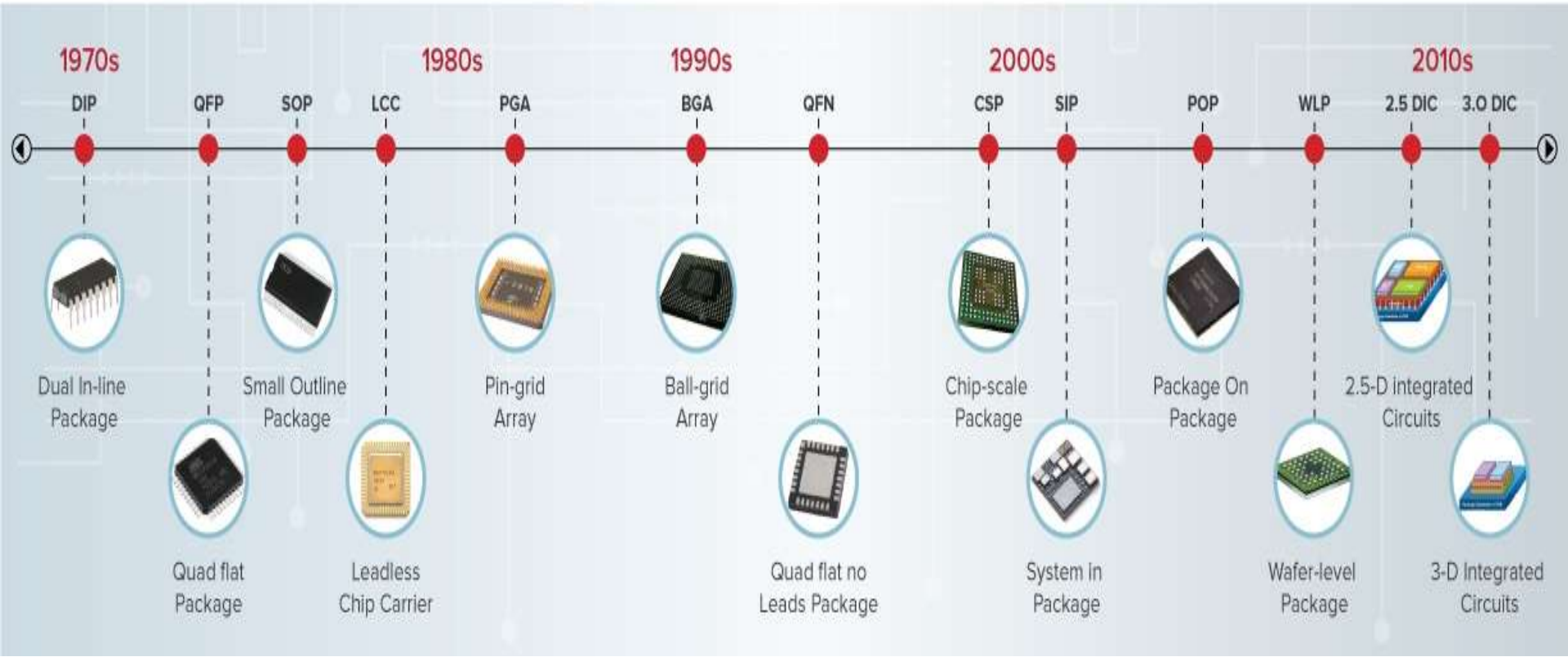
- Every chip communicates with the outside world.
- Signals flow onto and off the chip and we need to supply power.
- We need to consider the I/O and power constraints early in the floorplanning process.
- A silicon chip or die (plural die, dies, or dice) is mounted on a chip carrier inside a chip package . Connections are made by bonding the chip pads to fingers on a metal lead frame that is part of the package.
- The metal lead-frame fingers connect to the package pins . A die consists of a logic core inside a pad ring .
- On a **pad-limited die** we use tall, thin pad-limited pads , which maximize the number of pads we can fit around the outside of the chip.
- On a **core-limited die** we use short, wide core-limited pads .



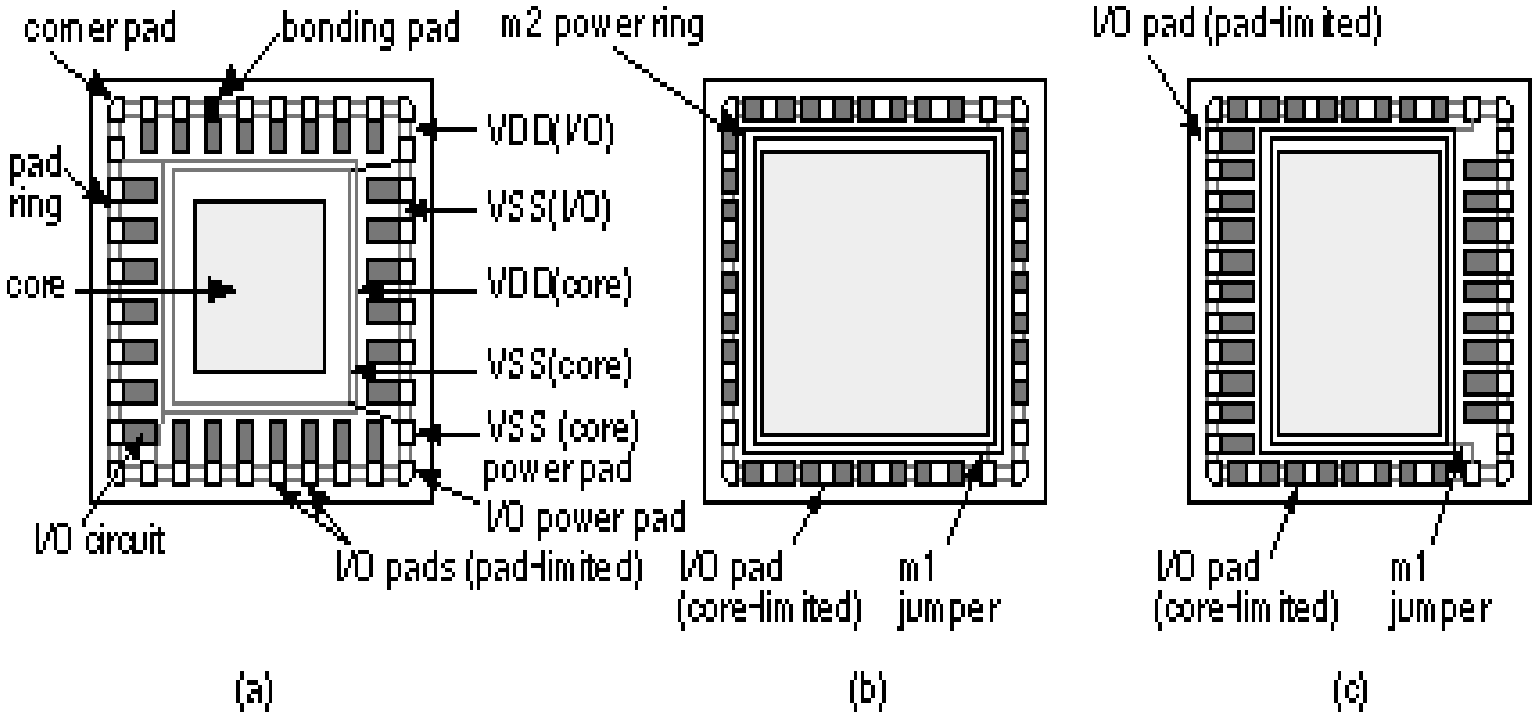


SEMICONDUCTOR PACKAGING HISTORY

anysilicon



I/O and Power Planning



•FIGURE 16.12 Pad-limited and core-limited die. (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.

I/O and Power Planning (contd.,)

- Special **power pads** are used for:1. positive supply, or VDD, power buses (or power rails) and
  2. ground or negative supply, VSS or GND.
  - one set of VDD/VSS pads supplies power to the **I/O pads** only.
  - Another set of VDD/VSS pads connects to a second power ring that supplies the **logic core.**
- I/O power as **dirty power**
  - It has to **supply large transient currents to the output transistors**.
  - Keep dirty power separate to avoid **injecting noise into the internal-logic power** (the **clean power**).
- I/O pads also contain **special circuits to protect against electrostatic discharge (ESD)**.
  - These circuits can withstand very short high-voltage (several kilovolt) pulses that can be generated during human or machine handling.

I/O and Power Planning (contd.,)

- If we make an electrical connection between the substrate and a chip pad, or to a package pin, it must be to VDD (n -type substrate) or VSS (p -type substrate). This **substrate connection** (for the whole chip) employs a **down bond** (or drop bond) to the carrier. We have several options:
  - *We can* ***dedicate one (or more) chip pad(s) to down bond to the chip carrier.***
  - *We can make a connection from a chip pad to the lead frame and* ***down bond from the chip pad to the chip carrier.***
  - *We can make a connection from a chip pad to the lead frame and down bond from the lead frame.*
  - *We can down bond from the lead frame without using a chip pad.*
  - *We can leave the substrate and/or chip carrier unconnected.*
- Depending on the package design, the type and positioning of down bonds may be fixed. This means we need to fix the position of the chip pad for down bonding using a **pad seed**

I/O and Power Planning (contd.,)

- A **double bond** connects two pads to one chip-carrier finger and one package pin. We can do this to save package pins or reduce the series inductance of bond wires (typically a few nanohenries) by parallel connection of the pads.

- **To reduce the series resistive and inductive impedance** of power supply networks, it is normal to use **multiple VDD and VSS pads**.

- This is particularly important with the **simultaneously switching outputs (SSOs)** that occur **when driving buses**

  - The **output pads can easily consume most of the power** on a CMOS ASIC, because the load on a pad (usually tens of picofarads) is much larger than typical on-chip capacitive loads.

  - Depending on the technology it may be necessary **to provide dedicated VDD and VSS pads for every few SSOs**. Design rules set how many SSOs can be used per VDD/VSS pad pair. These dedicated VDD/VSS pads must "follow" groups of output pads as they are seeded or planned on the floorplan.

I/O and Power Planning (contd.,)

- Using a **pad mapping,** we translate the **logical pad** in a netlist to a **physical pad** from a pad library. We might control pad seeding and mapping in the floorplanner.

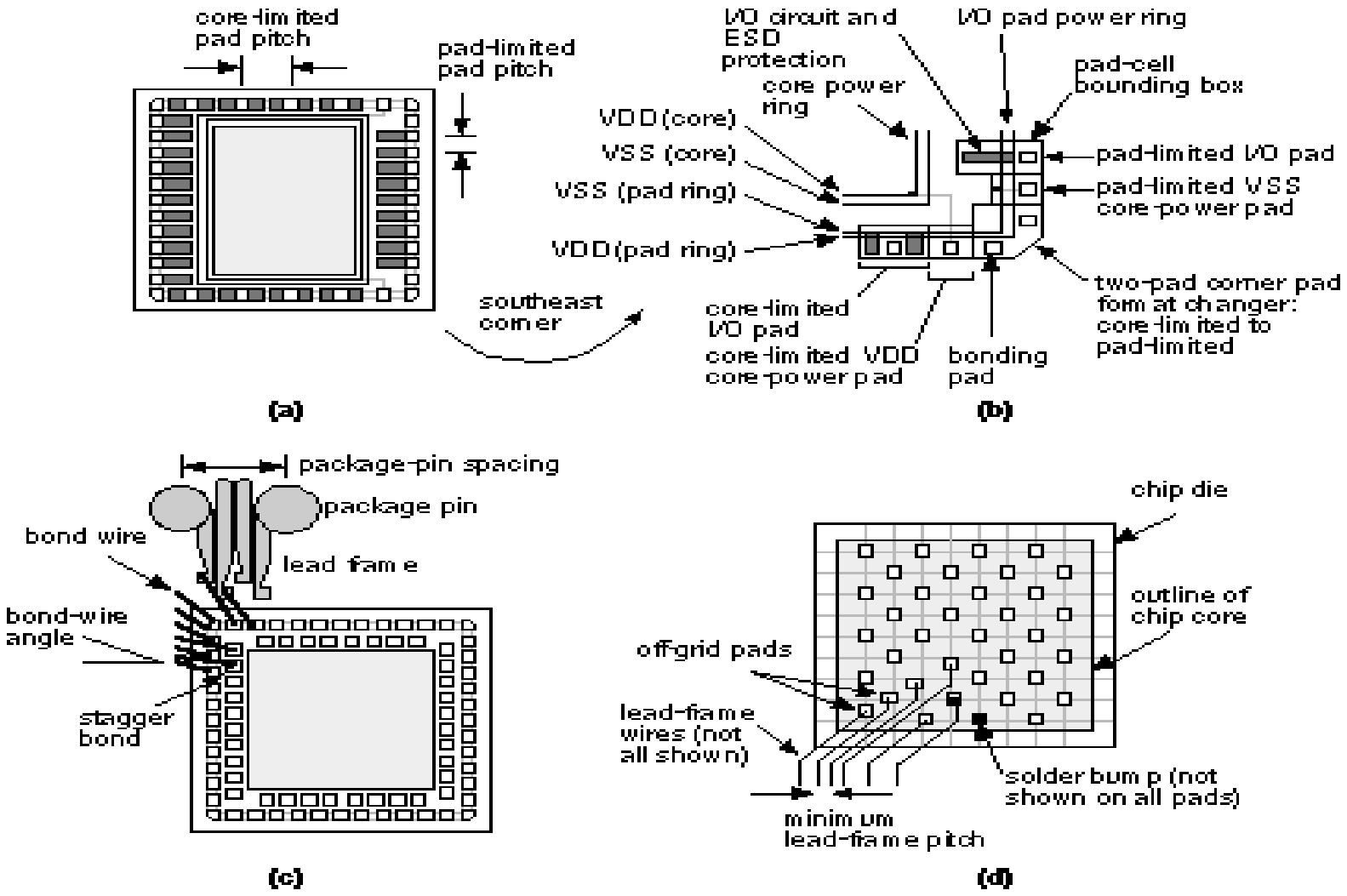
- There are several nonobvious factors that must be considered when generating a pad ring:

    - **Design library pad cells for one orientation**.
        - For example, an **edge pad for the south side of the chip**, and a **corner pad for the southeast corner**.
        - Generate other orientations by rotation and flipping (mirroring).
        - Some ASIC vendors will not allow rotation or mirroring of logic cells in the mask file. To avoid these problems we may need to have separate horizontal, vertical, left-handed, and right-handed pad cells in the library with appropriate logical to physical pad mappings.

    - **Mixing of pad-limited and core-limited edge pads in the same pad ring complicates the design of corner pads.**
        - In this case a corner pad also becomes a pad-format changer, or hybrid corner pad .

    - In single-supply chips we have one VDD net and one VSS net, both global power nets . It is also possible to **use mixed power supplies** (for example, 3.3 V and 5 V) or multiple power supplies (digital VDD, analog VDD).

I/O and Power Planning (contd.,)



•FIGURE 16.13 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

I/O and Power Planning (contd.,)

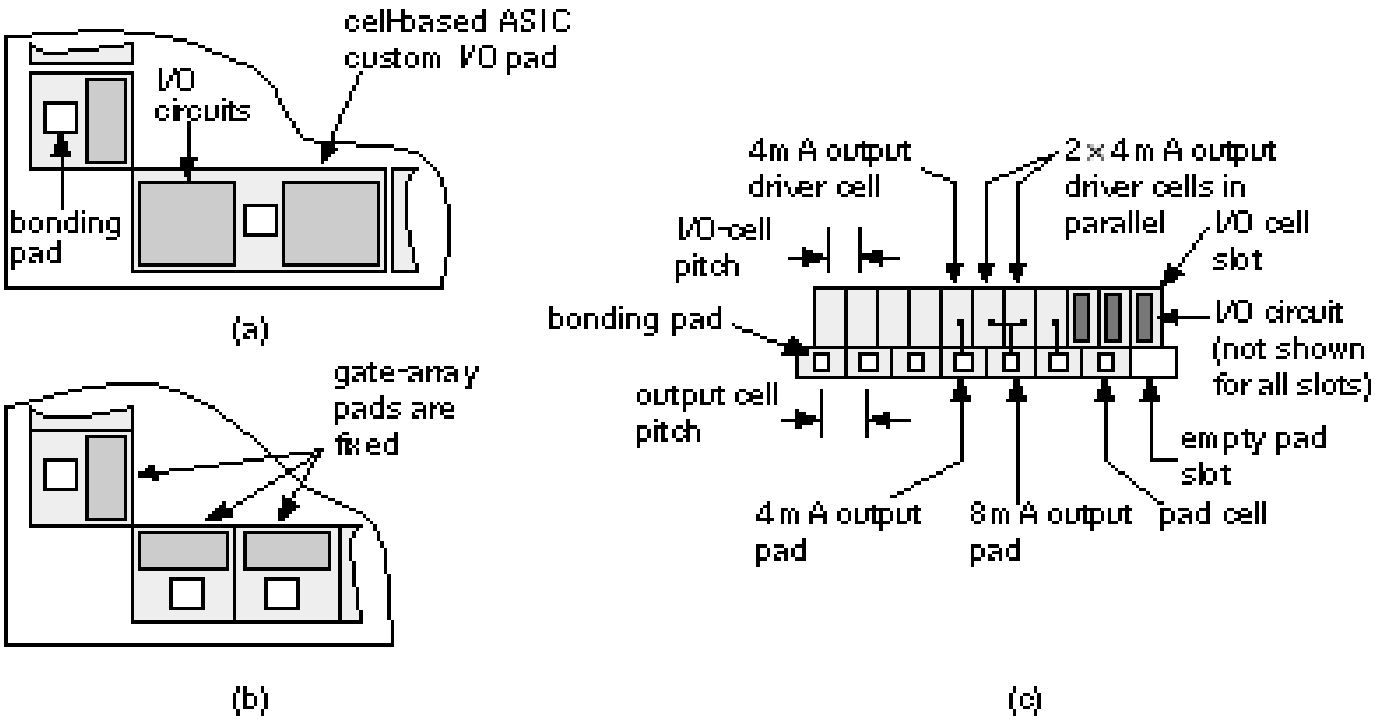
- **stagger-bond** arrangement using t**wo rows of I/O pads.**
  - In this case the design rules for bond wires (the spacing and the angle at which the bond wires leave the pads) become very important.

- **Area-bump** bonding arrangement (also known as flip-chip, solder-bump) used, for example, with **ball-grid array (BGA)** packages**.**

  - Even though the bonding pads are located in the center of the chip, **the I/O circuits are still often located at the edges of the chip** because of difficulties in power supply distribution and integrating I/O circuits together with logic in the center of the die.

- In an **MGA,** the pad spacing and I/O-cell spacing is fixed—each pad occupies a fixed pad slot (or pad site). This means that the properties of the pad I/O are also fixed but, if we need to, we can parallel adjacent output cells to increase the drive. To increase flexibility further the I/O cells can use a separation, the I/O-cell pitch , that is smaller than the pad pitch **.**

I/O and Power Planning (contd.,)



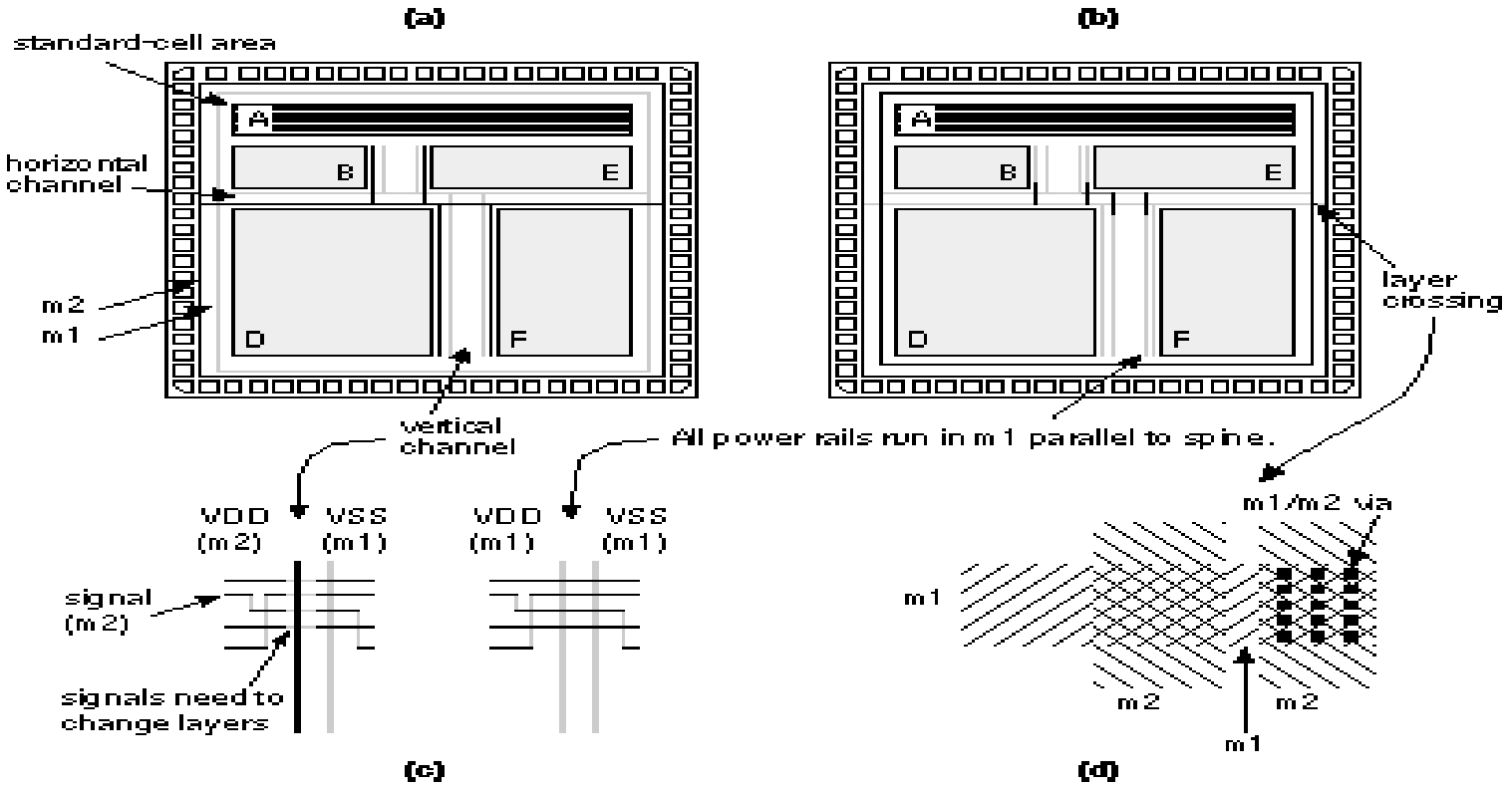
•FIGURE 16.14 Gate-array I/O pads. (a) Cell-based ASICs may contain pad cells of different sizes and widths. (b) A corner of a gate-array base. (c) A gate-array base with different I/O cell and pad pitches

I/O and Power Planning (contd.,)

- The long direction of a rectangular channel is the **channel spine .**

- Some automatic routers may require that metal lines parallel to a channel spine use a **preferred layer** (either m1, m2, or m3). Alternatively we say that a particular metal layer runs in a preferred direction .

