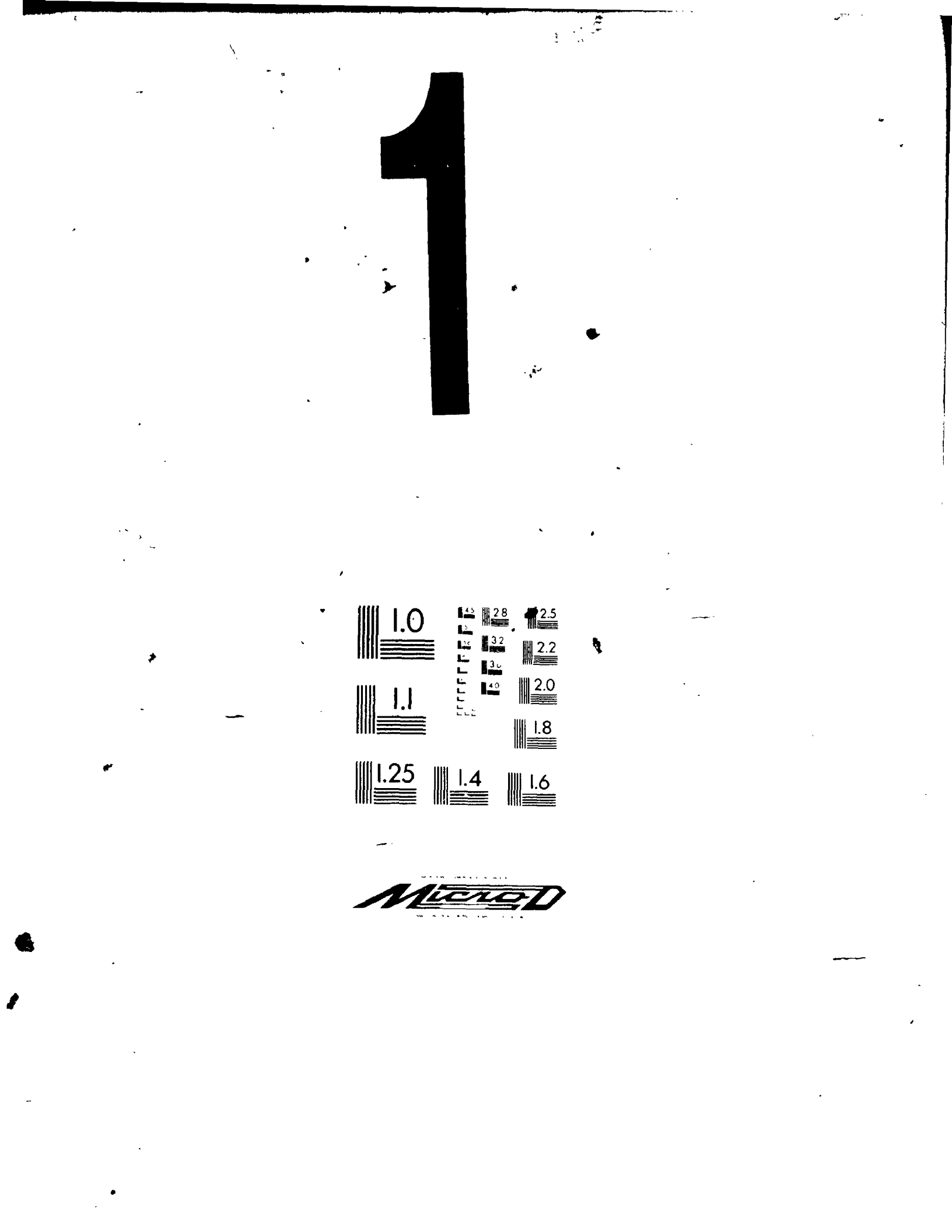

1
1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
MicroD

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, tests publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

Improving Interconnect and Register Allocation for the Behavioral Synthesis of Digital Circuits

by

JENNY MIDWINTER, B.ENG.

A Thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfilment of the requirements
for the degree of
Master of Engineering

Ottawa-Carleton Institute for Electrical Engineering
Faculty of Engineering
Department of Systems and Computer Engineering
Carleton University

March, 1988

ISBN 0-315-40639-9

The undersigned hereby recommends to the Faculty of Graduate Studies and Research acceptance of the thesis,

"Improving Interconnect and Register Allocation for the Behavioral Synthesis of Digital Circuits"

submitted by JENNY MIDWINTER, B.ENG., in partial fulfillment of the requirements for the degree of M.ENG.

John Knight
Thesis Supervisor

Chairman,
Department of Systems and
Computer Engineering

Department of Systems and Computer Engineering

Faculty of Engineering

Carleton University

April 1988

Abstract

A method of improving register and interconnect allocation in a register-transfer digital circuit synthesis system is described.

Weight-directed techniques for assigning registers and for assigning interconnect are developed. The weights are chosen such that the interconnect (buses and multiplexers) is minimized. Another weight-directed approach, that includes placement information, is introduced. All the developed techniques are of the order of N^2, where N is the number of data transfers, or variables, to be assigned. A cost function, which includes mux, bus, lead, and control costs, is developed and can be used to evaluate pre-layout interconnections.

These methods are suitable for use in a register-transfer synthesis system, or by designers, as stand-alone procedures, to optimize their circuits.

Acknowledgements

This thesis would not have been possible without the help of my supervisor, Dr. John Knight. His willingness to always make time in his busy schedule to discuss any aspect of this thesis was greatly appreciated. I especially thank him for his expert guidance that kept me focussed while still permitting me to explore things on my own.

The inspiration to try a weight-directed approach to register and interconnect allocation came from the work of Dr. Pierre Paulin on force-directed scheduling. I would like to thank him for our numerous discussions which provided me with many new ideas and insights.

For advice on many digital design issues, I often turned to Richard Griffiths. His encyclopedic knowledge and willingness to discuss any topic was much appreciated.

My fiance, Kevin Stille, was a constant source of optimism. I would like to thank him for his moral support and patience. Most of all, I would like to give him special thanks for all those "little things" that are too numerous to mention here.

Finally, I would like to thank my parents. They have always encouraged me in my academic pursuits and have always provided me with an environment conducive to learning.

Table of Contents

Chapter 1
Introduction

Since the development of the first integrated circuit in the 1950's, this device has become an essential element in today's advanced technology products. Integrated circuits (IC's) perform a wide range of functions, from monitoring the performance of cars to guiding missiles.

Due to advances in very large scale integration (VLSI) over the last decade, integrated circuits with almost a million transistors can now be fabricated. This new technology has created tremendous problems in the area of VLSI circuit design, as the complexity of creating these IC's has risen dramatically. Unfortunately, the IC designer's ability to keep track of evolving designs has not been able to keep pace with this increased complexity. This thesis attempts to improve this situation by:

1) Improving one aspect of automatically designed circuits,

2) Providing designers with a stand-alone procedure that could optimize their designs, with respect to interconnect and registers, given its register transfer description. This area of improvement is one that is often neglected by IC designers.

1.1 IC Design Domains

There are three domains in which an IC design can be represented:

- functional (behavioral),
- structural,
- geometric.

This has been neatly illustrated in [Gasj85] with the *Y-chart*, shown in Figure 1.1. Each axis in the Y represents one of these domains, while the *level* of abstraction is determined by its distance from the vertex - the farther from the vertex, the greater the abstraction.

The functional, or behavioral, domain describes what functions the circuit is to perform. In this domain the design should be considered as a *black box*, with a specified set of inputs, a set of outputs, and a set of functions describing the desired behavior of each output as a function of the inputs and time. For instance, at the *logic* level of abstraction a design may be represented by a Boolean equation and a statement of when the output should be available with respect to the inputs (Figure 1.2(a)).

The structural domain represents a one-to-many mapping of a functional representation onto a set of components and connections under such constraints as cost, area, and time. It serves as a *bridge* between the functional and the geometric dimensions. For example, a structural design at the logic level may be represented by a set of connected logic gates such as NAND's, or OR's (Figure 1.2(b)).

The geometric domain describes the design with respect to space, or silicon. It is responsible for capturing the placement and geometry of the design. For instance, the geometric design, at the logic level, will show the location and orientation of the gates and the interconnect (Figure 1.2(c)).

1.2 VLSI Design Process

VLSI design involves mapping a problem specification into mask descriptions for circuit fabrication. The goal of this process is to generate, in limited time and resources, an acceptable design with respect to:

- performance (speed of operation),
- chip area or package count,
- pin count,

Figure 1.1: **Y-CHART FOR VLSI DESIGN [Gajs85]**

The "Y-Chart" illustrates the three domains in which an IC design can be represented. Each domain, functional, structural, geometric, is represented by a different axis. The "level" of abstraction, of a representation, is determined by its distance from the vertex- the further from the vertex, the greater the abstraction.

The Y-Chart will be used to show the types of transformations that occur during the VLSI design process.

LEGEND:
- System Level
- Register Transfer Level
- Logic Level
- Circuit Level

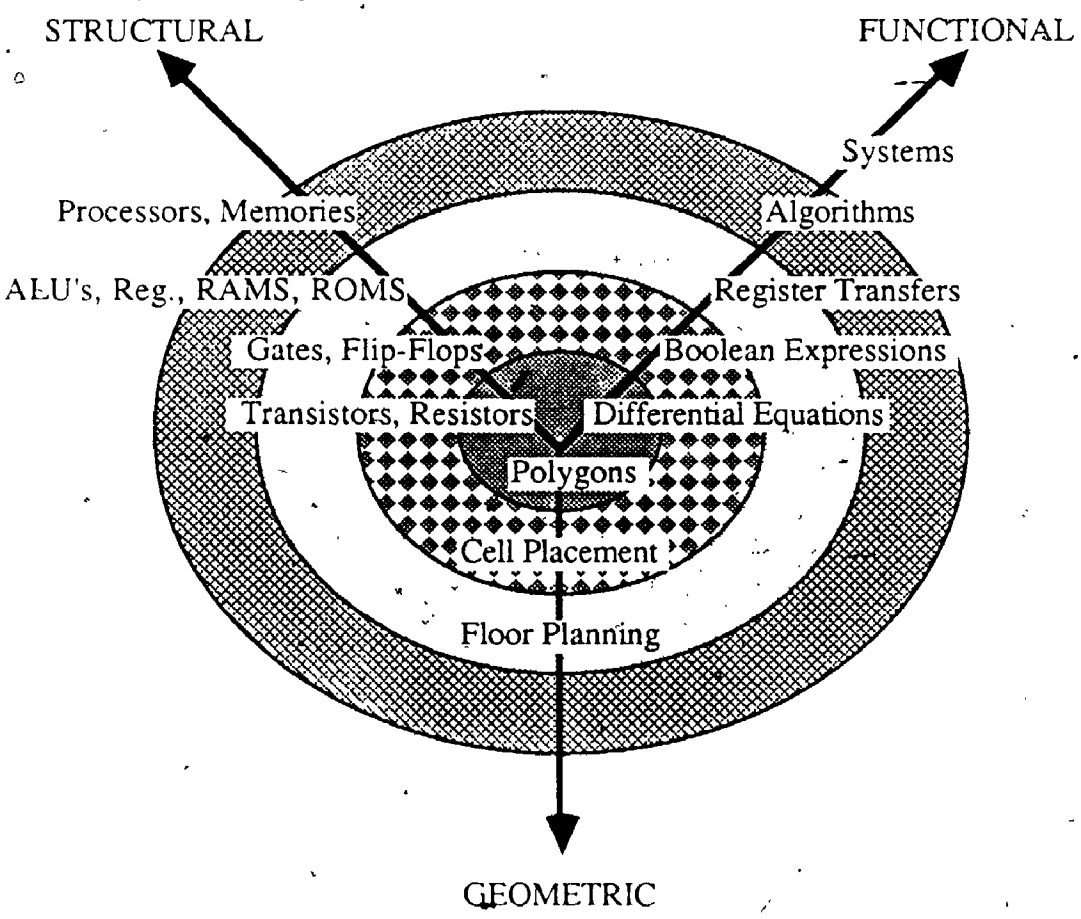


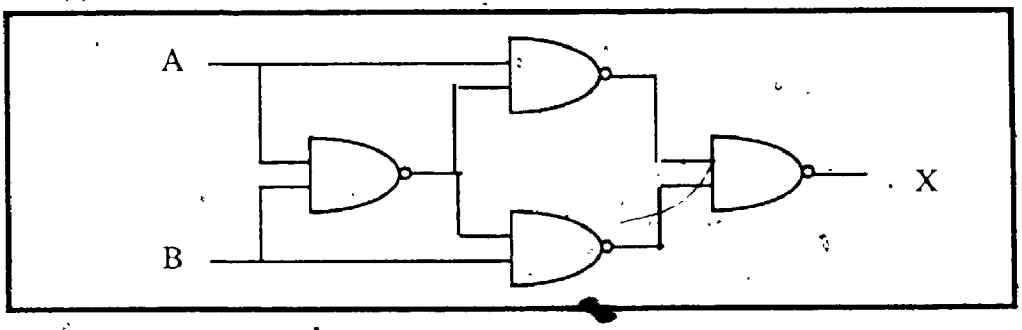
Figure 1.2: **REPRESENTATIONAL DOMAINS [Gajs85]**

This is an example of the three VLSI design domains at the logic "level". The functional domain (a) is represented by a boolean expression, while the same circuit is represented by a logic diagram (b) in the structural domain. Finally, the geometric domain captures the placement of the logic gates (c).

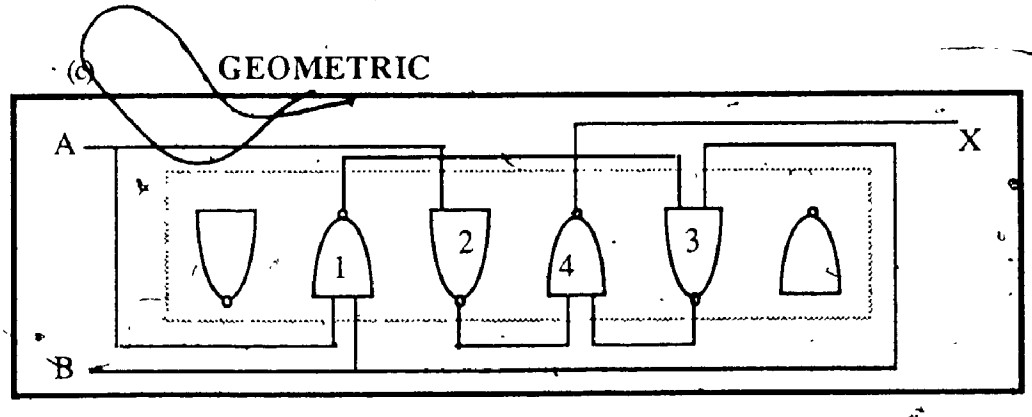
(a) **FUNCTIONAL**

$X = A'B + AB'$ after 30 ns.

(b) **STRUCTURAL**



(c) **GEOMETRIC**



- power consumption,
- design time,
- reliability,
- testability.

The design process requires a hierarchical, or *top-down*, approach to manage the massive amount of information. This approach involves partitioning and transforming the design from one level of representation to another (Figure 1.3). However, each representation may not always map neatly onto another lower level, or into one of different domain. All situations of incompatibility that might develop between levels can not be foreseen. The result of this phenomenon is that the designer must often backtrack to a previous level, or even domain, to where a critical design decision had been made, and pursue another design path. This results in an iterative design process .

Generally, production IC's will be designed in four major *phases* (Figure 1.3):

- specification, and algorithm design,
- logic design,
- physical design,
- test-set design.

1.2.1 Specification and Algorithm Design

Specification and algorithm design are the first two steps in the VLSI design process (Figure 1.3). During the specification step the desired circuit is defined. This will include:

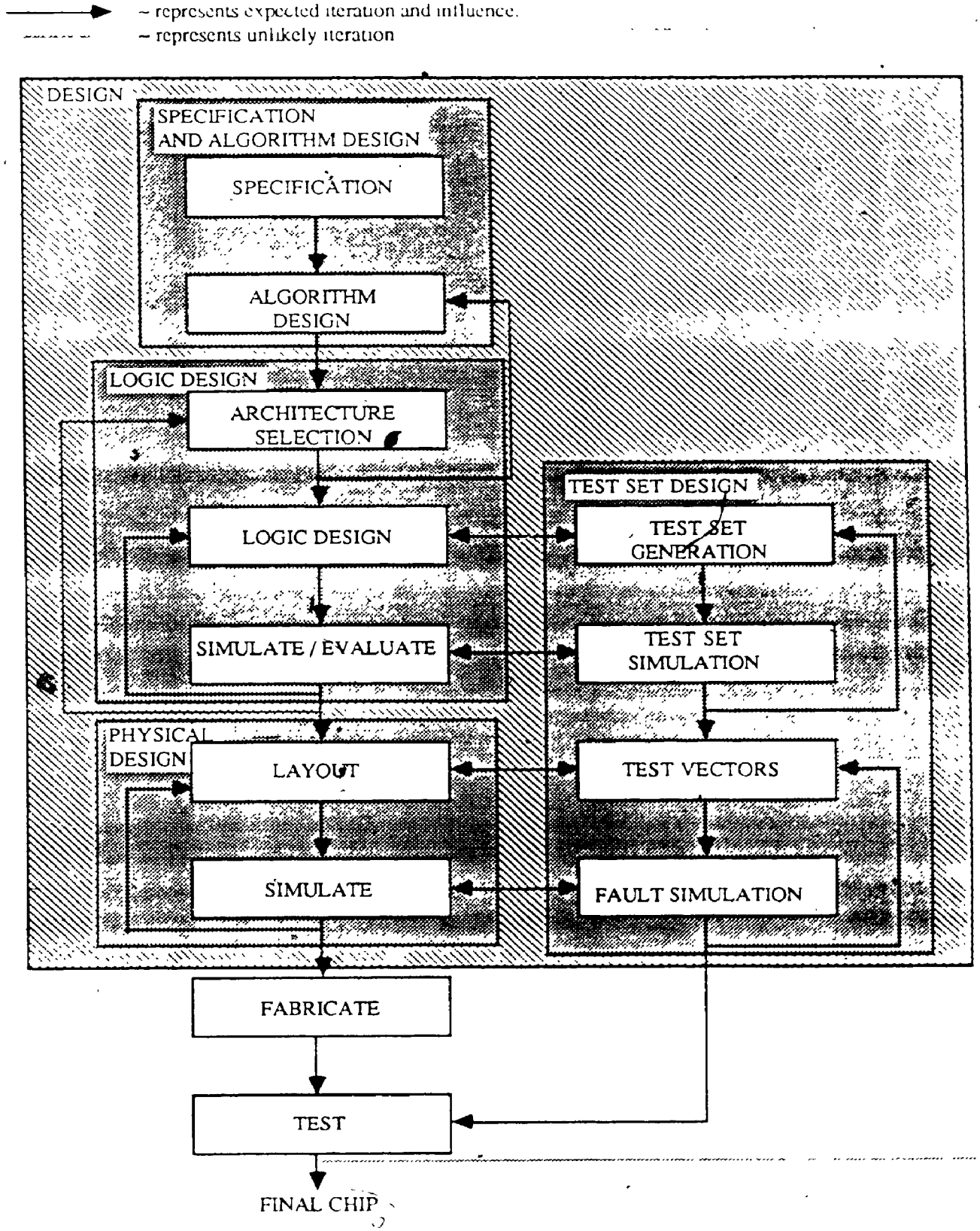
(a) A description of what the circuit should do,

(b) A list of constraints on the design, such as:

- time delays for and between significant events,
- area or package count upper bounds,

Figure 1.3 **VLSI DESIGN PHASES**

This figure illustrates the four major "phases" of the VLSI design process, and the iteration that will occur during and between these "phases"



- maximum number of pins,
- upper limit on power consumption,
- lower limit on reliability (mean time between failure),
- lower limit on testability (test coverage).

An example of a specification might be:

> Design a 16-pin IC to sort eight 8-bit numbers, with the output being available after 1 micro-second.

During the algorithm design *step*, a method of, or an algorithm for, satisfying the specification is developed. The algorithm may be represented by a computer program. For instance, using the sort example, specified above, at least two algorithms are possible: a *bubble sort* and an *insertion sort* (Figure 1.4).

1.2.2 Logic Design Phase

During the logic design *phase*, three activities (*steps*) occur (Figure 1.5):

- architecture selection,
- logic design,
- simulation.

The first step, architecture selection, involves developing, or selecting, an appropriate architecture to implement the algorithm. There may be some iteration between the architecture selection and algorithm design steps, as shown earlier in Figure 1.5. This is due to the fact that the algorithm, itself, will have an impact on the style of architecture (and vice versa), which will in turn affect the performance of the circuit. For instance, in the sort example, the pin limitation in the specification (16 pins) may imply that a bit-serial architecture be used (i.e.- the input of 8 numbers will be serial). Thus, the *bubble-sort* algorithm would probably be more appropriate, as it is most easily implemented in this architectural style (Figure 1.6).

The second step, logic design, transforms the architectural description to a gate *level* schematic (Figure 1.7). This will then be simulated to verify that it operates

Figure 1-4: Two Sort Algorithms

Shown, below, are two different algorithms that can perform sort operations. The algorithm will have an impact on the type of architecture that will be chosen (see Figure 1.6).

N = Number of numbers to sort

Array = Numbers to be sorted

(a) Bubble Sort Algorithm

```
FOR J=N-1,0 DO
 FOR I=0,J-1 DO
   IF Array(I) > Array(I+1) THEN
        Temp = Array(I)
        Array(I) = Array(I+1)
        Array(I+1) = Temp
   END-IF
 END-FOR
END-FOR
```

(b) An Insertion Sort Algorithm

```
FOR J=N-1,0 DO
   Array(0) = Array (I+1)
   WHILE Array(J) > Array(0) DO
        Array(J+1) = Array(J)
        J = J-1
   END-WHILE
   Array(J+1) = Array(0)
END-FOR
```

Figure 1.5: **LOGIC DESIGN "PHASE"**

The logic design phase encompasses three steps:
- Architecture Selection,
- Logic Design,
- Simulation.

The architecture selection step moves the design from a functional representation to a structural one, where the design will be defined in terms of logical blocks. Because the algorithm has an impact on the style of the architecture, which in turn will effect the performance, there will be some iteration between these two representations, as shown by the shaded line.

The logic design step transforms the architectural description to a gate level schematic.

Finally, the gate level circuit is simulated to verify that it operates correctly, and that it conforms to the design constraints. If the gate level design is unsatisfactory, then a new circuit may be designed, a different architecture may be selected, or another algorithm may even be developed, as shown by the shaded lines.

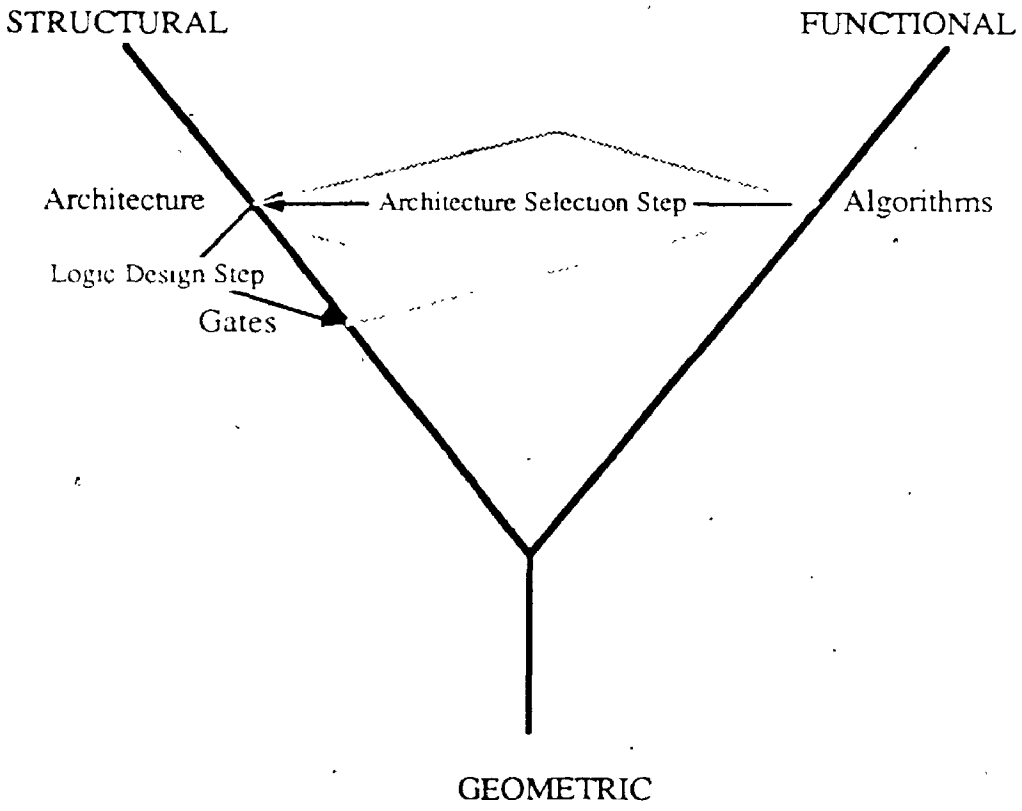
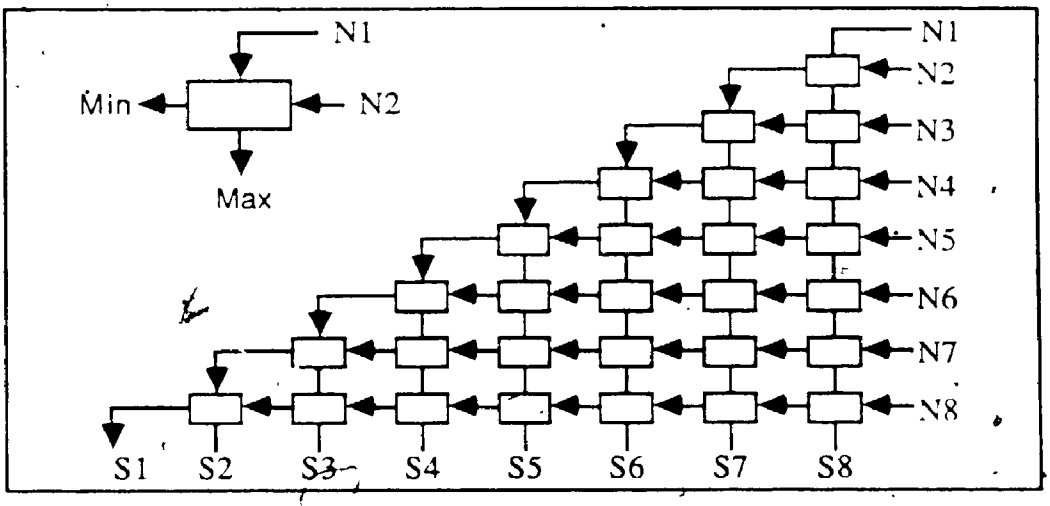


Figure 1.6 **BIT SERIAL ARCHITECTURE FOR 'BUBBLE SORT'**

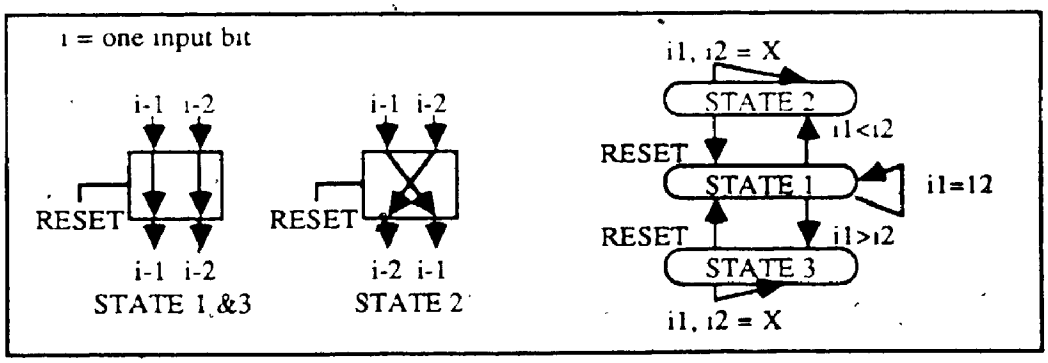
The 'bubble sort' algorithm can be implemented in a bit-serial fashion by using a matrix of 28 direction latch comparators(DLC's) to sort 8 two's-complement numbers. Each column of the matrix forces the largest number, of that column, to the bottom. The numbers must be input in unit delay intervals.

(a) **ARCHITECTURE**



(b) **OPERATION OF DLC**

The DLC compares two numbers, bit by bit (most sig. bit first). It remains in state 1 until a difference in the two numbers is established. It enters state 2, or 3, if it determines that the first number is less than the second number, or vice versa, respecuvely. It will then remain in one of these two states until it is reset.



correctly and that it meets the design constraints. If the gate *level* design is found to be unsatisfactory, then a new circuit may be designed, a different architecture may be selected, or another algorithm may even be developed.

1.2.3 Physical Design Phase

Once the design has been structurally defined, and its correct operation verified, a physical representation, or layout, is created (Figure 1.8). Once again, simulations are performed to verify correct operation, and that no new errors were introduced during this transformation.

1.2.4 Test Set Design Phase

While the IC is being designed, test-sets are generated that will verify that the fabricated chip operates to specification. There will also be some iteration involved in generating this test-set, to guarantee minimal test coverage. As well, the test-set design may influence the logical and physical design phases, as shown by the solid lines. Finally, the chip is fabricated and tested. Hopefully, all design errors were detected earlier, and a re-design will not be required.

1.3 Types of IC's

IC's can be divided into two categories:

- Application specific integrated circuits (ASIC's),
- General purpose integrated circuits (GPIC's).

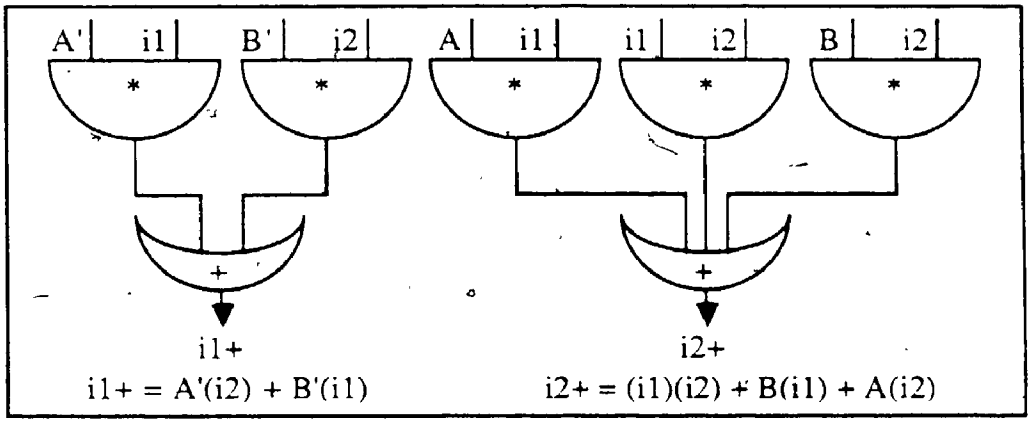
Application specific integrated circuits (ASIC's) are IC's which have been designed and fabricated for a specific application. They usually perform precise, and somewhat limited, functions as opposed to general purpose IC's (GPIC's). Examples of some of the types of applications for which ASIC's might be designed are a furnace temperature controller, or a digital telephone. Some examples of GPIC's include memory chips and microprocessors.

Figure 1.7: **LOGIC LEVEL DESCRIPTION OF DLC**

This is a gate level schematic of the DLC shown in Figure 1.6 (b). This is just one type of implementation (Mealy).

- A, B define the state,
- A+, B+ are the next states,
- i1, i2 are inputs at the bit level,
- i1+, i2+ are outputs at the bit level.