I/O and Power Planning (contd.,)



•FIGURE 16.15 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.

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.

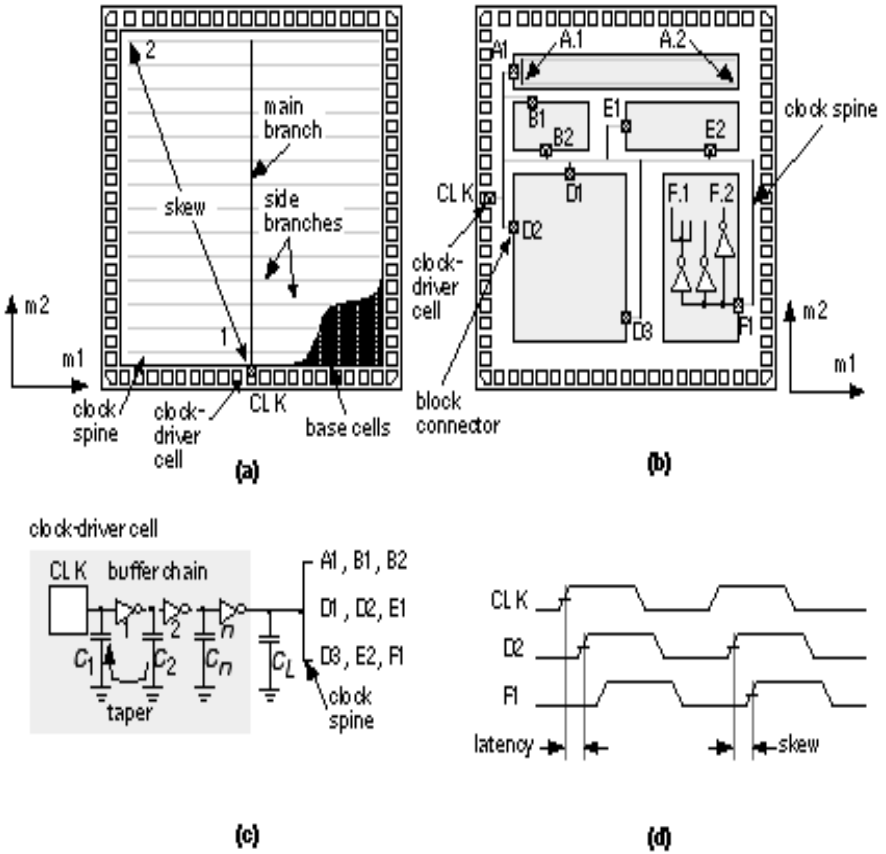
- (b) In this floorplan m1 is run parallel to the longest side of all channels, the channel spine.
  - This can make automatic routing easier
  - but may increase the number of vias and layer crossings.

- (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.

Clock Planning

- **clock spine** routing scheme with all clock pins driven directly from the clock driver. **MGAs and FPGAs** often use this **fish bone type of clock distribution** scheme



•FIGURE 16.16 Clock distribution.

•(a) A clock spine for a gate array.

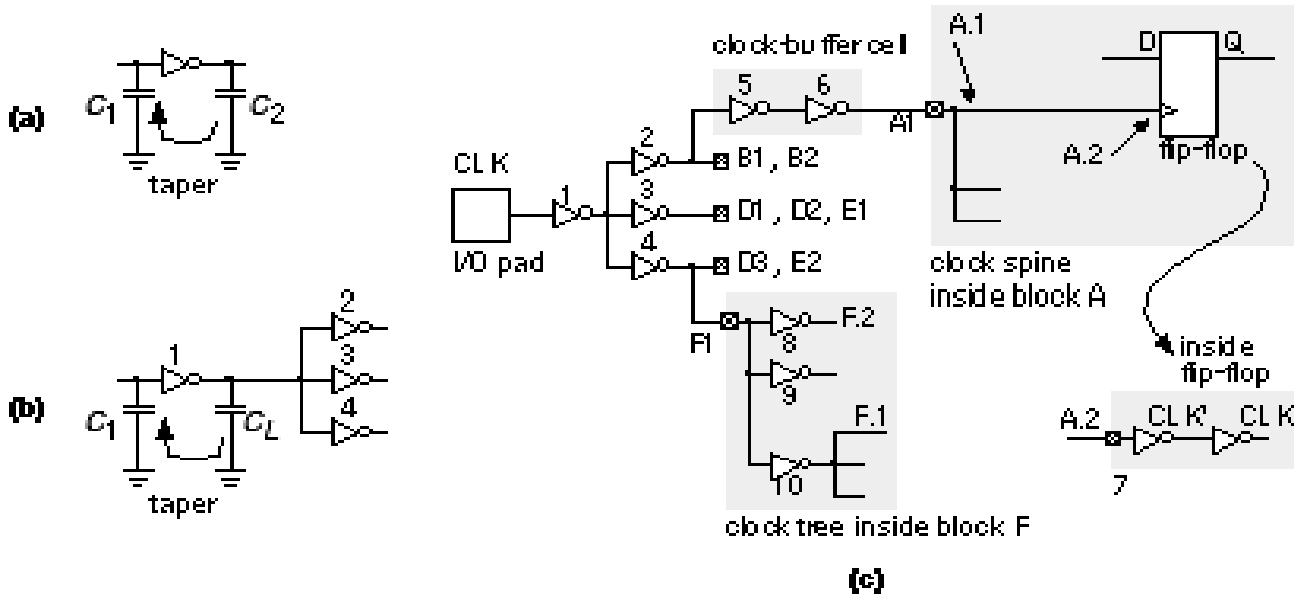
•(b) A clock spine for a cell-based ASIC (typical chips have thousands of clock nets).

•(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.

Clock Planning (cont.,)

- FIGURE 16.17 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 the cell-based ASIC of Figure 16.16 b. We have to balance the clock arrival times at all of the leaf nodes to minimize clock skew.



•Module III

•Placement

•Dr.(Mrs).D.Gracia Nirmala Rani
•Assistant Professor
•ECE Department
•Thiagarajar College of Engineering
•Madurai-15
•Email : gracia@tce.edu

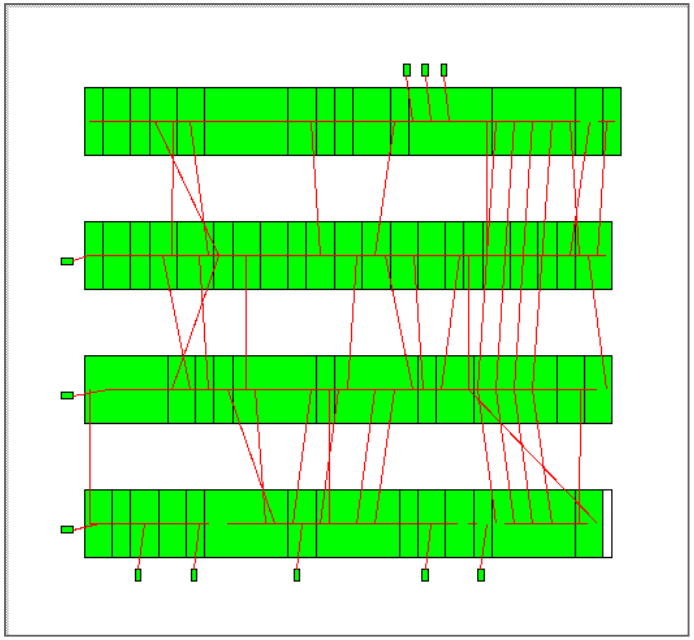
Content

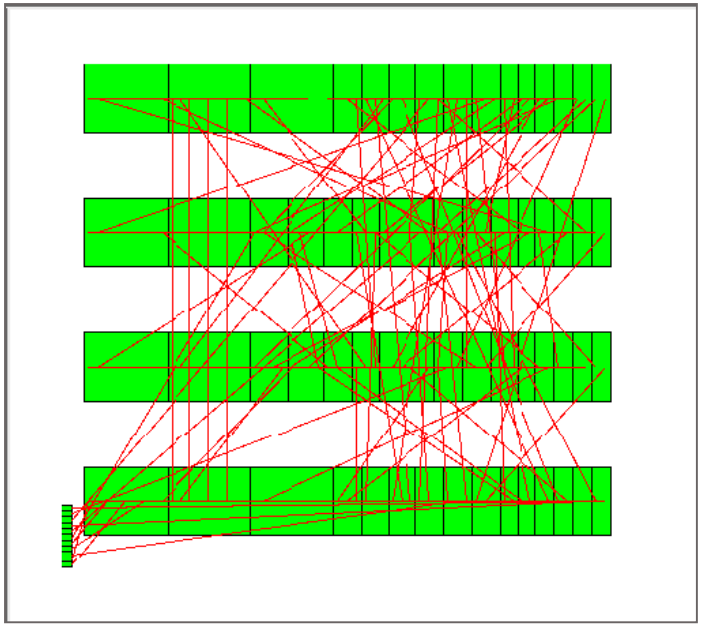
- Placement Definitions
- Placement Goals and Objectives
- Measurement of placement Goals and Objectives
- Placement Algorithms
- Simple placement Example
- Physical Design Flow

Placement

- The process of arranging circuit components on a layout surface undercertain constraints.
- Inputs : Set of fixed modules, netlist
- Output : Best position for each module based on various cost functions
- Cost functions include wirelength, wire routability, hotspots, performance, I/O pads.
- Placement is much more suited to automation than floorplanning.
- After we complete floorplanning and placement, we can predict both intrablock and interblock capacitances

Good placement vs Bad placement*



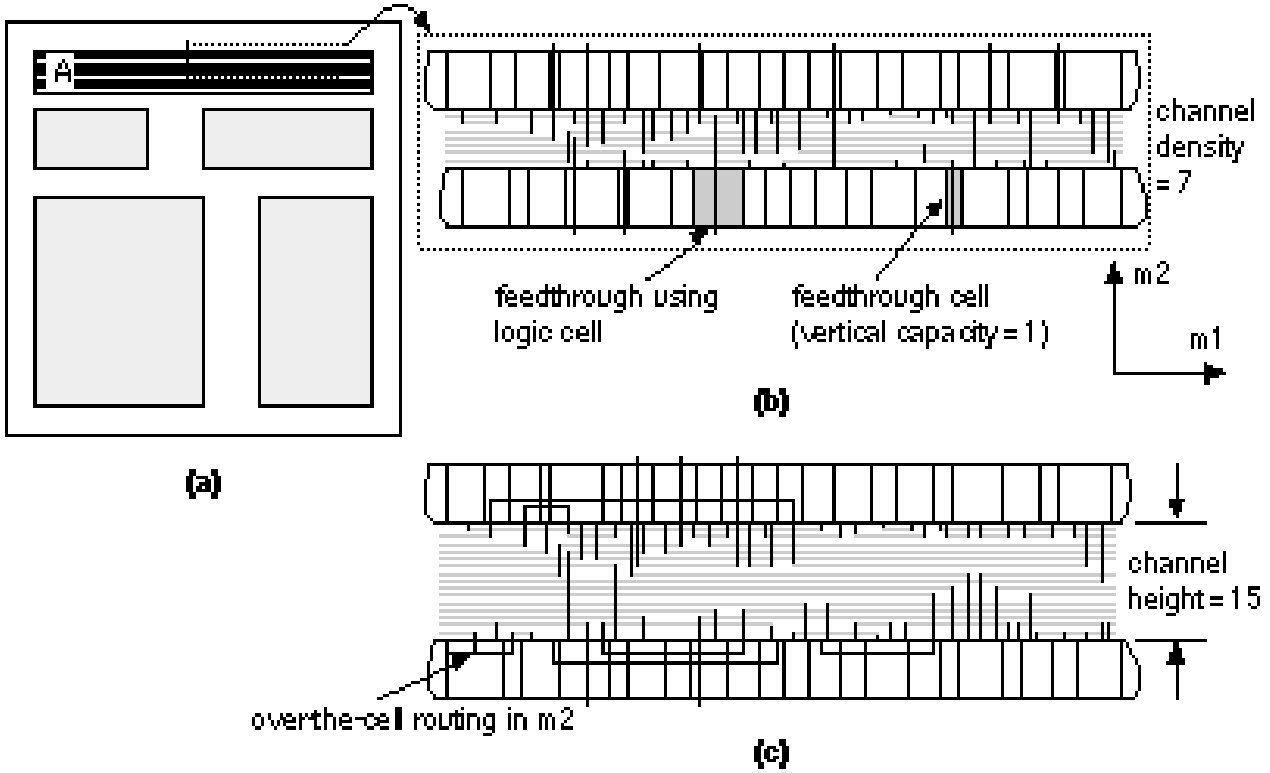


- Good placement
  - No congestion
  - Shorter wires
  - Less metal levels
  - Smaller delay
  - Lower power dissipation

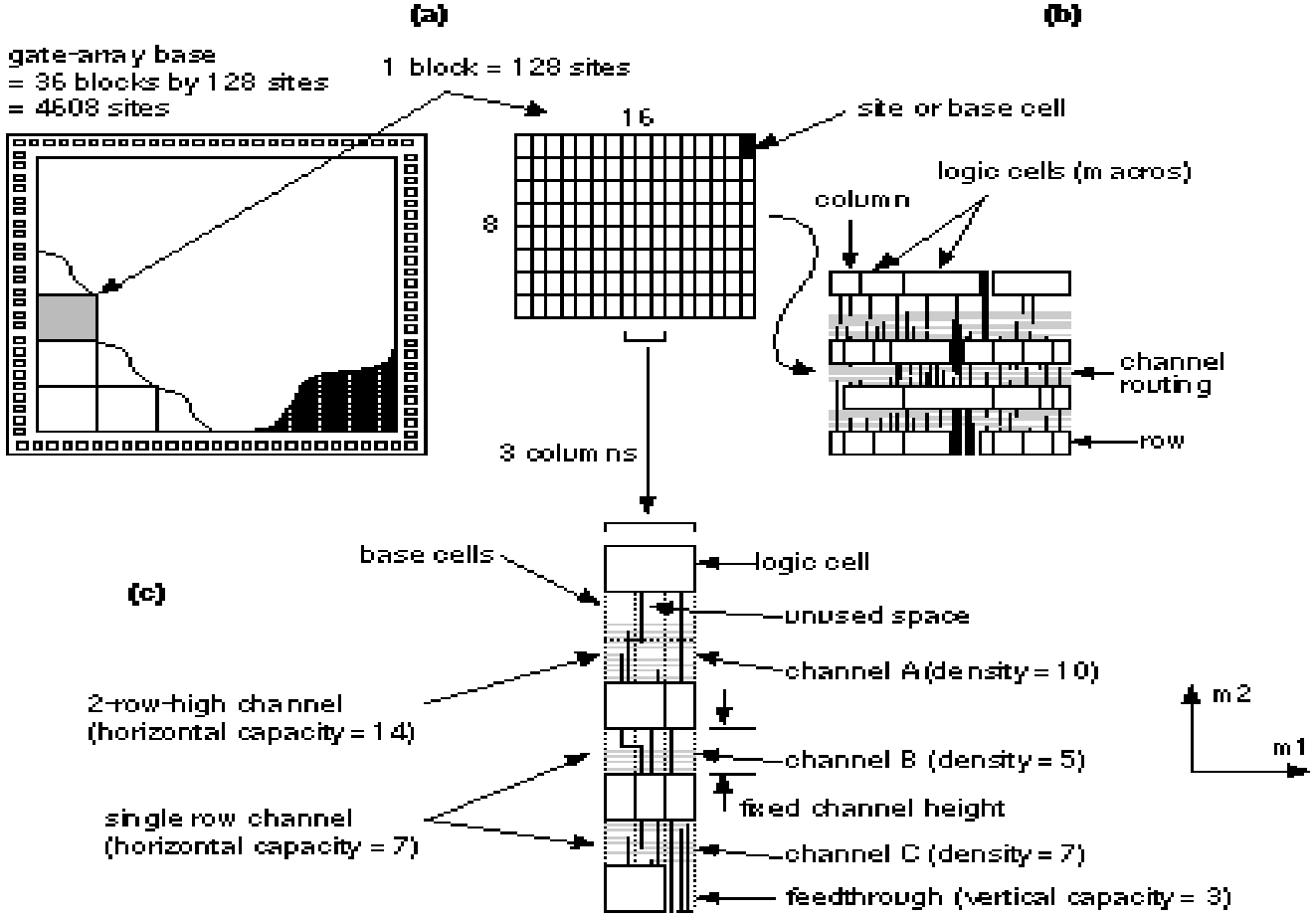
- Bad placement
  - Congestion
  - Longer wire lengths
  - More metal levels
  - Longer delay
  - Higher power dissipation

Placement Terms and Definitions

- CBIC, MGA, and FPGA architectures all have rows of logic cells separated by the interconnect—these are row-based ASICs



•FIGURE 16.18 **INTERCONNECT STRUCTURE**. (a) The two-level metal CBIC floorplan shown in Figure 16.11 b. (b) A channel from the flexible block A. This channel has a channel height equal to the maximum channel density of (there is room for seven interconnects to run horizontally in m1). (c) A channel that uses OTC (over-the-cell) routing in m2.



•FIGURE 16.19 **GATE-ARRAY INTERCONNECT**. (a) A small two-level metal gate array (about 4.6 k-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.

Vertical interconnect uses feedthroughs to cross the logic cells. Here are some commonly used terms with explanations (there are no generally accepted definitions):

- An unused vertical track (or just track) in a logic cell is called an **uncommitted feedthrough** (also **built-in feedthrough** , **implicit feedthrough** , or **jumper**).
- A vertical strip of metal that runs from the top to bottom of a cell (for double-entry cells), but has no connections inside the cell, is also called a **feedthrough** or **jumper**.
- Two connectors for the same physical net are **electrically equivalent connectors (or equipotential connectors).** for double-entry cells these are usually at the top and bottom of the logic cell.
- A dedicated **feedthrough cell** (or crosser cell) is an empty cell (with no logic) that can hold one or more vertical interconnects. These are used if there are no other feedthroughs available.
- A **feedthrough pin** or **feedthrough terminal** is an input or output that has connections at both the top and bottom of the standard cell.
- A **spacer cell** (usually the same as a feedthrough cell) is used to fill space in rows so that the ends of all rows in a flexible block may be aligned to connect to power buses, for example.

- There are also **LOGICALLY EQUIVALENT CONNECTORS (**or **FUNCTIONALLY EQUIVALENT CONNECTORS,** sometimes also called just **EQUIVALENT CONNECTORS**—which is very confusing).
- **Example:** The **two inputs of a two-input NAND gate** may be logically equivalent connectors. The placement tool can swap these without altering the logic (but the two inputs may have different delay properties, so it is not always a good idea to swap them).
- There can also be **LOGICALLY EQUIVALENT CONNECTOR GROUPS .** For example, in an OAI22 (OR-AND-INVERT) gate there are four inputs: A1, A2 are inputs to one OR gate (gate A), and B1, B2 are inputs to the second OR gate (gate B). Then group A = (A1, A2) is logically equivalent to group B = (B1, B2)—if we swap one input (A1 or A2) from gate A to gate B, we must swap the other input in the group (A2 or A1).

Interconnect Area for CBIC,MGA and FPGA

HORIZONTAL INTERCONNECT

- In the case of **channeled gate arrays and FPGAs**, the horizontal interconnect areas—the channels, usually on m1—have a **fixed capacity**.

- The channel capacity of **CBICs and channelless MGAs can be expanded** to hold as many interconnects as are needed. Normally we choose, as an objective, to minimize the number of interconnects that use each channel.

VERTICAL INTERCONNECT

- In the **vertical interconnect** direction, usually m2, **FPGAs still have fixed resources**.

- In contrast the placement tool can always **add vertical feedthroughs** to a **channeled MGA, channelless MGA, or CBIC.** These problems become less important as we move to three and more levels of interconnect.

Placement Goals and Objectives

The **goal of a placement** tool is to arrange all the logic cells within the flexible blocks on a chip.

Ideally, the **objectives** of the placement step are to

- Guarantee the router can complete the routing step
- Minimize all the critical net delays
- Make the chip as dense as possible

We may also have the following additional objectives:

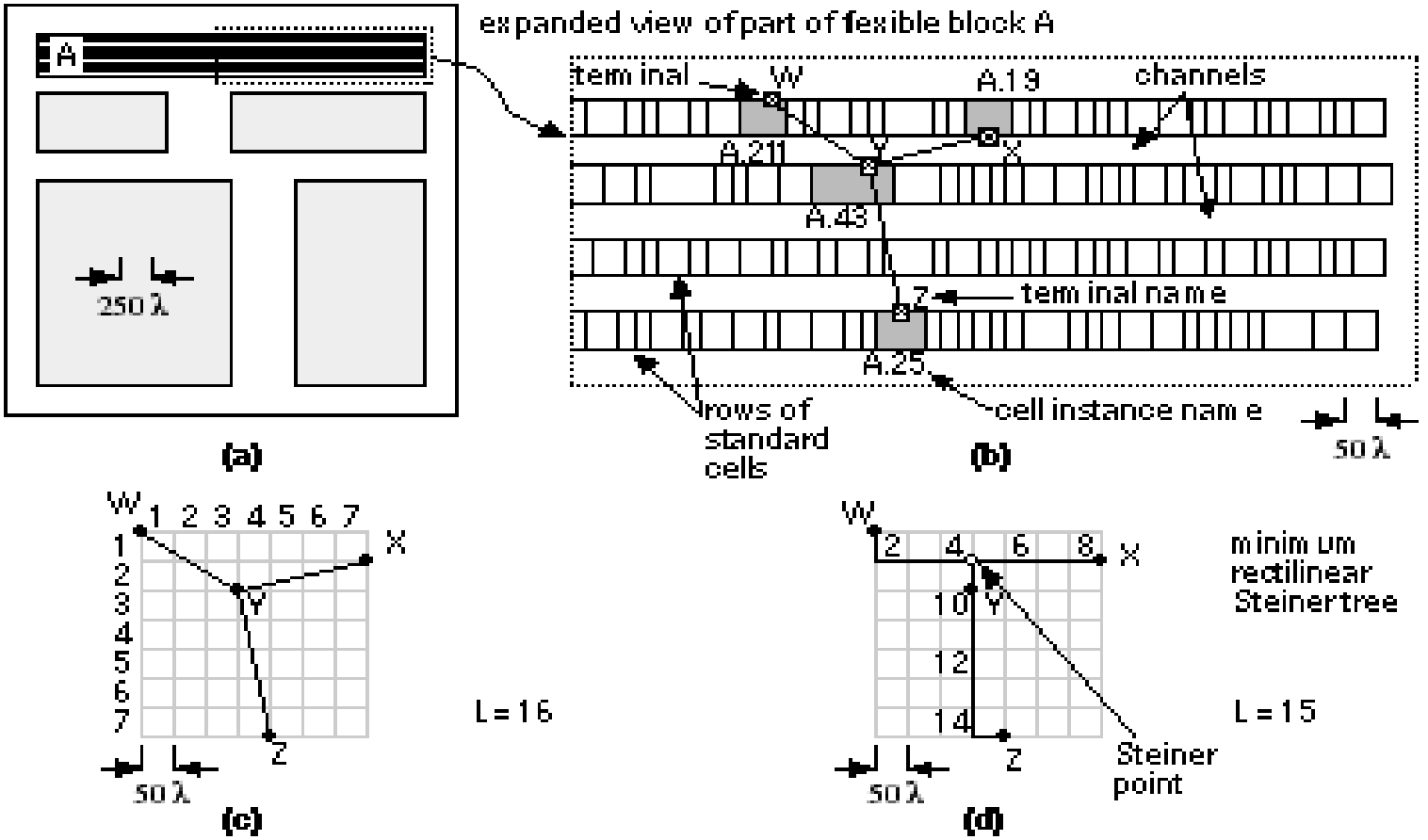
- Minimize power dissipation
- Minimize cross talk between signals

Current **placement tools** use more specific and achievable criteria. The most commonly used placement objectives are one or more of the following:

- Minimize the total estimated interconnect length
- Meet the timing requirements for critical nets
- Minimize the interconnect congestion

Measurement of Placement Goals and Objectives

- The graph structures that correspond to making **all the connections for a net** are known as **trees on graphs** (or just **trees**).
- Special classes of trees— **Steiner trees —minimize the total length of interconnect** and they are central to ASIC routing algorithms.
- **Minimum Steiner tree** - This type of tree **uses diagonal connections**—we want to solve a restricted version of this problem, using **interconnects on a rectangular grid**. This is called **rectilinear routing or Manhattan routing.**
- **Euclidean distance** between two points is the **straight-line distance**.
- The **Manhattan distance** (or **rectangular distance**) between two points is the distance we would have to walk in New York.



•FIGURE 16.20 Placement using trees on graphs. (a) The floorplan from Figure 16.11 b. (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

Measurement of Placement (contd.,)

- The **minimum rectilinear Steiner tree (MRST)** is the **shortest interconnect using a rectangular grid.** The determination of the MRST is in general an NP-complete problem—which means it is hard to solve.

- The **complete graph** has connections from each terminal to every other terminal.

- The **complete-graph measure** adds all the interconnect lengths of the complete-graph connection together and then divides by n /2, where n is the number of terminals.

  Complete graph = (n (n -1)) / 2)

- The **bounding box** is the smallest rectangle that encloses all the terminals.

- **half-perimeter measure** (or bounding-box measure) is one-half the perimeter of the bounding box.

  half perimeter $f = \frac{1}{2} \sum_{i=1}^{m} h_i$

  where m is the nets, h_i is the half perimeter measure for net i.

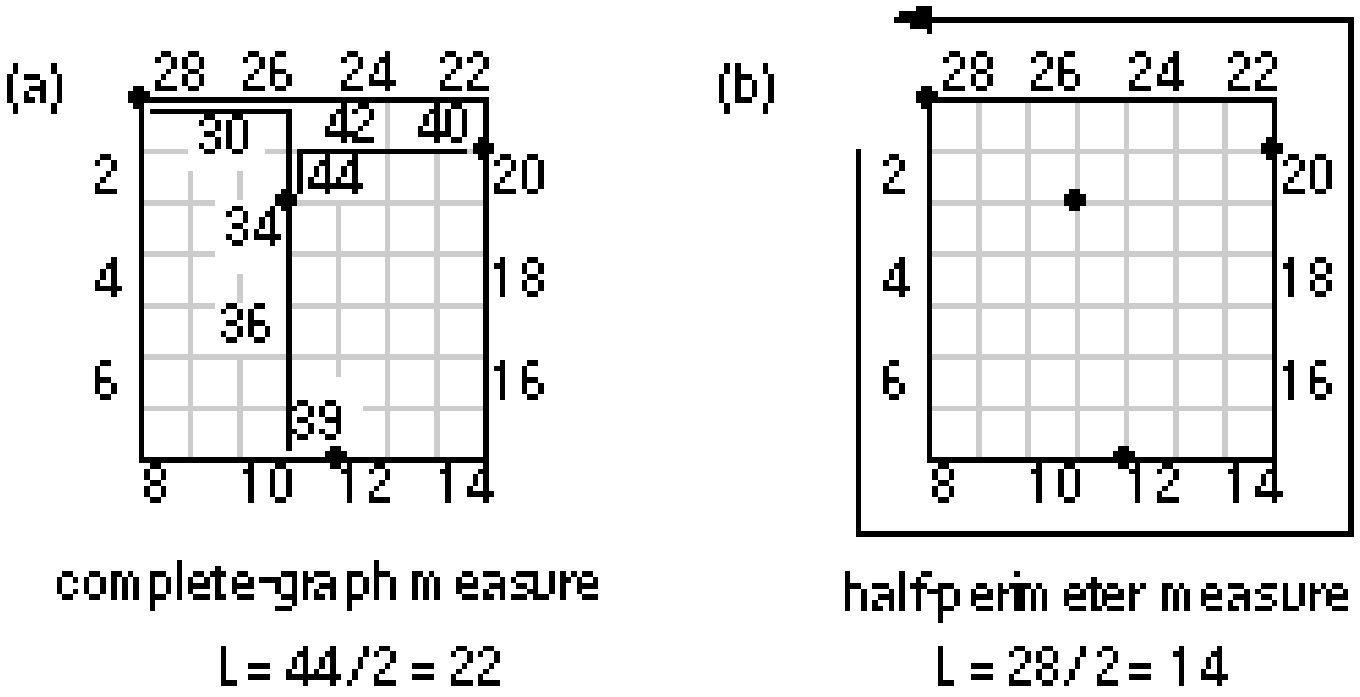
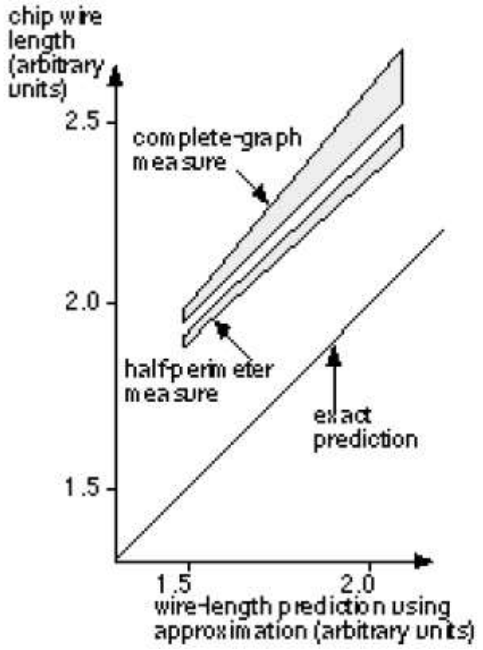


FIGURE 16.21 Interconnect-length measures. (a) Complete-graph measure. (b) Half-perimeter measure.

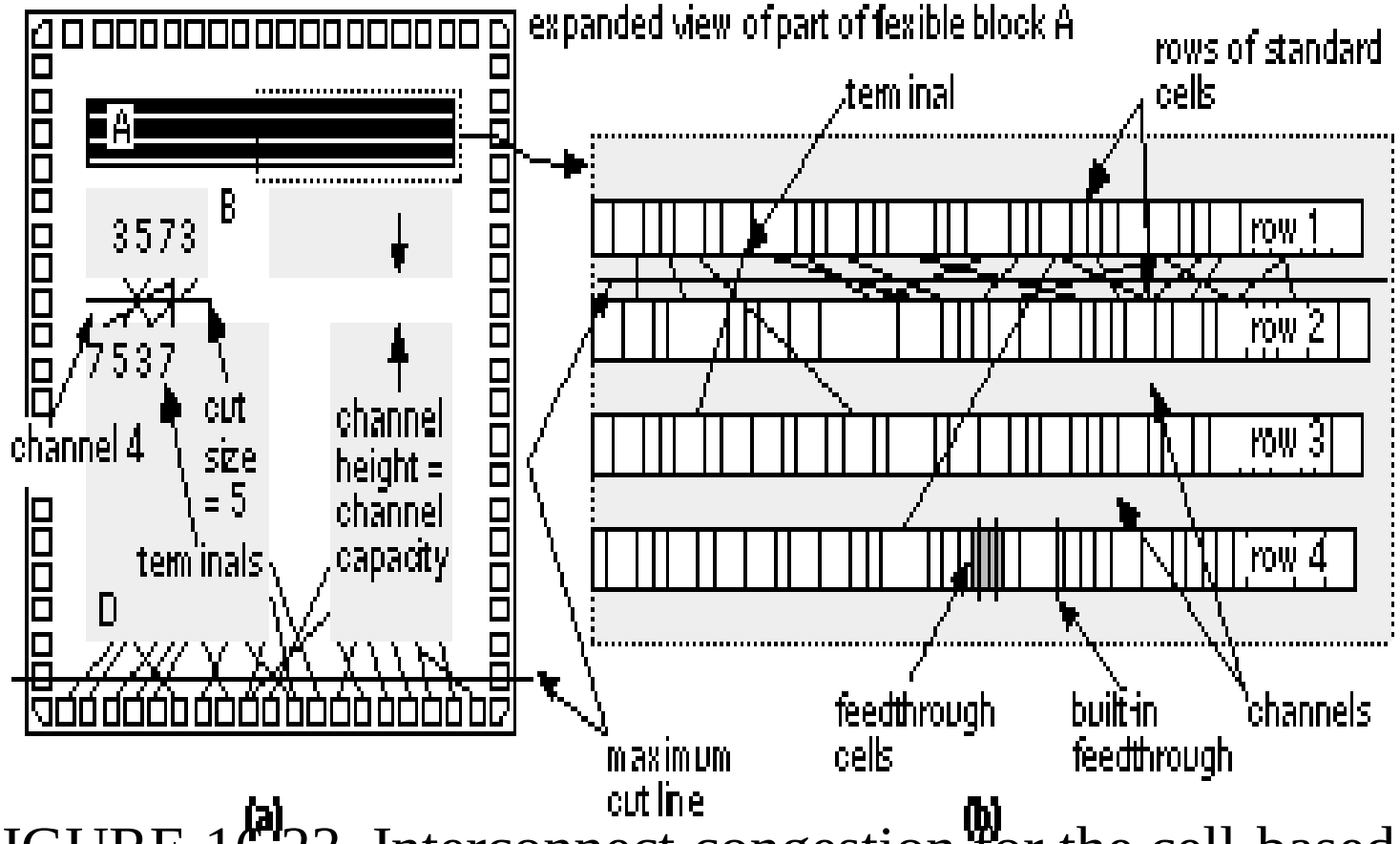
Correlation between total length of chip interconnect and the half-perimeter and complete-graph measures.



- **Meander factor** that specifies, on average, the ratio of the interconnect created by the routing tool to the interconnect-length estimate used by the placement tool.
- Another problem is that we have concentrated on finding estimates to the MRST, but the **MRST that minimizes total net length may not minimize net delay.**

Interconnect congestion

- There is no point in minimizing the interconnect length if we create a **placement that is too congested to route**.
- If we use **minimum interconnect congestion as an additional placement objective**, we need some way of measuring it.
- What we are trying to measure is **interconnect density**
- One **measure of interconnect congestion** uses the **maximum cut line** .
- **Maximum cut line:** Imagine a horizontal or vertical line drawn anywhere across a chip or block,
- **The number of interconnects that must cross this line** is the **cut size** (the number of interconnects we cut). The **maximum cut line has the highest cut size.**



•FIGURE 16.23 Interconnect congestion for the cell-based ASIC from Figure 16.11 (b). (a) Measurement of congestion. (b) An expanded view of flexible block A shows a maximum cut line.

Interconnect Delay

- Many **placement tools minimize estimated interconnect length or interconnect congestion as objectives**.
- The **problem with this approach is that a logic cell may be placed a long way from another logic cell to which it has just one connection.** This logic cell with one connection is less important as far as the total wire length is concerned than other logic cells, to which there are many connections. However, **the one long connection may be critical as far as timing delay is concerned.**
- As **technology is scaled, interconnection delays become larger relative to circuit delays** and this problem gets worse.

Interconnect Delay

- In **timing-driven placement we must estimate delay for every net for every trial** placement, possibly for hundreds of thousands of gates.

- Unfortunately, the **minimum-length Steiner tree does not necessarily correspond to the interconnect path that minimizes delay.** To construct a **minimum-delay path we may have to route with non-Steiner trees.**

- In the placement phase typically **we take a simple interconnect length approximation to this minimum-delay path (typically the half-perimeter measure).**

- Even when we can estimate the length of the interconnect, we **do not yet have information on which layers and how many vias the interconnect will use or how wide it will be.** Some tools allow us to include estimates for these parameters.
- Often we can specify **metal usage , the percentage of routing on the different layers to** expect from the router. **This allows the placement tool to estimate RC values and delays—and thus minimize delay.**

Placement Algorithms

There are two classes of placement algorithms commonly used in commercial CAD tools:

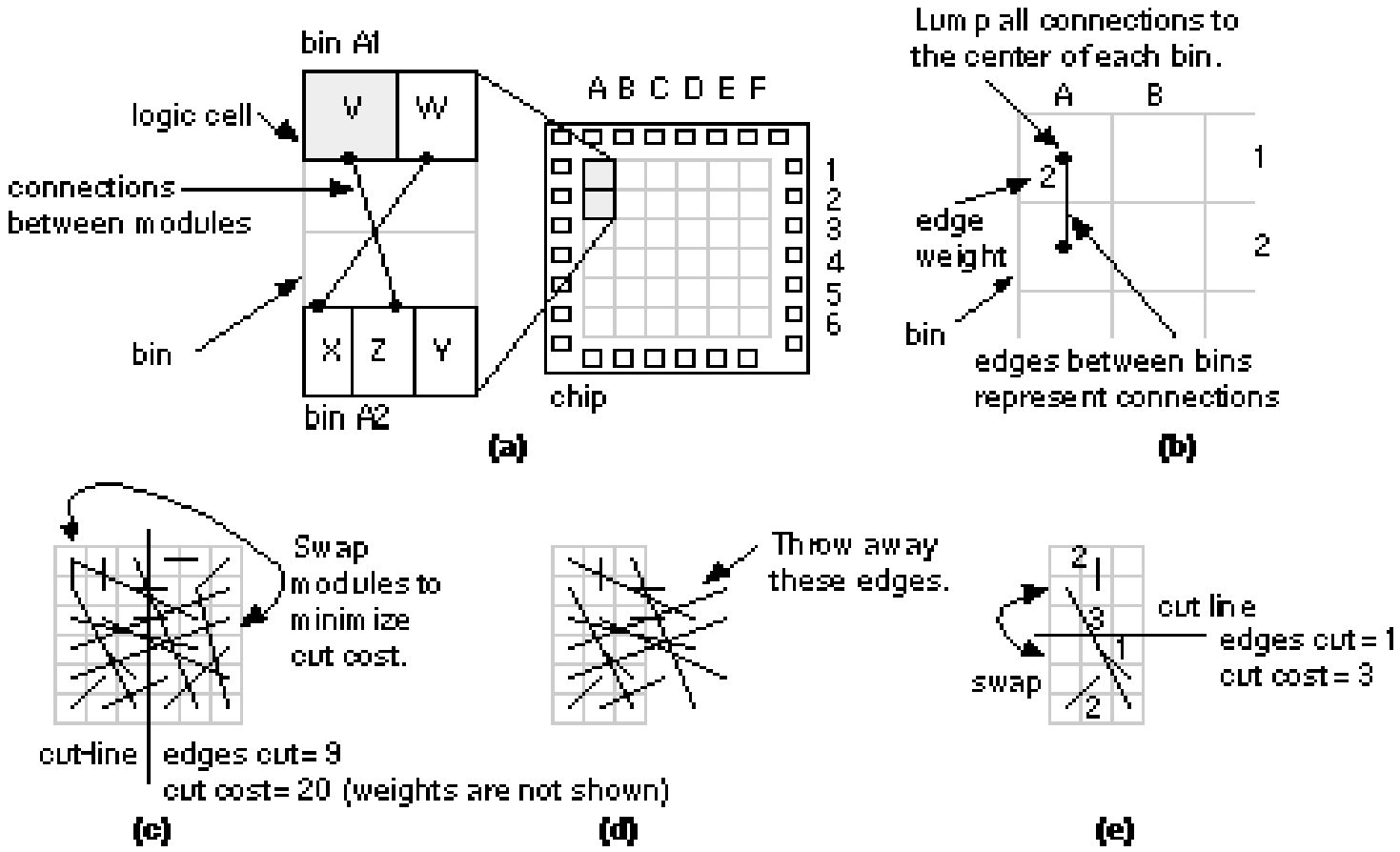
- **constructive placement** - uses a set of rules to arrive at a constructed placement.
  Example :**min-cut algorithm. Eigenvalue method**.
- **iterative placement improvement.**

As in system partitioning, placement usually starts with a constructed solution and then improves it using an iterative algorithm.