(a) **DATA PATH STRUCTURE**



(b) **CONTROL PATH STRUCTURE**

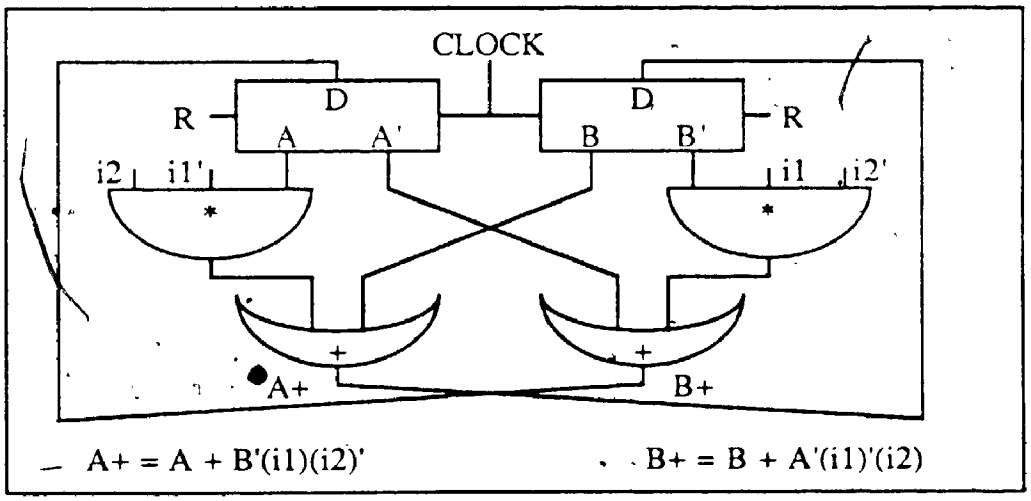
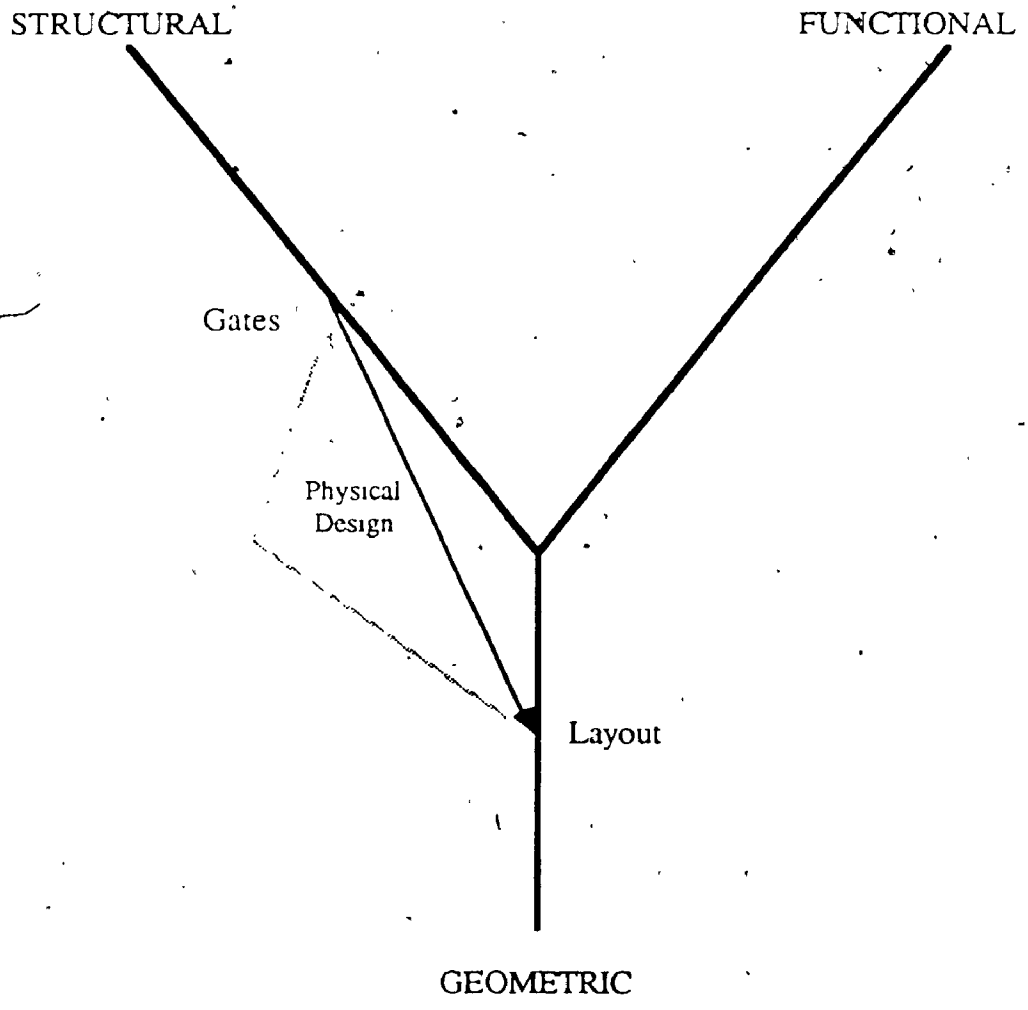


Figure 1.8: **PHYSICAL DESIGN "PHASE"**

The logic design phase encompasses two steps:
- Physical Design (creating layout),
- Simulation.

The physical design step transforms the the design from a gate level representation to a layout (ie- the transistors are placed). Then a simulation of the layout is performed to verify that no new errors were introduced during the transformations, and that the design still meets the constraints (ie- checks for parasitic effects).



1.4 ASIC Advantages and Disadvantages

The use of ASIC's offer a number of advantages over the use of GPIC's. These are:

- high speed,
- low cost where silicon area is important,
- smaller device size, hence higher yield,
- low power consumption,
- reduced parts inventory,
- greater design flexibility,
- protection of design.

The disadvantages associated with using ASIC's are:

- the cost inherent in designing them, such as requiring costly CAD tools, and heavy simulation, to produce a correct design,
- inflexible to design changes once ASIC is created.

1.5 Development of ASIC's

ASIC's can be divided into two types [Fuku88] (Figure 1.9):

- Application Specific Standard Products (ASSP's),
- Application Specific Custom Products (ASCP's).

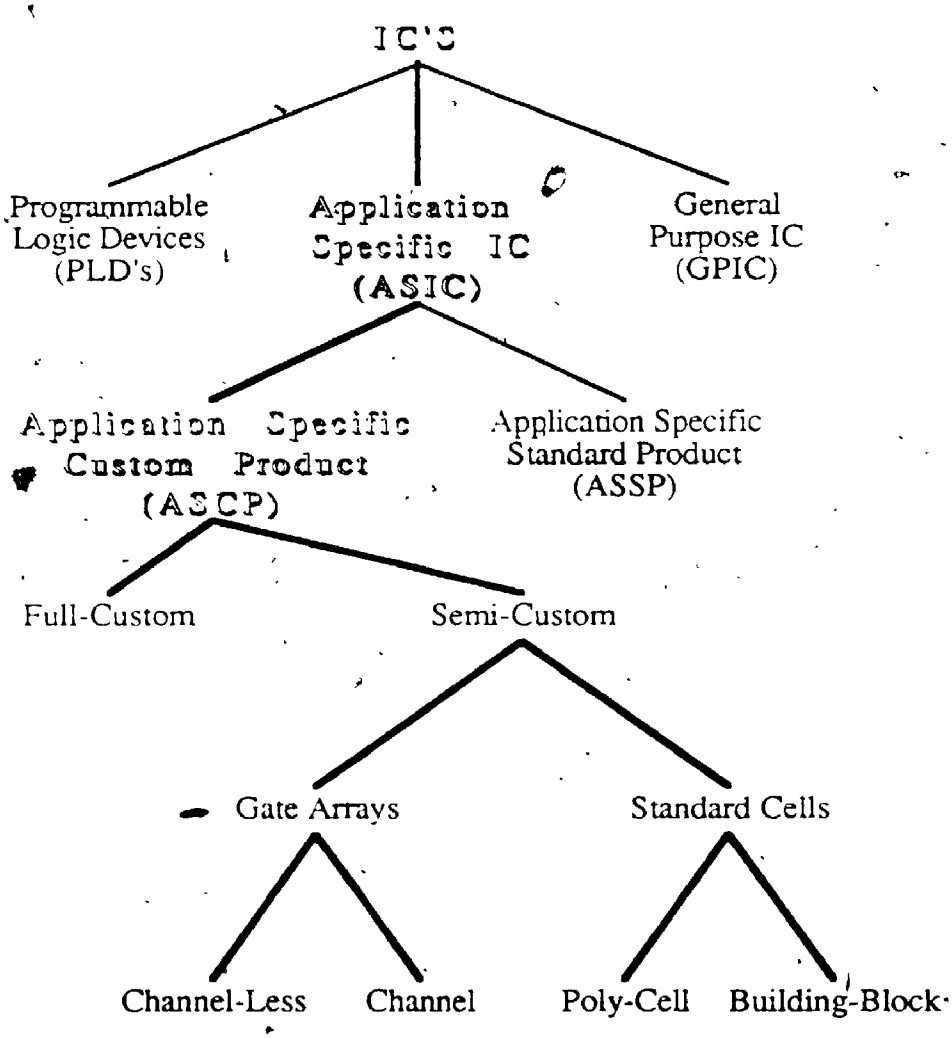
Application Specific Standard Products are IC's which have already been made for applications such as image processing or control.

An Application Specific Custom Product, ASCP, is an IC which has been made to an end-users specific order. It is the design of this type of IC that this thesis is addressing.

ASCP's can be developed using semi-custom or full-custom techniques. In the

Figure 1.9: **HIERARCHY OF IC TYPES**

ASIC's can be divided into two types:
- Application Specific Custom Products (ASCP),
- Application Specific Standard Products (ASSP).



semi-custom methodology, part of the development process is standardized, and involves the use of gate arrays or standard cells.

1.5.1 Gate Arrays

Gate array design requires customizing a gate array IC. This type of IC is one that has an array of unconnected gates (i.e.-it has been processed to a certain fabrication step). The final fabrication steps will connect the appropriate gates in the array to implement the desired function. There are two types of gate arrays, *channel* and *channel-less*, as explained in Figure 1.10.

The advantages of gate arrays include a fast development time with low development costs. The disadvantages are that many gates remain unused - 60 to 75 per cent for arrays with up to 15,000 gates and 25 to 30 per cent for those with over 50,000 gates [Fuku88]. As well, the cost of mass producing a design created with this method is relatively high.

1.5.2 Standard Cell

The standard cell design methodology requires assembling pre-designed and pre-tested circuit components (standard cells) to implement the design's function. The complexity of these standard cells can range from simple gates, latches, and flip-flops, to RAM's, ROM's, and PLA's (Figure 1.11(a)). The designer selects desired cells from the cell library and does the layout and interconnection between cells with a CAD tool (Figure 1.11(b)).

One of the advantages of using this popular technique is that the gate real estate efficiency (gates per unit area) approaches 100 per cent for designs with up to 50,000 gates [Fuku88]. As well, the price of mass producing the design is cheaper than if gate arrays are used. However, deliveries of devices designed with this method, range from one to six months depending on the complexity [Fuku88].

1.5.3 Full-Custom

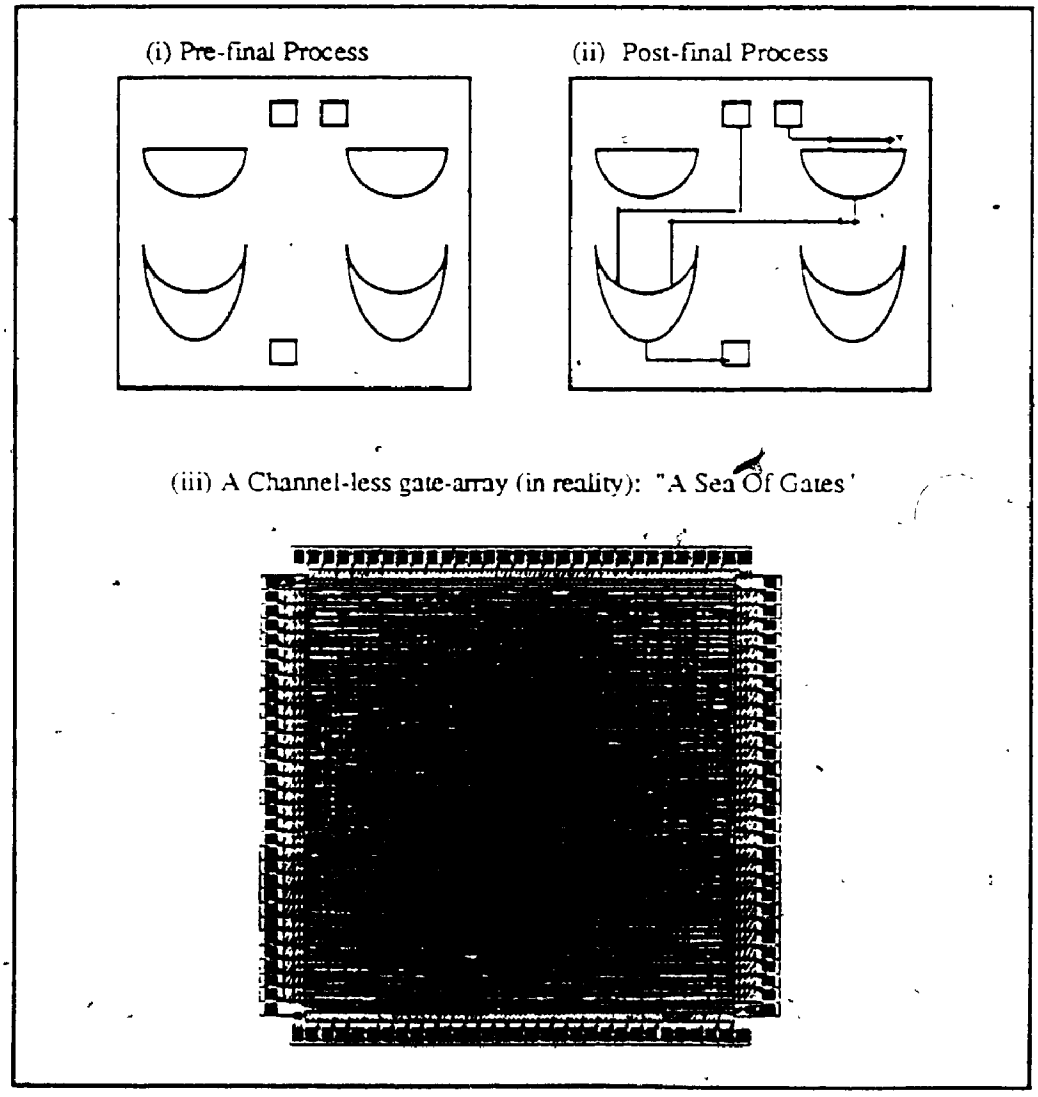
When a full-custom design methodology is used, each transistor is individually

Figure 1.10: **GATE ARRAY DESIGN**

Gate array design requires customizing a gate-array IC, of which there are two types:
- Channel-Less ...(a),
- Channel ...(b)

(a) CHANNEL-LESS GATE ARRAY DESIGN

In a channel-less gate array IC there is no exclusive "road" for wiring. Gates are layed all over the chip (i). A level of metal "personalizes" the IC by making the appropriate connections (ii). A picture of an actual channel-less gate array is shown in (iii).



(b) CHANNEL GATE ARRAY DESIGN

A channel gate-array IC is a gate-array in which in which the wiring "roads" (channels) are arranged among the gates (i). The final process makes the appropriate connections by using these "roads" (ii).

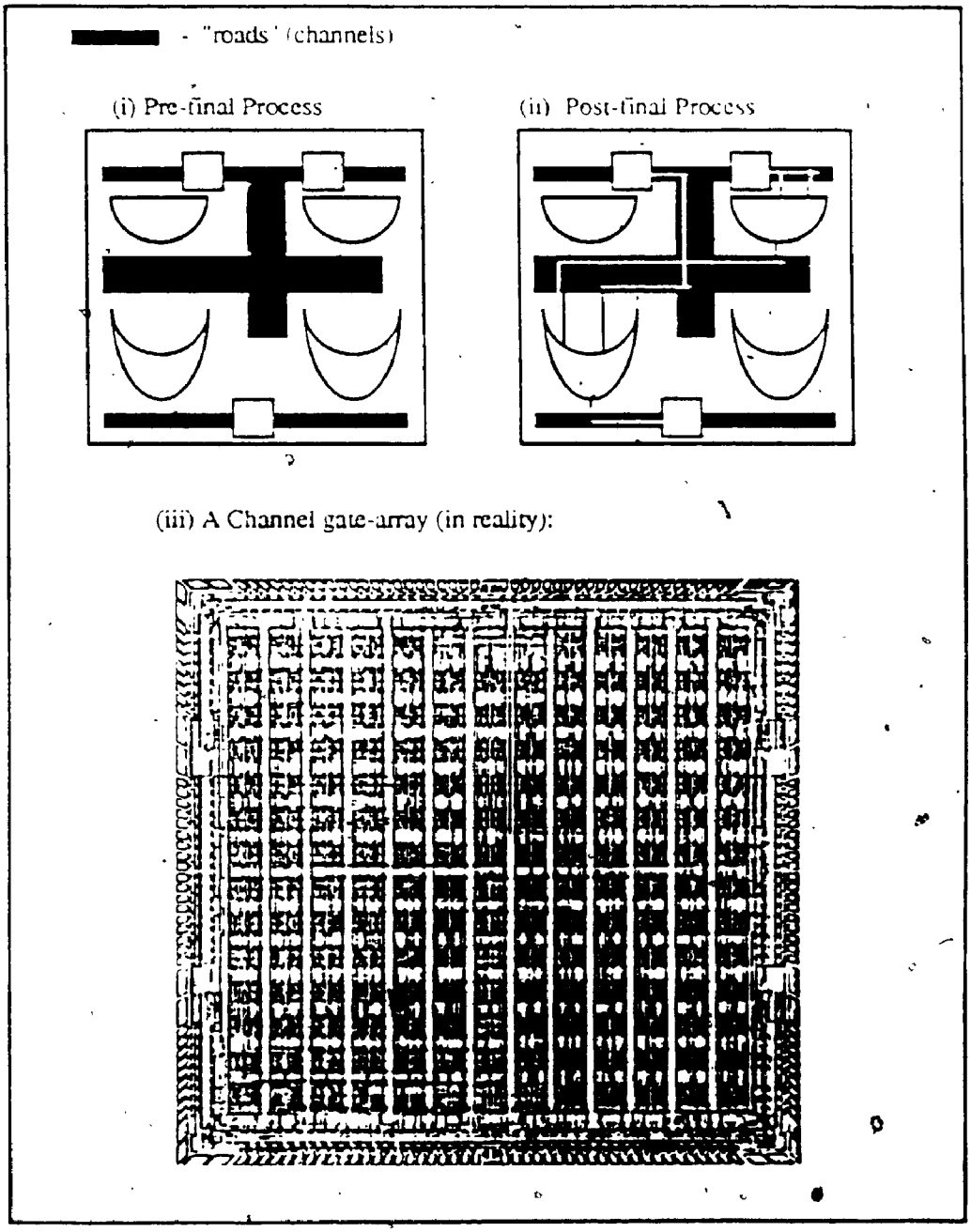


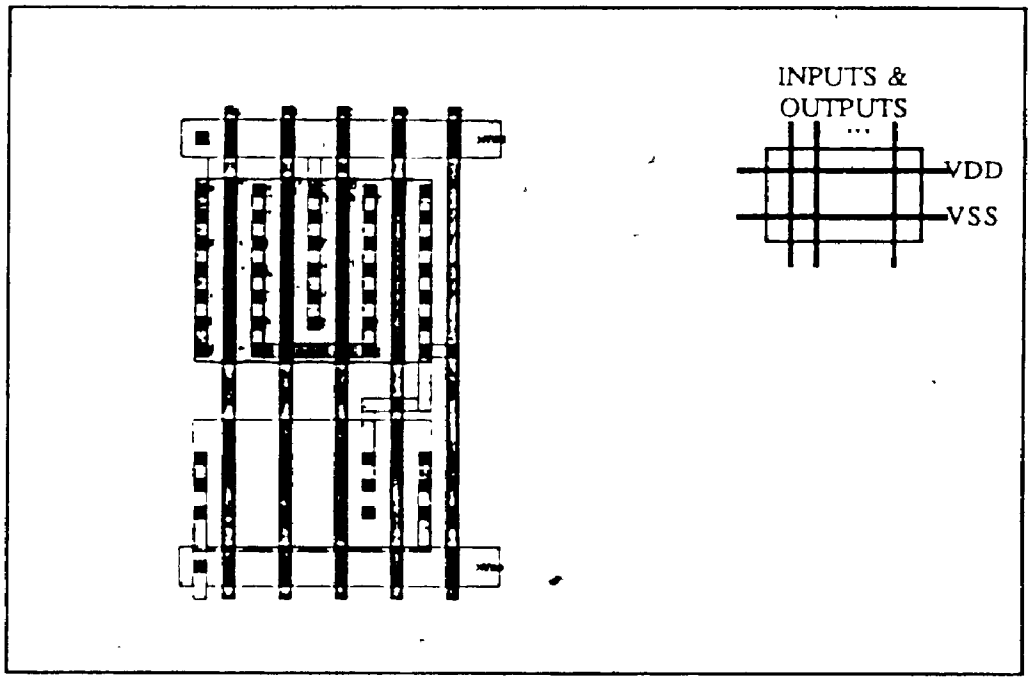
Figure 1.11: **STANDARD CELL DESIGN METHODOLOGY**

In a standard cell design, the desired cells are selected (a) from a cell library. The layout of, and interconnections between, cells can be done with a CAD tool (b).

(a) **TYPICAL STANDARD CELL [WeEs85]**

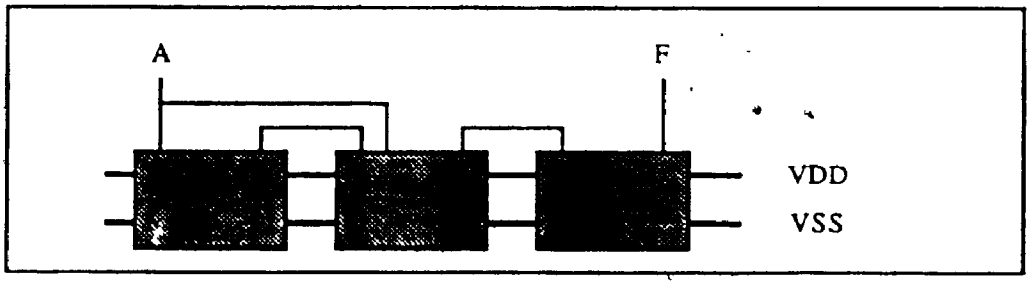
This CMOS standard cell implements the function:

$Z = (ABC + D)'$



(b) **LAYOUT AND INTERCONNECTION OF CELLS**

Standard cells, must have common height, VSS & VDD lines, and common input/output locations so that they can be easily abutted, as shown below.



designed in order to optimize the performance of the entire circuit. This style of designing will generate IC's with better performance and area utilization, than any of the other methods. However, it is also the most time consuming and costly method of design. Furthermore, it is next to impossible to create large designs using this technique.

1.5.4 Programmable Logic Devices

Another method of implementing an application specific function is through the use of programmable logic devices (PLD's). These devices integrate between four and ten TTL IC's, and can be designed in a few hours using an inexpensive development tool. For instance, in one such device, the *Field Programmable Logic Array* (FPLA), the designer can implement the desired function by *programming* the AND and OR arrays (Figure 1.12).

1.6 Design Automation

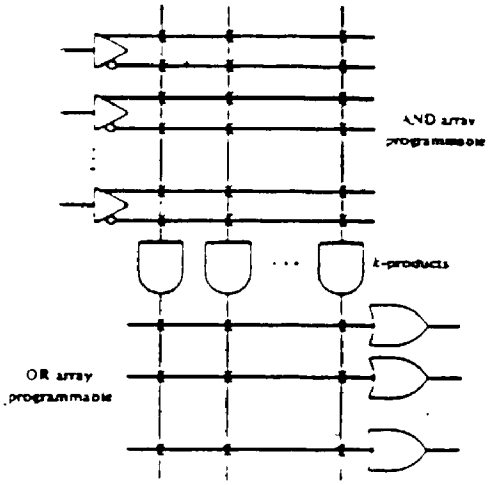
Designing an IC is a costly and complex process involving large amount of thought and human creativity. It requires engineers with a broad range of expertise from IC physics to system level design. As illustrated in Section 1.2, the bottleneck in this process is the design, evaluate (simulate) and re-design cycle. In order to reduce the overall design time and cost, several approaches are being used. The first attempts to improve the productivity of the expert designer through the use of *workstations* that capture, verify and evaluate generated designs. In the second, *compilers*, that automatically transform the design from one domain to another, or from one abstraction level to another, are being developed for specific and limited applications. These compilers offer the *promise* of:

- being able to support designers that don't have extensive low level design knowledge and experience,
- being able to support design at the system level,
- technology independent design,

Figure 1.12: FIELD PROGRAMMABLE LOGIC ARRAY (FPLA) [Klin82]

With the field programmable logic array (FPLA), the designer can implement the desired function by "programming" the AND - OR arrays. This programming can be realized by one of the following methods:

- ultraviolet rays erasure,
- electrical erasure,
- junction destruction,
- melt and cut of fuses.



- permitting exploration of the design space, (In other words, several designs for the same function can be developed, to determine which might be best.),
- fast design time,
- low design cost,
- designs that are *correct-by-construction*, thereby eliminating the iterative design cycle.

Some possible negative features of these compilers are that they currently, and will likely continue to, automatically create IC's that are both slower and require more silicon area than those that are hand-crafted. These compilers will be further described in Chapter 2.

1.7 Thesis Objective and Summary

A critical quantity in the physical layout of an IC is the amount of interconnection required [Dona79]. It has been observed that as the size of circuit increases, the amount of connections, within that circuit, increases exponentially (Figure 1.13). This will have an impact on:

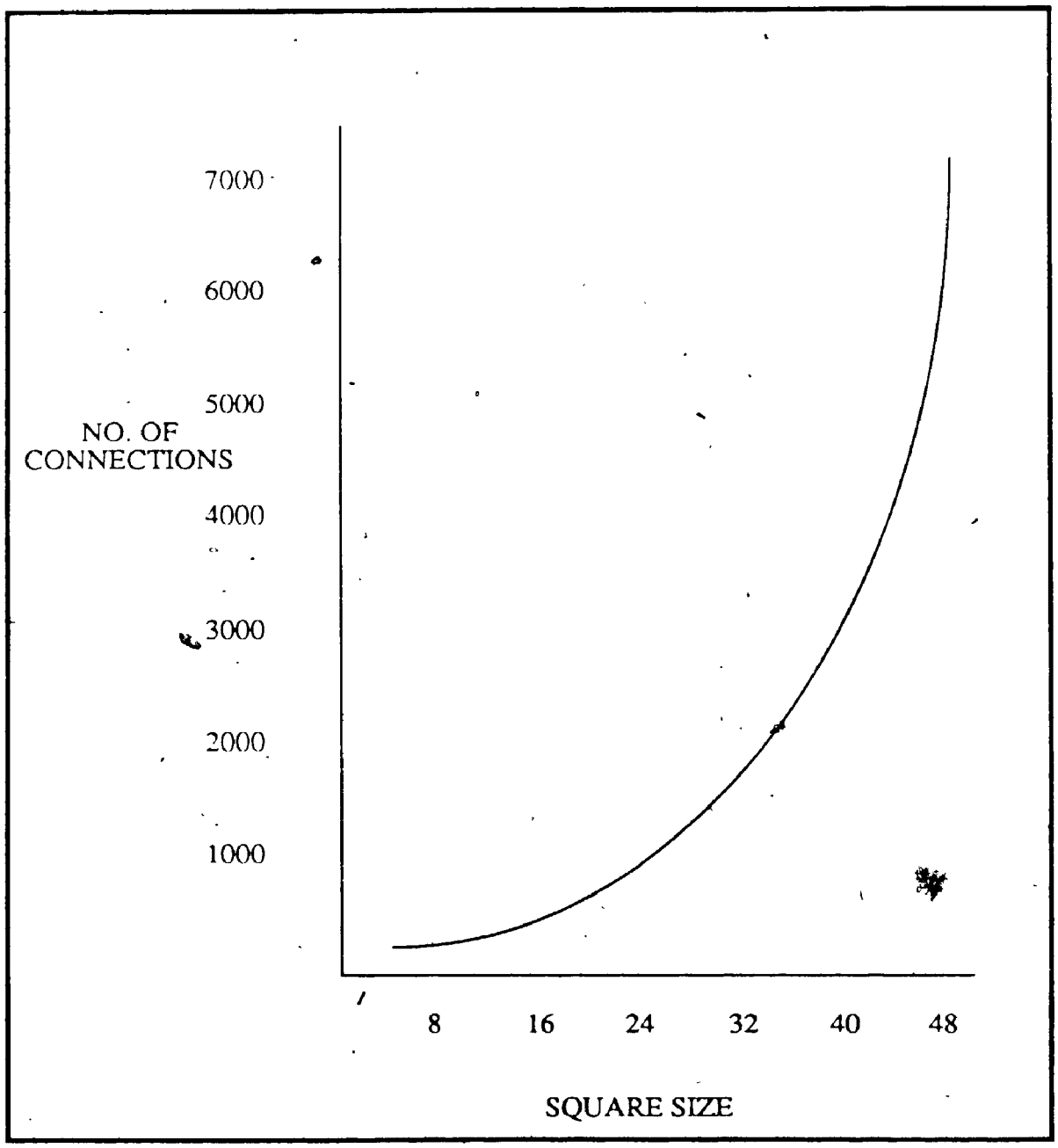
- IC Size,
- IC Performance,
- IC Power Requirements.

The amount of space required for interconnection will affect the total area, or size, of the IC. As well, extensive wiring may delay signals because of capacitive or transmission line effects. Moreover, such wiring will require power to drive the various signals over many wires, and to overcome any capacitive effects.

Considerable savings in power requirements, size, and performance can be obtained if the amount of interconnect is minimized. This thesis addresses the problem

Figure 1.13 NUMBER OF CONNECTIONS VERSES AREA

This figure illustrates that the number of connections on an IC increases exponentially with respect to its size. (data from [Dono79])



of minimizing interconnect in register-transfer level (RTL[1]) circuits. In particular:

- an algorithm to generate greatly reduced interconnect for RTL circuits (**BAL** - Bus Allocation Algorithm) will be shown,
- a criterion for minimizing RTL interconnect will be established,
- a method of evaluating the pre-layout interconnect cost will be developed,
- other methods of generating interconnect will be reviewed and comparisons will be made,
- an algorithm to assign variables to registers (**RAL**) will be developed,
- a brief comparison of **RAL** and two another methods of register allocation will be conducted.

It will be shown that:

- the interconnect generated by **BAL** is as good as, or better than, interconnect generated by other methods,
- the register allocation generated by **RAL** is better than that generated by another method,
- the complexities of **BAL** and **RAL** are of order N^2.

It will be shown that **BAL** is an effective method of generating interconnect, and that **RAL** is also an effective method of register allocation.

[1] A register-transfer level description defines the circuit's structure in terms of: 1) registers, 2) operators, such as ALU's and adders, 3) interconnect, such as muxes and buses, 4) and control (see Section 2.3).

Chapter 2
Silicon Compilation

Automating the design of digital hardware from high level specifications has been a research topic since the early fifties [Park84]. Today, the term *silicon compilation* is being used to describe the automatic design of IC's. More specifically, it has come to mean the transformation of a high level description into a layout.

Silicon compilers are classified as either structural or behavioral, depending on the abstraction level of their input design specification [PaGa87].

2.1 Structural Compilers

A structural silicon compiler converts the design from a structural definition to a geometric one [PaGa87]. This conversion process is called physical design (Figure 2.1).

The structural specification (i.e.- the input to the compiler) is defined in terms of a set of components and their interconnections (Figure 2.2). Each component cell is instantiated by retrieving the individual cell layout from a cell library. For instance, layouts of the NAND gate are copied from the cell library (Figure 2.3), placed and routed, to create the the layout for the structural specification.

2.2 Behavioral Compilers

A behavioral compiler transforms a functional description of a circuit into a geometric one (Figure 2.4).

This functional description (Figure 2.5) is specified by:

- a set of input and output ports,

Figure 2.1: **STRUCTURAL SILICON COMPILATION**

This figure shows that structural silicon compilation is that operation that transforms a structural description (as shown in Figure 2.2) to a geometric one. This transformation is called "physical design".

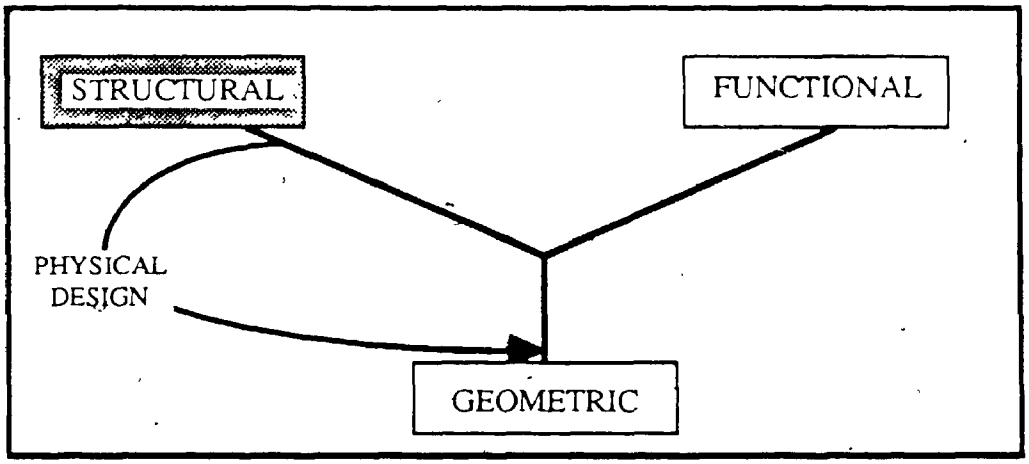


Figure 2.2: **AN EXAMPLE OF A STRUCTURAL DESCRIPTION**

This figure shows an example of a structural description of a circuit, at the logic level of abstraction, of the function " F = AB + CD ". This could be the type of input that a structural compiler would accept.

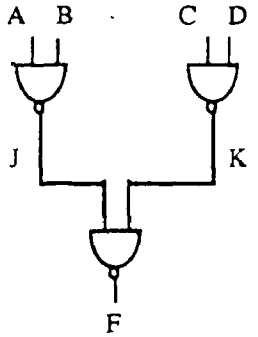
GRAPHIC REPRESENTATION	GATE	TYPE	NO. OF INPUTS	FAN-OUT	CONNEC-TIONS
	1	NAND	2	1	IN-1 → A IN-2 → B OUT → J
	2	NAND	2	1	IN-1 → C IN-2 → D OUT → K
	3	NAND	2	1	IN-1 → J IN-2 → K OUT → F

Figure 2.3: **GEOMETRIC REPRESENTATION OF A NAND GATE [WeEs85]**

This figure shows a geometric, or physical, representation of a NAND gate at the polygon / layout level of abstraction. This representation would be stored in a cell library.

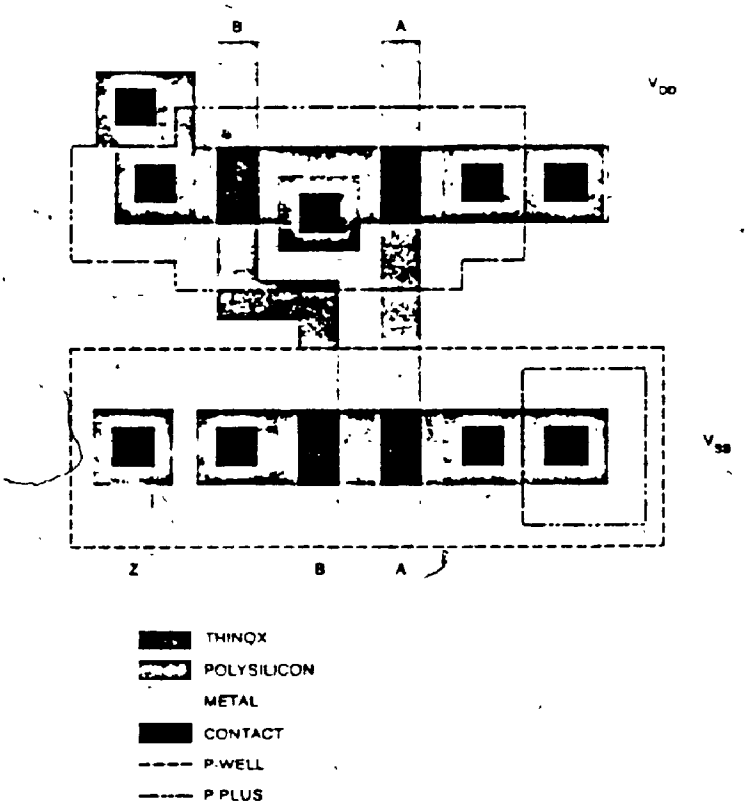


Figure 2.4: **BEHAVIORAL COMPILATION AND SYNTHESIS**

Behavioral compilation transforms a behavioral/functional description of a circuit into a geometric/physical representation. The synthesis process performs the functional to structural transformation (below). An example of the type of input to a behavioral compiler is shown in Figure 2.5.

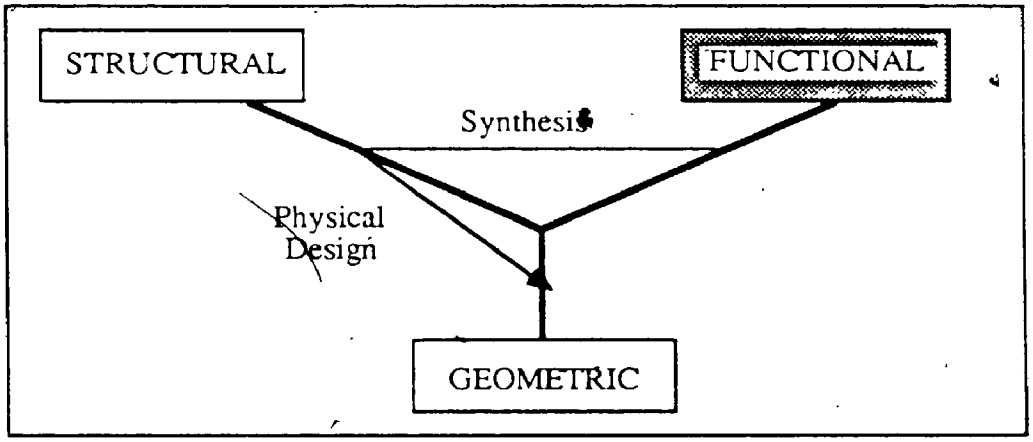
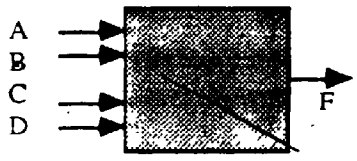


Figure 2.5: **INPUT TO A BEHAVIORAL COMPILER**

This is an example of a behavioral/functional description of a circuit (ie - input to a behavioral compiler). It is specified by:

- a set of input and output ports (ie- interface to circuit) ...(a),
- a behavior of the output ports with respect to input ports and time (ie- behavioral description) ...(b)
- a set of design constraints ...(c).

(a) INTERFACE DESCRIPTION	 A : Input Port, logic 1/0,1-bit. B : Input Port, logic 1/0,1-bit. C : Input Port, logic 1/0,1-bit. D : Input Port, logic 1/0,1-bit. F : Output Port, logic 1/0,1-bit.
(b) BEHAVIORAL DESCRIPTION	F = AB + CD
(c) CONSTRAINTS	- CMOS, 3m.

- a behavior of the output ports with respect to the input ports and time,
- a set of design constraints.

The transformation, from the functional level to that of the geometric, occurs in one of two ways:

1) Through the integration of a structural compiler, that permits the layout to be automatically generated from its functional description,
2) Through the transformation of a functional description to its structural equivalent. This pre-processing to a structural compiler is called *synthesis* (Figure 2.4).

2.3 Synthesis Of Register Transfer Level Circuits

A register transfer level (RTL) circuit consists of a data-path structure and a control-path structure (Figure 2.6).

The data-path structure is responsible for storing and operating on the data. It is defined by the primitive components:

- Functional Units,
- Storage Units,
- Interconnection Units.

Functional units, or operators , perform operations on the data such as add, subtract, and multiply. Typically, these operations are mapped to ALU's, multipliers, adders, subtractors and dividers. Data variables generated by the functional units need to be mapped to storage, or memory, units - which can be thought of as RAM, CACHE , or more simply as registers. Interconnection units, such as buses and muxes, transfer the data between functional and storage units.

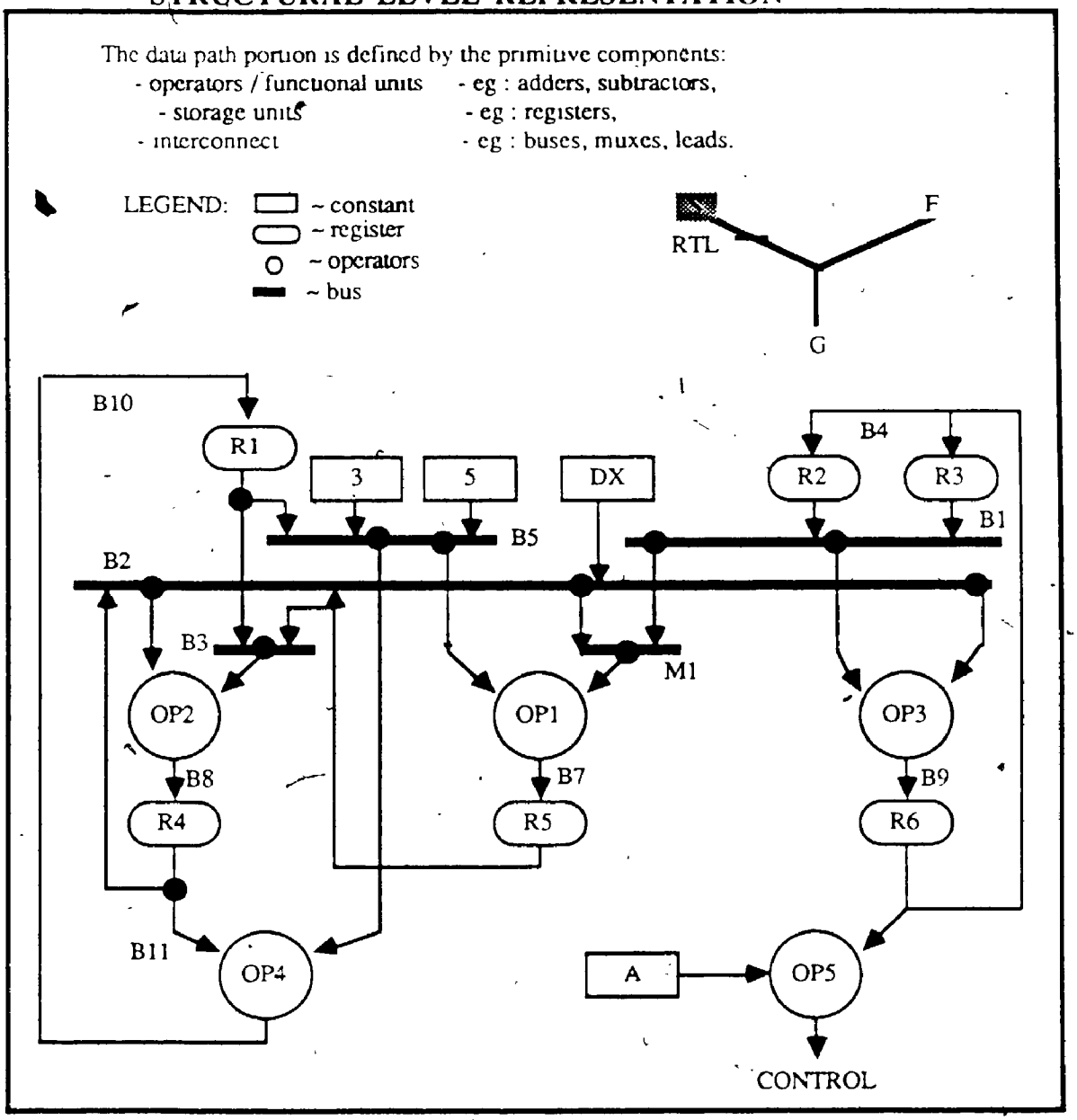
The control-path structure supervises, or controls, the sequence of events in the data-path. It may be defined by:

- a state or instruction sequencer,

Figure 2.6: EXAMPLE OF A REGISTER TRANSFER (RTL) CIRCUIT

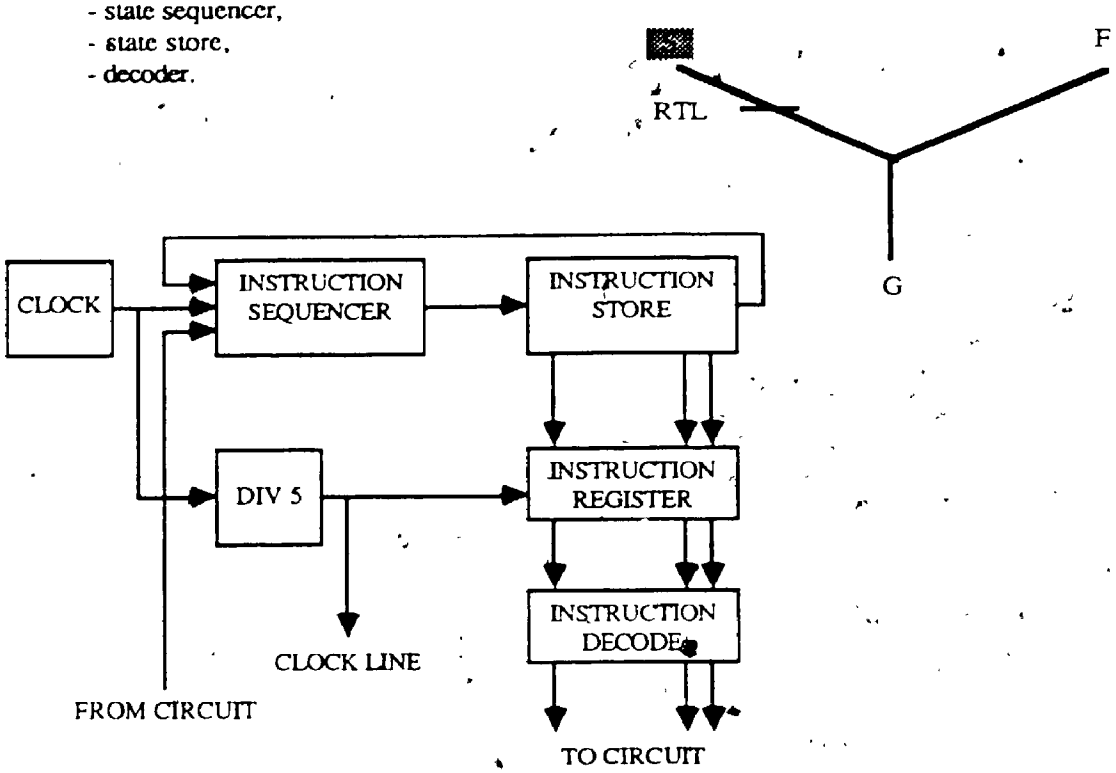
An RTL circuit consists of: (a) a data path structure,
(b) a control path structure.

(a) EXAMPLE OF DATA PATH STRUCTURE:
STRUCTURAL LEVEL REPRESENTATION



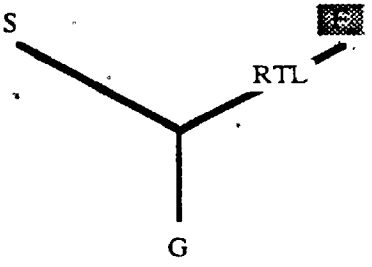
(b) EXAMPLE OF CONTROL PATH: STRUCTURAL REPRESENTATION

The control path portion, at the RTL, is defined by:
- state sequencer,
- state store,
- decoder.



(c) EXAMPLE OF CONTROL PATH: FUNCTIONAL REPRESENTATION

CONTROL STEP	RTL INSTRUCTION
1	R2 -> BUS 'B1' R1 -> BUS 'B3' DX -> BUS 'B2' OP2.OUT -> BUS 'B8' BUS 'B1' -> OP3.IN1 BUS 'B2' -> OP3.IN2, OP2.IN1 BUS 'B3' -> OP2.IN2 BUS 'B8' -> R4



- a state or instruction store,
- an instruction decoder.

Synthesis of an RTL structure is usually divided into three phases (Figure 2.7):

- problem specification (Section 2.3.1),
- data-path synthesis (Section 2.3.2),
- control-path synthesis (Section 2.3.3).

2.3.1 Phase 1: Problem Specification

The circuit specifications must be defined in terms of a high-level hardware description language (HDL), which describes the desired behavior of the hardware. For instance, Figure 2.8, below, shows a pascal-like definition of a circuit that is to solve the differential equation $y'' + 5xy' + 3y = 0$.

2.3.2 Phase 2: Data-Path Synthesis

There are several activities that occur during data-path synthesis:

- design transformation,
- resource allocation,
- decomposition,
- event scheduling.

Design transformation involves changing a design to achieve a goal, or to meet constraints, without adding further detail. Resource allocation involves selecting structures to implement functions. It is a many-to-many mapping as the same resources can be used to implement many functions. Decomposition occurs when there is no direct mapping between functional and structural components. The function must be decomposed until such a mapping is possible. Event scheduling assigns each operation to a *time-slot*.

Figure 2.7 TASKS TO BE PERFORMED DURING SYNTHESIS

Data path synthesis is usually performed before control path synthesis for ease of implementation, as shown below. However, if both processes were executed simultaneously, more optimum designs would result [Park84].

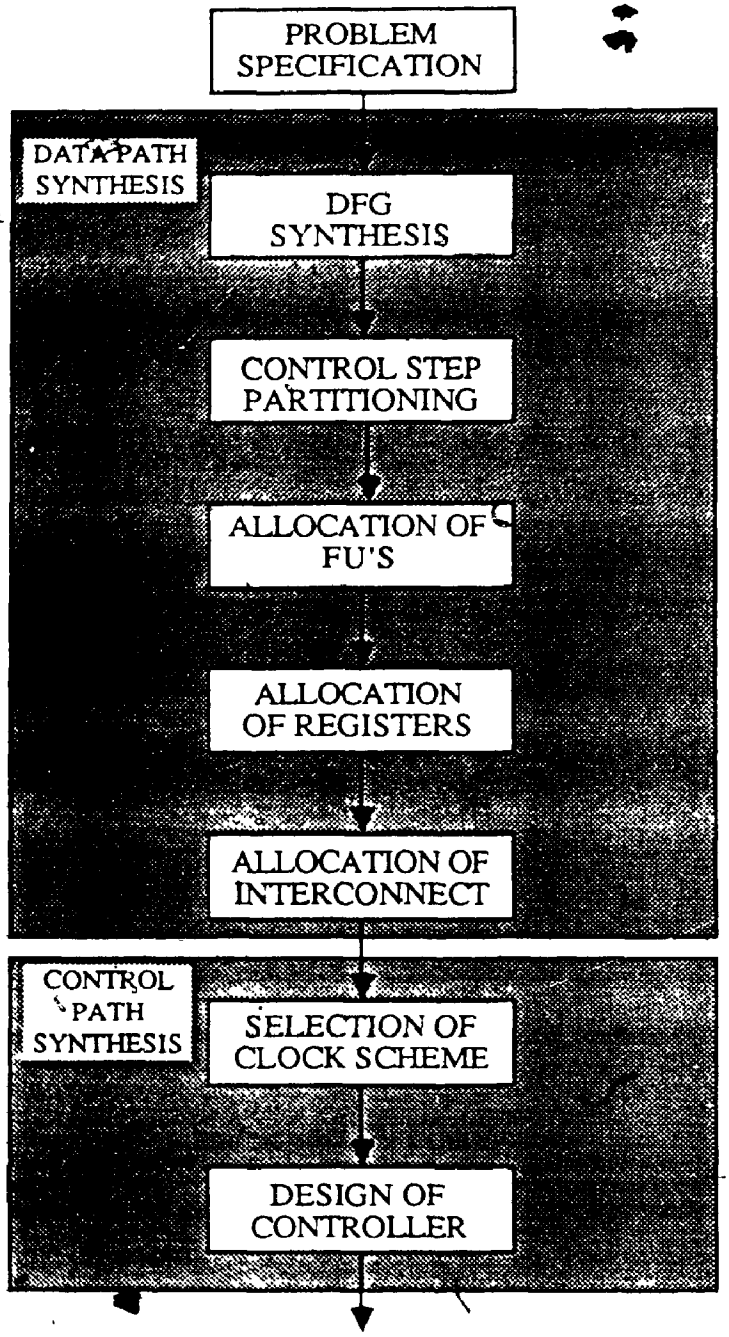


Figure 2-8: Example of a *Problem Specification* [Paul86]

```
PROGRAM Diffeq (INPUT,OUTPUT);
VAR a, dx, x, u, y, u1, y1: INTEGER;
BEGIN
   [Read in the ordinate, a, at which the value of the
function is wanted, the step size, dx, and boundary condition]
   WHILE x < a DO
           BEGIN
             x1 := x + dx ;
             u1 := u - 5uxdx -3ydx ;
             y1 := y + udx ;
             x  := x1 ;
             u  := u1 ;
             y  := y1 ;
           END ;
   WRITELN (y) ;
   END ;
```

Generally, during data-path synthesis many steps must be performed, as shown earlier in Figure 2.7:

- Data flow graph synthesis,
- Control step partitioning,
- Allocation of functional units,
- Allocation of registers,
- Allocation of interconnect.

2.3.2.1 Step 1: Data Flow Graph Synthesis

During this step, The behavioral description is transformed into a data flow graph (DFG). The DFG is a graphical representation of the data path. It defines

any precedence relations (i.e.- what operation must be performed before another). The DFG for the example above is given in Figure 2.9.

2.3.2.2 Step 2: Control Step Partitioning (Scheduling of DFG)

Once the DFG is established, the operations are assigned to discrete time steps. This scheduling of operations will have ramifications on chip area and performance, and is probably one of the most difficult tasks in data-path synthesis. To illustrate some of the complexities involved, the DFG, above, has been scheduled, with an imposed four cycle constraint, using three approaches:

- as soon as possible (ASAP),
- as late as possible (ALAP),
- load balanced (LB).

In ASAP scheduling (Figure 2.10(a)), operations are scheduled in a *greedy* fashion. An operation is assigned to the time step immediately following that in which the inputs, for that operation, become available. For instance, if the inputs to a *multiply* operation are ready (i.e.- available) in the first time step, then that *multiply* is assigned to the second time step.

In ALAP scheduling (Figure 2.10(b)), operations are scheduled in a *procrastinating* manner. An operation is assigned to the time step immediately preceeding that in which the data, produced by that operation, is required. For example, if the results from a particular *add* operation are needed in the fourth time step, then that *add* will be assigned to the third time step.

The problem, associated with using these two scheduling approaches, is that the operations tend to be grouped together, either in the first few time steps (ASAP) or the last few time steps (ALAP). This means that several functional units will be required, in order that many of the operations be performed in only one or two time steps. A better approach would be to *balance* the operations throughout all the time steps. This is the goal of LB scheduling (Figure 2.10(c)). With this type of scheduling the number of similar operations done in any time step is minimized.

Figure 2.9: **UNSCHEDULED DATA FLOW GRAPH**

This is an unscheduled data flow graph (ie- none of the operations have been scheduled to precise time steps). It represents the required flow of data through the system, and defines the order in which the operations, on this data, must be performed.

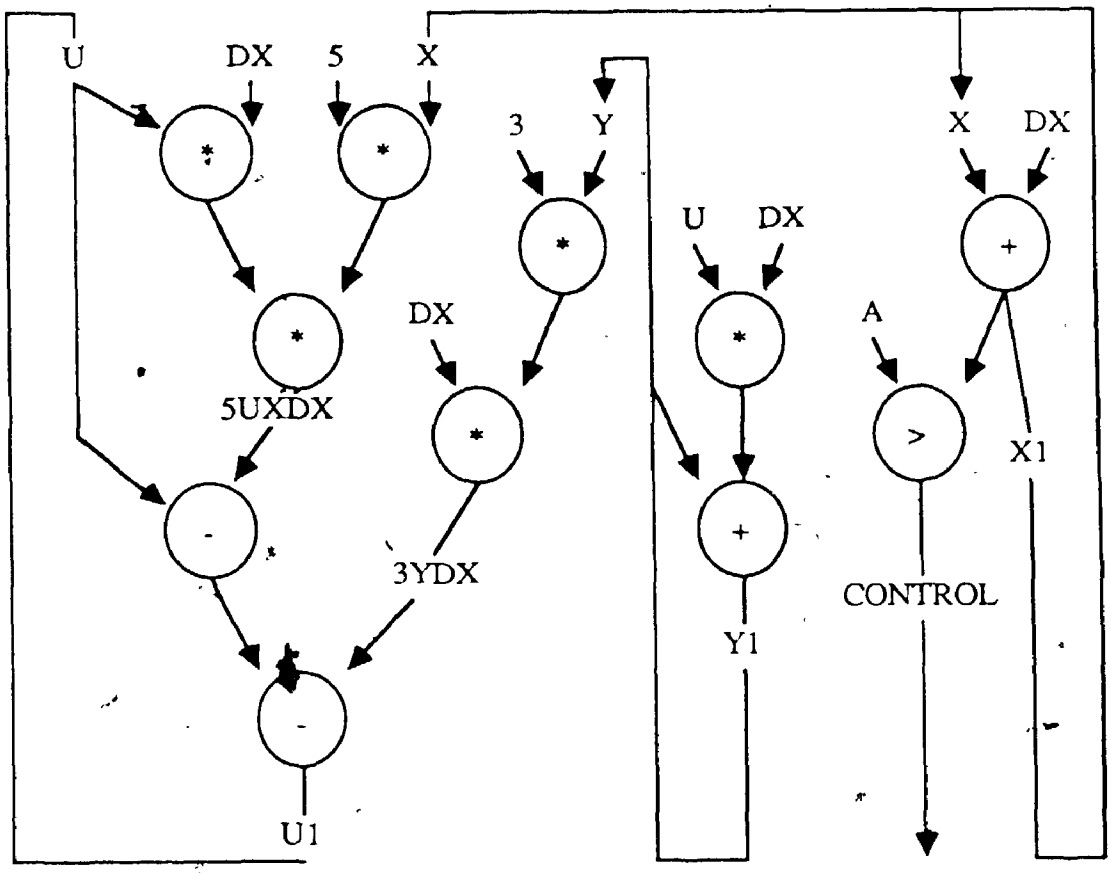
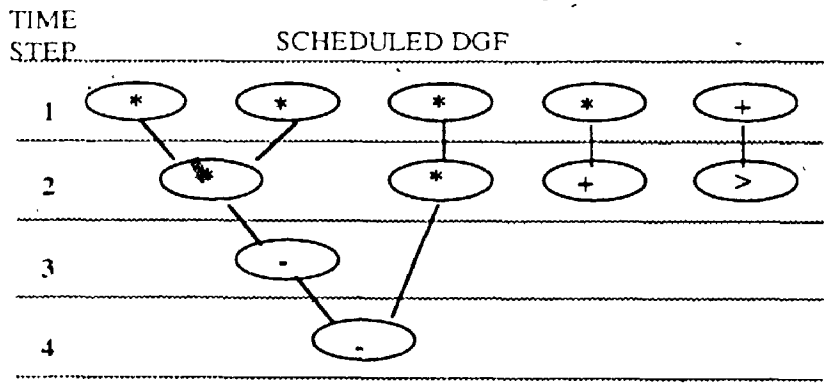


Figure 2.10 SCHEDULED DATA FLOW GRAPHS

The scheduled DFG has all operations (multiply, add, etc.) assigned to discrete time steps. Shown below are three differently scheduled versions of the same unscheduled DFG from Figure 2.9, with an imposed four step time constraint. Each version, shown in simplified form, will have an impact on the amount of hardware required.

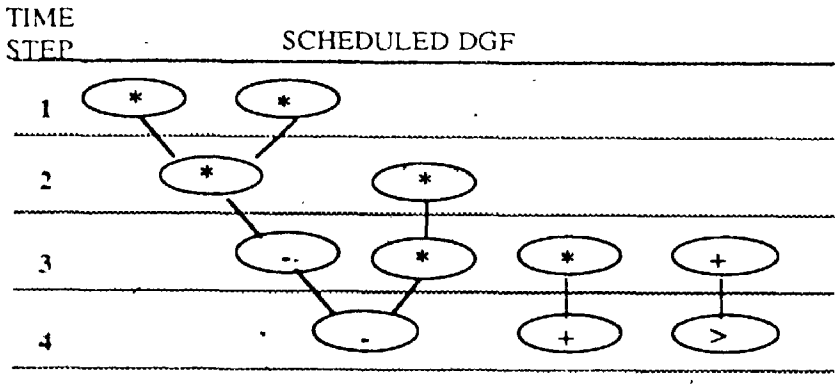
(a) ASAP SCHEDULED

In ASAP scheduling, operations are scheduled "as soon as possible". In other words, operations are assigned to the time steps immediately following those in which the data becomes available. In this example, 5 hardware operators will be required:



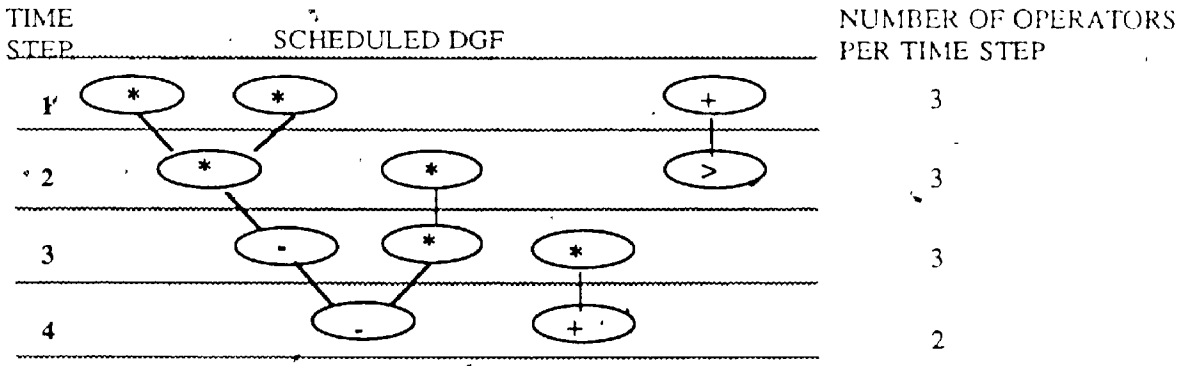
(b) ALAP SCHEDULED

In ALAP scheduling, operators are scheduled "as late as possible". This means that operations are assigned to the time steps immediately preceeding those in which data, from the operators, is required. In this example, at least 4 operators will be required.



(c) LB SCHEDULED

In LB scheduling, operators are scheduled in a "load balanced" fashion. This means that operations are assigned to time steps such that the number of operators per step is 'balanced' over all the steps. In this example ony 3 hardware operators are required.



This will minimize the number of standard cells necessary to do the operations.

2.3.2.3 Step 3: Allocation of Functional Units

Once the DFG is scheduled, the operations must be assigned to discrete functional units (FU's) as shown in Figure 2.11. There are many ways in which this allocation can be done, each with its own area and performance trade-offs. For instance, each type of operation may be performed by a unique FU. The benefit will be simpler and smaller FU's, however they will be more numerous. On the other hand, more than one type of operation may be performed by a single FU (e.g. an ALU may perform adds, subtracts, and compares) thereby requiring fewer FU's overall. However, these FU's will be more complex.

In addition to area, performance must be considered as well. The FU with the slowest computation time, will set the time step interval length. For instance, if an area-small adder (e.g.- a bit serial adder)is selected to perform an *add*, then the time required to perform this *add* could set the time step interval length. In other words, the time steps will be tailored to the slowest component.

2.3.2.4 Step 4: Allocation of Registers

Once functional units have been allocated, registers must be selected to store the variables generated by these functional units. Again, the goal is to minimize chip area. This can be accomplished by assigning more than one variable to a register (i.e.- sharing the register) such that the total number of registers and the amount of interconnect is minimized. For instance, Figure 2.12 presents different storage arrangements for the three data variables generated by *Operator1*, during time steps one through four. Arrangement a, the most area expensive solution, uses a unique register for each variable, while arrangement *c*, the most area efficient solution, shares one register among the three variables. This is possible since the three variables exist during different time steps. Variables throughout the DFG are assigned to registers in this same manner (Figure 2.13). However, it is not always as clear which variables should be chosen to share registers such that the

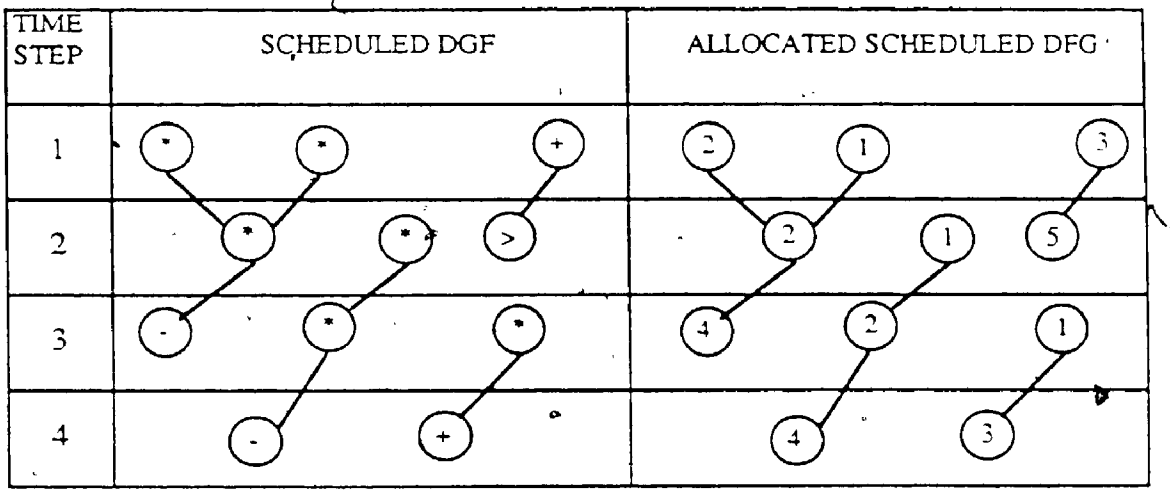
Figure 2.11: OPERATIONS TO BE PERFORMED BY THE FUNCTIONAL UNITS (FU'S)

From the scheduled DFG, the types of operations and when they are to be performed, can be determined. This information is used to select the appropriate functional units (FU's). For instance, in the example below, FU's are required that are capable of:

- multiply, add, compare, or some combination of two or more of these.

A simple allocation of operations to FU's is shown , below This is only one possible allocation out of many. The allocations are:

- muluply operations will take place on "MULTIPLIER-1", or "MULTIPLIER-2",
- add operations will take place on "ADDER-3",
- subtract operations will take place on "SUBTRACTOR-4",
- compare operations will take place on "COMPARATOR-5".



lowest area cost is achieved. Finding the best solution appears to be an exponential problem.

An algorithm to perform this task was developed by the author, and will be presented in Chapter 5. A further examination of the requirements of this process will also be discussed.

2.3.2.5 Step 5: Allocation of Interconnect

The allocation of interconnect involves assigning the data transfers, that occur between registers and operators, to particular buses and muxes. The goal is to group together the transfers from the DFG, each transfer in a group being assigned to the same bus or mux, such that the total amount of interconnect in the circuit is minimized and that concurrency constraints are not violated. For example, the six transfers, that are shaded in the DFG of Figure 2.13, have been divided into two groups using two different assignments (Figure 2.14(a). These two different assignments illustrate the types of choices that must be made during this allocation step. The first, *Choice-1*, requires one 3-input, 2-output mux and one straight lead (Figure 2.14(b)), while the second assignment, *Choice-2*, requires one 2-input, 3-output mux and one 2-input, 1-output mux (Figure 2.14(c)). Clearly, the former combination is best. All the transfers in the DFG are partitioned into groups in the same manner (Figure 2.15). However, finding the set of partitions that yields the minimum solution is an NP complete problem. The resulting data-path structure for this DFG is shown in Figure 2.6.

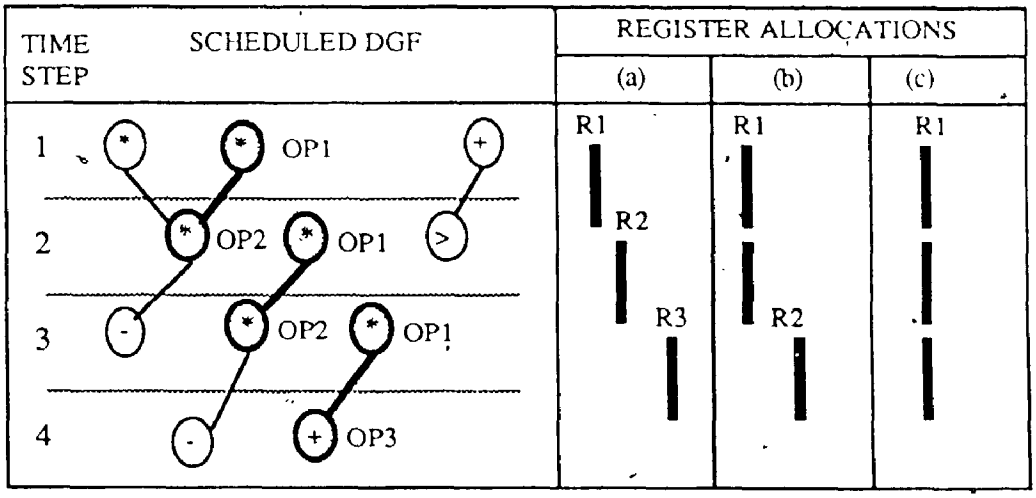
An algorithm to perform this task was developed and implemented, and forms the major objective of this thesis. The problem definition for the allocation of interconnect will be further defined in Chapter 3. As well, other approaches, or solutions, to this task will be presented.