The **min-cut placement method** uses successive application of partitioning. The following steps are,

- **Cut the placement area into two pieces.**
- **Swap the logic cells to minimize the cut cost.**
- **Repeat the process from step 1, cutting smaller pieces until all the logic cells are placed**

Usually we divide the **placement area into bins** . The size of a bin can vary, from a bin size equal to the base cell (for a gate array) to a **bin size that would hold several logic cells**. We can **start with a large bin size, to get a rough placement, and then reduce the bin size to get a final placement**.



•FIGURE 16.24 Min-cut placement. (a) Divide the chip into bins using a grid. (b) Merge all connections to the center of each bin. (c) Make a cut and swap logic cells between bins to minimize the cost of the cut. (d) Take the cut pieces and throw out all the edges that are not inside the piece. (e) Repeat the process with a new cut and continue until we reach the individual bins.

Eigen Value Placement Algorithm

The eigenvalue placement algorithm **uses the cost matrix or weighted connectivity matrix** (**eigen value methods** are also known as **spectral methods**) [Hall, 1970]. The measure we use is **a cost function f that we shall minimize**, given by ,

$$f = \frac{1}{2}\sum_{i=1}^{n} c_{ij} d_{ij}^2 \qquad (1)$$

where C = [c_{ij}] is the (possibly weighted) connectivity matrix, and d_{ij} is the Euclidean distance between the centers of logic cell i and logic cell j . Since we are going to minimize a cost function that is the square of the distance between logic cells, these methods are also known as **quadratic placement methods**. This type of cost function leads to a simple mathematical solution. We can rewrite the cost function f in matrix form:

$$f = \frac{1}{2}\sum_{i,j=1}^{n} c_{ij}\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2$$

$$f = x^T Bx + y^T By$$

B is a symmetric matrix, the disconnection matrix (also called the Laplacian).

B= D- C

C – Connectivity Matrix ; D – Diagonal Matrix or Degree Matrix

where,

$$d_{ii} = \sum_{j=1}^{n} C_{ij}$$

$$d_{ij} = 0, i \neq j$$

We can simplify the problem by noticing that it is symmetric in the **x - and y -coordinates**.

Let us **solve the simpler problem of minimizing the cost function for the placement of logic cells along just the x – axis first.** We can then apply this solution to the more general **two-dimensional placement problem**.

Before we solve this simpler problem, we introduce a constraint that the **coordinates of the logic cells must correspond to valid positions** (the cells do not overlap and they are placed on-grid). We make another simplifying assumption that **all logic cells are the same size and we must place them in fixed positions.** We can define a vector p consisting of the valid positions:

$$p = [p_1, p_2 .... p_n] \qquad (4)$$

For a valid placement the x -coordinates of the logic cells,

$$x = [x_1, x_2, ... x_n] \qquad (5)$$

must be a permutation of the fixed positions, p . We can show that requiring the logic cells to be in fixed positions in this way leads to a series of n equations restricting the values of the logic cell coordinates .If we impose all of these **constraint equations** the problem becomes very complex. Instead we choose just one of the equations:

$$\sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} p_i^2 \qquad (6)$$

Simplifying the problem in this way will lead to an approximate solution to the placement problem. We can write this single constraint on the x -coordinates in matrix form: ,

$$x^T x = P$$

$$P = \sum_{i=1}^{n} p_i^2$$

where P is a constant.

We can now summarize the formulation of the problem, with the simplifications that we have made, for a one-dimensional solution. We must minimize a cost function, g, where

$$g = x^T B x \tag{8}$$

subject to the constraint:

$$x^T x = p \tag{9}$$

This is a standard problem that we can solve using a Lagrangian multiplier:

$$\Lambda = x^T B x - \lambda\left[x^T x - p\right] \tag{10}$$

To find the value of x that minimizes g we differentiate L partially with respect to x and set the result equal to zero. We get the following equation:

$$\left[B - \lambda I\right]x = 0 \tag{11}$$

This last equation is called the characteristic equation for the disconnection matrix B and occurs frequently in matrix algebra (this l has nothing to do with scaling). The solutions to this equation are the eigenvectors and eigenvalues of B . Multiplying Eq.(11) by x^T we get:

$$\lambda x^T x = x^T B x$$

However, since we imposed the constraint $x^T x = I$ and $x^T Bx = g$, then

$$\lambda = \frac{g}{p}$$

The eigenvectors of the disconnection matrix B are the solutions to our placement problem.

calculate eigen value:

$$C = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix} \qquad D = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

$B = D - C$

$$B = \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 2 & -1 & -1 \\ 0 & -1 & 1 & 0 \\ -1 & -1 & 0 & 2 \end{bmatrix}$$

$$B - \lambda I = \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 2 & -1 & -1 \\ 0 & -1 & 1 & 0 \\ -1 & -1 & 0 & 2 \end{bmatrix} - \begin{bmatrix} \lambda & 0 & 0 & 0 \\ 0 & \lambda & 0 & 0 \\ 0 & 0 & \lambda & 0 \\ 0 & 0 & 0 & \lambda \end{bmatrix}$$

$$|B - \lambda I| = \begin{vmatrix} 1-\lambda & 0 & 0 & -1 \\ 0 & 2-\lambda & -1 & -1 \\ 0 & -1 & 1-\lambda & 0 \\ -1 & -1 & 0 & 2-\lambda \end{vmatrix} = 0$$

$$= (1-\lambda)\{(2-\lambda)\{(1-\lambda)(2-\lambda)\} + 1(\lambda-2) + 1(\lambda-1)\}$$

$$-1\{(2-\lambda)(1-\lambda) - 1\}$$

$$= (1-\lambda)\{4 - 4\lambda + \lambda^2 - 4\lambda + 4\lambda^2 - \lambda^3 + 2\lambda - 3\} - 1 + 3\lambda - \lambda^2$$

$$= (1-\lambda)\{-\lambda^3 + 5\lambda^2 - 6\lambda + 1\} - 1 + 3\lambda - \lambda^2$$

$$= \lambda^4 - 6\lambda^3 + 10\lambda^2 - 4\lambda = 0$$

$$\lambda(\lambda^3 - 6\lambda^2 + 10\lambda - 4) = 0 \qquad \lambda_1 = 0,\ \lambda_2 = 3.414,\ \lambda_3 = 0.585$$

$$\lambda_4 = 2$$

Iterative Placement Improvement

An iterative placement improvement algorithm takes an existing placement and tries to improve it by moving the logic cells. There are two parts to the algorithm:

- The **selection criteria** that decides which logic cells to try moving.
- The **measurement criteria** that decides whether to move the selected cells.

There are several interchange or iterative exchange methods that differ in their selection and measurement criteria:

- Pair wise interchange,
- force-directed interchange,
- force-directed relaxation, and
- force-directed pair wise relaxation.

All of these methods usually consider only pairs of logic cells to be exchanged. A source logic cell is picked for trial exchange with a destination logic cell

Iterative Placement Improvement

An iterative placement improvement algorithm takes an existing placement and tries to improve it by moving the logic cells. There are two parts to the algorithm:

- The **selection criteria** that decides which logic cells to try moving.
- The **measurement criteria** that decides whether to move the selected cells.

There are several interchange or iterative exchange methods that differ in their selection and measurement criteria:

- Pair wise interchange,
- force-directed interchange,
- force-directed relaxation, and
- force-directed pair wise relaxation.

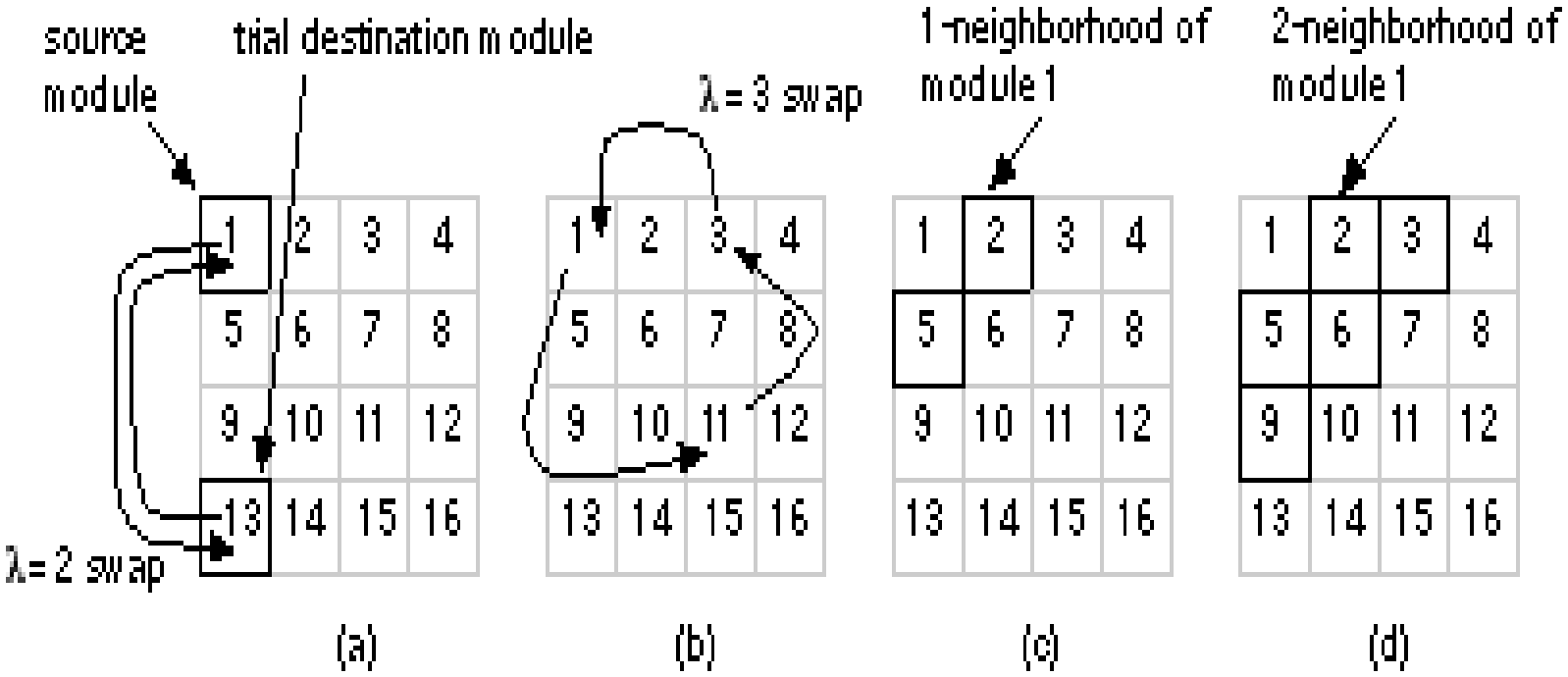
All of these methods usually consider only pairs of logic cells to be exchanged. A source logic cell is picked for trial exchange with a destination logic cell

Iterative Placement Improvement (contd.,)

The **pair wise-interchange** algorithm is similar to the interchange algorithm used for iterative improvement in the system partitioning step:

- Select the source logic cell at random.
- Try all the other logic cells in turn as the destination logic cell.
- Use any of the measurement methods we have discussed to decide on whether to accept the interchange.
- The process repeats from step 1, selecting each logic cell in turn as a source logic cell.

The **neighborhood exchange algorithm** is a modification to pairwise interchange that considers only destination logic cells in a neighborhood —cells within a certain distance, e, of the source logic cell. Limiting the search area for the destination logic cell to the e -neighborhood **reduces the search time**.



•FIGURE 16.26 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 e of a logic cell form an e-neighborhood.

•(c) A one-neighborhood.

•(d) A two-neighborhood.

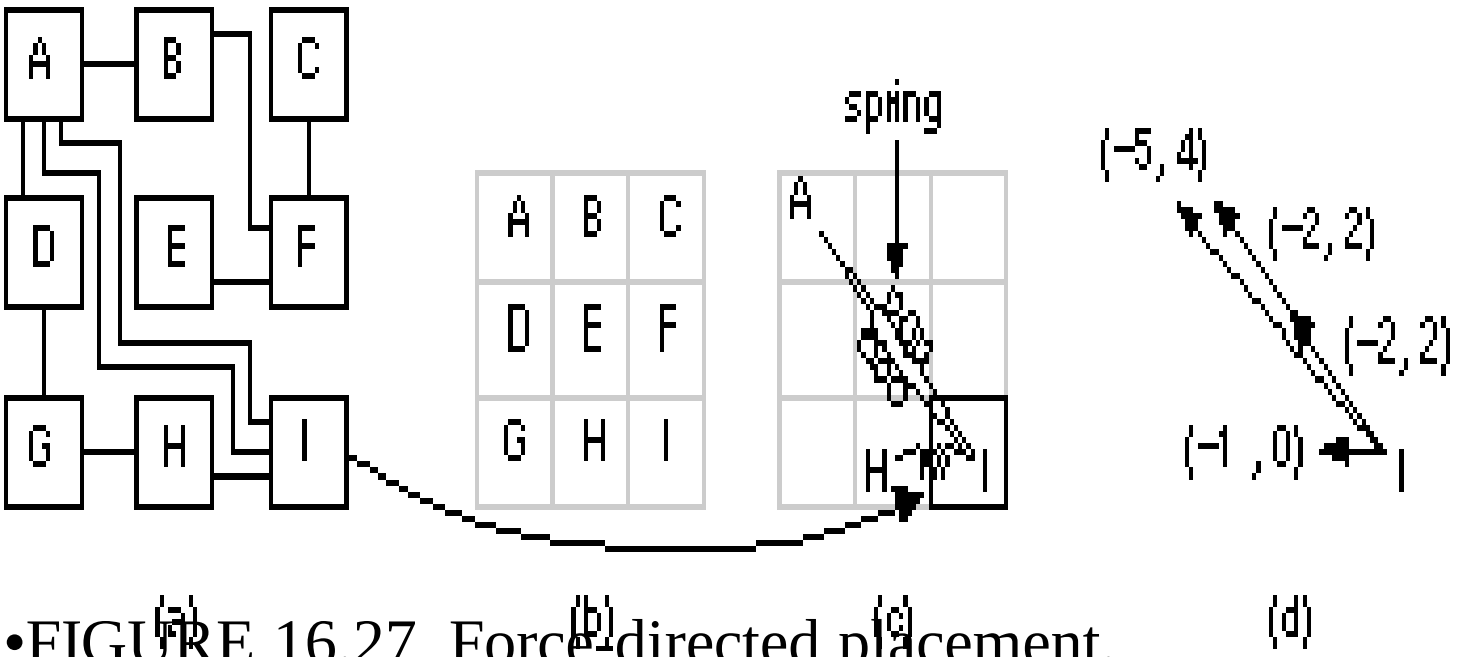
Iterative Placement Improvement (contd.,)

Force-directed placement methods:

Imagine identical **springs connecting all the logic cells** we wish to place. The **number of springs is equal to the number of connections** between logic cells. The effect of the springs is to pull connected logic cells together. The more highly connected the logic cells, the stronger the pull of the springs. The **force on a logic cell i due to logic cell j** is given by **Hooke's law** , which says the force of a spring is proportional to its extension:

$$F_{ij} = -c_{ij} x_{ij} .$$

- The vector component x_{ij} is directed from the center of logic cell i to the center of logic cell j .
- The vector magnitude is calculated as either the **Euclidean or Manhattan distance** between the logic cell centers.
- The c_{ij} form the connectivity or cost matrix (the matrix element c_{ij} is the number of connections between logic cell i and logic cell j).



•FIGURE 16.27 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.

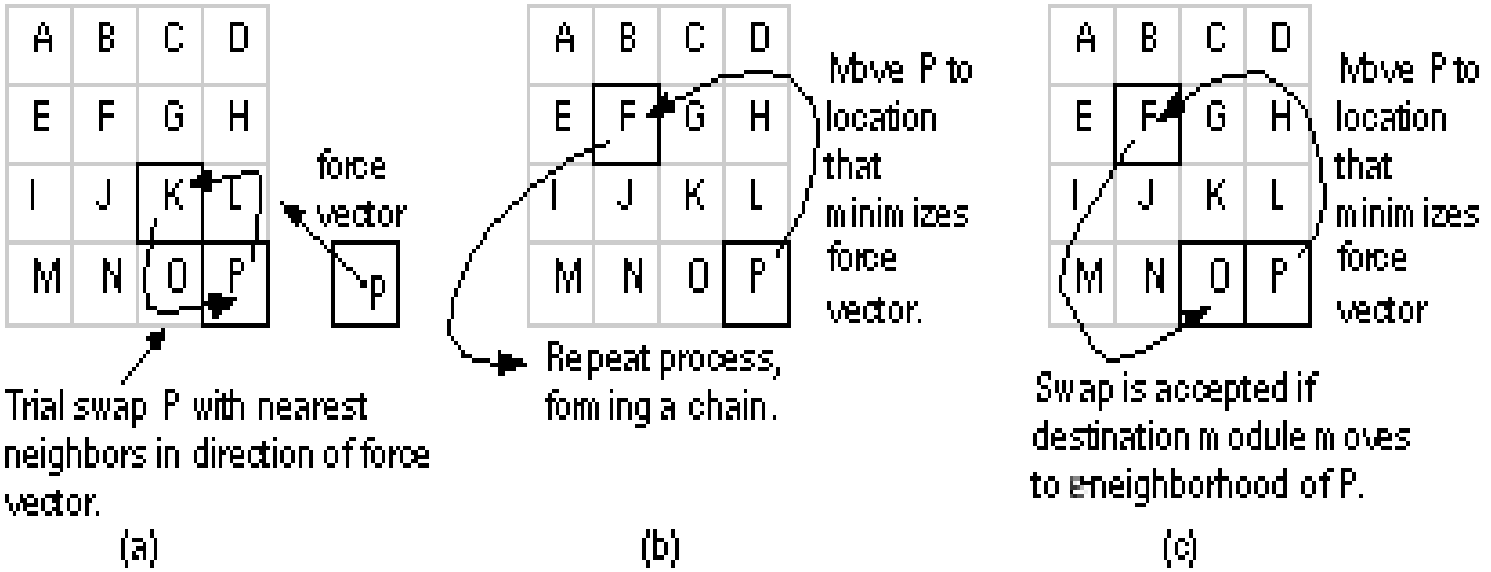
•(d) The forces are proportional to the spring extensions.

Iterative Placement Improvement (contd.,)

Force-directed placement algorithms:

- The **force-directed interchange algorithm uses the force vector** to select a pair of logic cells to swap.
- The **force-directed relaxation** a chain of logic cells is moved.
- The **force-directed pairwise relaxation algorithm** swaps one pair of logic cells at a time.

We reach a force-directed solution when we **minimize the energy of the system, corresponding to minimizing the sum of the squares of the distances separating logic cells.** Force-directed placement algorithms thus also use a quadratic cost function.



•FIGURE 16.28 Force-directed iterative placement improvement.

•(a) Force-directed interchange.

•(b) Force-directed relaxation.

•(c) Force-directed pairwise relaxation.

Placement Using Simulated Annealing

Applying simulated annealing to placement, the algorithm is as follows:

- Select logic cells for a trial interchange, usually at random.
- Evaluate the objective function E for the new placement.
- If D E is negative or zero, then exchange the logic cells. If D E is positive, then exchange the logic cells with a probability of exp(– D E / T).
- 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.