2.3.3 Phase 3: Control-Path Synthesis

Synthesis of the control-path for an RTL described circuit begins with a scheduled and allocated DFG. Two activities occur during control-path synthesis (Fig-

Figure 2.12: ALLOCATION OF VARIABLES TO REGISTERS

This example shows a partial assignment of variables to registers for the DFG from Figure 2.11 (shaded regions in DFG shown below). Three different storage arrangements for the variables produced by 'Operator-1' (OP1) are illustrated (a), (b) and (c). Arrangement (a) is the most area expensive, as each variable has been assigned to a unique register. Arrangement (c) is the most area economical as only one register is being used to store data over 4 time steps.



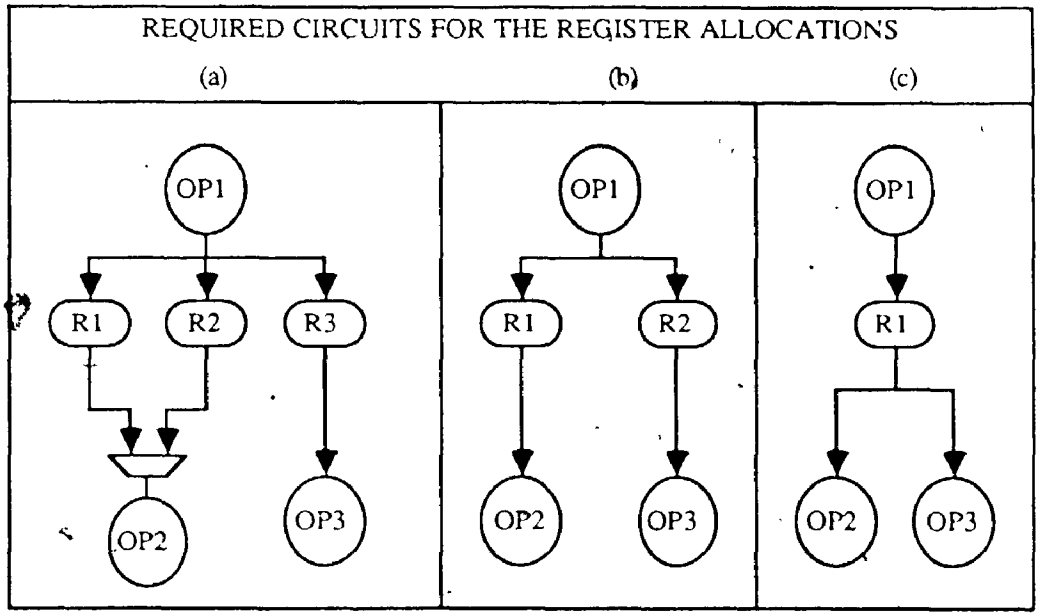


Figure 2.13: **DFG (PRE-INTERCONNECT-ALLOCATION OR POST-REGISTER-ALLOCATION)**

This is the "post-register-allocation" DFG that solves the differential equation described in Figure 2.8. Note that:

- operations have been scheduled to discrete time steps,
- operations have been assigned to specific components,
- all the variables have been assigned to registers.

The shaded lines represent the transfers that will be assigned to interconnect in Figure 2.14.

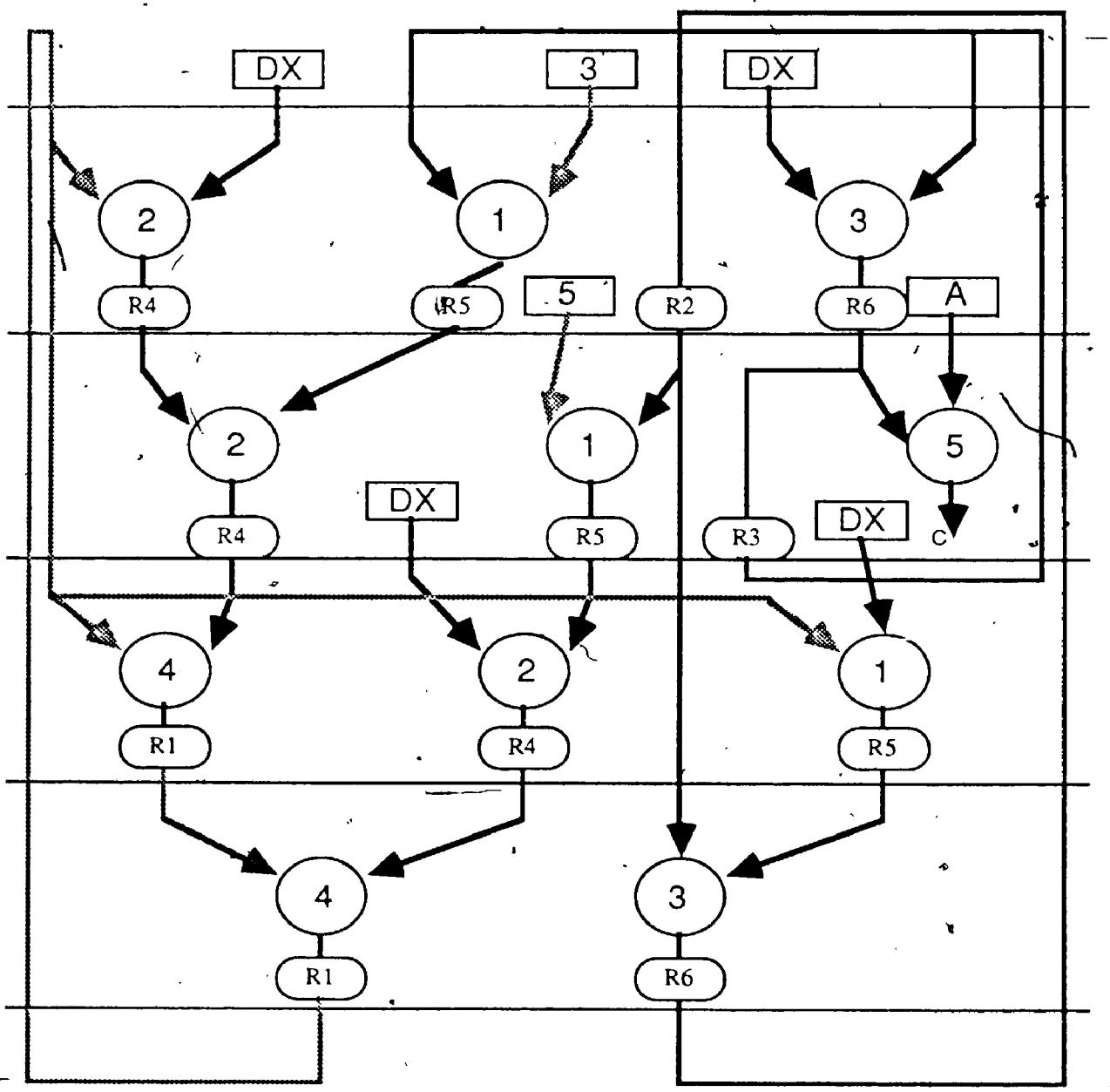


Figure 2.14. **ASSIGNMENT OF TRANSFERS TO MUXES**

This example illustrates the different ways in which transfers can be assigned to interconnection units. The transfers (a) have been partitioned into two groups in two different ways:

Choice-1 (see part b): Group1 - Transfer -2
Group2- Transfers 1,3-6.

Choice-2 (see part c): Group1- Transfer-1
Group2- Transfers 2-6.

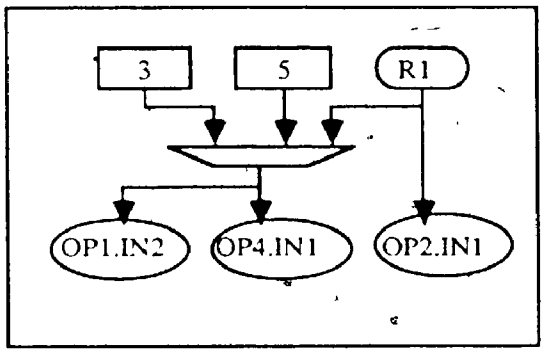
(a) **TRANSFERS TO BE ASSIGNED** (Shaded transfers from Figure 2.13)

TRANSFER INDEX	SOURCE	DESTINATION	TIME STEP	CHOICE-1	CHOICE-2
1	3	OP1.IN2	1	Group-2	Group-1
2	R1	OP2.IN1	1	Group-1	Group-2
3	5	OP1.IN2	2	Group-2	Group-2
4	R1	OP4.IN1	3	Group-2	Group-2
5	R1	OP1.IN2	3	Group-2	Group-2
6	R1	OP4.IN1	4	Group-2	Group-2

(b) **'CHOICE-1'**

If transfers 1, 3-6 are combined, the following interconnect will be required:

- 3-input, 2-output MUX,
- straight through lead.



(c) **'CHOICE-2'**

If transfers 2-6 are combined, the following interconnect will be required:

- 2-input, 3-output MUX,
- 2-input, 1-output MUX.

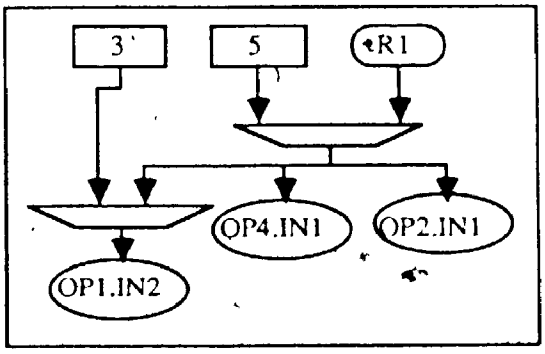
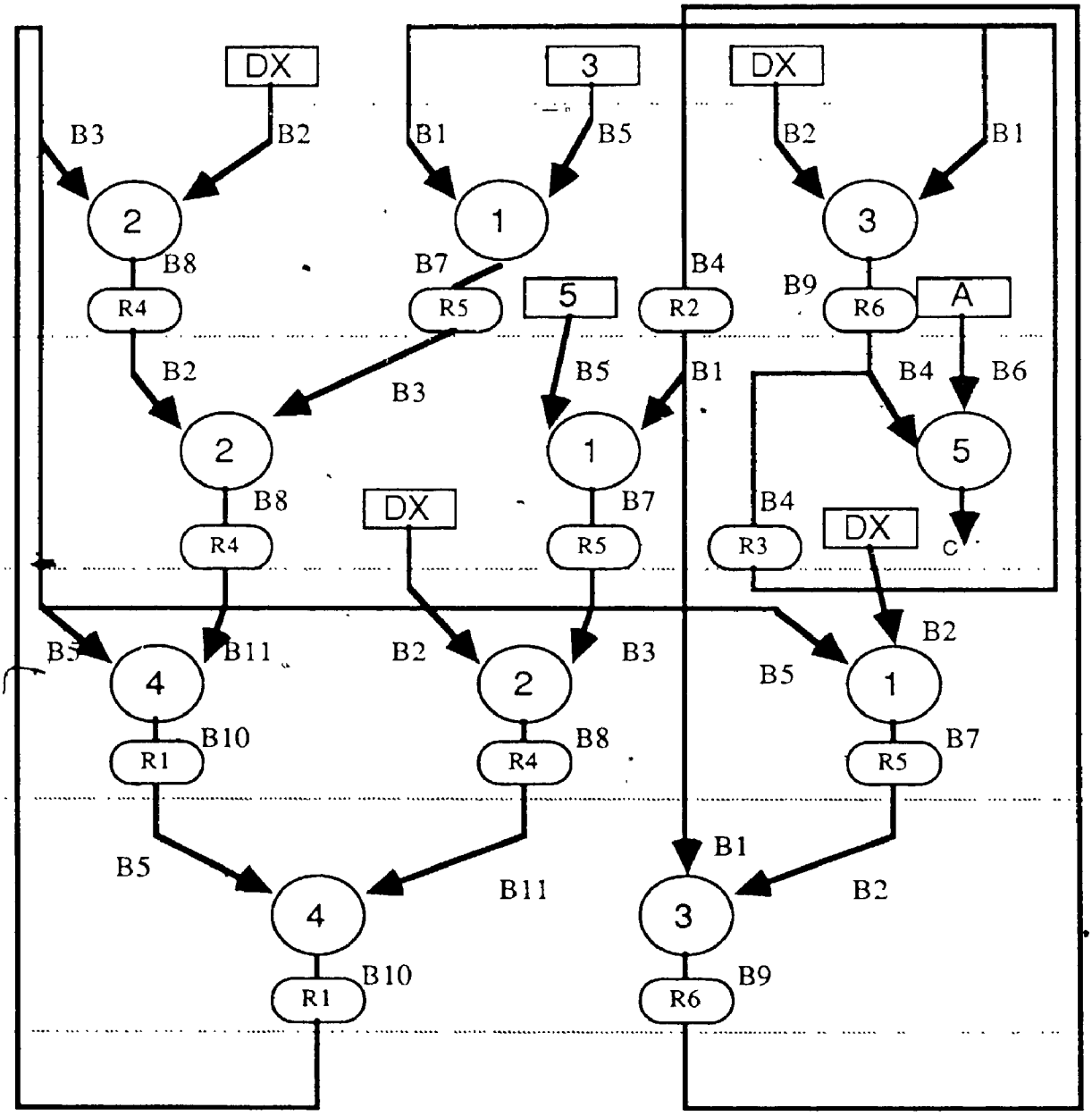


Figure 2.15: **DFG POST-INTERCONNECT-ALLOCATION**

This is the DFG for the data path structure of Figure 2.6. This DFG solves the differential equation of Figure 2.8. The transfer variables have been assigned to buses B1, B2 etc. For example B8 connects operator-2 output to register R4 input. Note, it is used for 3 time steps.



ure 2.7):

- selection of a clocking scheme,
- design of a controller.

Selection of a clocking scheme involves determining the number and lengths of clock phases, and the allocation of events to clock phases. Most synthesis procedures assume one or two clock phases of equal duration [Park84].

The design of the controller usually involves selecting the control style, such as: random logic, microcode, or PLA, and implementing those control functions that are outlined in the scheduled and allocated DFG. The control-path synthesis task has been extensively explored in [Kala86].

2.4 Implementation of Synthesis Systems

There have been almost as many different methods of synthesis as there are systems that attempt this task. The systems, to date, may be categorized by the following criteria:

- Application,
- Method of synthesis,
- Approaches to data-path synthesis operations.

The application of the synthesis system refers to the type of hardware it is attempting to generate. Some systems exclusively design microprocessors [KoTh84] [KoTh85], while others have been developed to produce signal processing hardware [PaKG86] [TsSi86] [TsSi83]. Still others are able to generate solutions for both applications [PaGa87][BrGa87].

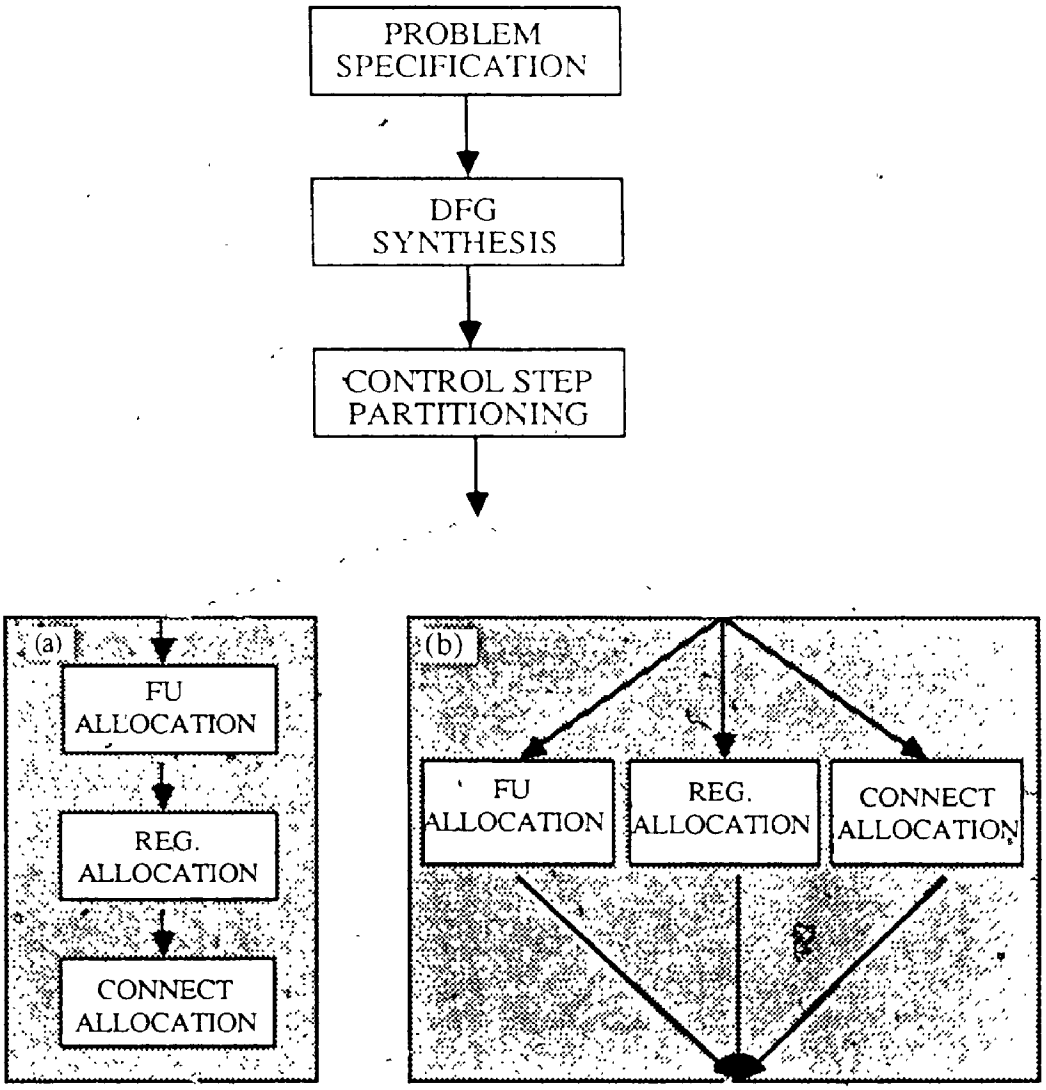
The method of synthesis refers to the type of software techniques employed by the hardware compiler. Such techniques include: heuristic/rule-based [BrGa87] [PaGa], algorithmic [TsSi 83,86], or various combinations of these two [PaKG86].

Finally, there are various approaches to performing those data-path synthesis steps that were outlined in section 2.3.2. Functional units, registers, and inter-

connect may be allocated sequentially [PaKG86][TsSi86] or in parallel [BrGa87] [PaGa87] as shown in Figure 2.16.

Figure 2.16: APPROACHES TO DATA PATH SYNTHESIS

There are at least two different approaches to data path synthesis. One approach, (shown in (a)), performs FU, register, and interconnect allocations sequentially. Another approach (shown in (b)), these three allocations are performed in parallel.



STRUCURAL DESCRIPTION
AT THE
REGISTER TRANSFER LEVEL

Chapter 3
Problem Formulation and Possible Solutions

3.1 Formulation of the Task

The problem of allocating minimal interconnect has been formulated as an assignment one:

> To assign data transfers, with fixed time constraints, to interconnection units using the least amount of area.

In other words, the goal is to group transfers together, each transfer in a group being assigned to the same bus or mux, such that the interconnect has:

- fast transfer time,
- low power consumption,
- small silicon area.

During data path synthesis, the data transfers are obtained from the data flow graph. Each transfer is represented by a transfer variable (TV), which defines the source and destination of the transfer, and the time slots in which that transfer occurs (Figure 3.1.1).

3.1.1 Interconnection Primitives

Many different types of interconnect units are used in circuit design. Some of these have been identified by [TsSi81] as (Figure 3.1.2):

(a) A set of simple wires: - used to connect one source with one destination,

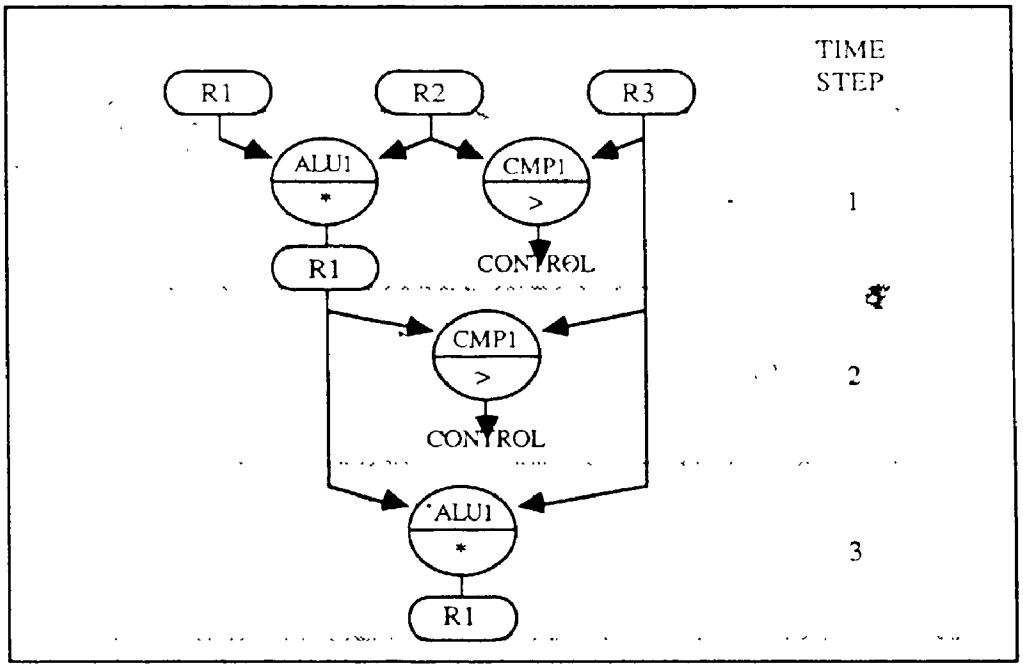
(b) Wired-Broadcast tree: - used to connect one source with many outputs,

(c) Multiplexer (MUX): - has several sources, only one of which is active at a time, and one sink,

Figure 3.1.1 **TRANSFER VARIABLES (TV'S)**

Transfer varaiables (TV's) represent the transfers of data that occur in the circuit. During data path synthesis, the TV's can be obtained from the data flow graph (DFG), that has registers and functional units already allocated (a). The TV's indicate the source and destination of a transfer, and the time steps in which that transfer occurs (b).

(a) **DATA FLOW GRAPH (DFG)**

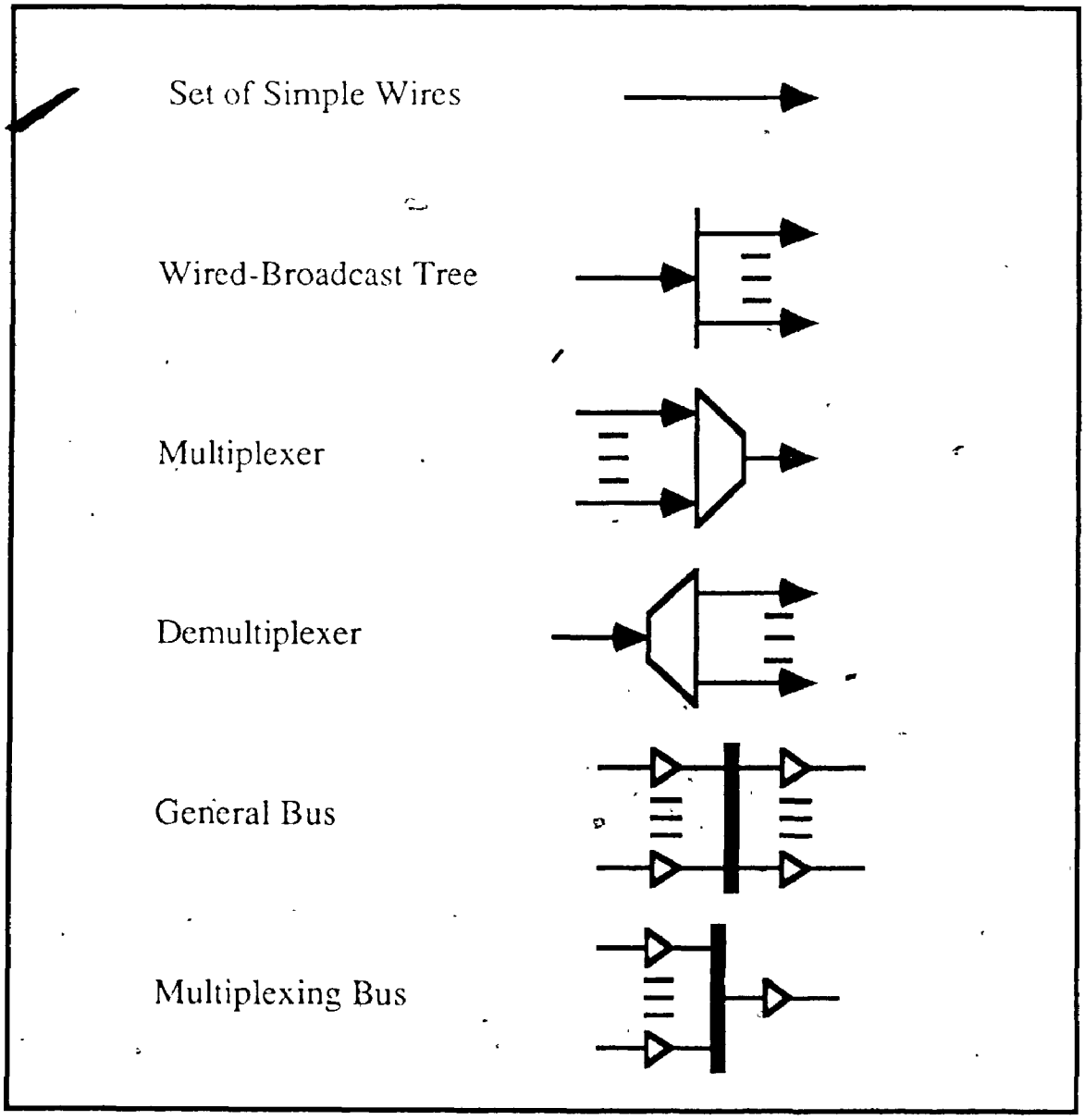


(b) **TRANSFER VARIABLE LIST**

INDEX	SOURCE	DESTINATION	STEP
1	R1	ALU.IN1	1, 3
2	R2	ALU1.IN2	1
3	R2	CMP.IN1	1
4	R3	CMP.IN2	1, 2
5	R3	ALU.IN2	3
6	ALU.OUT	R1	1, 3

Figure 3.1.2 - INTERCONNECT TYPES (FROM [TsSi81])

This figure illustrates the different types of interconnect as defined by [TsSi81].



(d) Demultiplexer (DEMUX): - has a single input and multiple outputs,

(e) General bus: - has multiple inputs and outputs, each of which is connected to the bus through a gating element (pass transistor). Only the gating elements from one source may be active at a time, while those to more than one sink may be active.

(f) Multiplexing bus: - is a general bus, but with only one input.

It was found, however, that these connection components could be reduced to a set of three connection primitives. These are:

- a set of simple wires,
- a wired-broadcast tree,
- a bus and/or mux (*bus/mux*).

The set of simple wires and the wired-broadcast tree primitives (Figures 3.1.3(a) and (b)) are the same as defined previously. The former being used to connect a single source with single destination, with the latter being used to connect a single source with multiple destinations.

The mux, demux, general bus, and multiplexing bus, that were identified by [TsSi81], however, can be further reduced to the one primitive, the bus/mux (Figure 3.1.3(c)). This primitive connects multiple sources with multiple destinations.

3.1.2 Interconnect Cost

Each of the three types of connection primitives incur global and local costs. These costs are associated with the area and power consumed for pass transistors, and area for the leads (wires).

A simple lead connection (Figure 3.1.3(a)) between a source and a destination consumes just the area for the wire, and poses no concurrency problems for the transfer. However, assigning all the transfer variables to straight leads would incur a large area cost.

The wire broadcast tree (Figure 3.1.3(b)) would reduce some of the area as-

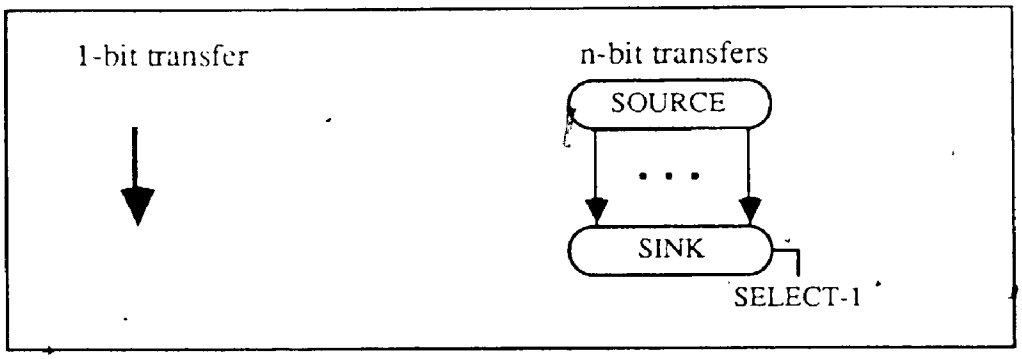
Figure 3.1.3 **INTERCONNECTION PRIMITIVES**

Three interconnection primitives have been identified and used in this thesis:

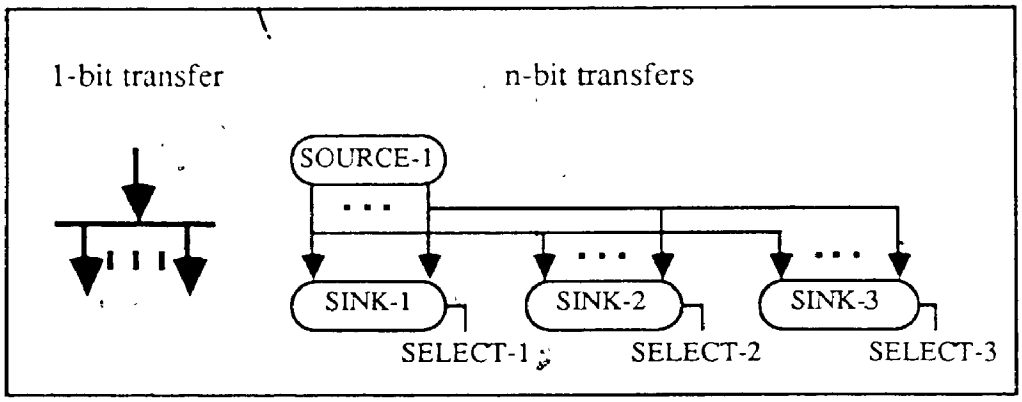
- straight leads (a),
- wired-broadcast tree (b),
- bus/mux (c).

The algorithm developed as part of this thesis partitions the list of TV's, with each partition being assigned to one of these primitives.

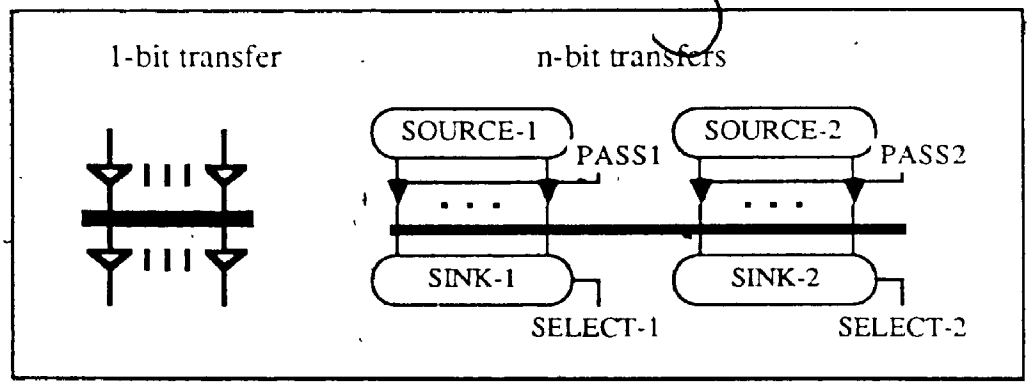
(a) **A SET OF SIMPLE WIRES**



(b) **A WIRED-BROADCAST TREE**



(c) **A BUS (MUX)**



sociated with the straight leads, as wire from one source is shared among many destinations.

A mux/bus (Figure 3.1.3(c)) has leads shared among several sources, and a set of pass transistors for each source. However, only one transfer can occur at a time.

3.2 Other Approaches

This section presents some existing approaches and solutions to the problem of allocating minimal interconnect. These are:

1) **FACET** System (Section 3.2.1):

   This RTL synthesis system was developed at Carnegie-Mellon University. It uses a modified clique partitioning approach to allocate interconnect.

2) **HAL** System (Section 3.2.2):

   This RTL synthesis package, was developed at Carleton University and Bell Northern Research. It uses both a heuristic and clique partitioning algorithm to allocate interconnect.

3) **CHIPPE** System (Section 3.2.3):

   This system, developed at the University of Illinois, generates RTL circuits using a knowledge based approach.

4) **REAL** System (Section 3.2.4):

   This system uses the *left-edge* algorithm to perform register allocation. The possibility of using this same approach to allocate interconnect will be explored.

A comparison of these solutions and **BAL**, the bus allocation algorithm developed as part of this thesis, will be presented in Section 4.4, once the **BAL** algorithm is described (Section 4.1-4.3).

3.2.1 Modified Clique Partitioning Approach

Tseng and Siewiorek use a *clique partitioning* like approach to minimize interconnect in their data path synthesis system, **FACET** [TsSi86][TsSi83]. The bus minimization problem is represented as follows:

> Each transfer variable is represented as a node in a graph, while the combinability of two transfers is represented by an edge between two nodes (Figure 3.2.1). Two transfers are combinable if they occur in different time steps, or if they have the same source. The graph is partitioned into a near minimum number of cliques. Each clique represents a bus.

The general idea of clique partitioning is to divide the graph into a minimum number of disjoint clusters (cliques). A clique can loosely be defined as a non-empty collection of nodes in which each node is connected to every other node in that clique. As well, none of the nodes appear in any other cliques in the graph G (Figure 3.2.2). Finding the minimum number of cliques in a graph is an *NP complete* problem [TsSi86].