Experiments show that simple **min-cut based constructive placement is faster** than simulated annealing but that **simulated annealing is capable of giving better results at the expense of long computer run times**. The iterative improvement methods that we described earlier are capable of giving results as good as simulated annealing, but they **use more complex algorithms.**

Timing-Driven Placement Methods

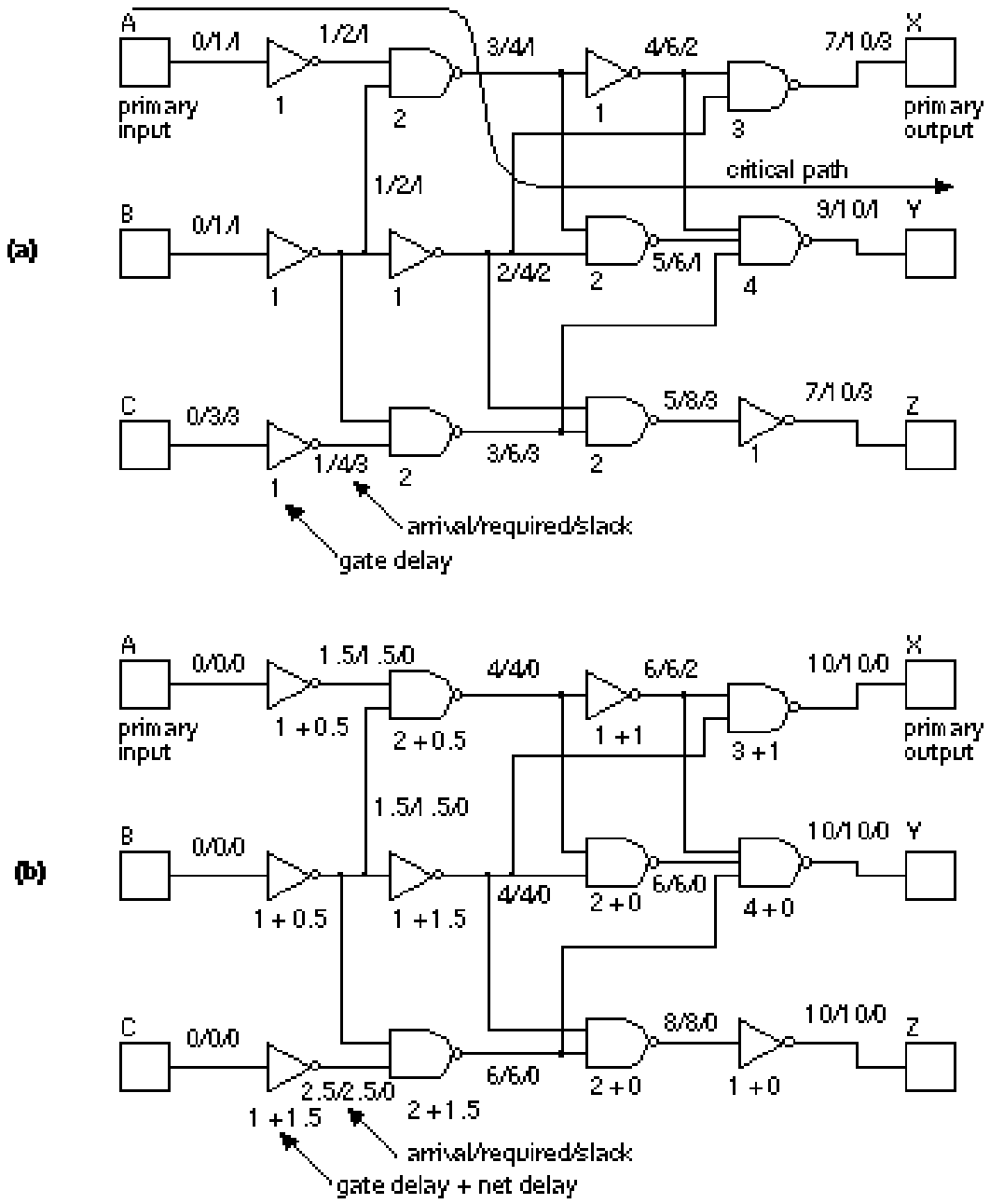
- Minimizing delay is becoming more and more important as a placement objective.
- There are two main approaches:

  – net based

  – path based
- We can use net weights in our algorithms.
- The problem is to calculate the weights.
- One method finds the n most critical paths (using a timing-analysis engine, possibly in the synthesis tool).
- The net weights might then be the number of times each net appears in this list. Another method to find the net weights uses the **zeroslack algorithm.**

Timing-Driven Placement Methods

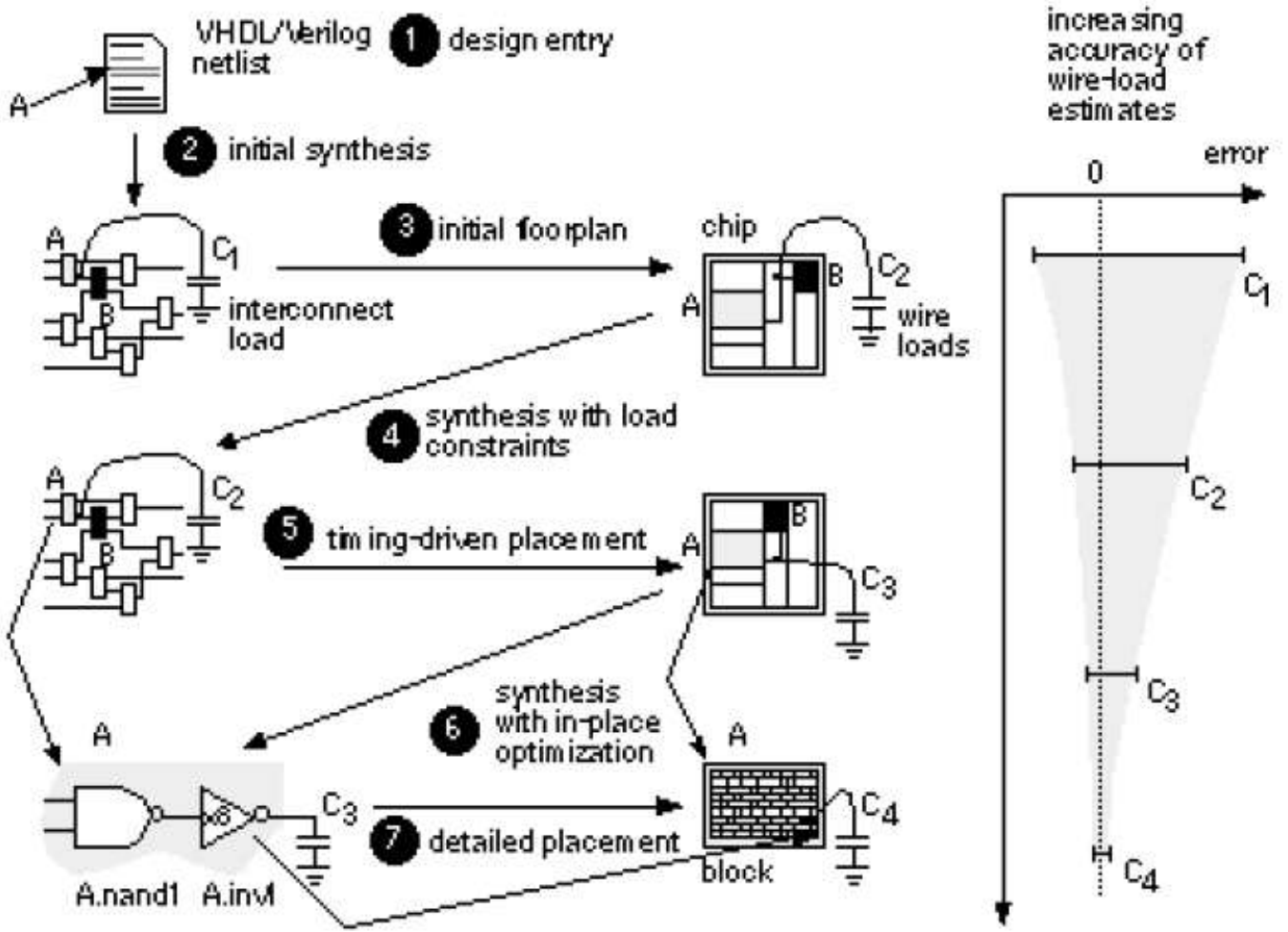
- Figure 16.29 (a) shows a circuit with primary inputs at which we know the arrival times (actual times) of each signal.
- We also know the required times for the primary outputs the points in time at which we want the signals to be valid.
- We can work forward from the primary inputs and backward from the primary outputs to determine arrival and required times at each input pin for each net.
- The difference between the required and arrival times at each input pin is the slack time (the time we have to spare).
- The zero-slack algorithm adds delay to each net until the slacks are zero, as shown in Figure 16.29 (b).
- The net delays can then be converted to weights or constraints in the placement.



•FIGURE 16.29 The zero-slack algorithm. (a) The circuit with no net delays.

•(b) The zero-slack algorithm adds net delays

Physical design flow



Module IV
Routing

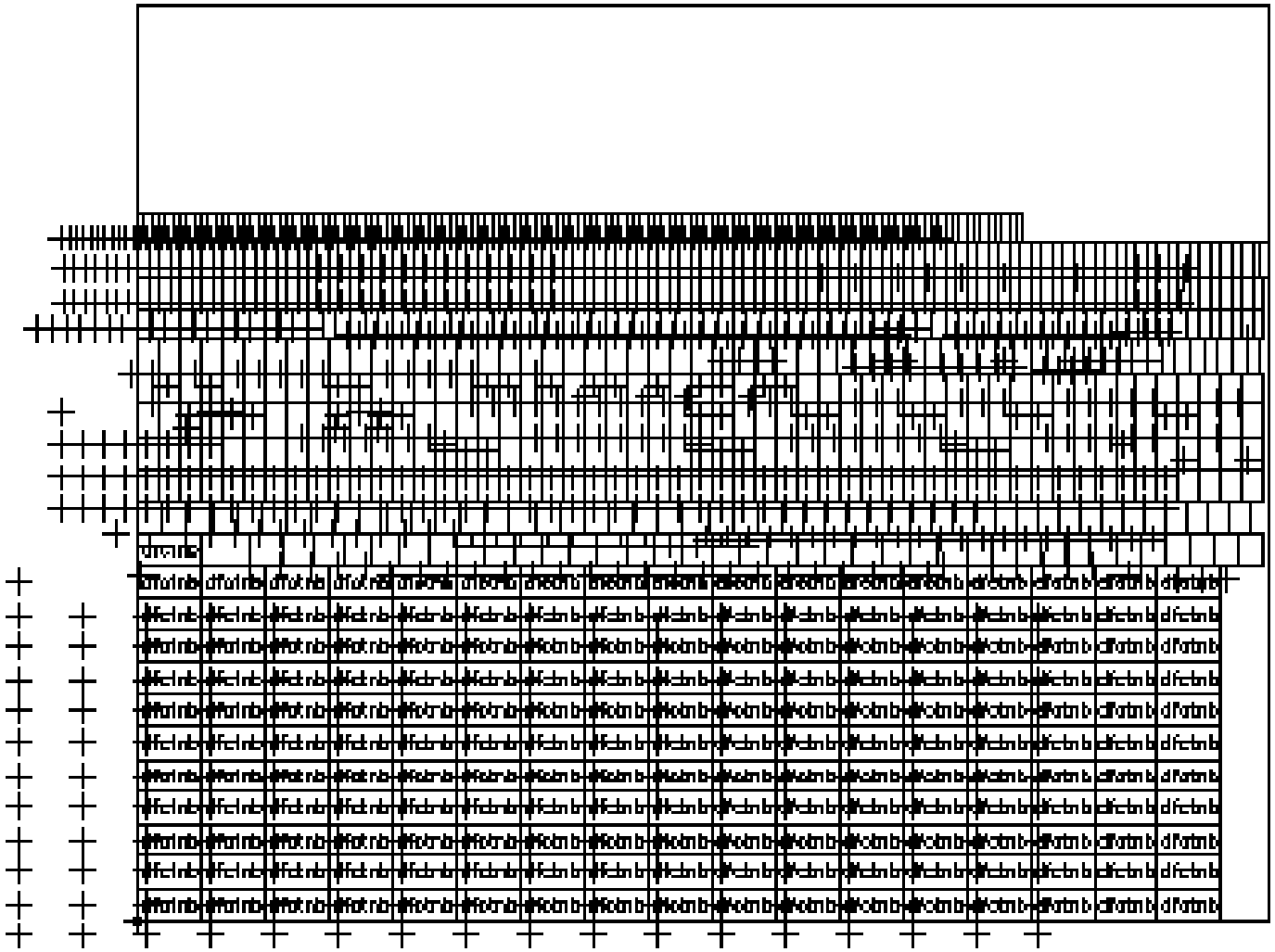
Dr.(Mrs).D.Gracia Nirmala Rani
Assistant Professor
ECE Department
Thiagarajar College of Engineering
Madurai-15
Email : gracia@tce.edu

Introduction

- Once the designer has
  - **Floorplanned** a chip
  - The logic cells within the flexible blocks have been **placed**
  - Time to **make the connections by routing** the chip.

- This is still a hard problem that is made easier by dividing it into smaller problems.

- Routing is usually split into
  - **Global routing** followed by **detailed routing** .

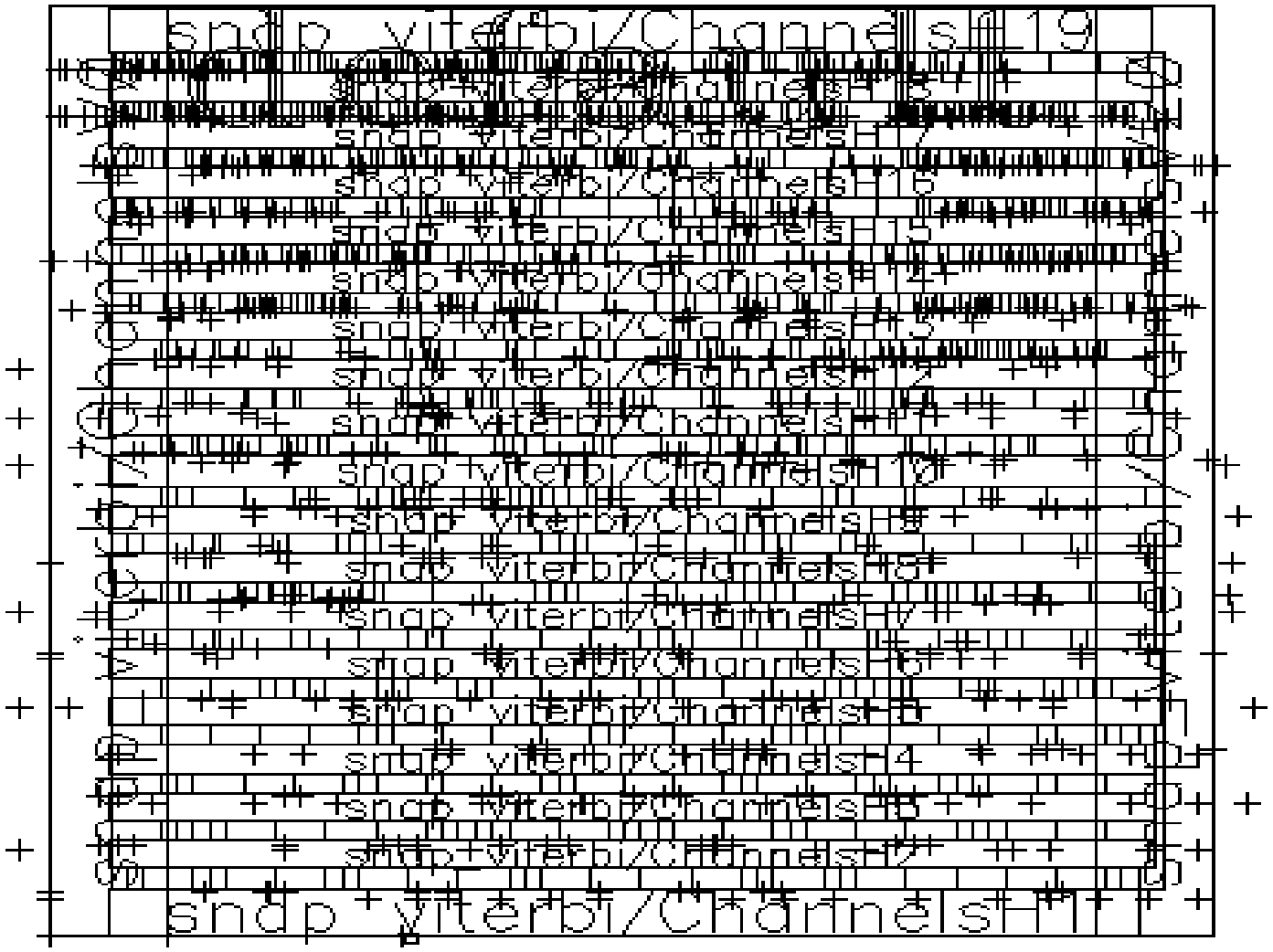


- The starting point of floorplaning and placement steps for the viterbi decoder

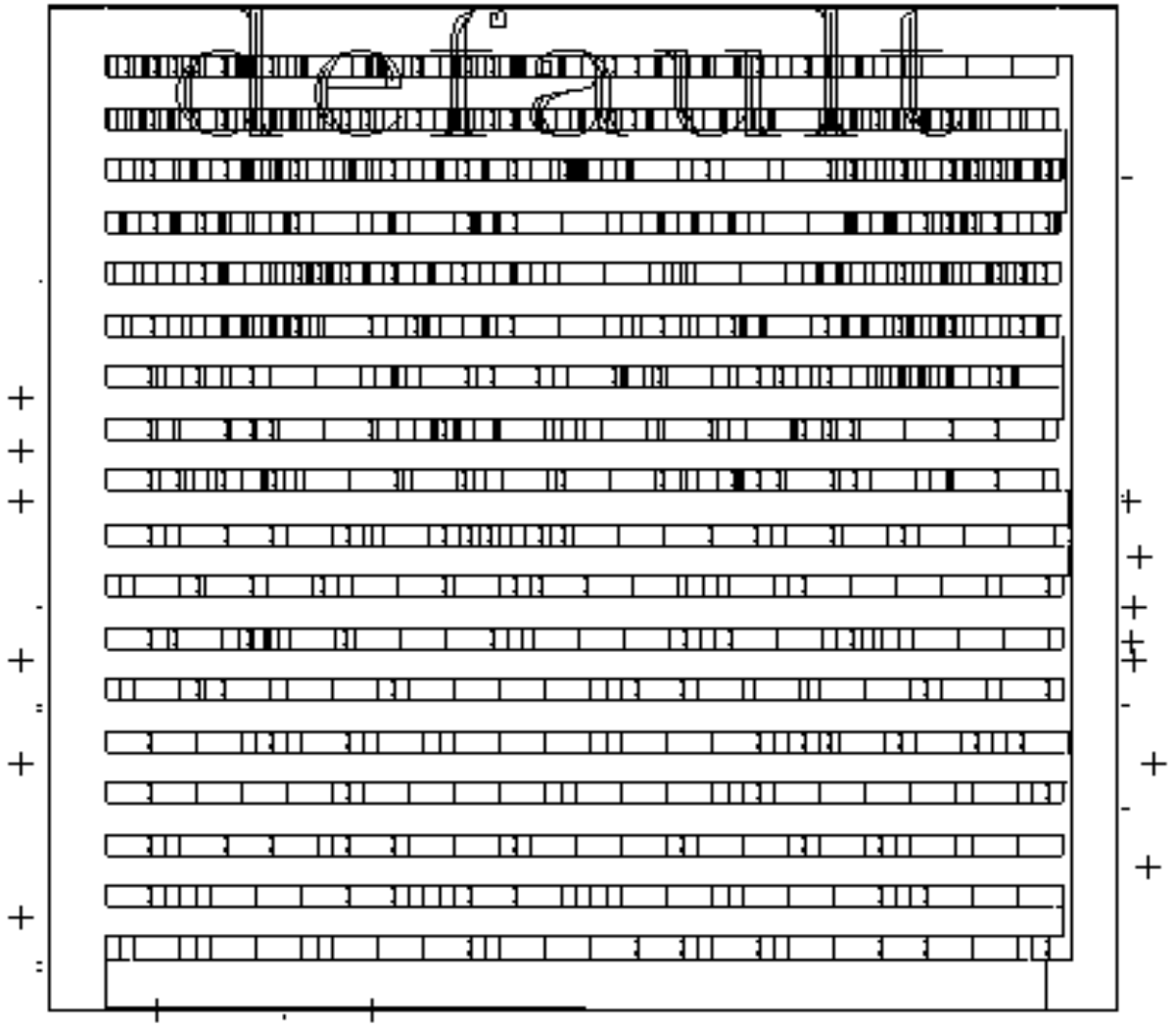
- **-collection of standard cells with no room set aside yet for routing.**

The starting point of floorplaning and placement steps for the viterbi decoder

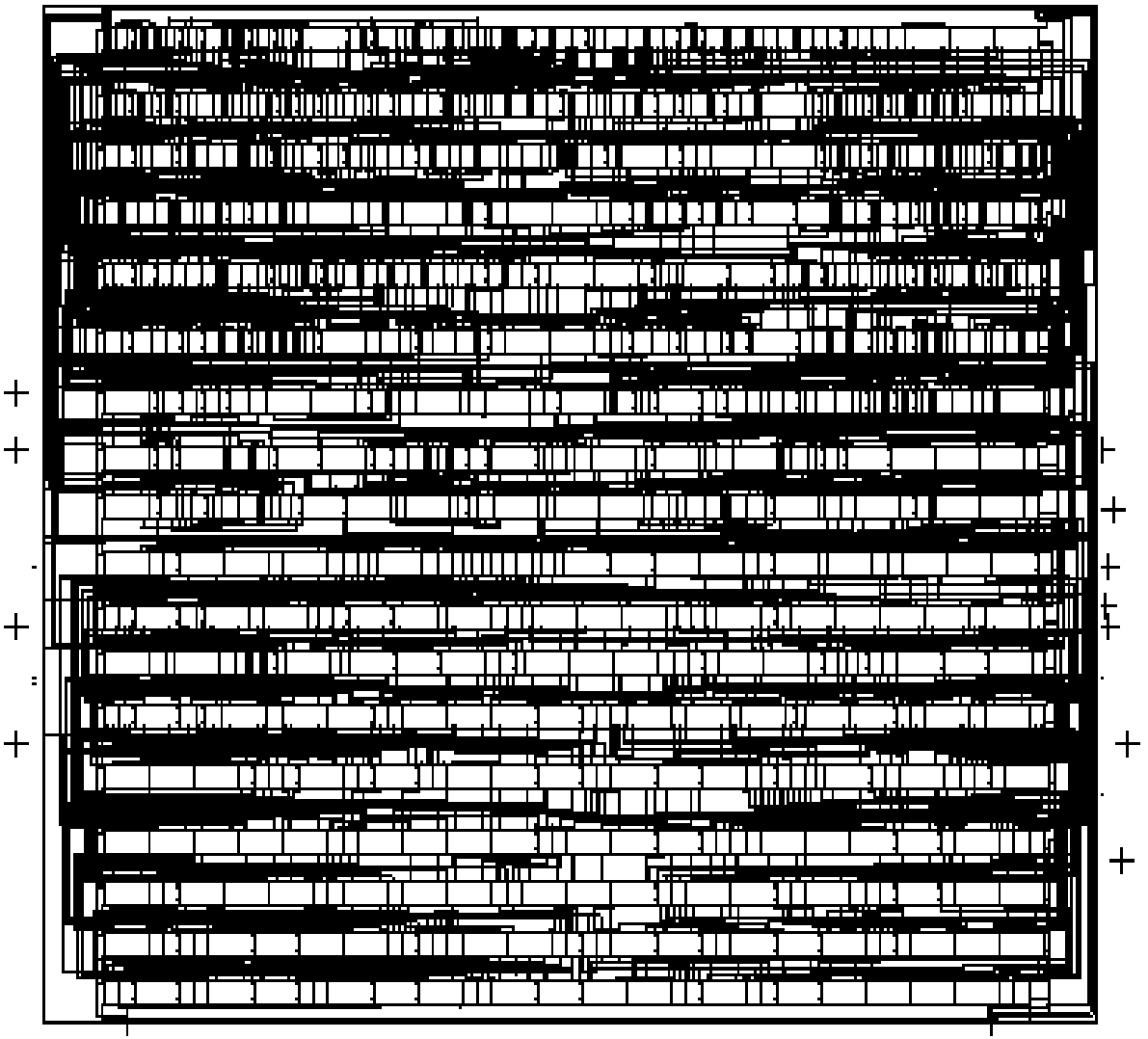
- **Small boxes that look like bricks** - outlines of the standard cells.
- **Largest standard cells, at the bottom of the display** (labeled dfctnb) - 188 D flipflops.
- **'+' symbols** -drawing origins of the standard cells—for the D flip-flops they are shifted to the left and below the logic cell bottom left-hand corner.
- **Large box surrounding all the logic cells** - estimated chip size.
- (This is a screen shot from Cadence Cell Ensemble.)



The viterbi decoder after floorplanning



•FIGURE 17.1 The core of the Viterbi decoder chip after placement (a screen shot from Cadence Cell Ensemble)



•FIGURE 17.2 The core of the Viterbi decoder chip after the completion of global and detailed routing (a screen shot from Cadence Cell Ensemble). This chip uses two-level metal. Although you cannot see the difference, m1 runs in the horizontal direction and m2 in the vertical direction.

Global Routing

- The details of **global routing differ** slightly between
  - **cell-based ASICs, gate arrays, and FPGAs**, but the principles are the same.

- A global router does not make any connections, **it just plans them.**

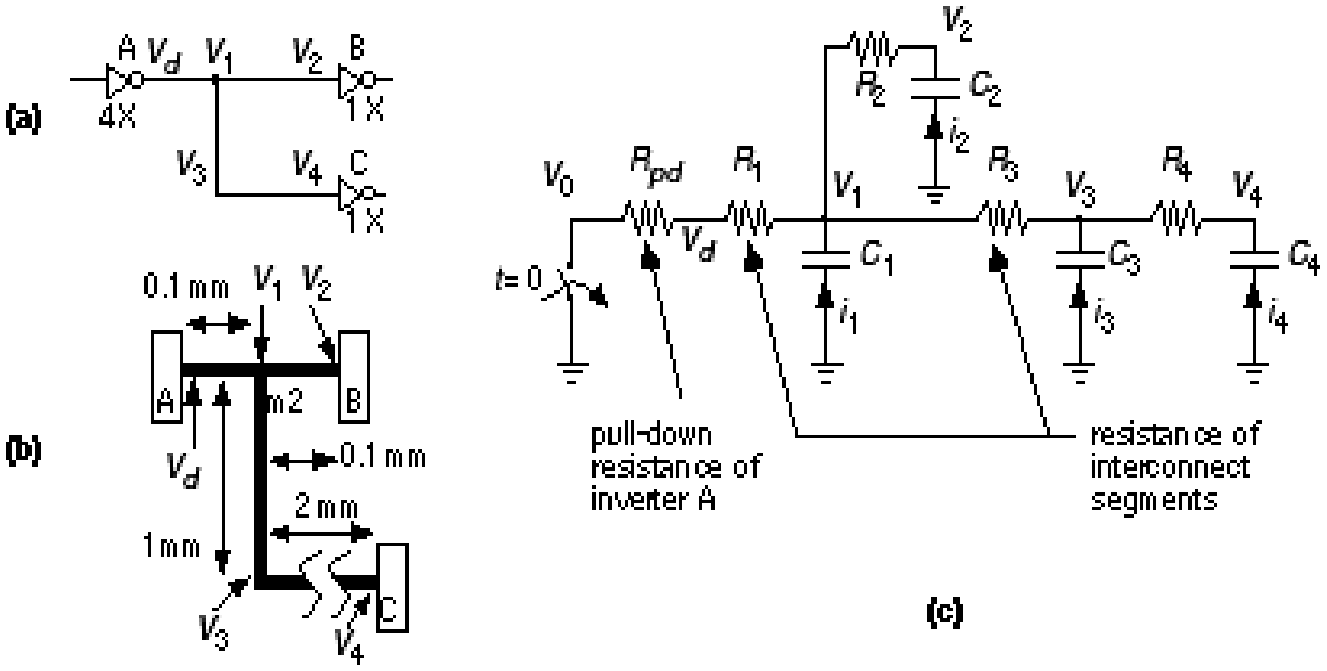
- Global route the whole chip (or large pieces if it is a large chip) **before detail routing the whole chip (or the pieces).**

Goals and Objectives

- **Input to routing**
  - **Floorplan** that includes the locations of all the fixed and flexible blocks;
  - **Placement information** for flexible blocks;
    - **Locations of all the logic cells.**
- **Goal** of global routing
  - **To provide complete instructions to the detailed router** on where to route every net.
- **Objectives** of global routing
    - ***Minimize the total interconnect length.***
    - ***Maximize the probability that the detailed router can complete the routing.***
    - ***Minimize the critical path delay.***

Measurement of Interconnect Delay

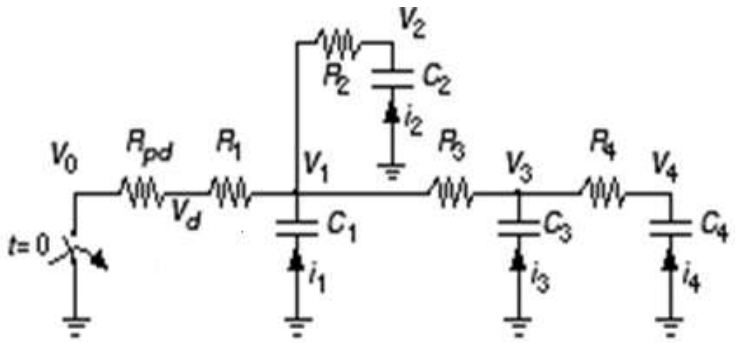
- After placement, the logic cell positions are fixed and the global router can afford to use better estimates of the interconnect delay.
- To illustrate one method, we shall use the **Elmore constant to estimate the interconnect delay for the circuit shown in** Figure 17.3 .



•FIGURE 17.3 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.

The problem is to find the voltages at the inputs to logic cells B and C taking into account the parasitic resistance and capacitance of the metal interconnect. Figure 17.3 (c) models logic cell A as an ideal switch with a pull-down resistance equal to R pd and models the metal interconnect using resistors and capacitors for each segment of the interconnect.

- The Elmore constant for node 4 (labeled V_4) in the network shown in Figure 17.3 (c) i



$$\zeta_4 = \sum_{k=1}^{4} R_{k4} C_k \qquad (17.1)$$

$$= R_{14} C_1 + R_{24} C_2 + R_{34} C_3 + R_{44} C_4 ,$$

- where,

$$R_{14} = R_{pd} + R_1 \text{ (resistance to } V_0 \text{ shared by node 1 and 4)} \qquad (17.2)$$

$$R_{24} = R_{pd} + R_1$$

$$R_{34} = R_{pd} + R_1 + R_3$$

$$R_{44} = R_{pd} + R_1 + R_3 + R_4$$

In Eq. 17.2 notice that R 24 = R pd + R 1 (and not R pd + R 1 + R 2) because R 1 is the resistance to V 0 (ground) shared by node 2 and node 4.

Suppose we have the following parameters (from the generic 0.5 m m CMOS process, G5) for the layout shown in Figure 17.3 (b):

- **m2 resistance is 50 m Ω /square.**
- **m2 capacitance (for a minimum-width line) is 0.2 pFmm –1 .**
- **4X inverter delay is 0.02 ns + 0.5 $C_L$ns (C_L is in picofarads).**
- **Delay is measured using 0.35/0.65 output trip points.**
- **m2 minimum width is 3 λ = 0.9 µm.**
- **1X inverter input capacitance is 0.02 pF (a standard load).**

First we need to find the pull-down resistance, R_{pd} , of the 4X inverter. If we model the gate with a linear pull-down resistor, R_{pd} , driving a load C_L , the output waveform is exp – t /($C_L R_{pd}$) (normalized to 1V).

The output reaches 63 percent of its final value when t = $C_L R_{pd}$, because exp (–1) = 0.63. Then,because the delay is measured with a 0.65 trip point, the constant 0.5 nspF –1 0.5kW is very close to the equivalent pull-down resistance. Thus, **R_{pd} = 500 Ω .**

$$R_1 = R_2 = \frac{(0.1\ \text{mm})\ (50 \infty 10^{-3}\ \Omega)}{0.9\ \mu\ \text{m}} = 6\ \Omega$$

$$R_3 = \frac{(1\ \text{mm})\ (50 \infty 10^{-3}\ \Omega)}{0.9\ \mu\ \text{m}} = 56\ \Omega$$

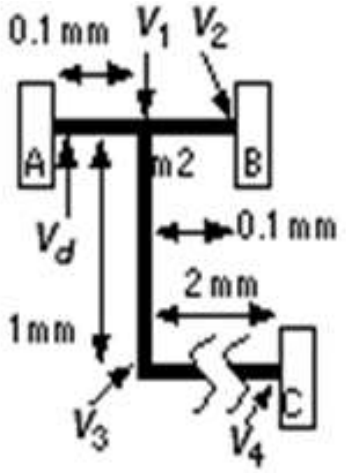
$$R_4 = \frac{(2\ \text{mm})\ (50 \infty 10^{-3}\ \Omega)}{0.9\ \mu\ \text{m}} = 112\ \Omega$$

$$C_1 = (0.1\ \text{mm})\ (0.2 \infty \text{pFmm}^{-1}) = 0.02\ \text{pF}$$

$$C_2 = (0.1\ \text{mm})\ (0.2 \infty \text{pFmm}^{-1}) + 0.02\ \text{pF} = 0.04\ \text{pF}$$

$$C_3 = (1\ \text{mm})\ (0.2 \infty \text{pFmm}^{-1}) = 0.2\ \text{pF}$$

$$C_4 = (2\ \text{mm})\ (0.2 \infty \text{pFmm}^{-1}) + 0.02\ \text{pF} = 0.42\ \text{pF}$$



- **m2 resistance is 50 m Ω square.**
- **m2 capacitance (for a minimum-width line) is 0.2 pFmm –1 .**
- **4X inverter delay is 0.02 ns + 0.5 $C_L$ns (C_L is in picofarads).**
- **Delay is measured using 0.35/0.65 output trip points.**
- **m2 minimum width is 3 λ = 0.9 µm.**
- **1X inverter input capacitance is 0.02 pF (a standard load).**

- $R_1 = R_2 = 6\ \Omega$
- $R_3 = 56\ \Omega$
- $R_4 = 112\ \Omega$
- C 1=0.02 pF
- C 2 =0.04 pF
- C 3=0.2 pF
- C 4=0.42 pF

Now we can calculate the path resistance, R_{ki}, values (notice that $R_{ki} = R_{ki}$):

$$R_{14} = 500\ \Omega + 6\ \Omega = 506\ \Omega$$
$$R_{24} = 500\ \Omega + 6\ \Omega = 506\ \Omega$$
$$R_{34} = 500\ \Omega + 6\ \Omega + 56\ \Omega = 562\ \Omega$$
$$R_{44} = 500\ \Omega + 6\ \Omega + 56\ \Omega + 112\ \Omega = 674\ \Omega \qquad (17.5)$$

Finally, we can calculate Elmore's constants for node 4 and node 2 as follows:

$$\zeta_{D4} = R_{14} C_1 + R_{24} C_2 + R_{34} C_3 + R_{44} C_4 \quad (17.6)$$

$$= (506)(0.02) + (506)(0.04)$$

$$+ (562)(0.2) + (674)(0.42)$$

$$= 425 \text{ ps} .$$

$$\zeta_{D2} = R_{12} C_1 + R_{22} C_2 + R_{32} C_3 + R_{42} C_4 \quad (17.7)$$

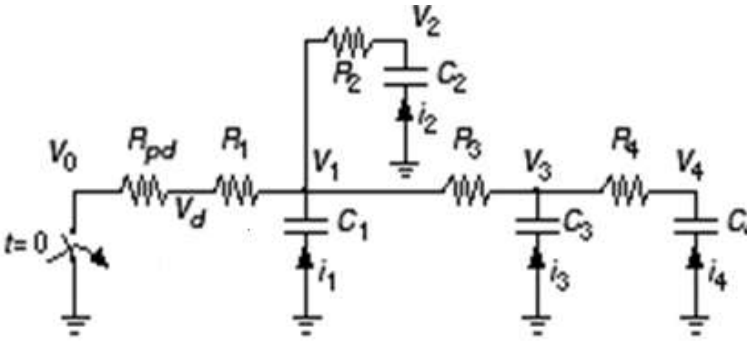
$$= (R_{pd} + R_1)(C_1 + C_3 + C_4)$$

$$+ (R_{pd} + R_1 + R_2) C_2$$

$$= (500 + 6 + 6)(0.04)$$

$$+ (500 + 6)(0.02 + 0.2 + 0.2)$$

$$= 344 \text{ ps} .$$



•and $\zeta_{D4} - \zeta_{D2} = (425 - 344) = 81$ ps.

•A lumped-delay model neglects the effects of interconnect resistance and simply sums all the node capacitances (the lumped capacitance) as follows:

- $\zeta_D = R_{pd} (C_1 + C_2 + C_3 + C_4)$ (17.8)
- $= (500) (0.02 + 0.04 + 0.2 + 0.42)$
- $= 340$ ps

Measurement of delay

The delay of the inverter can be assigned as follows:

- 20 ps (the intrinsic delay, 0.2 ns, due to the cell output capacitance),
- 340 ps (due to the pull-down resistance and the output capacitance),
- 4 ps (due to the interconnect from A to B), (ζ_{D2-} ζ_{D})
- 85 ps (due to the interconnect from A to C) (ζ_{D4-} ζ_{D}).

Measurement of Interconnect Delay (contd.,)

- Even using the Elmore constant we still made the following assumptions in estimating the path delays:
  - **A step-function waveform drives the net.**
  - **The delay is measured from when the gate input changes.**
  - **The delay is equal to the time constant of an exponential waveform that approximates the actual output waveform.**
  - **The interconnect is modeled by discrete resistance and capacitance elements.**

- The global router could use more sophisticated estimates that **remove some of these assumptions**, but there is a limit to the accuracy with which delay can be estimated during global routing

- **When the global router attempts to minimize interconnect delay, there is an important difference between a path and a net.**

- **The path that minimizes the delay between two terminals on a net is not necessarily the same as the path that minimizes the total path length of the net.**

Global Routing Methods

- Many of the methods used in global routing are based on the solutions to the **tree on a graph problem.**

- **sequential routing :**

  One approach to global routing takes each net in turn and calculates the shortest path using tree on graph algorithms—with the added restriction of using the available channels.

Disadvantage:

- *As a sequential routing algorithm proceeds,* ***some channels will become more congested*** *since they hold more interconnects than others.*
- *In the case of* ***FPGAs and channeled gate arrays****, the channels have a fixed channel capacity and can only hold a certain number of interconnects.*

Global Routing Methods (contd.,)

- There are **two different ways** that a global router normally handles this problem.
  1. Order independent Routing
  2. Order dependent Routing

- **Order-independent routing**, a global router proceeds by routing each net, ignoring how crowded the channels are. Whether a particular net is processed first or last does not matter, the channel assignment will be the same.

- **Order-independent routing**, after all the interconnects are assigned to channels, the global router returns to those channels that are the most crowded and reassigns some interconnects to other, less crowded, channels.

- **order dependent** :A global router can consider the number of interconnects already placed in various channels as it proceeds. In this case the global routing is order dependent —the routing is still sequential, but now the order of processing the nets will affect the results.

- Iterative improvement or simulated annealing may be applied to the solutions found from both order-dependent and order-independent algorithms.

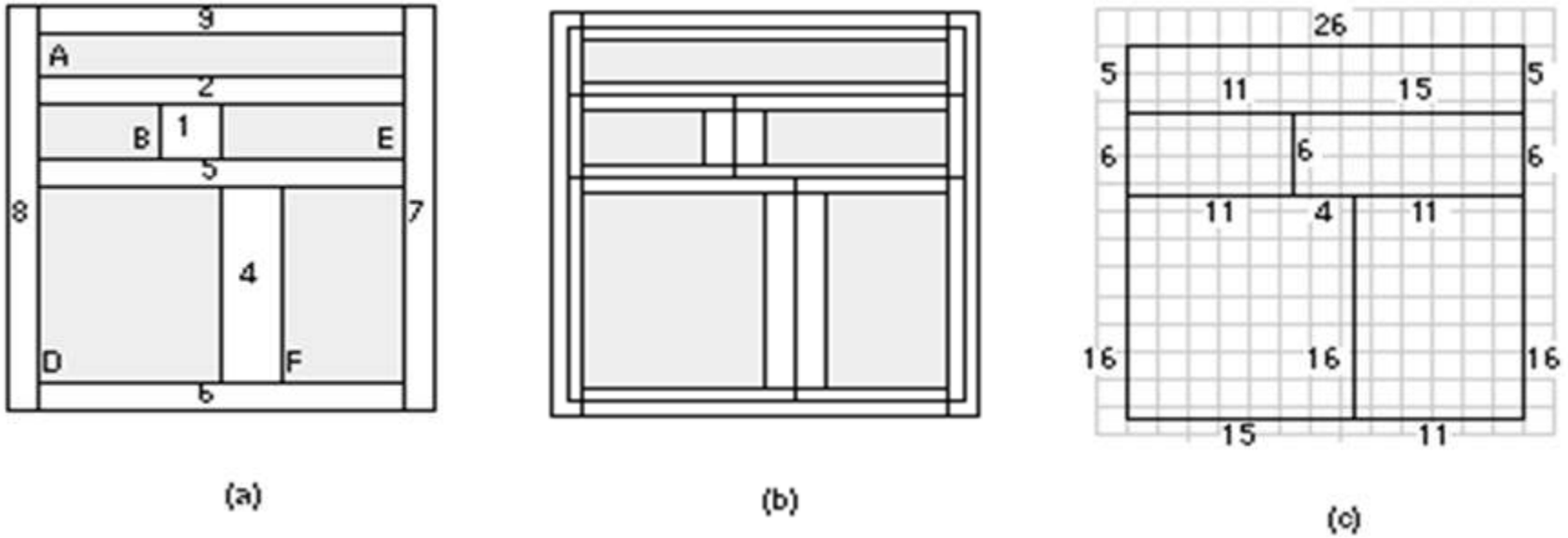
Global Routing Methods (contd.,)

- **Hierarchical routing** handles all nets at a particular level at once.
- Rather than handling all of the nets on the chip at the same time, the global-routing problem is made more tractable by dividing the chip area into levels of hierarchy.
- By considering only one level of hierarchy at a time the size of the problem is reduced at each level.
- There are two ways to traverse the levels of hierarchy.
    - **top-down approach** :- Starting at the whole chip, or highest level, and proceeding down to the logic cells is the.
    - The **bottom-up approach** starts at the lowest level of hierarchy and globally routes the smallest areas first.