Tseng and Siewiorek use a number of notions to direct the partitioning task to reduce the complexity of the algorithm, which yields a near minimum number of cliques. These notions are:

- divide and conquer,
- neighbourhood property,
- transitive property.

The notion of divide and conquer is used to avoid merging pairs of nodes randomly. The nodes of the graph G are divided into several categories, which can be defined as groups of two or more nodes that have either the same source or destination. Only nodes from the same category will be selected to be merged. The nodes from graph G, that are in one of these categories, will form the graph G1 (Figure 3.2.3(a)).

Figure 3.2.1 **BUS MINIMIZATION INTO THE CLIQUE PARTITIONING PROBLEM**

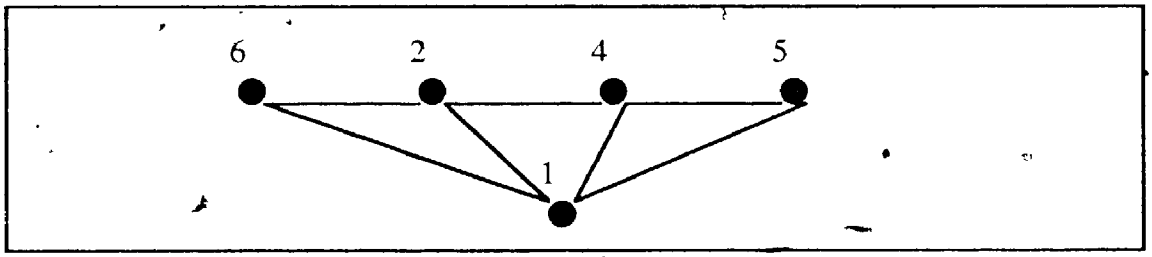
The bus minimization problem can be formulated into the clique partitioning problem. Each transfer variable in the TV list (a) is represented as a node in the graph 'G' (b). Two TV's , nodes, can be combined (ie- grouped together) if they either occur in different cycles, or have the same source. This situation is represented as an edge (line) between two nodes in the graph 'G'.

(a) **TRANSFER VARIABLE LIST**

INDEX	SOURCE	SINK	STEP
1	V1	V12	1
2	V1	ALU1.IN1	1, 3
3	V2	ALU1.IN2	1, 3, 4
4	V3	ALU1.IN1	2
5	V3	ALU2.IN1	3, 4
6	V4	ALU2.IN2	2
7	V5	ALU2.IN2	3
8	V5	ALU3.IN2	3
9	V6	ALU1.IN2	2
10	V10	ALU3.IN1	3
11	V12	ALU1.IN1	4
12	ALU1.OUT	V2	2, 3, 4
13	ALU1.OUT	V3	1
14	ALU2.OUT	V1	4
15	ALU2.OUT	V3	3
16	ALU2.OUT	V5	2
17	ALU3.OUT	V10	3

(b) **GRAPH 'G': GRAPHICAL REPRESENTATION**

Only a partial graph of the TV's listed above is shown, for clarity. The complete graph is shown in "matrix" form in (c).



(c) GRAPH 'G': MATRIX REPRESENTATION

This figure shows the graph 'G' , from Figure 3.2.1(b) in a matrix representation.

LEGEND: ■ - the two TV's i-j cannot be merged.
□ - the two TV's i-j can be merged, but they share neither source nor destination.
□ - the two TV's i-j can be combined, and they share either source or destination.

Of Common Neighbours Between i & j = Number of □ or □ Spaces Common to both i & j.

Of Edges Deleted if i & j are merged = # Of □ & □ in i + # Of □ & □ in j - # Common Neighbours - 1.

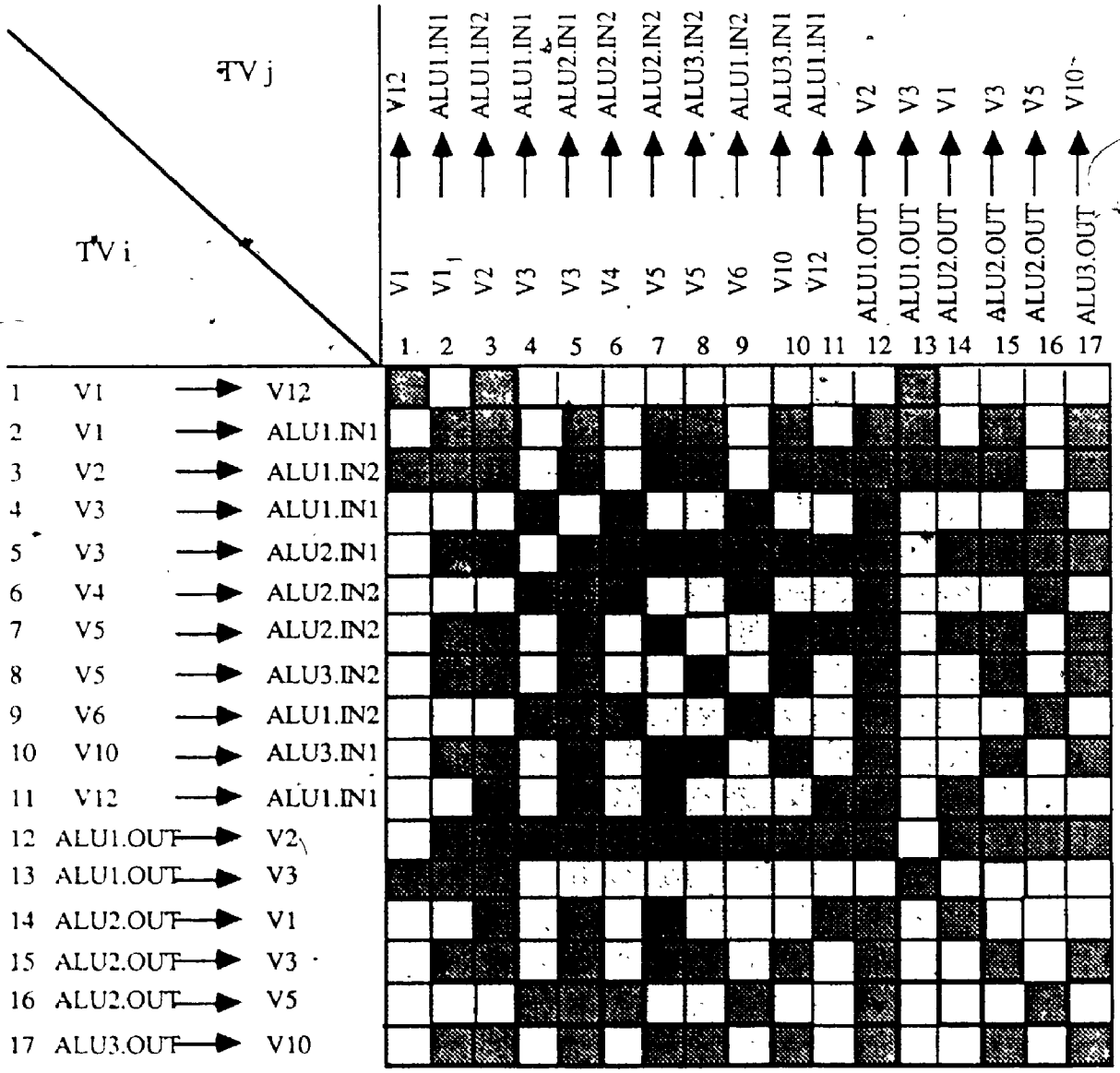
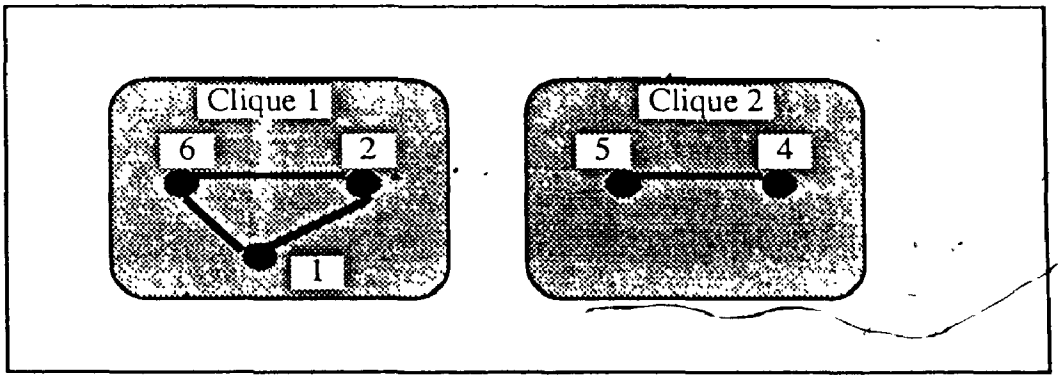


Figure 3.2.2 **CLIQUE PARTITIONING**

Clique partitioning divides a graph into a minimum number of cliques. A clique is a non-empty set of nodes, in which each node in the clique is:

- connected to every other node in that clique,
- only in that one clique (ie- a node can only be in one clique at a time)

(a) **MINIMUM # OF CLIQUES FOR FIGURE 3.2.1(b)**



(b) **ANOTHER MINIMUM # OF CLIQUES**

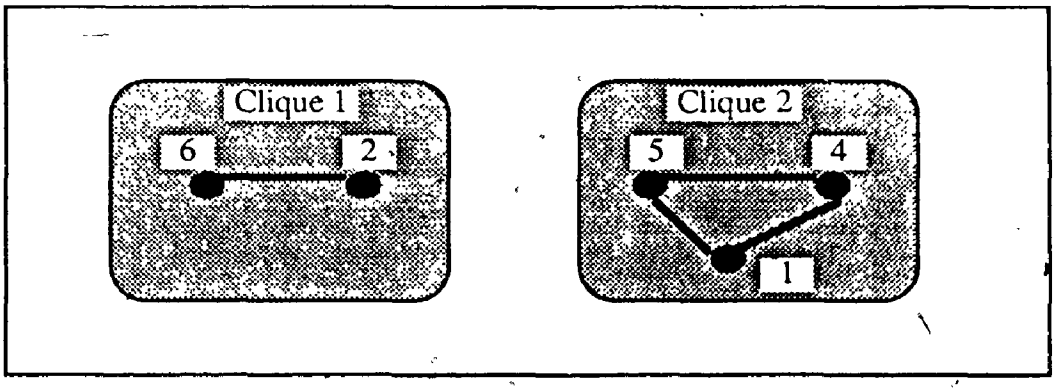
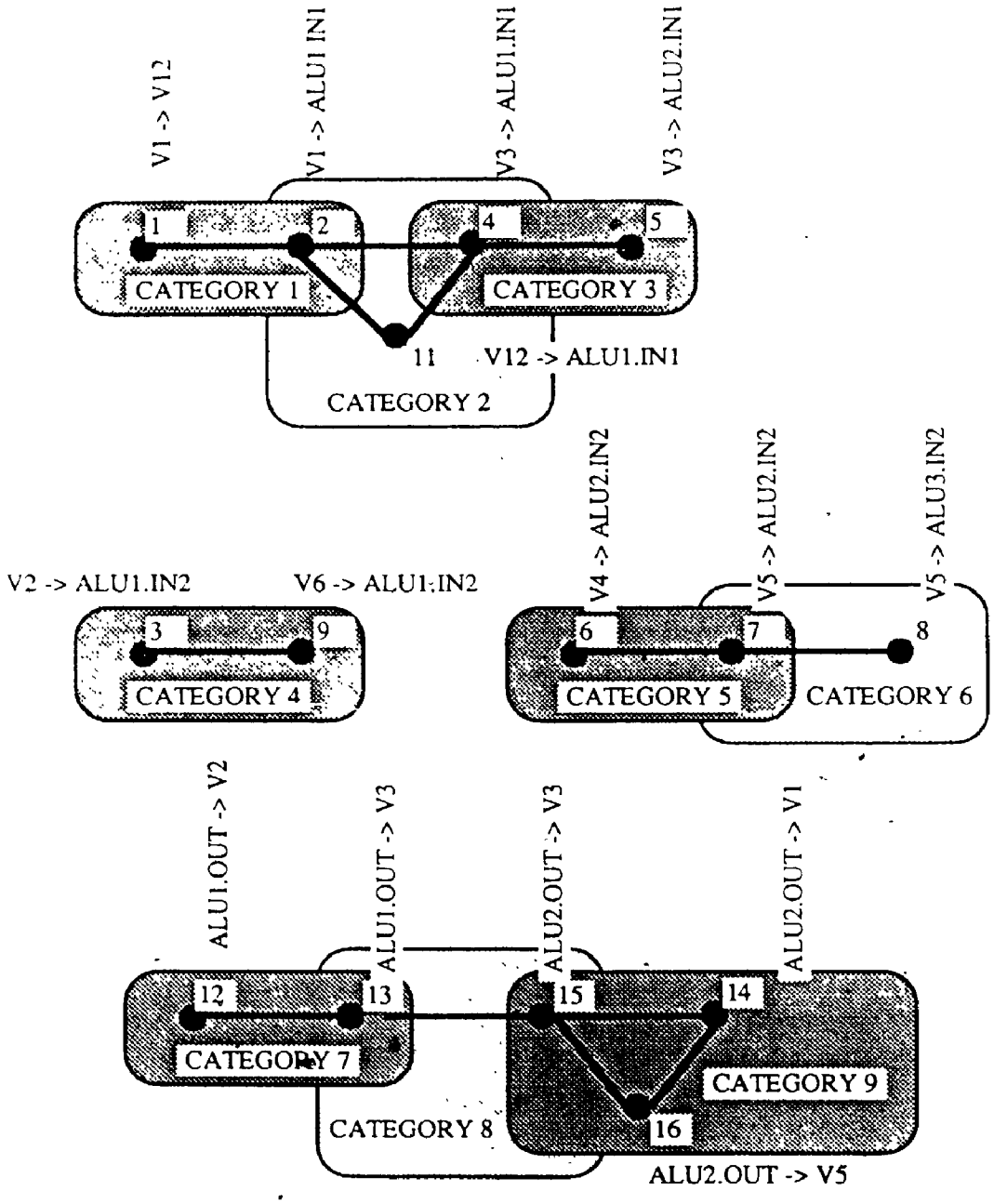


Figure 3.2.3 **CATEGORIES AND NEIGHBOURHOODS**

(a) GRAPH 'G1': CATEGORIES OF GRAPH 'G'

G1 is the subgraph of G. It contains only transfers which share sources or destinations with another transfers, and is divided into categories. A category is defined as a group of two or more nodes with the same source or destination. The nodes shown below are also shown in the matrix form of graph G (Figure 3.2.1c).



(b) NEIGHBOURHOOD PROPERTY

This table illustrates the neighbourhood property which is used to determine which pair of nodes from the graph G1 (graph of categories) should be merged first. The pair with the most common neighbours in G is selected. If there is more than one pair with the most common neighbours, then that pair which would delete the fewest edges in G is selected. The number of common neighbours can easily be determined by using the formula shown in the graph G, in Figure 3.2.1(c).

Pair '14, 16' would be selected first since it has the most number of common neighbours and would delete the fewest edges.

CATEGORY IN GRAPH G1 (Fig. 3.2.3a)	COMMON ELEMENT	EDGE (NODE PAIR)	NO. OF COMMON NEIGH-BOURS (Fig. 3.1.2c)	NO. OF EDGES DELETED (Fig. 3.1.2c)
1	V1	1, 2	6	*
2	ALU1.IN1	2, 4	3	*
		2, 11	5	*
		4, 11	7	15
3	V3	4, 5	2	*
4	ALU1.IN2	3, 9	0	*
5	ALU2.IN2	6, 7	3	*
6	V5	7, 8	6	*
7	ALU1.OUT	12, 13	0	*
8	V3	13, 15	2	*
9	ALU2.OUT	14, 15	6	*
		14, 16	7	14
		15, 16	4	*

* not needed, therefore not calculated.

The neighbourhood property is used to decide which pair of nodes, from the graph G1 (graph of categories), should be combined first, while the transitive property is used to move nodes, under special circumstances from one category to another. These properties will be illustrated, later, with an example.

A description of the modified clique partitioning algorithm as applied to the interconnect minimization problem is:

Step 1: A compatibility graph G is constructed. All transfer variables that are compatible have an edge between them (Figure 3.2.1(c)).

Step 2: All edges in G that connect two nodes having the same source or destination are collected to form the graph G1 (Figure 3.2.3(a)).

Step 3: From the graph, G1, the edge with the largest profit is combined. An edge is determined to have the largest profit if its nodes have the most number of common neighbours in G. If there is a tie, the nodes that would cause the fewest edges to be deleted are selected. In the example, Figure 3.2.3(b), two edges, *4-11* and *14-16*, have the same best number of common neighbours. Thus the number of edges deleted must be determined for those two edges. Since edge *14-16* will have the fewest edges deleted, nodes 14 and 16 are selected to be merged first.

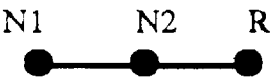
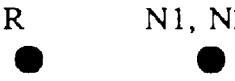
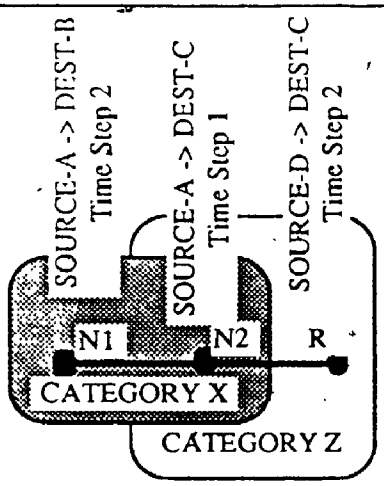
Step 4: The graphs G and G1 are then updated in the following manner. (Let *N1* and *N2* be the nodes that are to be combined, determined from Step 3.)

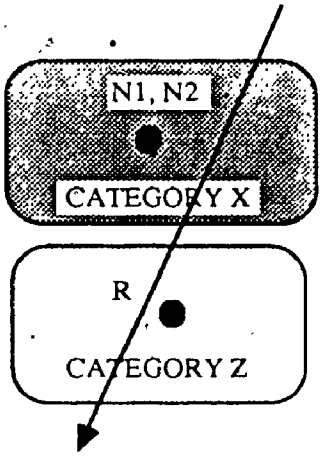
i) If a node, R, is connected to either of the nodes N1 or N2, but not both, then the edge *N1-R* or *N2-R* is deleted from the graphs G and G1 (Figure 3.2.4).

ii) If a node, S, is connected to both nodes N1 and N2, then one of those edges is deleted in G and G1 (Figure 3.2.5(a)). As well, if the transfer variable, at node R, has source or destination in common with the transfer variables N1 or N2, then edge *S-N1* or edge *S-N2* is moved into that category in graph G1 (Figure 3.2.5(b)). (This is an example of the

Figure 3.2.4 **ILLUSTRATION OF STEP 4 (i)**

If node R is connected to either of the nodes N1 or N2, but not to both, then the edge N1, R (as shown in a) or N2, R (as shown in b) should be deleted.

<table>
<tr><th>GRAPH</th><th>CASE</th><th>BEFORE</th><th>AFTER</th></tr>
<tr><td rowspan="2">G</td><td>b</td><td> 
</td><td> 
</td></tr>
<tr><td>a</td><td> 
</td><td> 
</td></tr>
<tr><td>G1
(CATEGORIES)</td><td>b</td><td> 

R cannot be merged with N1, N2 because of time step conflicts.</td><td> 

Both these categories would now be removed because they contain only one node.</td></tr>
</table>

use of the transitive property.)

iii) The edge between N1 and N2 is deleted in graph G and G1. Figures 3.2.6 and 3.2.7 illustrate the result of merging nodes 14 and 16 on graph G and G1, respectively.

iv) The common number of neighbours in graph G1 are updated.

Step 5: Steps 3 and 4 are repeated until either the category, or G, or G1 is empty.The nodes 14 and 16, that have been chosen to be merged earlier are in category 9. This category must be empty before nodes in another category can be merged. (In otherwords, as many transfers as possible are assigned to a bus, before another bus is created.) Since node 15 is the only node remaining in this category, it is chosen to be merged next (Figures 3.2.8 and 3.2.9).

Step 6: Steps 3, 4, and 5 are repeated until G1 (all categories) or G is empty.

The interconnection variables are eventually partitioned into eight groups (Figure 3.2.10):

1) [13,14,15,16],

2) [1,2,4,11],

3) [6,7,8],

4) [3,9],

5) [5],

6) [10],

7) [12],

8) [17].

3.2.2 Heuristic and Clique Partitioning Approach

The **HAL** data path synthesis system uses both heuristics and clique partitioning to minimize the amount of interconnect generated during the bus allocation phase. Generally, one mux is assigned to each unique destination. Clique partitioning is then applied to merge these muxes, with heuristics being used to help reduce

Figure 3.2.5 **ILLUSTRATION OF STEP 4 (ii)**

If node S is connected to both nodes N1 and N2, then one of the edges N1, S or N2, S must be deleted (shown in a). The edge N1, N2 is also deleted.

If the TV at node S has source or destination in common with the TV's of N1 or N2, then edge S, N1, N2 should be moved into the same category of N1, N2 (as shown in b).

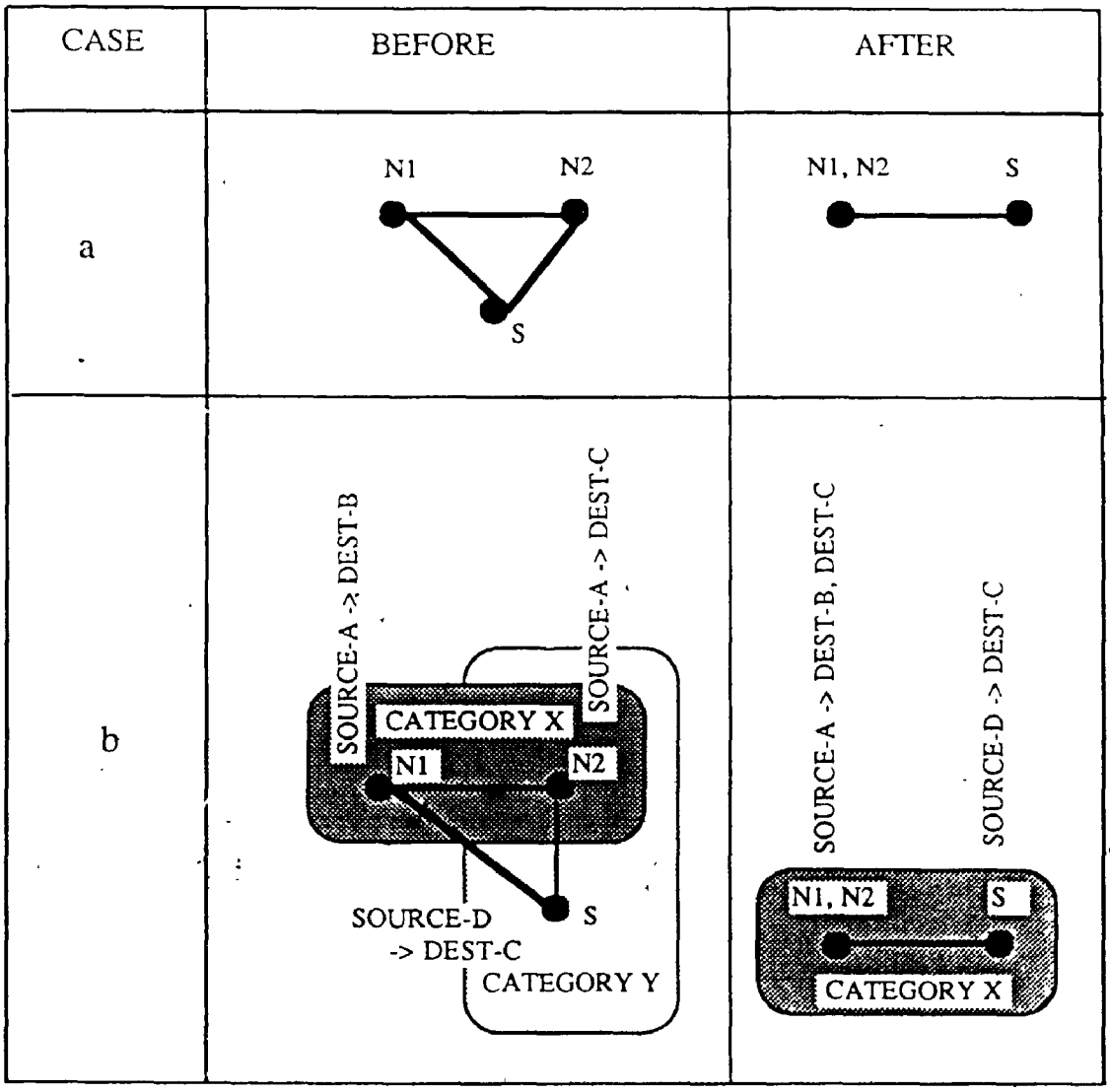


Figure 3.2.6 **GRAPH 'G' WITH NODES 14 AND 16 MERGED**

After nodes 14 & 16 are merged, the graph G is updated. In partucular:

■ merged with anything yields ■

□ merged with □ yields □

□ merged with □ or □ yields □

- delete row 16 and column 16,
- rename row and column 14-16.

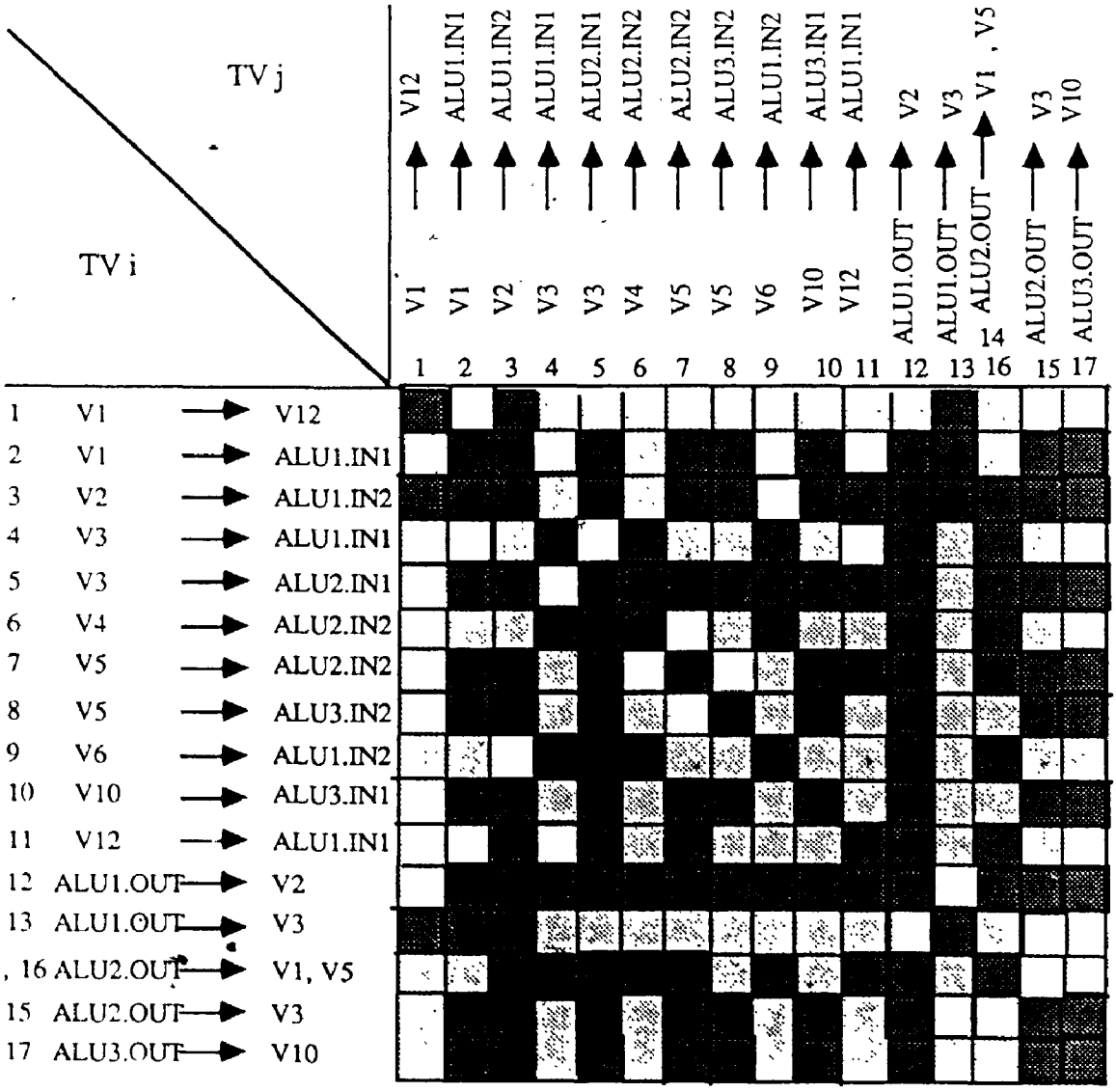


Figure 3.2.7 CATEGORY 9 AFTER MERGING NODES 14 AND 16

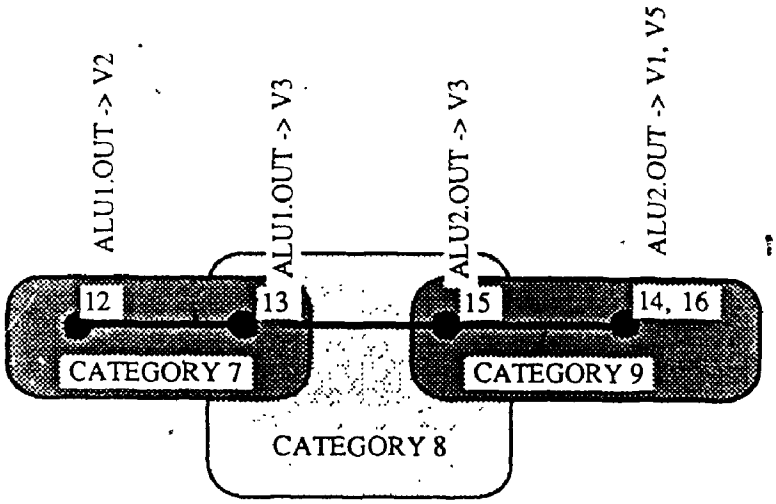


Figure 3.2.8 CATEGORY 9 AFTER MERGING NODES 15 AND 14, 16

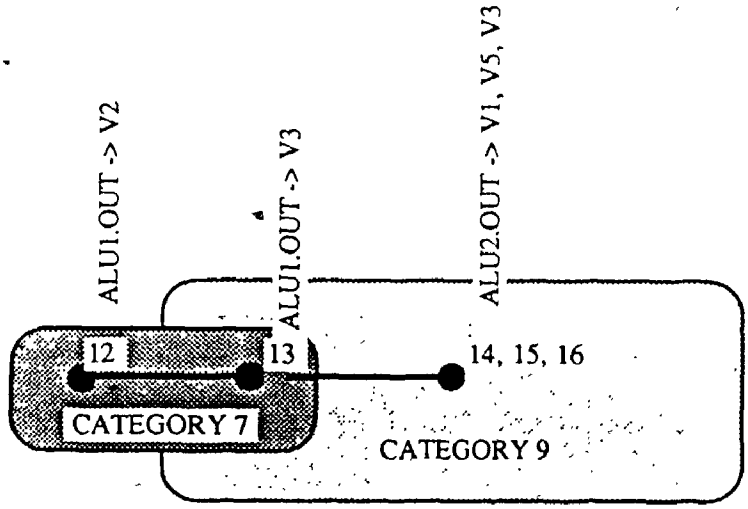


Figure 3.2.9 **GRAPH 'G' WITH NODES 15 AND 14, 16 MERGED**

- moving all ■ from row 15 into the corresponding row 14, 16
- moving all ■ from column 15 into the corresponding spot in column 14, 16
- moving all □ from row 15 into any □ in row 14, 16
- moving all □ from column 15 into any □ in column 14, 16
- delete row 15 and column 14, 16
- rename row and column 14, 16 as row and column 14, 15, 16.

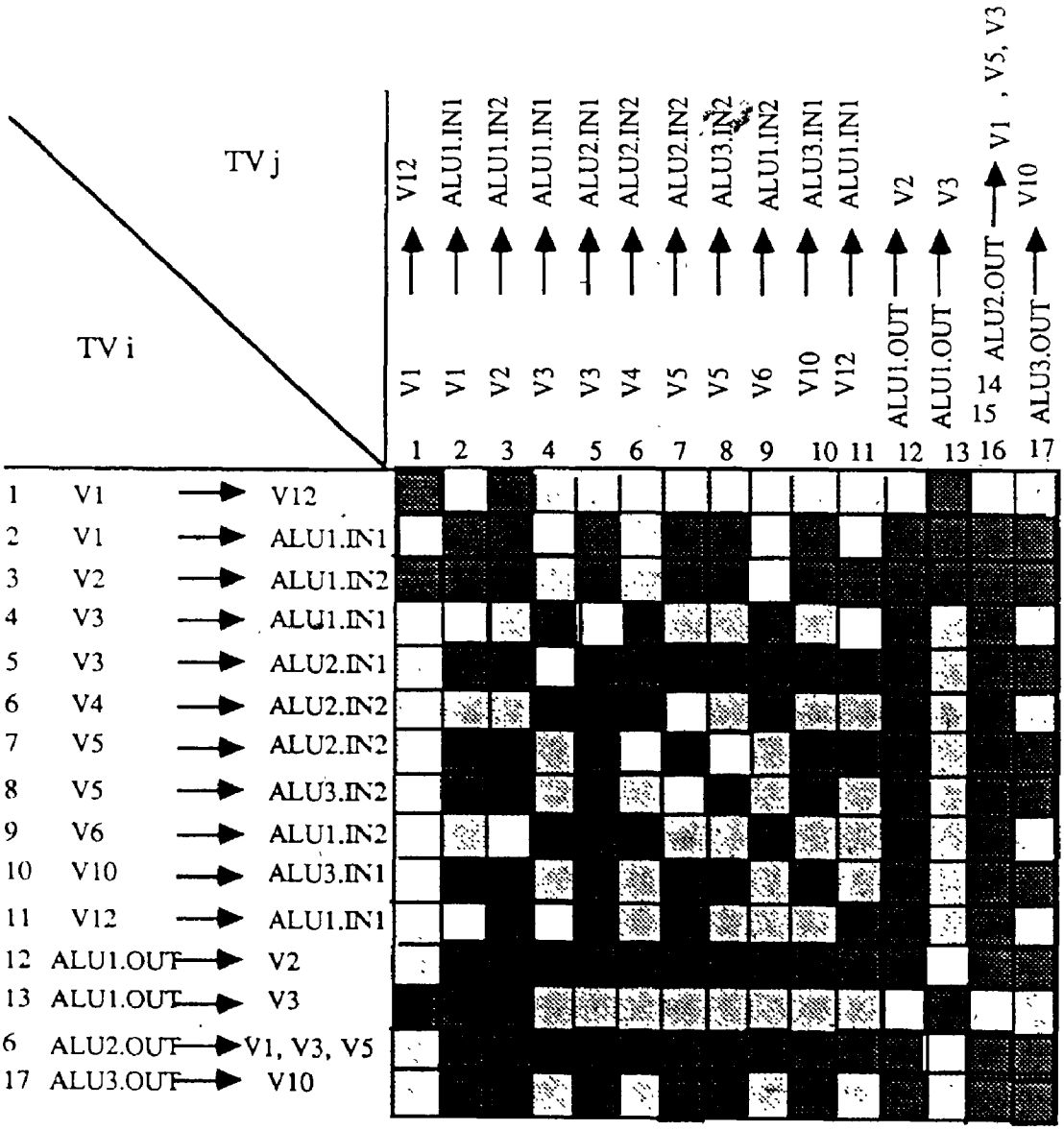
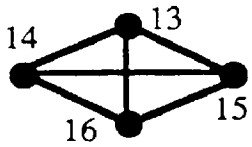
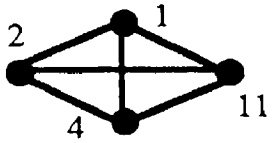
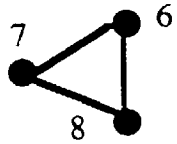
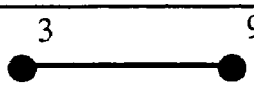


Figure 3.2.10 **FINAL RESULTS**

This figure shows the clique partitioned graph G.

CLIQUE	TRANSFER VARIABLE INDEX	SOURCE → SINK	TIME STEP
	13	ALU1.OUT → V3	1
	14	ALU2.OUT → V1	4
	15	ALU2.OUT → V3	3
	16	ALU2.OUT → V5	2
	1	V1 → V12	1
	2	V1 → ALU1.IN1	1,3
	4	V3 → ALU1.IN1	2
	11	V12 → ALU1.IN1	4
	6	V4 → ALU2.IN2	2
	7	V5 → ALU2.IN2	3
	8	V5 → ALU3.IN2	3
	3	V2 → ALU1.IN2	1,3,4
	9	V6 → ALU1.IN2	2
5	5	V3 → ALU2.IN1	2,3,4
10	10	V10 → ALU3.IN1	3
12	12	ALU1.OUT → V2	2,3,4
17	17	ALU3.OUT → V10	3

the search space.

The **HAL** approach is somewhat similar to that of the **FACET** system (Section 3.2.1) in that they both use clique partitioning. The differences are that:

1) **HAL** forms the categories (muxes) from transfers with the same destination, while **FACET** forms categories from transfers with either the same source or destination,
2) **HAL** uses clique partitioning to merge whole categories, while **FACET** uses clique partitioning to merge transfers within categories.

The steps required to minimize the interconnect are:

Step 1: Assign a mux to each unique destination. All transfers of data will then pass through a mux. The result of this step on the TV's listed in Figure 3.2.11 is shown in Figure 3.2.12.

Step 2: Determine the *compatible* muxes for each mux assigned in step 1 (Figure 3.2.13). Two muxes are considered to be *compatible* if:

- one or more of their inputs are common to both,
- the remaining, non-common, inputs of both muxes are disjoint in time (i.e.- the inputs do not transfer their contents at the same time).

Formulate the problem of merging the *compatible* muxes into that of clique partitioning. Each node in the graph G will represent a mux, while each edge will represent the compatability between two muxes (Figure 3.2.14).

Step 3: Determine the cost of the interconnect for every combination, or way, of clique partitioning the Graph G (Figure 3.2.15) (See Section 3.2.1 for a brief explanation on clique partitioning). The cost of the interconnect can be determined using the costing algorithm developed in Section 4.1.2.

Step 4: Select the combination of merged muxes that yielded the lowest interconnect cost during step 3 (Figure 3.2.16).

The clique partitioning problem is NP complete and every combination must

Figure 3.2.11: **TV'S TO BE ASSIGNED BY 'HAL'**

TRANSFER VARIABLES			
INDEX	SOURCE	DESTINATION	TIME STEPS
1	R16	MULT1.2	2
2	R16	ADD1.1	4
3	R17	MULT2.1	1
4	R17	MULT1.1	3
5	R17	SUB1.1	4
6	R18	MULT2.1	2, 3
7	R18	ADD1.2	4
8	R21	SUB1.2	3,4
9	R21	MULT2.2	2
10	R24	MULT1.2	1
11	R24	ADD1.1	1
12	R24	COMP1.2	2
13	C1	MULT2.2	1, 3
14	C1	MULT1.2	3
15	C1	ADD1.2	1
16	C2	MULT1.1	1, 2
17	C6	COMP1.1	2
18	MULT1	R18	1,2,3
19	MULT2	R21	1,2,3
20	ADD1	R16	4
21	ADD1	R24	1
22	SUB1	R17	3,4

Figure 3.2.12: **STEP1: ASSIGN ONE MUX FOR EACH UNIQUE DESTINATION**

This figure shows the results of step 1 on the TV list of Figure 3.2.11. Each unique destination has been assigned a mux. All transfers of data to that destination will pass through that mux. For example, 'R1', 'R24', 'C1' all pass through 'Mux-1' to get to destination 'MULT1.2'. Note that even unique source to unique destination TV's are assigned to muxes, as well.

MUX NO.	INDEX NO.	INPUT(S)	OUTPUT	TIME STEPS
1	1	R16	MULT1.2	2
	10	R24		1
	14	C1		3
2	4	R17	MULT1.1	3
	16	C2		1, 2
3	3	R17	MULT2.1	1
	6	R18		2, 3
4	9	R21	MULT2.2	2
	13	C1		1, 3
5	7	R18	ADD1.2	4
	15	C1		1
6	2	R16	ADD1.1	4
	11	R24		1
7	5	R17	SUB1.1	4
8	8	R21	SUB1.2	3,4
9	17	C6	COMP1.1	2
10	12	R24	COMP1.2	2
11	18	MULT1	R18	1,2,3
12	19	MULT2	R21	1,2,3
13	20	ADD1	R16	4
14	21	ADD1	R24	1
15	22	SUB1	R17	3,4

Figure 3.2.13 **STEP 2: DETERMINE COMPATIBLE MUXES**

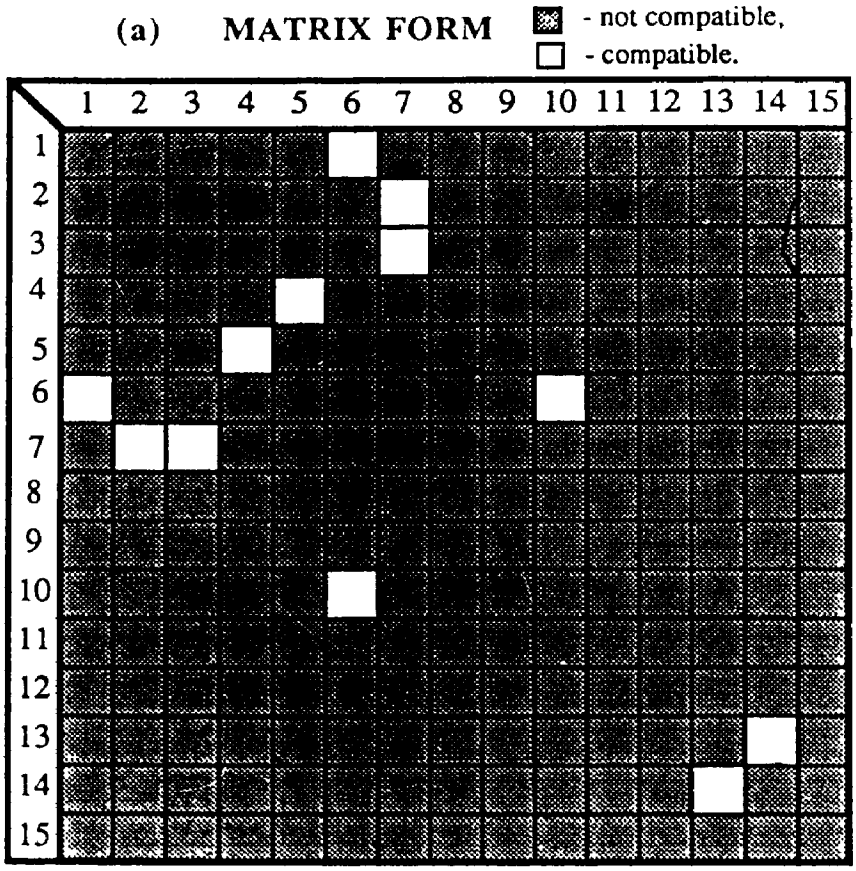
This figure illustrates the result of determining the compatible muxes for each mux in Figure 3.2.12. Two muxes can be considered as compatible (and hence can be merged) if:

- one or more of their inputs is common to both,
- the remaining, non-common, inputs are disjoint in time (ie- do not transfer at the same time.

MUX NO.	INDEX NO.	INPUT(S)	OUTPUT	TIME STEPS	COMPATIBLE MUXES	
					MUX NO.	COMMON INPUTS
1	1	R16	MULT1.2	2	6	R16, R24
	10	R24		1		
	14	C1		3		
2	4	R17	MULT1.1	3	7	R17
	16	C2		1, 2		
3	3	R17	MULT2.1	1	7	R17
	6	R18		2, 3		
4	9	R21	MULT2.2	2	5	C1
	13	C1		1, 3		
5	7	R18	ADD1.2	4	4	C1
	15	C1		1		
6	2	R16	ADD1.1	4	1	R16, R24
	11	R24		1	10	R24
7	5	R17	SUB1.1	4	2, 3	R17
8	8	R21	SUB1.2	3,4	-	
9	17	C6	COMP1.1	2	-	
10	12	R24	COMP1.2	2	6	R24
11	18	MULT1	R18	1,2,3	-	
12	19	MULT2	R21	1,2,3	-	
13	20	ADD1	R16	4	14	ADD1
14	21	ADD1	R24	1	13	ADD1
15	22	SUB1	R17	3,4	-	

Figure 3.2.14 GRAPH G OF COMPATIBLE MUXES

(a) MATRIX FORM



(b) GRAPH FORM

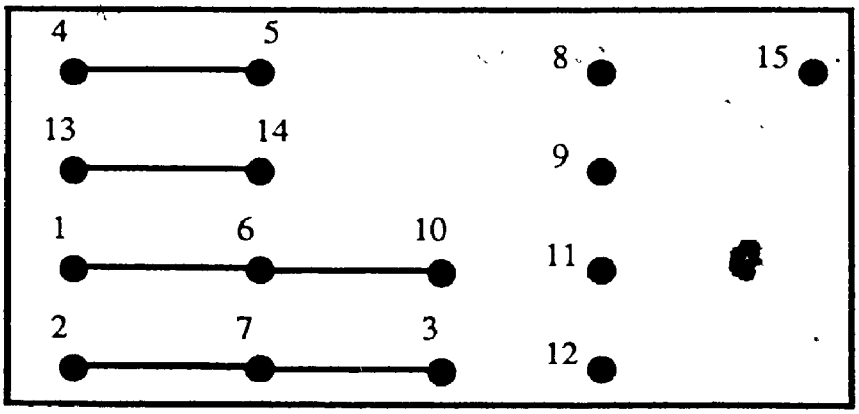
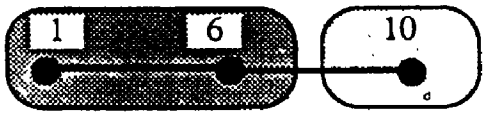
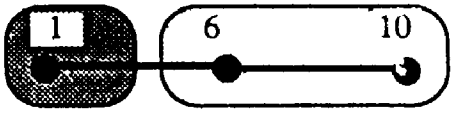
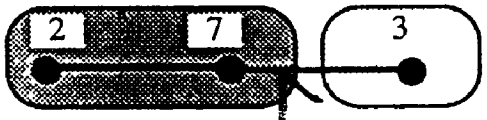
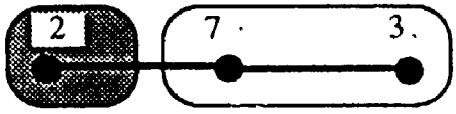


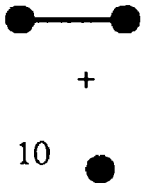
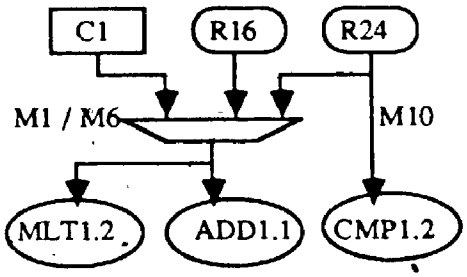
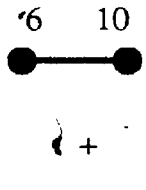
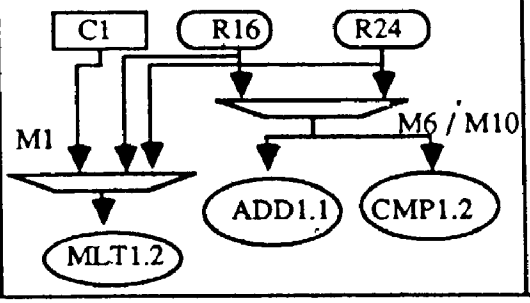
Figure 3.2.15 **STEP3: COST EVALUATION FOR EVERY WAY OF PARTITIONING GRAPH G**

There are only two decisions to be made when partitioning the graph G, shown in (a). The "cost" of the interconnect for each way of partitioning the graph is determined , using the algorithm developed in section 4.1.2. The partitions that produce the least expensive interconnectis then chosen.

(a) PARTITIONING THE GRAPH

	CHOICE # 1	CHOICE # 2
DECISION # 1		
DECISION # 2		

(b) DECISION # 1: COST EVALUATION

MUX #'S	INPUT(S)	OUTPUT(S)	CIRCUIT	COST
	R16, R24, C1 R24	MULT1.2, ADD1.1 COMP1.2		9.5
	R16, R24, C1 R16, R24	MULT1.2 ADD1.1, COMP1.2		14

(c) DECISION # 2: COST EVALUATION

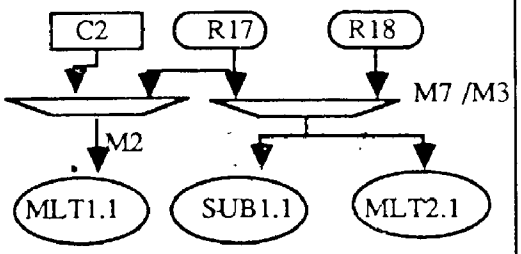
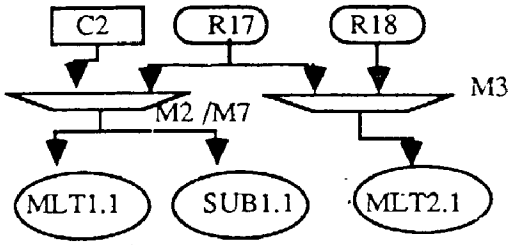
MUX #'S	INPUT(S)	OUTPUT(S)	CIRCUIT	COST
7 3 + 2	R17, R18 R17, C2	SUB1.1 MULT2.1 MULT1.1		13.5
2 7 + 3	R17, C2 R17, R18	MULT1.1, SUB1.1 MULT2.1		13.5

Figure 3.2.16 **STEP 4: CHOOSE COMBINATION THAT YIELDS LOWEST INTERCONNECT COST**

MUX NO.	INDEX NO.	INPUT(S)	OUTPUT	TIME STEPS
M1 / M6	1	R16	MULT1.2	2
	2	R16	ADD1.1	4
	10	R24	MULT1.2	1
	11	R24	ADD1.1	1
	14	C1	MULT1.2	3
M2 / M7	4	R17	MULT1.1	3
	5	R17	SUB1.1	4
	16	C2	MULT1.1	1, 2
M3	3	R17	MULT2.1	1
	6	R18	MULT2.1	2, 3
M4 / M5	7	R18	ADD1.2	4
	9	R21	MULT2.2	2
	13	C1	MULT2.2	1, 3
	15	C1	ADD1.2	1
M8	8	R21	SUB1.2	3,4
M9	17	C6	COMP1.1	2
M10	12	R24	COMP1.2	2
M11	18	MULT1	R18	1,2,3
M12	19	MULT2	R21	1,2,3
M13 /M14	20	ADD1	R16	4
	21	ADD1	R24	1
M15	22	SUB1	R17	3,4

be determined before the best can be selected. However, the author of **HAL** claims that because of the requirement for muxes to be *compatible* (disjoint in time and have at least one common input) the number of combinations that must actually be evaluated will be small [Paul88].

3.2.3 Knowledge Based Approach

The **CHIPPE** System [BrGa87,PaGa87] uses a knowledge based approach to automatically design register-transfer level circuits. A design critic (an expert system) guides the design process, evaluates generated designs, and selects strategies to improve the design. A designer (**Slicer** and **Splicer** modules) are responsible for generating the designs, under the direction of this critic (Figure 3.2.17).

The design critic accepts input from both the user and the automatic designer (**Slicer** and **Splicer**). The input from the user is in terms of constraints, while that from the **Slicer** and **Splicer** modules describes the generated circuit in terms of its operation (scheduled control data flow graph) and its components and connections (micro-architecture), respectively. The design critic, itself, is composed of two modules, a **Planner**, and an **Evaluator**.

The **Evaluator** determines the quality of a generated design in terms of the:

- area,
- cost,
- performance,
- component usage frequency,
- component overlap,
- and others.

The **Planner** is responsible for:

(a) Setting the proper design style, such as:

- sequencer style,
- number of addresses in each micro-instruction,

Figure 3.2.17: THE CHIPPE DESIGN SYSTEM

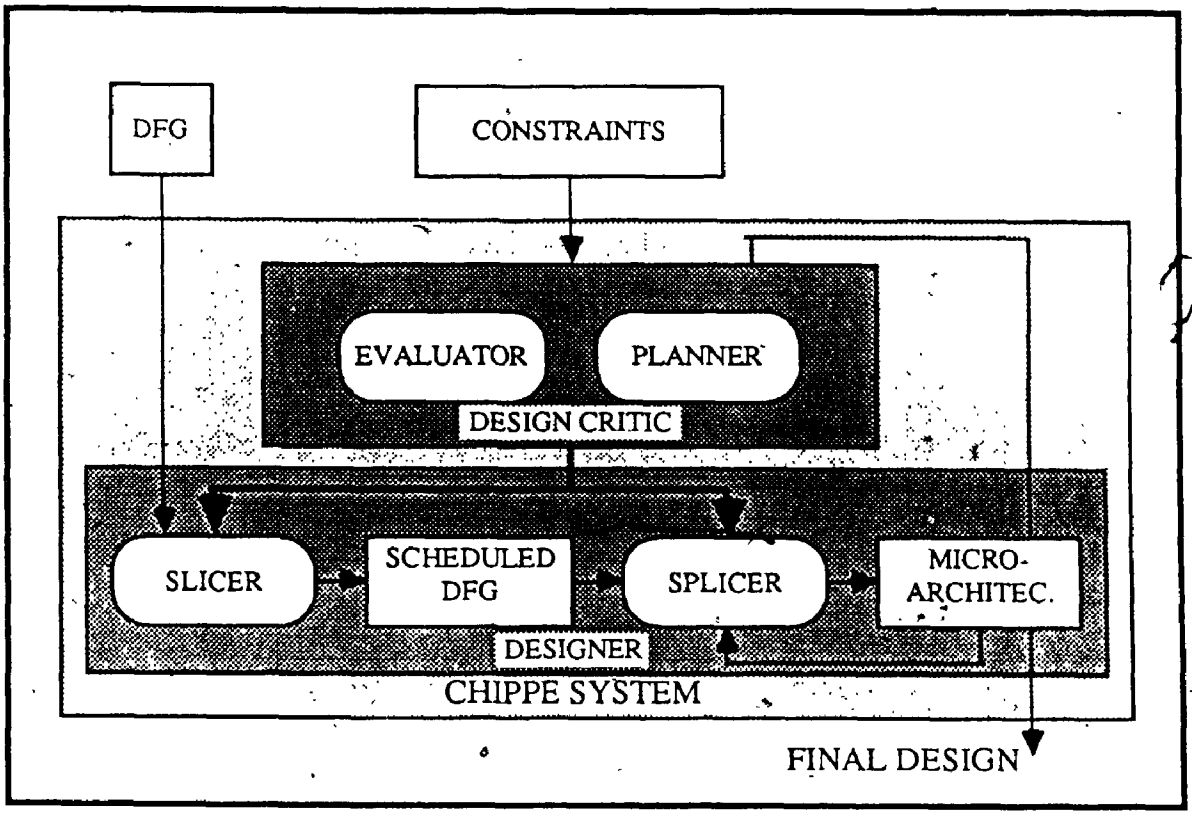
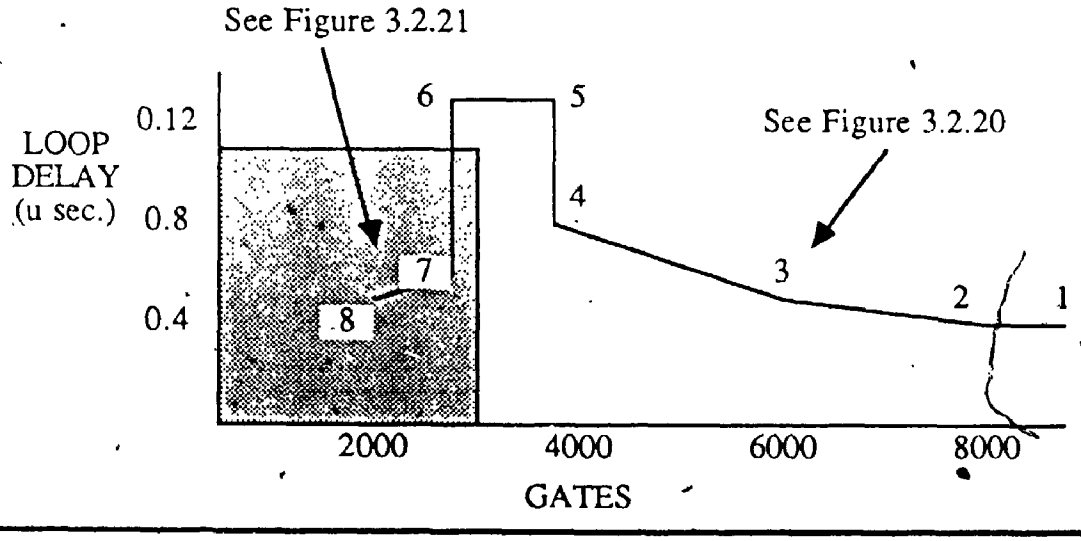


Figure 3.2.18: EVOLUTION OF 'CHIPPE' CREATED DESIGNS

This figure shows a typical evolution of a design by the CHIPPE AUTOMATIC DESIGN SYSTEM. The shaded box illustrates the area and performance goals that are to be acheived. The designs are ordered from 1-8. Design-1 was created first, the following designs were then created on the evaluation of previous designs. It is only as the design approaches the final global cost constraints that the interconnect is optimized.



- connectivity style,
- how many and what type of units are in data path,
- pipelining level,
- clock cycle.

(b) Setting the design strategy, such as:

- level of exploration costs,
- ranking trade-offs,
- optimization order.

(c) Re-defining the design plan, based on measurements from the Evaluator, in order to move the design closer to its goal.

The Slicer module transforms the control data flow graph (DFG that includes control information) into one that is scheduled, a state graph. Operations in the data flow sections are placed into specific machine states (control steps) in the state graph. This is done using an algorithmic method.

The Splicer module transforms the state graph, (i.e.- the scheduled DFG, into a register-transfer level circuit. It allocates operations to specific operators (adders, ALU's, etc.), data variables to registers, and data transfers to specific transfer paths (muxes, buses, etc.). It performs these three tasks simultaneously by making the assignments on a state by state basis. The assignments are made in four steps (Figure 3.2.18):

Step 1: Register to input bus connections:

The registers are placed in a sorted list, with multiple occurrences of the same register being removed. The registers are then taken, in their sorted order, and assigned to buses. If any of the registers have already been connected to buses during a previous state, then those register assignments will be tried first. Registers and buses must match in bit widths for a register to use a bus. Specific cost requirements guide any decisions made during this step.

Step 2: Input bus to function unit connections:

This is the most complex of the four steps, as it performs the actual operation-to-functional-unit bindings. If the input is tagged to an operation that already exists, then the connection is made. Otherwise, if it is the first input to a particular operation, then there may be a choice between structural components. A greedy strategy is used to assign the input, and operation, to a component that already exists (i.e.- a new add operation to an existing ALU), if it is not already busy, has proper bit widths and performance. Again, a cost function will drive this process.

Step 3: Function unit outputs to output buses:

This step follows the same algorithm as that of the first step.

Step 4: Output buses to registers:

It is during this step that the data variable to register assignments are made. As well, the data variables' *lifetime* (length of time that the data is required) is stored in a table associated with that register. This information will then be used in successive Step 4's. As with all the other steps, cost requirements guide the decisions.

Once the circuit has been designed, it is passed to the **Evaluator**, where measurements are made to determine if the circuit meets the specifications. If it doesn't, then the **Planner** decides how to improve the circuit. It may decide that certain portions of the circuit should be optimized, a new circuit be generated with more than one state considered simultaneously during the assignment phase, different structural units be used, or a new scheduled DFG be used. This process is an iterative one, with the circuit slowly approaching the goals. If a wrong choice is made at one point, the design critic can backtrack to a previous design, and take a different path.

Figure 3.2.18 shows a typical evolution of a design. The shaded box illustrates the area and performance goals that are to be achieved. The designs are ordered

Figure 3.2.19: CHIPPE DESIGN STEPS

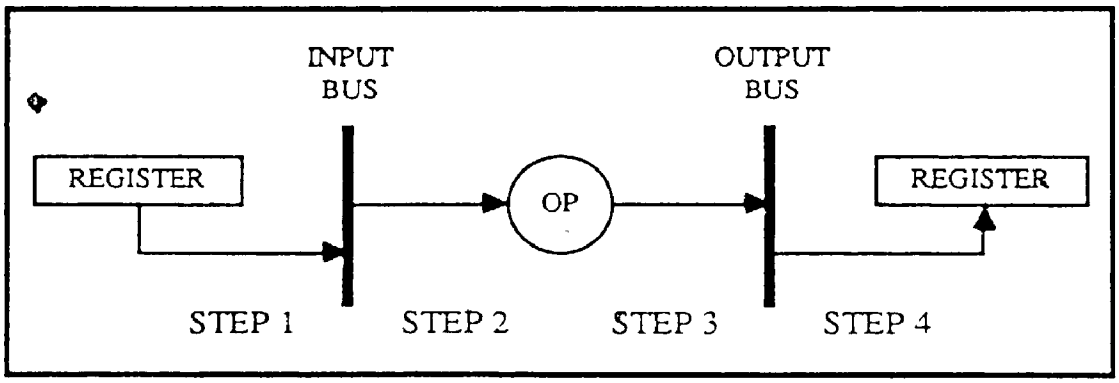


Figure 3.2.20: DESIGN AT STAGE 3

This figure shows the design at the pre-interconnect optimization stage. Since the constraints are not yet close to being met, a minimal interconnect has not been generated (note the number of muxes).

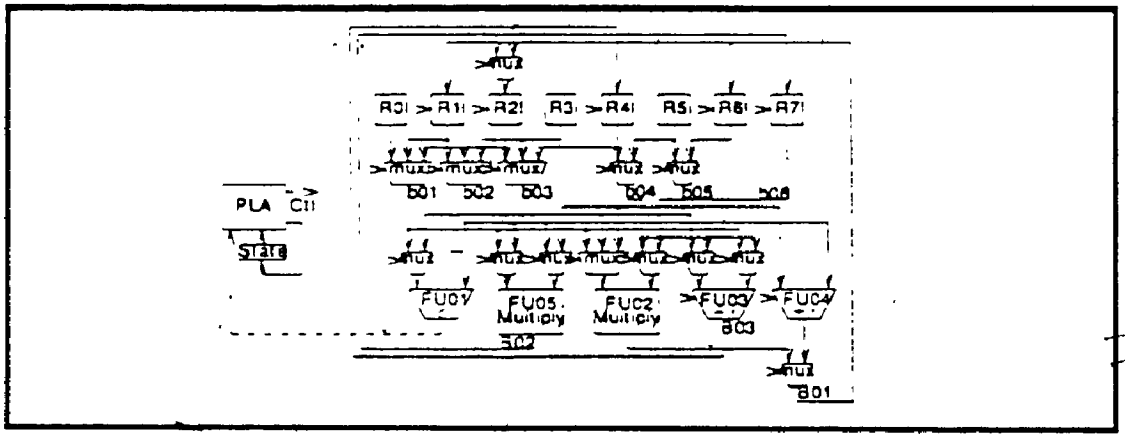
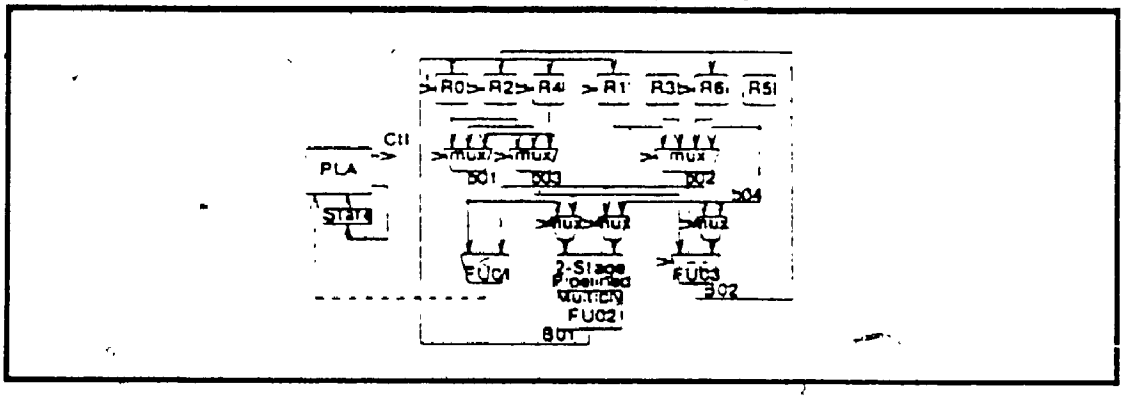


Figure 3.2.21: DESIGN AT STAGE 4

This figure shows the final design, with a Splicer optimized interconnect.



from 1-8. Design-1 was created first, the following designs were then created on the evaluation of previous designs. It is only as the design approaches the final stages (i.e.- gets close to global goals), that the interconnect is optimized. Figure 3.2.20 shows the design at the a pre-interconnect optimization stage. Since the constraints are not yet close to being met, a minimal interconnect has not been generated (note the number of muxes). Figure 3.2.21 shows the final design, with a Splicer optimized interconnect. It is not clear, from presently available references, how this optimization is actually performed.

3.2.4 Left-Edge Algorithm Approach

The left-edge algorithm is a well known CAD algorithm. It is best known in the area of channel routing, where it has been effectively used to perform track assignment. In this context, its goal is to allocate wire segments to tracks, so as to minimize the total number of needed tracks. It has been found to work well and is of order N^2 [KuPa87[HaSt71]]. It is this algorithm that has formed the basis for **REAL**, a register allocation program [KuPa87]. This section presents the approach used by **REAL** to assign data variables to registers, and illustrates why this approach is not entirely suitable for the for bus allocation task.

The register allocation problem can be formulated into that of the left-edge: The data variables (DV's), which are represented by the edges in the data flow graph (Figure 3.2.22), are listed in the *lifetime table* (Figure 3.2.23(a)). The information in this table indicates the time steps during which the variables must be stored (i.e.- *lifetime*) in terms of its *time of birth* (left edge) and *time of death* (right edge). A DV may be assigned to a register if its *lifetime* does not overlap with those of any other DV's previously assigned to that register.

The left-edge algorithm, as applied to register allocation is:

Step 1: Sort the data variables (DV's), from left to right, according to their *time of birth* (left edge) and *lifetimes* (length), as shown in Figure 3.2.23(b).

Step 2: Assign the first DV in the *lifetime table* to a register. This register then be-

Figure 3.2.22:

DATA FLOW GRAPH FOR THE LEFT EDGE EXAMPLE

This is the DFG used to illustrate the Left-Edge Algorithm as applied to register allocation. The data variables (DV's) that must be assigned are shown as nodes (ie [redacted]). They are also shown in the lifetime table in Figure 3.2.23(a).