Global Routing

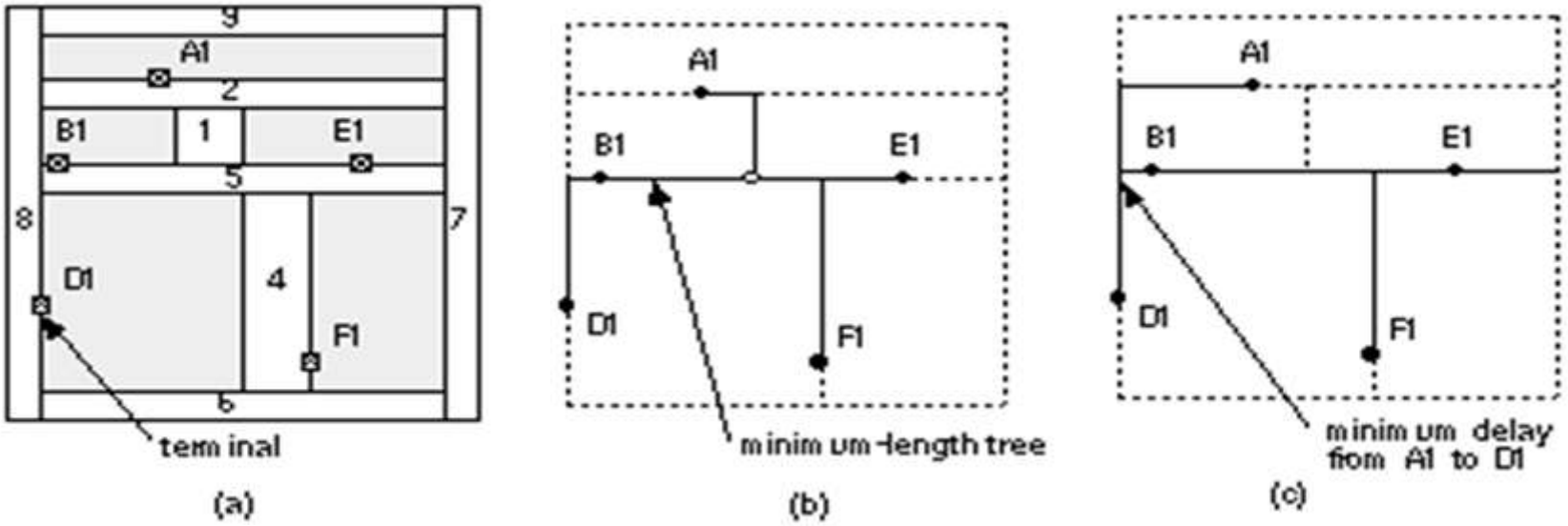
- There are **two types of areas to global route:**
  - ***between blocks***
  - ***inside the flexible blocks***

Global Routing Between Blocks



•FIGURE 17.4 Global routing for a cell-based ASIC formulated as a graph problem. (a) A cell-based ASIC with numbered channels. (b) The channels form the edges of a graph. (c) The channel-intersection graph. Each **channel corresponds to an edge** on a graph whose **weight corresponds to the channel length**.

Global Routing Between Blocks (contd.,)



•FIGURE 17.5 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.

Global Routing Between Blocks (contd.,)

- Global routing is very **similar for cell-based ASICs and gate arrays**, but there is a very important **difference between the types of channels in these ASICs.**

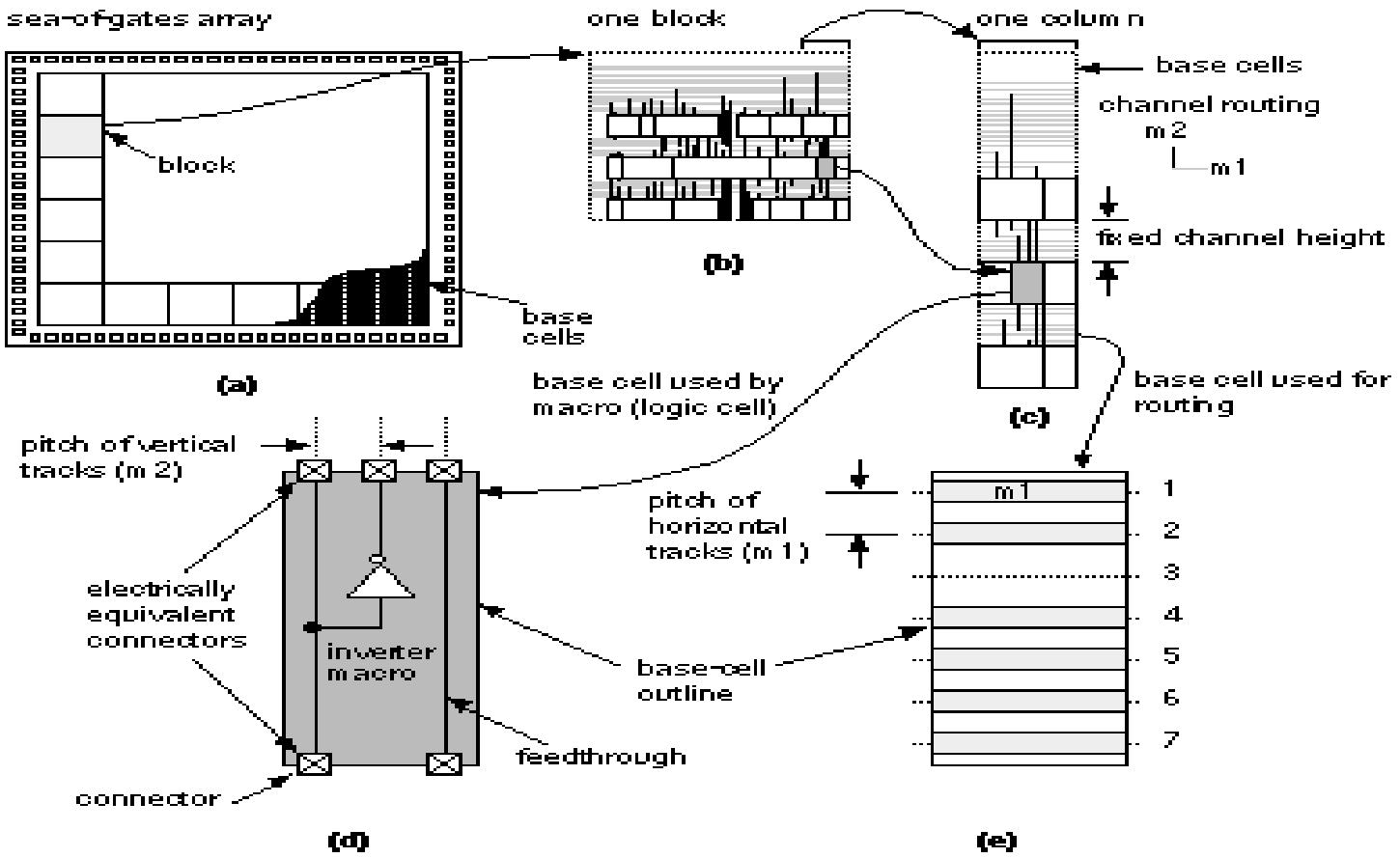
- In **channeled gate-arrays and FPGAs the size, number, and location of channels are fixed.**

- **Advantage - the global router can allocate as many interconnects** to each channel as it likes, since that space is committed anyway.

- **Disadvantage - there is a maximum number of interconnects that each channel can hold.**

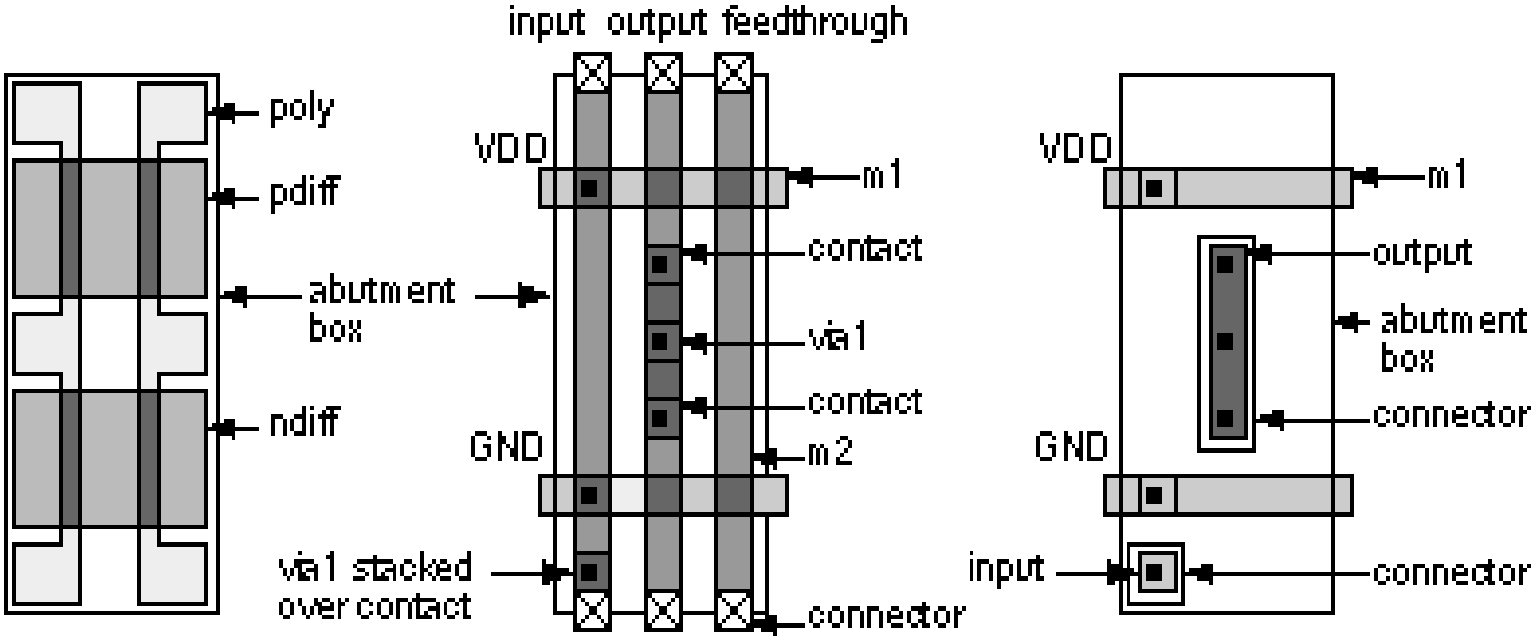
- If the global router needs more room, even in just one channel on the whole chip, the designer has to repeat the placement-and-routing steps and try again (or use a bigger chip).

Global Routing Inside Flexible Blocks



•FIGURE 17.6 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**

Global Routing Inside Flexible Blocks (contd.,)



•FIGURE 17.7 The gate-array inverter from Figure 17.6 d. (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 Inside Flexible Blocks

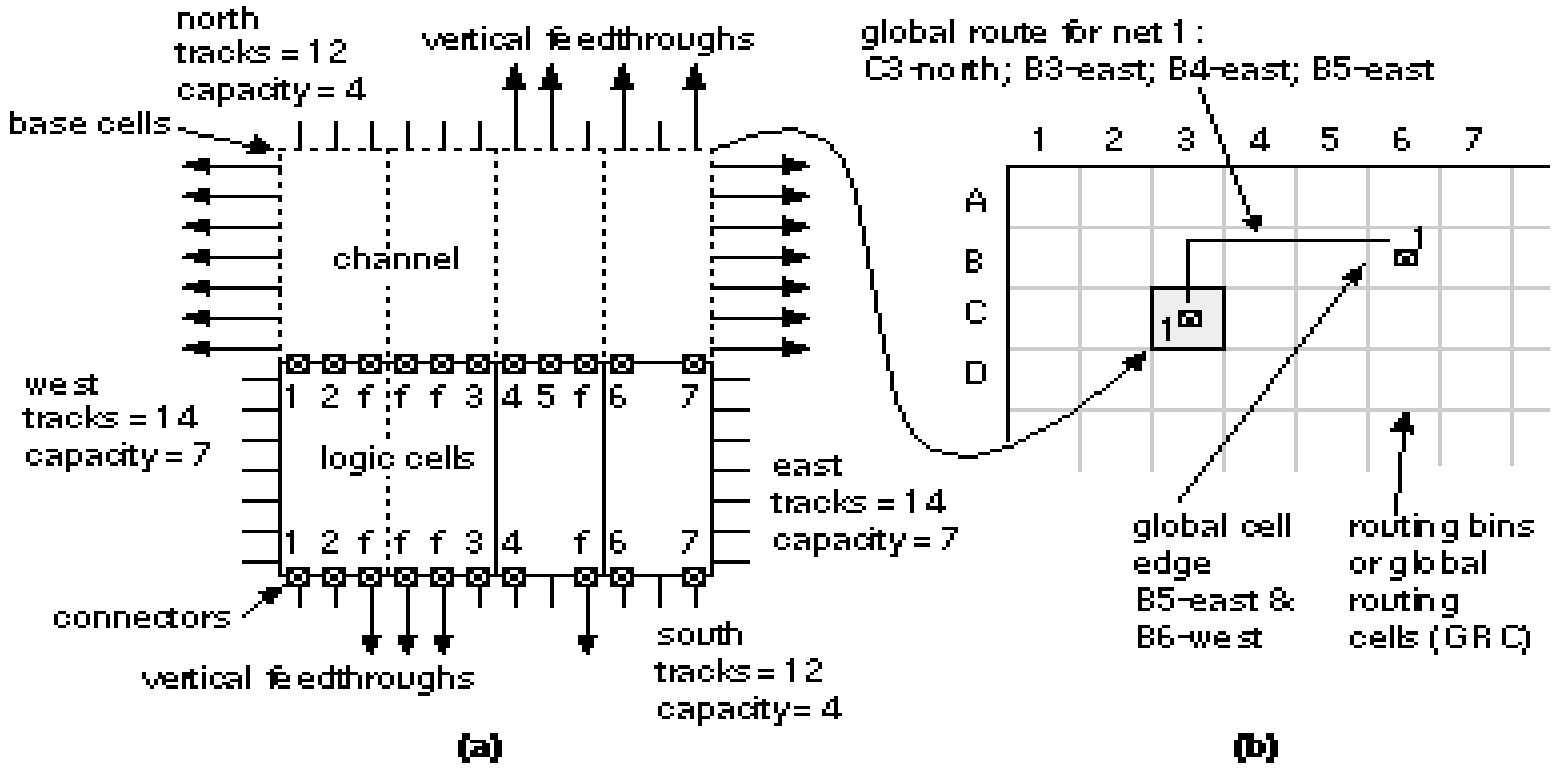


FIGURE 17.8 Global routing a gate array. (a) A single global-routing cell (GRC or routing bin) containing 2-by-4 gate-array base cells. For this choice of routing bin the maximum horizontal track capacity is 14, the maximum vertical track capacity is 12. The routing bin labeled C3 contains three logic 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.

Timing-DrivenMethods

- As in timing-driven placement, there are two main approaches to timing-driven routing:
  - **net-based and path-based.**

- **Path-based methods are more sophisticated**.

  For example, if there is a critical path from logic cell A to B to C, the global router may increase the delay due to the interconnect between logic cells A and B if it can reduce the delay between logic cells B and C.

- **Placement and global routing tools may or may not use the same algorithm to estimate net delay.** If these tools are from different companies, the algorithms are probably different.

- The algorithms must be compatible, however. **There is no use performing placement to minimize predicted delay if the global router uses completely different measurement methods.**

- **Companies that produce floorplanning and placement tools make sure that the output is compatible with different routing tools**—often to the extent of using different algorithms to target different routers.

Back-annotation

- The global router can give **not just an estimate of the total net length** (which was all we knew at the placement stage), but **the resistance and capacitance of each path in each net.** This **RC information** is used to calculate **net delays.**

- **Back-annotate this net delay information**
  - **to the synthesis tool** for in-place **optimization** or
  - to a **timing verifier** to make sure **there are no timing surprises**.

- **Differences in timing predictions** at this point **arise due to the different ways in which the placement algorithms estimate the paths and the way the global router actually builds the paths.**

Detailed Routing

Goal:

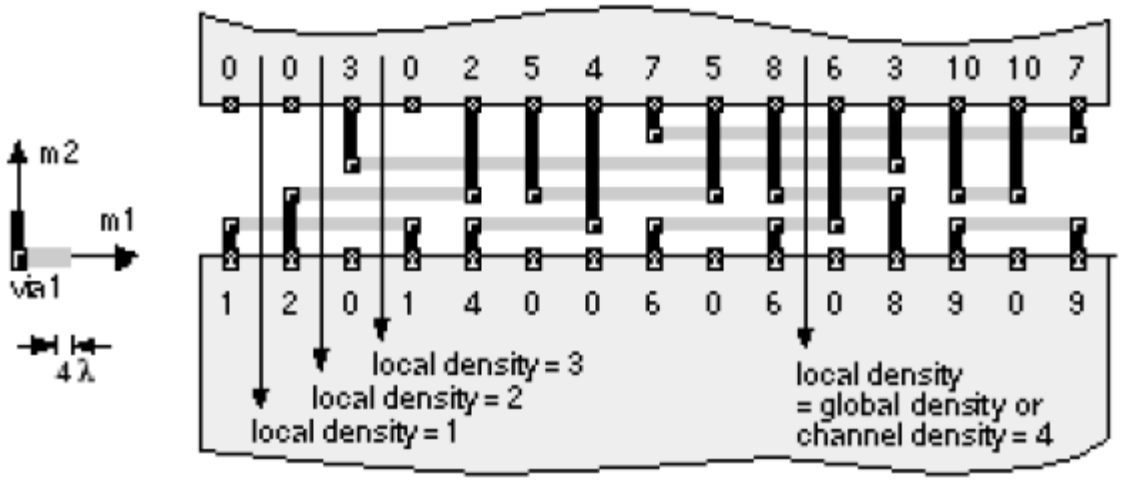
- The goal of detailed routing is to **complete all the connections between logic cells.**

Objectives:

- The most common objective is to minimize one or more of the following:
  - The total **interconnect length and area**
  - The **number of layer changes** that the connections have to make
  - The **delay of critical paths**

- Minimizing the number of layer changes corresponds to minimizing the number of vias that add parasitic resistance and capacitance to a connection.

Measurement of Channel Density
Definition of Local and Global channel density



- **Maximum local density of channel is Global density**
- **Channel density is less than or equal to Channel capacity.**

Left-edge algorithm

The **left-edge algorithm (LEA) is the basis for several routing algorithms [** Hashimoto and Stevens, 1971].

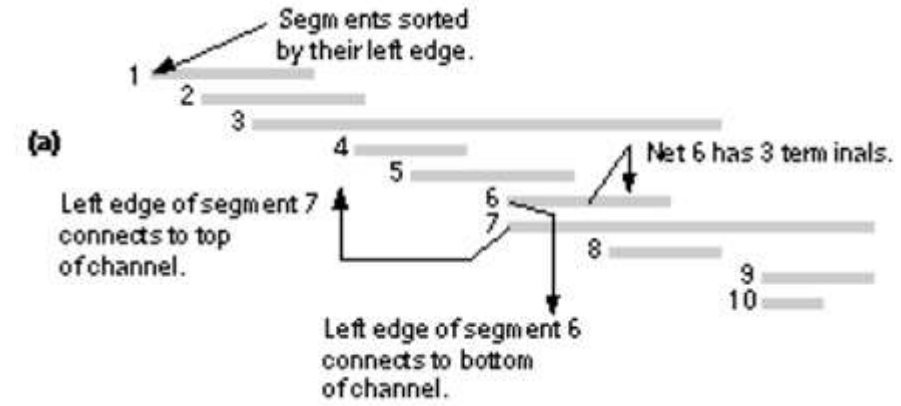
The LEA applies to two-layer channel routing, using one layer for the trunks and the other layer for the branches.

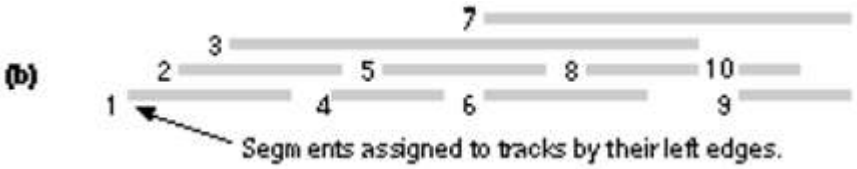
For example, m1 may be used in the horizontal direction and m2 in the vertical direction.