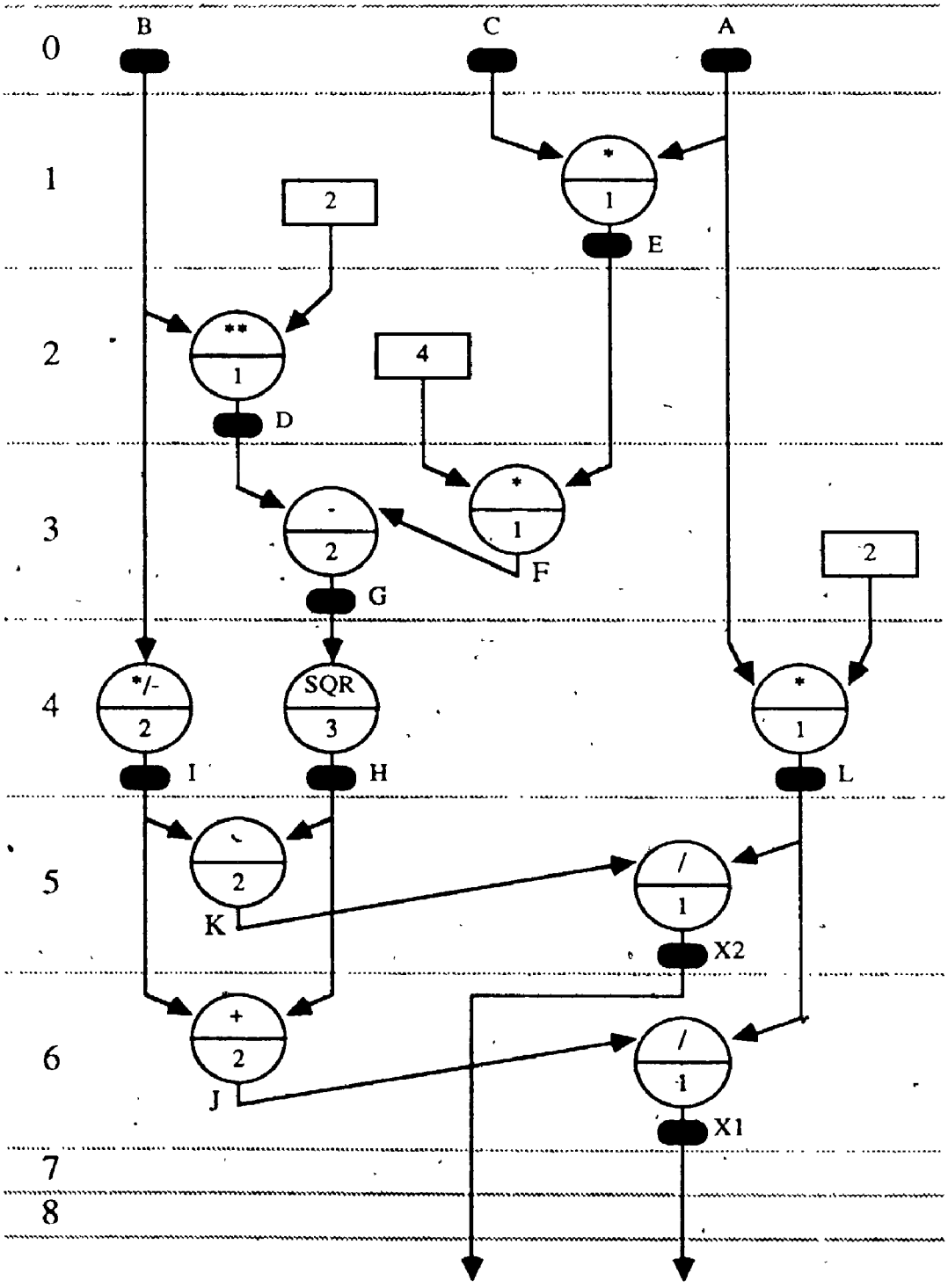
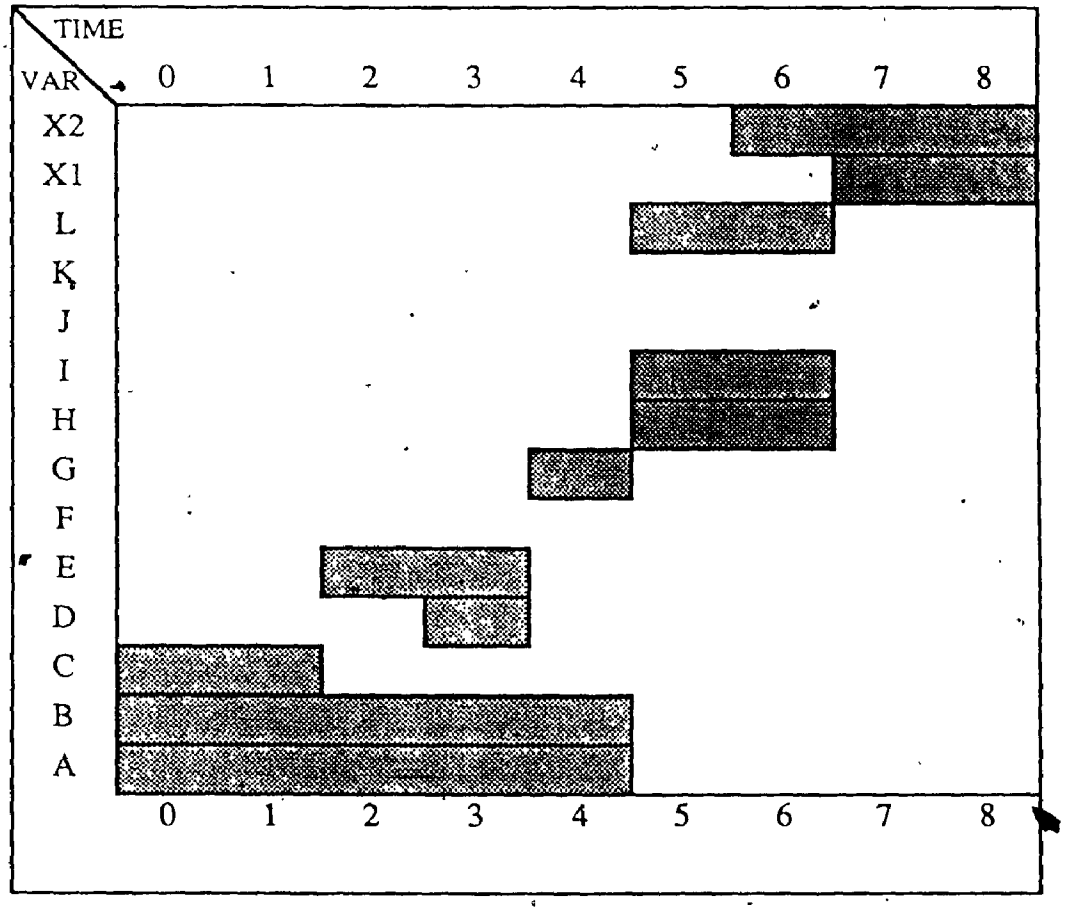


Figure 3.2.23 **LEFT-EDGE ALGORITHM EXAMPLE**

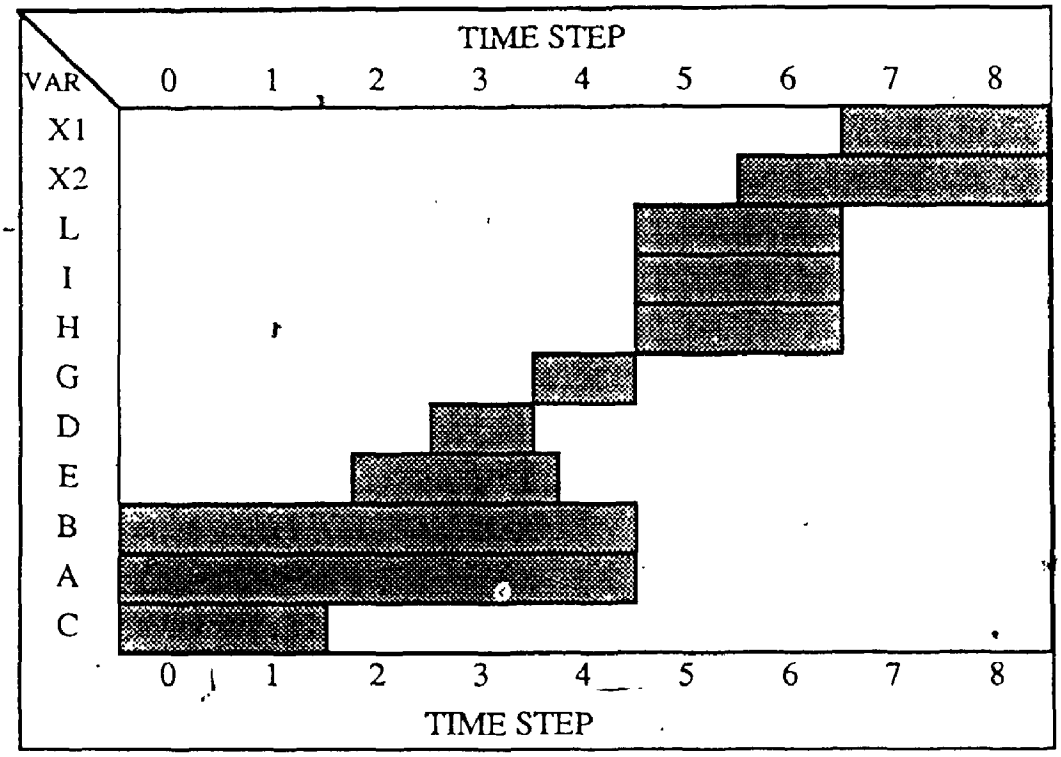
The lifetime table (a) shows the number of time steps for which each data variable (DV) must be stored. The left edge represents the variable's "time of birth" while the right edge represents the variables "time of death".

(a) **LIFETIME TABLE**



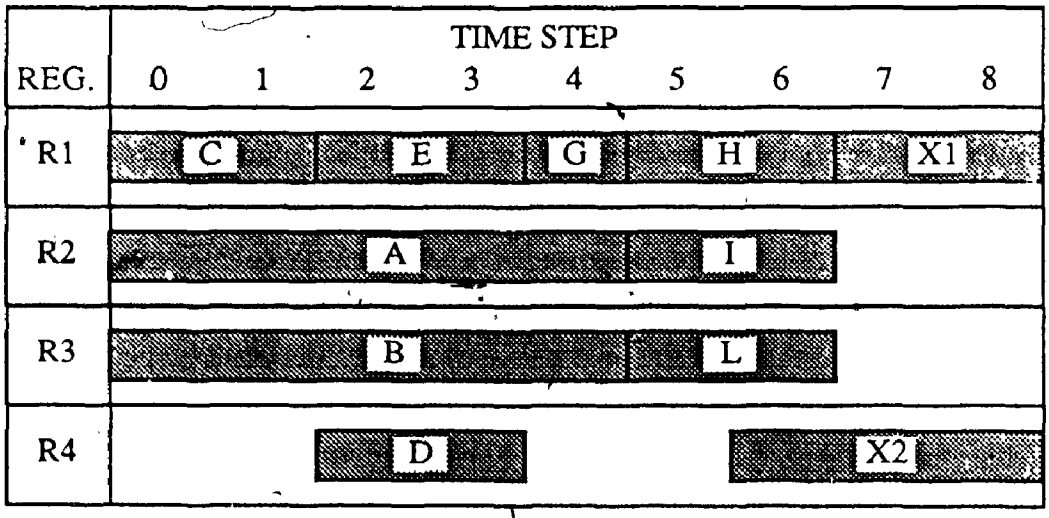
(b) SORTED LIFETIME TABLE

The DV's are then sorted, according to their left edges, and length of their lifetimes. The shortest left-most DV's are first (b).



(c) LEFT EDGE ALLOCATION

The DV's are then assigned to registers. The left-most, shortest DV is selected to be assigned to a register first. Other variables are then assigned to this register, accordingly, until no more variables will fit into this register (c), and then another register is selected.



becomes the current register.

(DV *c* is assigned to register-1 in Figure 3.2.23(c)).

Step 3: Find the next DV, whose *time of birth* is to the right of the last selected DVs *time of death* (right edge), and assign it to the current register. Repeat this step until no more DV's can be assigned to the current register. (DV's *e*, *g*, *h*, *x1* are assigned next in register-1 in Figure 3.2.23(c)).

Step 4: Repeat steps 2-4 until no more DV's remain in the *lifetime table* (Figure 3.2.23.(c)). The results of this allocation is shown in Figure 3.2.24.

Continuing with this example, a formulation of the bus allocation problem into that of the left-edge or **REAL** may be illustrated. The transfer variables (TV's) would be conceptually the same as the data variables (DV's), while buses could be modelled as registers. The TV's would also be listed in a *lifetime table*, like that of the DV lifetime table. The information presented in this table would indicate the time steps during which the data transfers are to occur (Figure 3.2.25(a)).

The result of the first step, sorting the *lifetime table* of TV's is shown in Figure 3.2.25(b). On examining this table, it is noticed that some of the TV's have discontinuous time steps. For instance, *TV1* exists for time step 1 and 3, but it does not exist during time step 2. This just means that *R1* (source) transfers its data to *OP1.IN2* (destination) in time steps 1 and 3, but not during time step 2. If the algorithm were to be applied, as is, it could not take advantage of these gaps. For instance, it would produce a solution with eight buses (Figure 3.2.25(b)), which is quite far from the minimum solution of six buses (Figure 3.2.25(c)). Since, there is no parallel for this type of situation (i.e.- more than one birth and death times) in the left edge algorithm, it would have to be modified to take advantage of these gaps. Moreover, the algorithm would also have to be modified to consider common sources or common destinations when making assignments. These modifications would seriously complicate an otherwise simple greedy algorithm. It is for these reasons that the left edge algorithm is unsuitable for the task of assigning TV's to buses.

Figure 3.2.24 **DFG WITH ALLOCATED REGISTERS**

This DFG (from Figure 3.2.22) shows the results of assigning variables to registers using the left-edge algorithm.

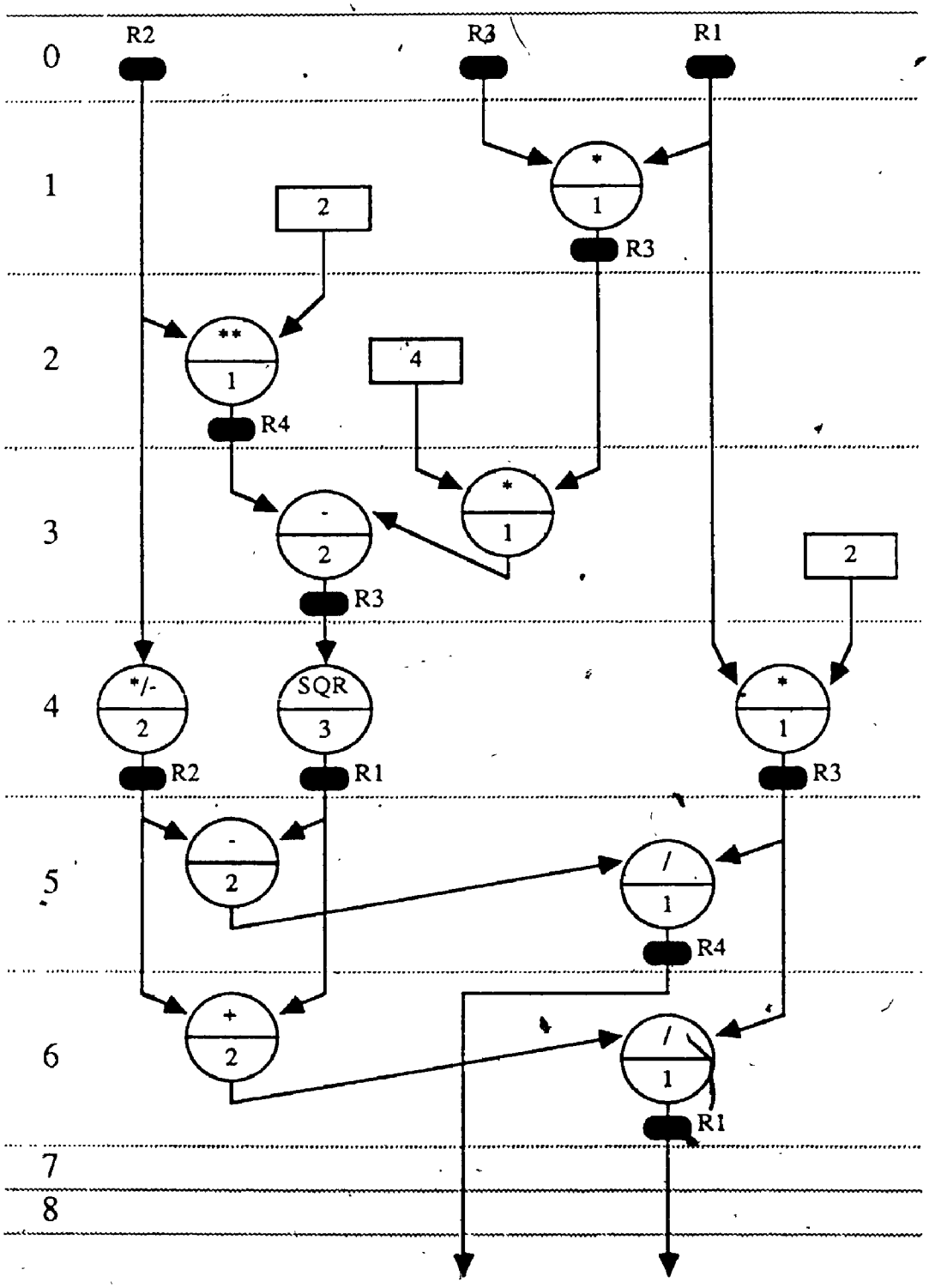
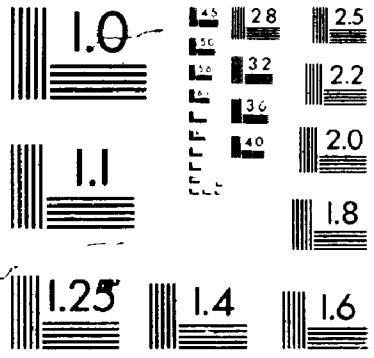


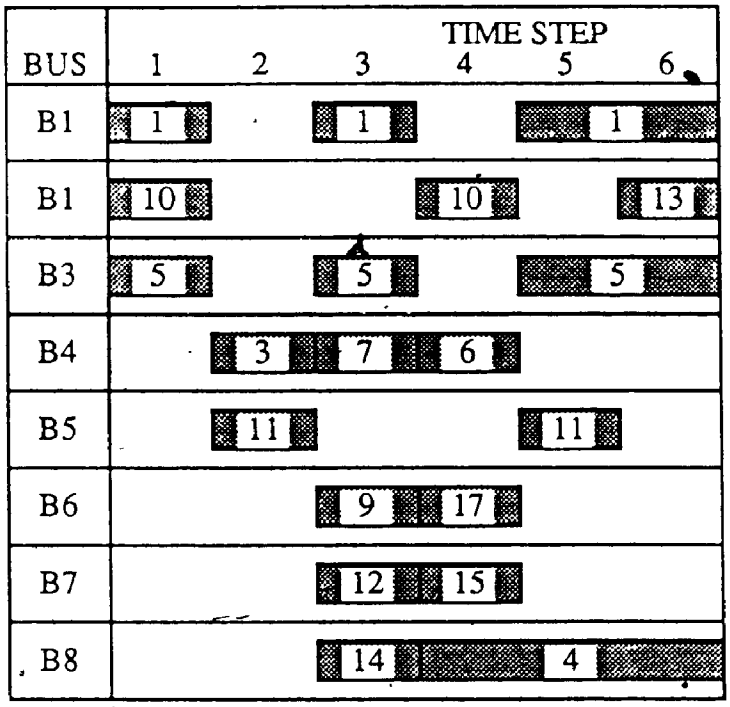
Figure 3.2.25: LEFT-EDGE ALGORITHM APPLIED TO INTERCONNECTION ALLOCATION

(a) SORTED LIFETIME TABLE

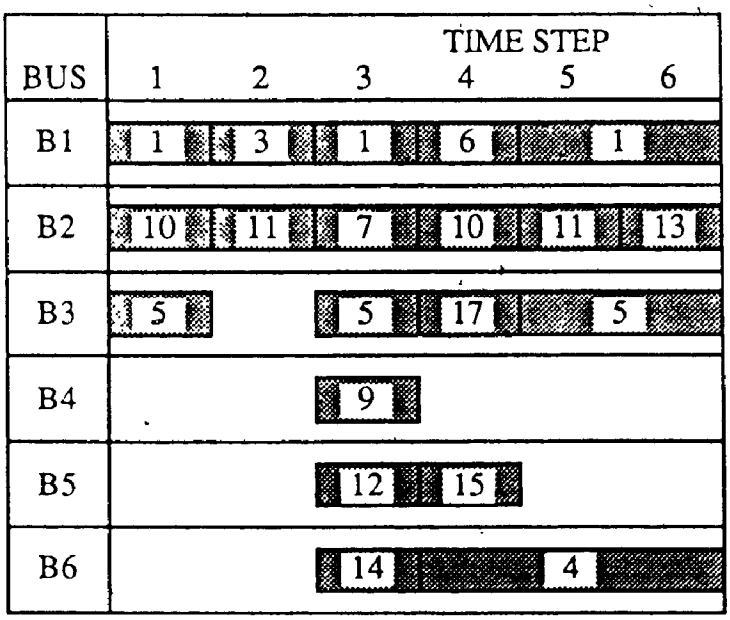
TV#	SRC. → DST.		TIME STEP 1	2	3	4	5	6
1	R1	OP1.1						
10	OP1	R3						
5	R3	OP1.2						
3	R2	OP1.1						
8	C2	OP1.2						
11	OP1	R4						
7	R4	OP2.2						
9	C4	OP1.2						
12	OP1	OP2.1						
14	OP2	R3						
6	R3	OP3						
15	OP2	R2						
17	OP3	R1						
4	R2	OP2.2						
2	R1	OP2.1						
16	OP2	OP1.1						
13	OP1	R1						



(b) ASSIGNING TV'S TO BUSES BY LEFT-EDGE



(c) ASSIGNING TV'S TO BUSES BY INSPECTION



Chapter 4
BAL Description, Analysis, and Comparison

4.1 BAL - Bus Allocation Algorithm

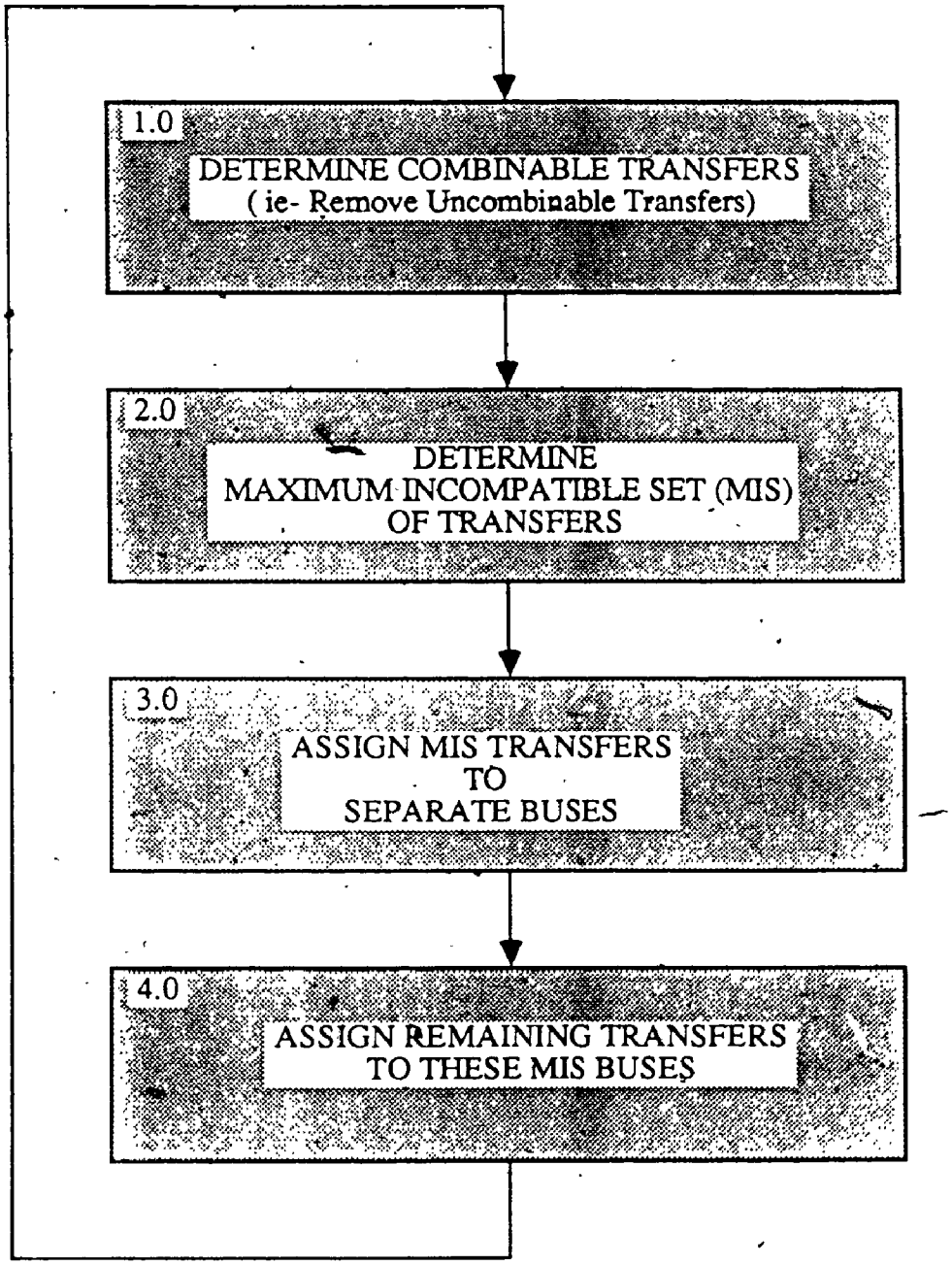
BAL's object is to partition the transfer variables (TV's) into the minimum number of groups, with each TV in a group being assigned to the same bus, such that there is a minimum amount of area consumed by interconnect, and concurrency constraints are not violated. A set of priority factors (*primary weights*) directs the partitioning process.

4.1.1 BAL - Step 1

The first step is to remove the *uncombinable* TV's from the list of TV's. An *uncombinable* TV is defined as a transfer variable that has neither source nor destination in common with another TV. Note that although it may physically be able to be combined with other TV's, there would be zero or negative profit in doing so. Therefore, these TV's are removed from further consideration. For instance, in Figure 4.1.2, TV-16 is an *uncombinable* TV, as its source, *OP-2*, only feeds the destination *R4*, and destination *R4* only receives data from source *OP-2*.

The combinable transfers are determined by searching the TV list, and verifying that each TV has at least the same source or sink as another TV in this list (Figure 4.1.3). Both TV's would then be labelled as combinable. After one main pass through the list, all TV's, that are not labelled (i.e.- isolated transfers), are discarded from the list (Figure 4.1.4). These TV's are assigned to separate buses, one bus per uncombinable TV.

Figure 4.1.1: "BAL" - BUS ALLOCATION ALGORITHM



UNTIL Number-of-Combinable-Transfers = NULL

Figure 4.1.2: **LIST OF TRANSFER VARIABLES (TV'S) TO ~~BE~~ ASSIGNED TO BUSES**

(Obtained from the DFG shown in Figure 2.15)

This figure shows the list of TV's that are to be assigned to buses. The "uncombinable TV's" have neither source nor destination in common with other TV's, and are removed from this list. For instance, TV-16 will be removed (ie- it is "uncombinable") since its source 'OP-2' only feeds its destination 'R4', and destination 'R4' only receives data from 'OP-2'.

TRANSFER VARIABLES				SAME SOURCE OR SINK AS ANOTHER TRANSFER ?
INDEX	SOURCE	DESTINATION	CYCLES	
1	R1	OP1.IN2	3	YES
2	R1	OP2.IN1	1	YES
3	R1	OP4.IN1	3,4	YES
4	R2	OP1.IN1	2	YES
5	R2	OP3.IN2	4	YES
6	R3	OP1.IN1	1	YES
7	R3	OP3.IN2	1	YES
8	R4	OP2.IN2	2	YES
9	R4	OP4.IN2	3,4	YES
10	R5	OP2.IN1	2,3	YES
11	R5	OP3.IN1	4	YES
12	R6	OP5.IN2	2	YES
13	R6	R2	1	YES
14	R6	R3	2	YES
15	OP1	R5	1,2,3	NO
16	OP2	R4	1,2,3	NO
17	OP3	R6	1,4	NO
18	OP4	R1	3,4	NO
19	DX	OP1.IN1	3	YES
20	DX	OP2.IN2	1,3	YES
21	DX	OP3.IN1	1	YES
22	C3	OP1.IN2	1	YES
23	C5	OP1.IN2	2	YES
24	A	OP5.IN1	2	NO

Figure 4.1.3: **STEP 1.0: DETERMINE COMBINABLE TV'S**

```
FOR I = 1 TO Number_Of_TV's DO
    FOR J = I + 1 TO Number_Of_TV's DO
        IF { ( TV[I].Source = TV[J].Source ) OR
             ( TV[I].Sink   = TV[J].Sink )      } THEN

            TV[I].Combinable = TRUE
            TV[J].Combinable = TRUE
        {END-IF}
    {END-FOR}
{END-FOR}
```

Figure 4.1.4: **LIST OF COMBINABLE TV'S DETERMINED FROM FIGURE 4.1.2**

The table, below, contains all the combinable TV's from the list of TV's in Figure 4.1.2 (ie- all "incombinable TV's" have been removed). A "combinable" TV is one that has either its source or destination in common with another TV's source or destination, respectively.

INDEX	SOURCE	DESTINATION	CYCLES
1	R1	OP1.IN2	3
2	R1	OP2.IN1	1
3	R1	OP4.IN1	3,4
4	R2	OP1.IN1	2
5	R2	OP3.IN2	4
6	R3	OP1.IN1	1
7	R3	OP3.IN2	1
8	R4	OP2.IN2	2
9	R4	OP4.IN2	3,4
10	R5	OP2.IN1	2,3
11	R5	OP3.IN1	4
12	R6	OP5.IN2	2
13	R6	R2	1
14	R6	R3	2
19	DX	OP1.IN1	3
20	DX	OP2.IN2	1,3
21	DX	OP3.IN1	1
22	C3	OP1.IN2	1
23	C5	OP1.IN2	2

4.1.2 BAL - Step 2

The second step is to determine the maximum incompatible set (MIS) of combinable TV's. An MIS is defined as the largest set of unique source TV's that may not share a bus due to concurrency constraints (i.e.- all require a bus during the same time step). The MIS is found by finding the time step in which the largest number of unique sources are required to transfer their contents (Figure 4.1.5 and 4.1.6). If there is more than one MIS, then one is chosen arbitrarily.

4.1.3 BAL - Step 3

This step involves assigning the MIS TV's to separate buses, and determining the remaining combinable transfers. The MIS transfers for the example are TV's 4, 8, 10, 12, 14 and 23, which have been assigned to buses 1 through 5, respectively (top half of Figure 4.1.7). The remaining combinable TV's {1, 2, 3, 5, 6, 7, 9, 11, 13, 19, 20, 21, 22} are shown in the bottom half of this figure.

4.1.4 BAL - Step 4

In this step, the remaining combinable TV's are assigned to the MIS buses. This is accomplished in several sub-steps (Figure 4.1.8):

Step 4.1: This is responsible for determining the *primary weights* of all of the remaining TV's with respect to each of the MIS buses. The *primary weight* of a TV with respect to an MIS bus is inversely proportional to the cost of assigning that TV to that bus, the greater the weight, the smaller the cost. This weight can be determined by comparing the sources, sinks, and time steps, of the TV with those that have already been assigned to that bus (Figure 4.1.9). Seven cases have been established and are summarized in Figure 4.1.10.

Case 1: The TV has the same time step as one of the assigned TV's and they do not have the same source. This situation indicates that there would be a

Figure 4.1.5: **STEP 2.0: DETERMINE THE MIS TV'S**

```
FOR J = I + 1 TO Number_Of_Time-Steps DO
  FOR I = 1 TO Number_Of_Combinable_TV's DO
      IF { ( Time-Step[J].Time-Step MEMBER_OF TV[I].Time-Step ) AND
           (.TV[I].Source NOT_MEMBER_OF Time-Step[J].Source ) } THEN
          Time-Step[J].TVs = ( ADD TV[I] )
          Time-Step[J].Mis = Time-Step[J].Mis
      {END-IF}
  {END-FOR}
  IF { Time-Step[J].Mis > Mis_Maximum } THEN
      Mis-Maximum = Time-Step[J].Mis
      Mis-Set  = Time-Step[J].TVs
  {END-IF}
{END-FOR}
```

Figure 4.1.6: **DETERMINING THE *MIS* FROM THE COMBINABLE LIST OF TV'S**

The MIS set of TV's indicates the minimum number of buses that will be required for the "combinable" TV's. It is the maximum set of unique-source TV's in any time step. For instance, time steps 1 & 2 both have 5 unique-source TV's, therfore one of them is selected, arbitrarily, to be the MIS-BUS.

TIME STEP	INDEX NO.	SOURCE OF TRANSFER	NO. OF UNIQUE-SOURCE TRANSFERS
1	2	R1	5
	6	R3	
	7	R3	
	13	R6	
	20	DX	
	21	DX	
	22	C3	
2	4	R2	5
	8	R4	
	10	R5	
	12	R6	
	14	R6	
	23	C5	
3	1	R1	4
	3	R1	
	9	R4	
	10	R5	
	19	DX	
	20	DX	
4	3	R1	4
	5	R2	
	9	R4	
	11	R5	

CHOOSE

Figure 4.1.7: **STEP 3.0: ASSIGN THE MIS TV'S TO SEPARATE BUSES**

STEP4.1: DETERMINE THE PRIMARY WEIGHTS OF THE REMAINING TV'S WITH RESPECT TO THE 'MIS' BUSES.

LEGEND: ▨ - TV is permanently assigned to that MIS BUS,

□ ⊡ - Primary Weight of TV with repect to MIS BUS.