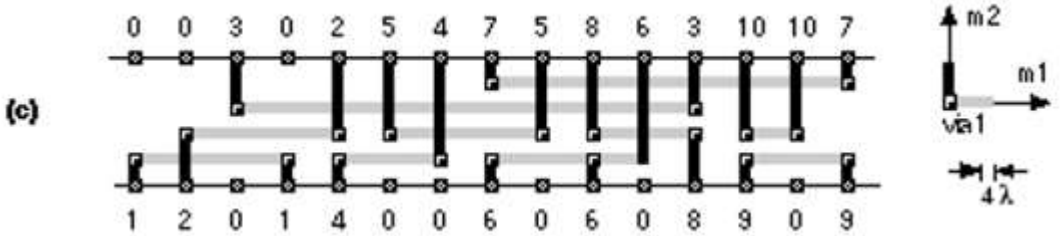
The LEA proceeds as follows:

1. Sort the nets according to the leftmost edges of the net's horizontal segment.
2. Assign the first net on the list to the first free track.
3. Assign the next net on the list, which will fit, to the track.
4. Repeat this process from step 3 until no more nets will fit in the current track.
5. Repeat steps 2–4 until all nets have been assigned to tracks.
6. Connect the net segments to the top and bottom of the channel.

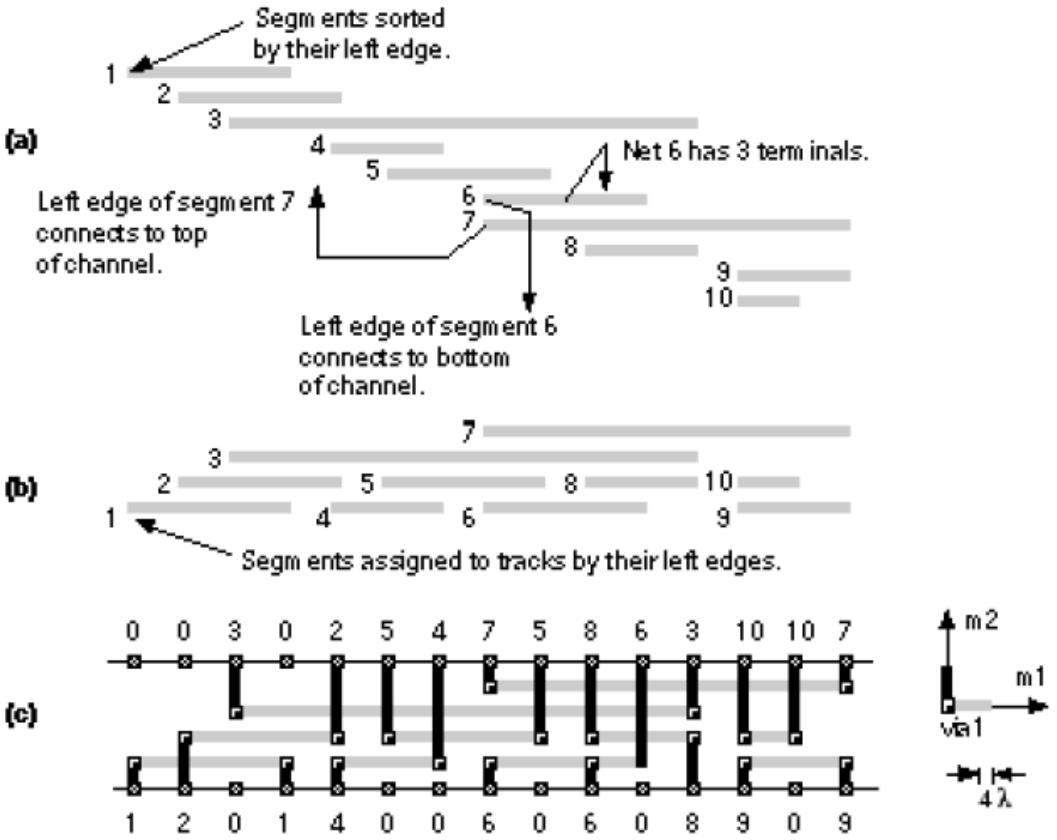
Left-edge algorithm





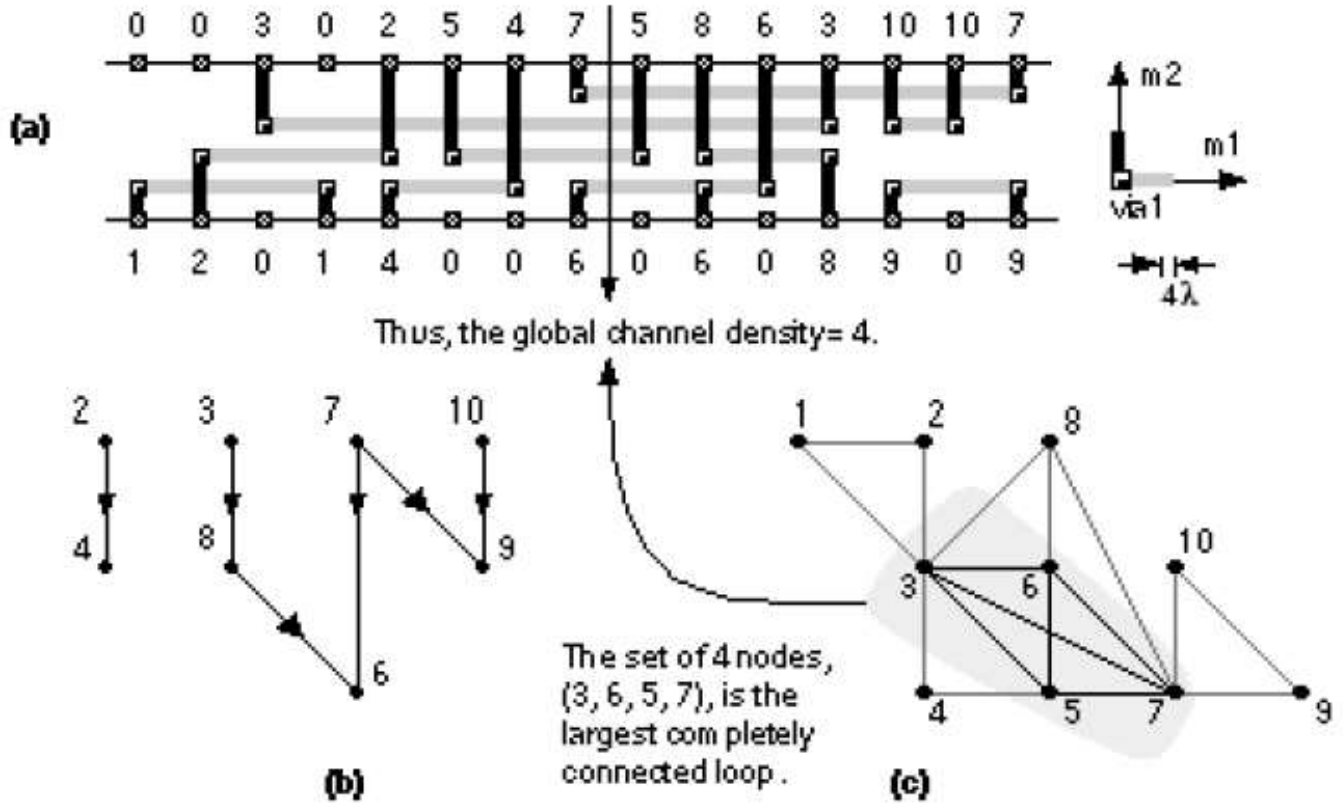


Left-edge algorithm

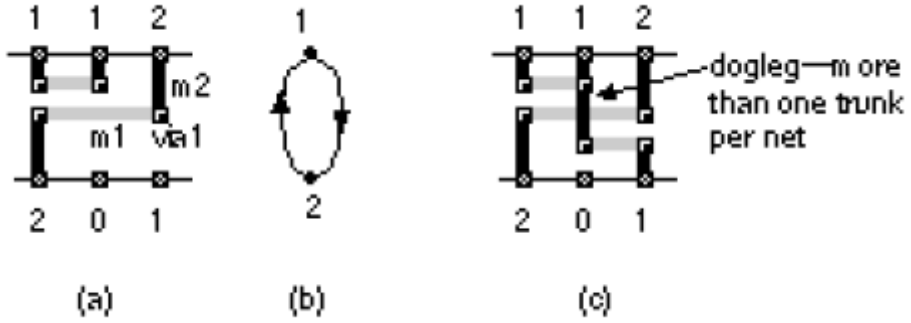


Constraints and Routing Graphs

- Two terminals that are in the same column in a channel create a **vertical constraint .**
- Overlap between the trunks of nets is called **horizontal constraint.**



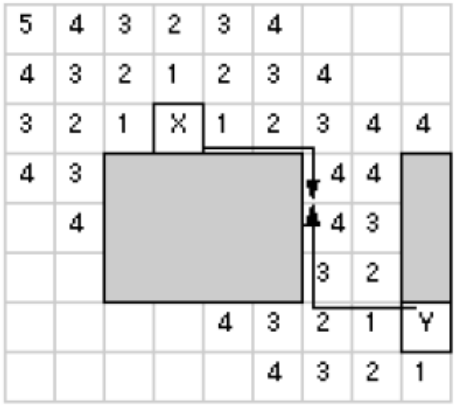
Dog-Leg router



- A **dogleg router removes the restriction that each net** can use only one track or trunk.

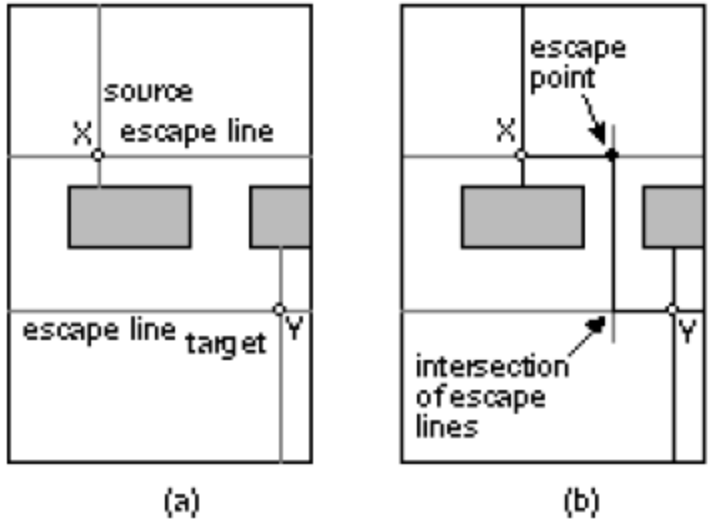
Area Routing Algorithm- Lee-Maze algorithm

[For general shaped areas]



- **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, choose the bin that avoids changing direction). (The original form of the Lee algorithm uses a single wave.)

Hightower or line search-Area routing algorithm [For general shaped areas]



• 1. Extend lines from both the source and target toward each other.

•2. When an extended line, known as an **escape line , meets an obstacle, choose a** point on the escape line from which to project another escape line at right angles to the old one. This point is the **escape point .**

Special routing- CLK routing

- **Gate arrays** normally use a **clock spine (a regular grid)**, eliminating the need for special routing.
- The **clock distribution grid** is designed at the same time as the gate-array base to **ensure a minimum clock skew and minimum clock latency**—given power dissipation and clock buffer area limitations.

- **Cell-based ASICs** may use either a **clock spine, a clock tree, or a hybrid approach.**
- Figure shows how a clock router may **minimize clock skew in a clock spine** by making the **path lengths, and thus net delays, to every leaf node equal—using jogs in the interconnect paths if necessary.**

- More sophisticated clock routers perform **clocktree synthesis (automatically choosing the depth and structure of the clock tree) and clock-buffer insertion (equalizing the delay to the leaf nodes by balancing** interconnect delays and buffer delays).

Special routing- CLK routing

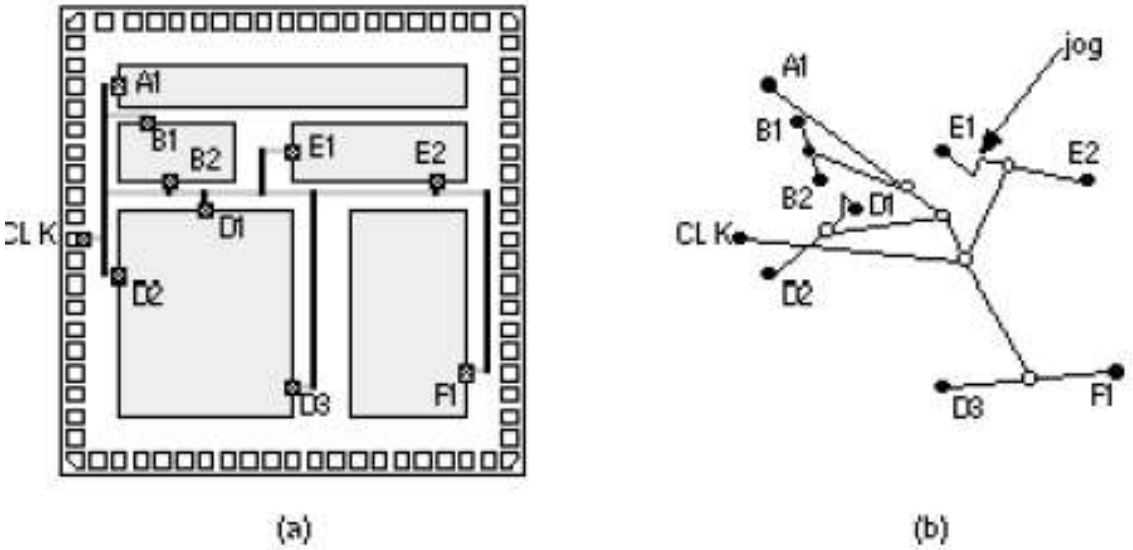


FIGURE: Clock routing. (a) A clock network for the cellbased ASIC

(b) **Equalizing the interconnect segments between CLK and all destinations** (by **including jogs if necessary**) **minimizes clock skew.**

Special routing- Power routing

- **Power bus width**
- Each of the power buses has to be **sized according to the current it will carry.**
- **Too** much current in a power bus can lead to a failure through a mechanism known as **electromigration.**
- **The required power-bus widths can** be estimated automatically from library information, from a separate **power simulation tool, or by entering the power-bus widths to the routing software by hand.**
- Many **routers use a default power-bus width** so that it is quite easy to complete routing of an ASIC without even knowing about this problem.

Special routing- Power routing

- **Gate-Array ASIC**

- **Gate arrays** normally use a regular **power grid as part of the gate-array base.**

- **The** gate-array logic cells contain two fixed-width power buses inside the cell, running horizontally on m1.

- The **horizontal m1 power buses are then strapped in a vertical direction by m2 buses, which run vertically across the chip.**

Special routing- Power routing

- **Cell-based ASIC**
- **Standard cells** are constructed in a similar fashion to **gate-array cells**, **with power buses running horizontally in m1 at the top and bottom of each cell.**
- A **row of standard cells** uses **end-cap cells that connect to the VDD and VSS power buses** placed by the power router.
- **Power routing of cell-based ASICs** may include the option **to include vertical m2 straps at a specified intervals.**
- In a three-level metal process, power routing is similar to two-level metal ASICs. **Power buses inside the logic cells are still normally run on m1**. **Using HVH routing** it would be possible to run **the power buses on m3 and drop vias all the way down to m1** when power is required in the cells.

Circuit Extraction

- After detailed routing is complete, **the exact length and position of each interconnect for every net is known.**
- Now **the parasitic capacitance and resistance associated with each interconnect, via, and contact** can be calculated.
- This data is generated by a **circuit-extraction tool in one of the formats.**
- **standard parasitic format (SPF)**
- The **standard parasitic format (SPF)** describes interconnect delay and loading due to parasitic resistance and capacitance**.**
- There are three different forms of SPF:
  - Two of them (**regular SPF** and **reduced SPF) contain the same information, but in different formats, and model the behavior of interconnect**;
  - Third form of SPF (**detailed SPF) describes the actual parasitic**

Circuit Extraction

- **The load at the output of gate A is represented by one of three models: lumped-C, lumped- RC, or PI segment.**

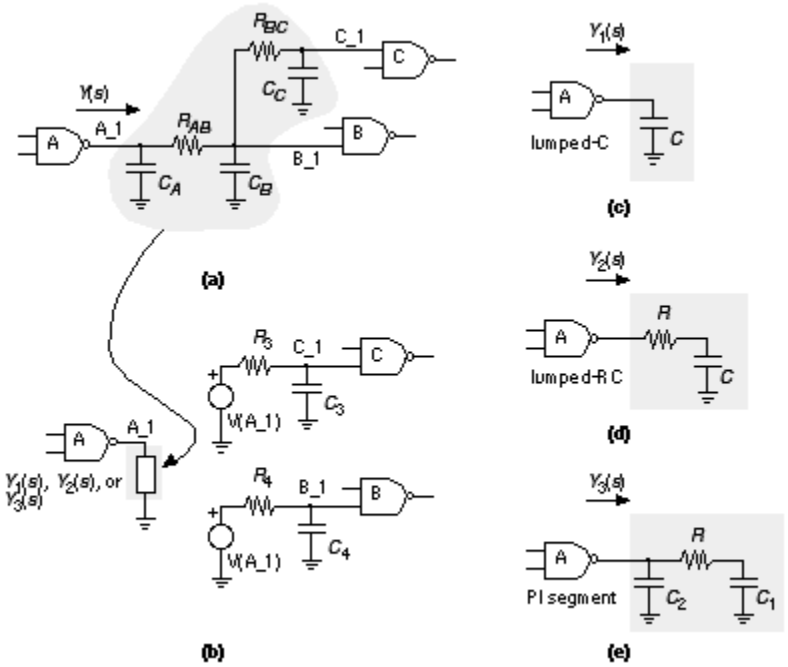


Figure: **The regular and reduced standard parasitic format (SPF) models for interconnect.** (a) An example of an interconnect network with fanout. The driving-point admittance of the interconnect network is *Y (s). (b) The* ***SPF model of*** **the interconnect.** (c) The **lumped-capacitance interconnect model.** (d) The **lumped-RC interconnect model.** (e) The **PI segment interconnect model** .

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).*

Circuit Extraction

The key features of regular and reduced SPF are as follows:

- The **loading effect of a net** as seen by the driving gate is represented by choosing one of three different RC networks: lumped-C, lumped-RC, or PI segment (selected when generating the SPF) [O'Brien and Savarino, 1989].
- The **pin-to-pin delays of each path in the net** are modeled by a simple RC delay (one for each path). This can be the Elmore constant for each path, but it need not be.
- The **reduced SPF (RSPF) contains the same information as regular SPF, but uses the** SPICE format.
- **Detailed SPF:**
- The **detailed SPF (DSPF) shows the resistance and capacitance of each segment in a** net, again in a SPICE format. There are no models or assumptions on calculating the net delays in this format.

Design-Rule Check (DRC)

- ASIC designers perform **two major checks before fabrication.**
- **DRC:**
- The first check is a **design-rule check (DRC) to ensure that nothing has gone wrong in the process of assembling the logic cells and routing.**
- The DRC may be performed at two levels.
- **Phantom-Level DRC:**
- The first level of DRC is a **phantom-level DRC , which checks for shorts, spacing violations, or other** design-rule problems between logic cells.
- This is principally **a check of the detailed router.**
- If the real library-cell layouts (sometimes called **hard layout) can be accessed, we can instantiate the phantom cells and perform a second-level DRC at the** transistor level.

Design-Rule Check (DRC)

- **Dracula check:**

- This is principally **a check of the correctness of the library cells.**
- Normally **the ASIC vendor will perform this check** using its own software as a type of incoming inspection.
- The **Cadence Dracula software** is one de facto standard in this area, and you will often hear reference to a **Dracula deck that consists of the Dracula code** describing an ASIC vendor's design rules.
- Sometimes ASIC vendors will give their Dracula decks to customers so that **the customers can perform the DRCs themselves**.

Design-Rule Check (DRC)

- **Layout Vs Schematic check:**
- To ensure that **what is about to be committed to silicon is what is really wanted.**
- An electrical schematic is extracted from the physical layout and compared to the netlist.
- This closes a loop between the logical and physical design processes and ensures that both are the same.
- The LVS check is not as straightforward as it may sound, however.

Design-Rule Check (DRC)

- **Problems in LVS check:**
- The **first problem is transistor-level netlist** for a large ASIC forms an **enormous graph.**
- LVS software essentially has to **match this graph against a reference graph** that describes the design**.**
- Ensuring that **every node corresponds exactly to a corresponding element in the schematic (or HDL code) is a very difficult task.**
- The first step is normally to match certain key nodes (such as the power supplies, inputs, and outputs), but the process can very quickly become bogged down in the **thousands of mismatch errors that are inevitably generated initially.**

Design-Rule Check (DRC)

- **Problems in LVS check:**
- The second problem with an LVS check is **creating a true reference.**
- The **starting point may be HDL code or a schematic.**
- Logic synthesis, test insertion, clock-tree synthesis, logical-to-physical pad mapping, and several **other design steps each modify the netlist.**
- The **reference netlist may not be what we wish to fabricate.**
- In this case designers increasingly **resort to formal verification that extracts a Boolean description of the function of the layout and compare that to a known good HDL description.**

Thank you