TRANSFER I S → D	BUS1 1 2 3 4	BUS2 1 2 3 4	BUS3 1 2 3 4	BUS4 1 2 3 4	BUS5 1 2 3 4
4 R2 → OP1.IN1	▨				
8 R4 → OP2.IN2		▨			
10 R5 → OP2.IN1			▨		
12 R6 → OP5.IN2				▨	
22 C5 → OP1.IN2					▨
1 R1 → OP1.IN2	0	0		0	3
2 R1 → OP2.IN1	0	0	3	0	
3 R1 → OP4.IN1	0	0		0	0
5 R2 → OP3.IN2	1	0	0	0	0
6 R3 → OP1.IN1	3	0	0	0	
7 R3 → OP3.IN2	0	0	0	0	
9 R4 → OP4.IN2	0	1		0	0
11 R5 → OP3.IN1	0	0	1	0	0
13 R6 → R2	0	0	0	1	
14 R6 → R3				2	
19 DX → OP1.IN1	3	0		0	0
20 DX → OP2.IN2	0 0	3 3		0 0	
21 DX → OP3.IN1	0	0	0	0	
23 C3 → OP1.IN2	0	0	0	0	3

Figure 4.1.8 STEP 4.0: ASSIGN REMAINING COMBINABLE TV'S TO MIS BUSES

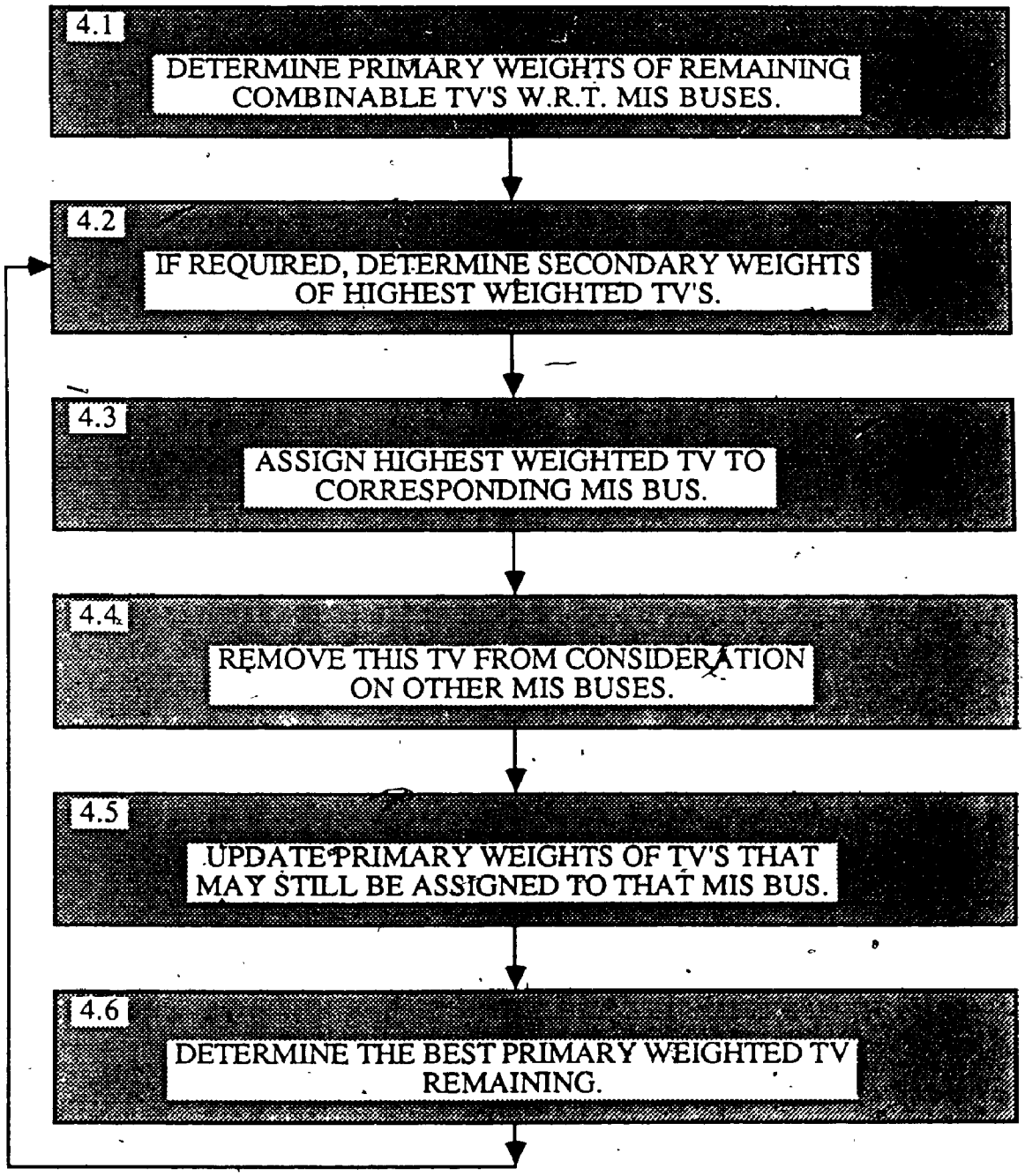
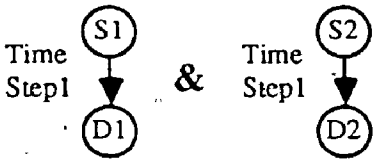
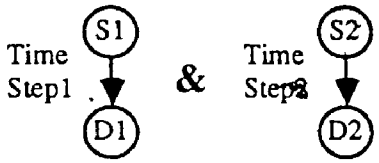
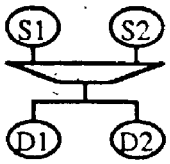
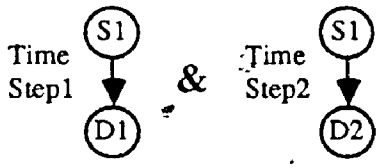
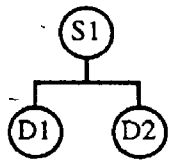
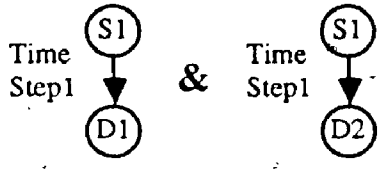
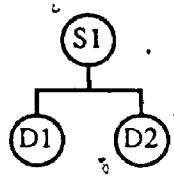
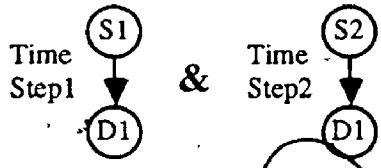
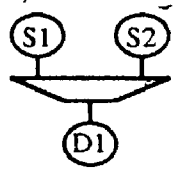
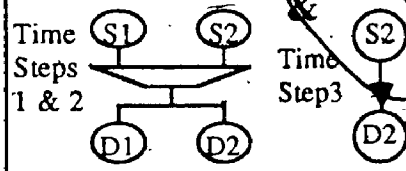
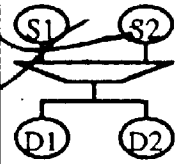
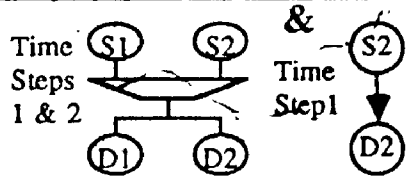
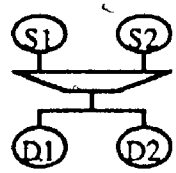


Figure 4.1.9: **STEP 4.1: DETERMINE PRIMARY WEIGHTS OF REMAINING COMBINABLE TV'S WITH RESPECT TO EACH MIS BUS.**

```
FOR I = 1 TO Number_Of_Remaining_TV's DO
        FOR J = I + 1 TO Number_Of_MIS_Buses DO
          Determine-Primary-Weight ( TV[I]  MIS_Bus[J] )
        {END-FOR}
{END-FOR}
```

Figure 4.1.10: **SEVEN CASES FOR DETERMINING PRIMARY WEIGHTS**

CASE NO.	DESC.	BEFORE	AFTER	WEIGHT ASSIGNED
1	The two transfers occur in the same time step, therefore, they cannot be combined.			NIL
2	The two transfers have nothing in common.			0
3	The two transfers have the same source, but occur at different times.			1
4	The two transfers have the same source, and occur at the same time.			2
5	The two transfers have the same sink, and occur at different times.			3
6	The 3rd transfer has the same source& sink, & occurs at different times.			4
7	The 3rd transfer has the same source& sink, & occurs at the same time.			5

time conflict (bus contention) if this TV were to be assigned to the bus (WEIGHT = NIL).

Case 2: The TV has neither the same source, destination, nor time step, as any of those TV's that have already been assigned to the bus (WEIGHT = 0).

Case 3: The TV has neither the same sink, nor time step, as any of the assigned TV's. However, it does have the same source (WEIGHT = 1).

Case 4: The TV has the same time step as one of the assigned TV's and they have the same source (WEIGHT = 2).

Case 5: The TV has neither the same source, nor time step, as any of those TV's that have already been assigned to that bus. However, the TV does have the same sink as one of the assigned ones (WEIGHT = 3).

Case 6: The TV has the same source as one, and the same sink as another, of the assigned TV's. As well, it does not have the same time step of any of those assigned TV's (WEIGHT = 4).

Case 7: The TV has the same source and time step as one, and the same destination as another, of the assigned TV's (WEIGHT = 5).

The weights assigned to each of these cases, yielded the best, most consistent, results. This will be justified in Section 4.3.

Step 4.2: Once the primary weights of all of the TV's have been determined (Figure 4.1.7), the highest weighted TV is assigned to the corresponding MIS bus. However, if there is more than one TV with the highest weight (i.e.- a tie), then the *secondary weights* of these tying TV's must be determined. For instance, in Figure 4.1.7, there are five TV's, {1, 2, 6, 19, 20}, that have a primary weight of 3. The secondary weights of these TV's attempt to determine which of these should be assigned first, by predicting what future savings may result. These *secondary weights* are determined in the following manner (Figure 4.1.11).

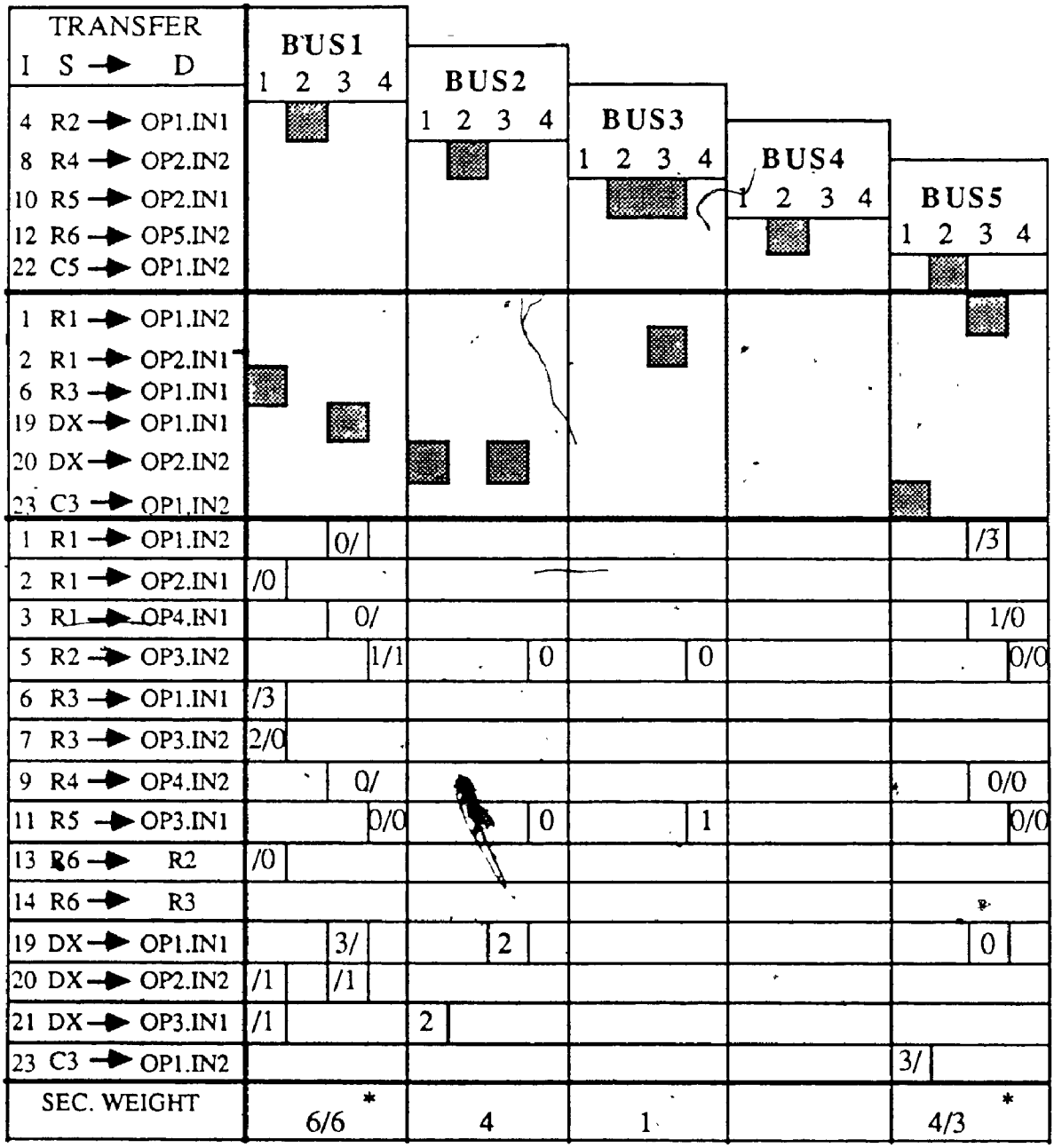
Step 4.2.1: Each TV under consideration is temporarily assigned to the corresponding MIS bus, as shown in Figure 4.1.12.

Figure 4.1.11 **STEP 4.2: DETERMINE SECONDARY WEIGHTS OF THOSE TV'S WITH HIGHEST PRIMARY WEIGHTS.**

```
FOR I = 1 TO Number_Of_Highest_PW_TVs DO

        Secondary_Weight = 0
        Temporarily-Assign ( TV[I]  TV[I].Mis_Bus )

        FOR J = 1 TO TV[I].Mis_Bus.Potential_TVs DO

                Secondary_Weight = Secondary_Weight +
                        Determine-Primary-Weight ( TV[J]  TV[I].Mis_Bus )

        {END-FOR}

        IF { Secondary_Weight >= Max_Secondary_Weight } THEN

                Max_Secondary_Weight = Secondary_Weight
                Best_Secondary_Weighted_TV = TV[I]

        {END-IF}
{END-FOR}
```

Figure 4.1.12: **STEP 4.2: CALCULATE SECONDARY WEIGHTS FOR TV'S WITH HIGHEST PRIMARY WEIGHT**

Each of the TV's, having equal highest primary weights is temporarily assigned to the corresponding MIS-BUS, one at a time (TV's 1, 2, 6, 19, 20, 23 to BUSES 5, 3, 1, 1, 2, 5, respectively). For each of the temporarily assigned TV's, the primary weights of any remaining TV's, on the corresponding MIS bus, are summed. This sum represents the secondary weight of the temporarily assigned TV. For instance, TV-20 is temporarily assigned to MIS-BUS-2. Its secondary weight is 4 (0+0+2+2).



TRANSFER I S → D	BUS1 1 2 3 4	BUS2 1 2 3 4	BUS3 1 2 3 4	BUS4 1 2 3 4	BUS5 1 2 3 4
4 R2 → OP1.IN1					
8 R4 → OP2.IN2					
10 R5 → OP2.IN1					
12 R6 → OP5.IN2					
22 C5 → OP1.IN2					
1 R1 → OP1.IN2					
2 R1 → OP2.IN1					
6 R3 → OP1.IN1					
19 DX → OP1.IN1					
20 DX → OP2.IN2					
23 C3 → OP1.IN2					
1 R1 → OP1.IN2	0/				/3
2 R1 → OP2.IN1	/0				
3 R1 → OP4.IN1	0/				1/0
5 R2 → OP3.IN2	1/1	0	0		0/0
6 R3 → OP1.IN1	/3				
7 R3 → OP3.IN2	2/0				
9 R4 → OP4.IN2	0/				0/0
11 R5 → OP3.IN1	0/0	0	1		0/0
13 R6 → R2	/0				
14 R6 → R3					
19 DX → OP1.IN1	3/	2			0
20 DX → OP2.IN2	/1 /1				
21 DX → OP3.IN1	/1	2			
23 C3 → OP1.IN2					3/
SEC. WEIGHT	* 6/6	4	1		* 4/3

* X/Y : X IS THE SECONDARY WEIGHT OF THE TOPMOST TV

Step 4.2.2: The primary weights, of those TV's that may still be assigned to the MIS bus, after the temporary assignment, are summed. For instance, if TV-20 is assigned to MIS-BUS-2 (Figure 4.1.12), the only TV's that may then be considered for assignment to this bus are: TV-5, TV-11, TV19, and TV-21. The sum of the primary weights of these TV's with respect to MIS-BUS-2 is 4 (0 + 0 + 2 + 2, respectively). Thus the *secondary weight* of TV-20 is 4.

Step 4.3: The TV with the highest secondary weight is selected to be permanently assigned to the corresponding MIS-BUS. If more than one has the highest weight (i.e.- a tie) then one should be chosen arbitrarily. For instance, TV-6 and TV-19, both on BUS-1, have the same highest secondary weight. TV-19 is arbitrarily selected to be assigned to BUS-1.

Step 4.4: This step involves removing the newly assigned TV from being considered on other buses, as outlined in Figure 4.1.13.

Step 4.5: This step is responsible for re-calculating the primary weights of those TV's that may still be assigned to the newly adjusted MIS bus (Figure 4.1.14). For instance, in Figure 4.1.15, the primary weight of the TV's in column 1 (i.e.- MIS-BUS-1) are re-calculated to reflect the fact that TV-19 was permanently assigned to BUS-1.

Step 4.6: Finally, the highest weighted, non-zero, TV(s) are found. No new calculations are required. If non-zero, primary weighted TV's remain, then the algorithm proceeds to step 4.2. The remaining TV's are assigned in the same manner (Figure 4.1.17).

If more than one TV remains after assigning all non-zero primary weighted TV's to the MIS buses, then steps 1.0 to 4.0 are repeated with those remaining transfers. These steps may be repeated G times. However, for all observed cases $G \leq 2$.

Finally, all *uncombinable* transfers are assigned to separate buses. Ideally, a preliminary placement program could then decide if it would be profitable for any

Figure 4.1.13 **STEP 4.4: REMOVE THE PREVIOUSLY ASSIGNED TV FROM BEING CONSIDERED ON OTHER MIS BUSES.**

```
FOR I = 1 TO Number_Of_MIS-Buses DO
        IF { TV MEMBER-OF MIS-Bus[I].Potential-TVs } THEN
                REMOVE ( TV MIS-Bus[I].Potential-TVs )
        {END-IF}
{END-FOR}
```

Figure 4.1.14: **STEP 4.5: UPDATE PRIMARY WEIGHTS OF THE OTHER TV'S THAT MAY STILL BE ASSIGNED TO THE MIS-BUS.**

```
FOR I = 1 TO Number_Of_MIS-Bus.Potential-TVs DO
        Determine-Primary-Weight ( TV[I] MIS-Bus )
{END-FOR}
```

Figure 4.1.15: **STEPS 4.3 - 4.5: ASSIGN THE HIGHEST WEIGHTED TV TO THE CORRESPONDING MIS-BUS.**

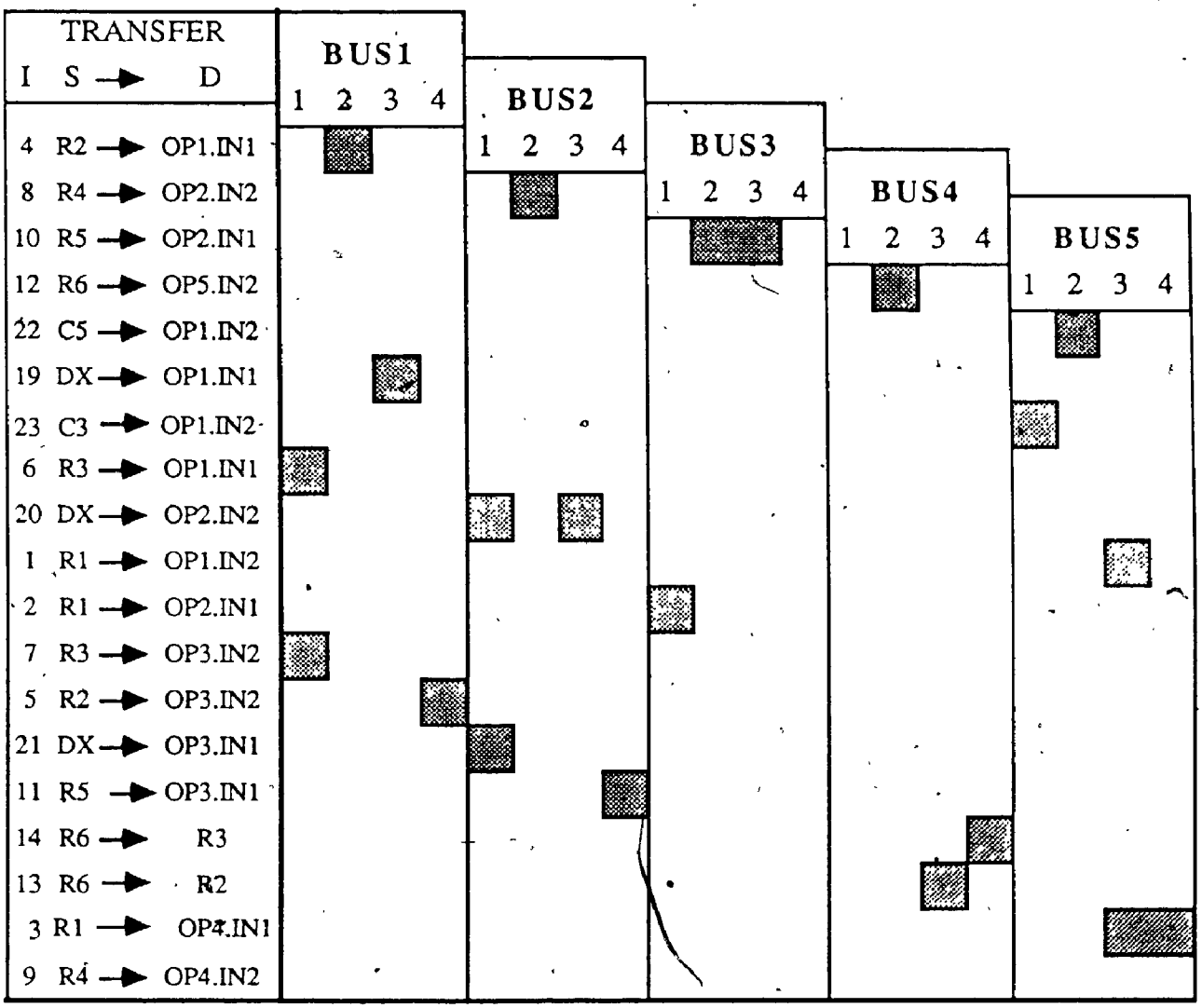
This table shows that 'TV-19' was chosen to be assigned to 'BUS1'. Note that the weights of the remaining transfers on 'BUS1' have been adjusted to reflect this new addition to the bus and that TV-19 has been removed from being considered on other buses.

TRANSFER I S → D	BUS1 1 2 3 4	BUS2 1 2 3 4	BUS3 1 2 3 4	BUS4 1 2 3 4	BUS5 1 2 3 4
4 R2 → OP1.IN1					
8 R4 → OP2.IN2					
10 R5 → OP2.IN1					
12 R6 → OP5.IN2					
22 C5 → OP1.IN2					
19 DX → OP1.IN1					
1 R1 → OP1.IN2		0		0	3
2 R1 → OP2.IN1	0	0	3	0	
3 R1 → OP4.IN1		0		0	0
5 R2 → OP3.IN2	1	0	0	0	0
6 R3 → OP1.IN1	3	0	0	0	
7 R3 → OP3.IN2	0	0	0	0	
9 R4 → OP4.IN2		1		0	0
11 R5 → OP3.IN1	0	0	1	0	0
13 R6 → R2	0	0	0	1	
14 R6 → R3				2	
20 DX → OP2.IN2	1 1	3 3		0 0	
21 DX → OP3.IN1	1	0	0	0	
23 C3 → OP1.IN2	0	0	0	0	3

Figure 4.1.16: **STEP 4.6: DETERMINE BEST PRIMARY WEIGHED TV'S REMAINING.**

```
Number_Of_Best_TVs = MIS-Bus[1].Num_Best_TVs
Best_Weight = MIS-Bus[1].Best_Weight
Best_TVs = MIS-Bus[1].Best_TVs

FOR I = 2 TO Number_Of_MIS-Buses DO

    IF { MIS-Bus[I].Best_Weight > Best_Weight } THEN
      Number_Of_Best_TVs = MIS-Bus[I].Num_Best_TVs
      Best_Weight = MIS-Bus[I].Best_Weight
      Best_TVs = MIS-Bus[I].Best_TVs

      ELSE

        IF { MIS-Bus[I].Best_Weight > Best_Weight } THEN
            Best_TVs = ADD ( MIS-Bus[I].Best_TVs
                             Best_TVs)
            Number_Of_Best_TVs = Number_Of_Best_TVs
                             + MIS-Bus[I].Num_Best_TVs


          { END-IF }

    { END-IF }
{ END-FOR }
```

Figure 4.1.17: RESULT OF BAL



of these *uncombinable* transfers to share a bus. Normally, if an operator and a register are connected only to each other, they will be placed very close together.

A block diagram for the example is shown in Figure 2.6(a).

4.1.5 Complexity of BAL (With Secondary Weights)

A cursory outline of **BAL** is given in Figure 4.1.18. Included in this outline is the complexity of each step. From this figure, the complexity of the **BAL** Algorithm can be stated as :

$$O_{BAL} = G \times \{O_{Step\text{-}1} + O_{Step\text{-}2} + O_{Step\text{-}3} + O_{Step\text{-}4}\} \qquad (4.1-1)$$

Where: O $\stackrel{def}{=}$ order of complexity of,

$G \stackrel{def}{=}$ the number of times steps 1-4 must be executed to assign all the TV's to buses.

$$O_{Step\text{-}1} = N(N-1) \qquad (4.1-2)$$

Where: $N \stackrel{def}{=}$ Number of TV's to be assigned.

$$O_{Step\text{-}2} = C \times T \qquad (4.1-3)$$

Where: $C \stackrel{def}{=}$ the number of combinable TV's,

$T \stackrel{def}{=}$ the number of time steps in the DFG.

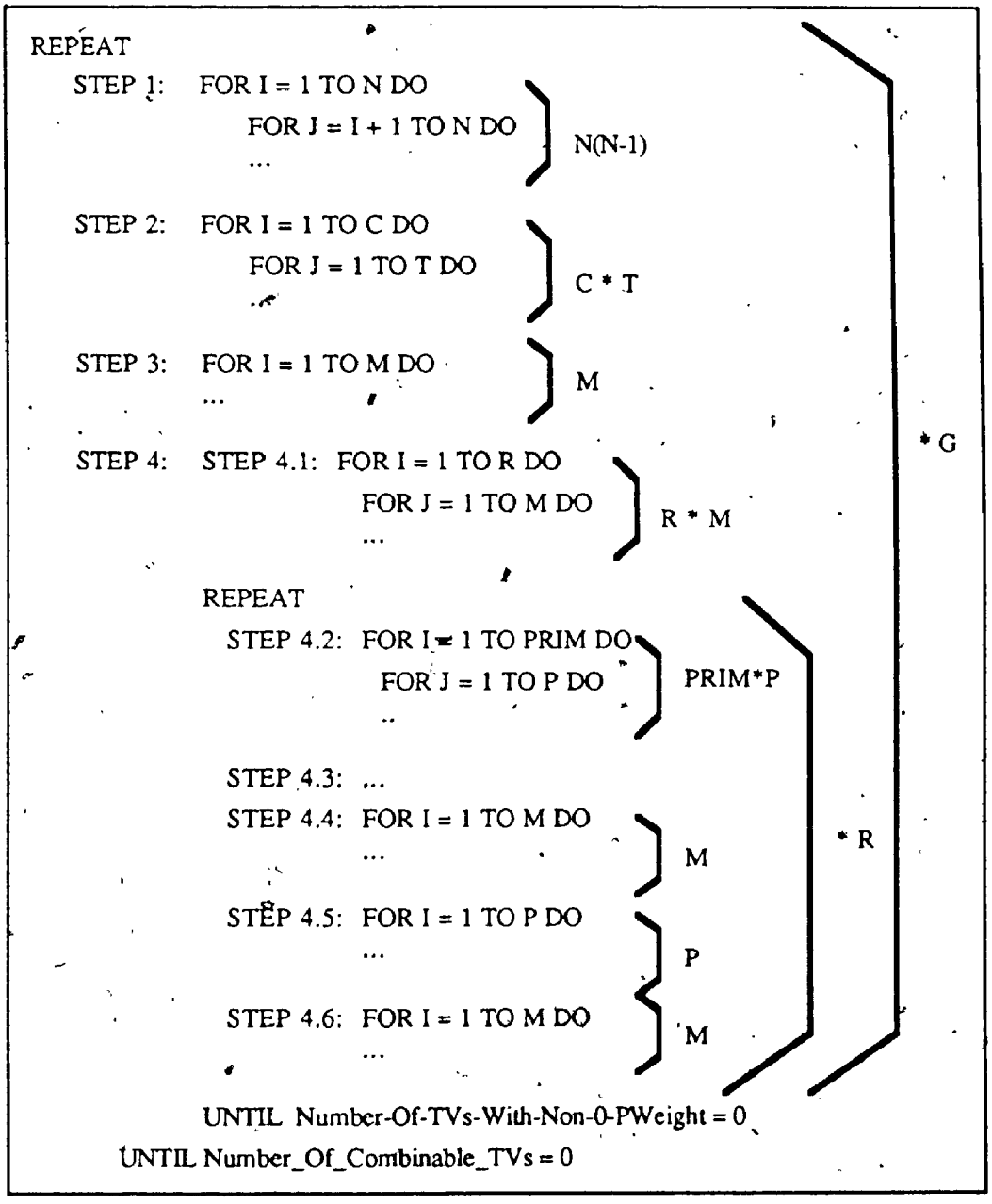
$$O_{Step\text{-}3} = M \qquad (4.1-4)$$

Where: $M \stackrel{def}{=}$ the maximum number of TV's in the *Maximally Incompatible Set* (MIS), i.e.- the minimum number of buses required for the combinable TV's.

$$O_{Step\text{-}4} = R \times \{O_{Step\text{-}4.1} + O_{Step\text{-}4.2} + O_{Step\text{-}4.4} + O_{Step\text{-}4.5} + O_{Step\text{-}4.6}\} \qquad (4.1-5)$$

Figure 4.1.18: **OUTLINE OF BAL FOR COMPLEXITY ANALYSIS**

N = Number of TV's to be assigned,
G = Number of times steps 1-4 must be executed to assign all TVs
T = Number of time steps,
C = Number of combinable TV's,
R = Number of combinable TV's that must be assigned after the MIS TV's have been assigned, R = C - M
M = Number of TV's in the MIS (i.e.- minimum number of buses),
PRIM = Number of TV's having equal and highest primary weight,
P = Number of TV's that may be assigned to any MIS bus.



Where: $R \stackrel{\text{def}}{=}$ the number of combinable TV's that must be assigned after the MIS-TV's have been assigned.

$$O_{\text{Step-4.1}} = M \qquad (4.1-6)$$

Where: M is as defined earlier.

$$O_{\text{Step-4.2}} = \text{Prim} \times P \qquad (4.1-7)$$

Where: $P \stackrel{\text{def}}{=}$ the number of TV's that may be assigned to one MIS-BUS,
Prim $\stackrel{\text{def}}{=}$ the number of TV's that have equal and highest primary weights.

$$O_{\text{Step-4.4}} = M \qquad (4.1-8)$$

Where: M is as defined earlier.

$$O_{\text{Step-4.5}} = P \qquad (4.1-9)$$

Where: P is as defined earlier.

$$O_{\text{Step-4.6}} = M \qquad (4.1-10)$$

Where: M is as defined earlier.

Therefore, re-stating equation (4.1-5), and simplifying, yields:

$$O_{\text{Step-4}} = R \times \{3M + P + \text{Prim} \times P\} \qquad (4.1-11)$$

Re-stating equation (4.1-1) and simplifying, yields:

$$O_{\text{BAL}} = G \times \{N(N-1) + C \times T + M + R(3M + P + \text{Prim} \times P)\} \qquad (4.1-12)$$

For a worst case situation, the following assumptions can be made :

- All the TV's under consideration are combinable:

$$C = N$$

- Half of the combinable TV's are in the MIS:

$$M = C/2 = N/2$$

$$R = N - N/2 = N/2$$

- Each of the remaining TV's can be assigned to each of the MIS buses (i.e.- no bus contention):

$$P = R = N/2$$

- All of the remaining TV's have equal primary weight:

$$\text{Prim} = P \times M = N/2 \times N/2 = (N^2)/4$$

- The number of times that steps 1-4 must be repeated:

$G \leq 2$ (i.e.- it certainly isn't N).

Thus re-stating (4.1-12) to include these assumptions, and simplifying, yields:

$$O_{\text{BAL}} = N^4 \qquad (4.1-13)$$

Thus, the complexity of **BAL**, with secondary weights, is of order N^4.

4.1.6 Complexity of BAL Without Secondary Weights

If the secondary weight calculations are not included in **BAL**, then the complexity would be further reduced. A cursory outline of **BAL**, without secondary weights, is given in Figure 4.1.19. From this Figure, the complexity of step-4 can be stated as:

$$O_{\text{Step-4}} = R \times \{O_{\text{Step-4.1}} + O_{\text{Step-4.4}} + O_{\text{Step-4.5}} + O_{\text{Step-4.6}}\} \quad (4.1-16)$$

Using the same assumptions as for the average case, and simplifying, yields,

$$O_{\text{BAL}} = K(N^2) \quad (4.1-17)$$

Where: $K \stackrel{\text{def}}{=}$ a constant $\ll N$.

Thus, the complexity of **BAL**, when secondary weights are not used, is of order N^2. The argument for not using secondary weights will be presented in Section 4.3.2.

4.1.7 Implementation of BAL

BAL was implemented using VAX LISP 2.0. A data flow diagram of this algorithm is shown in Figure 4.1.20. The implementation consists of four main packages:

Discard: This contains the functions necessary to determine the set of combinable TV's, and the corresponding set of uncombinable TV's, from the given set of TV's.

Determine_MIS: This package contains those functions required to determine the maximum incompatible set (MIS) of transfer variables.

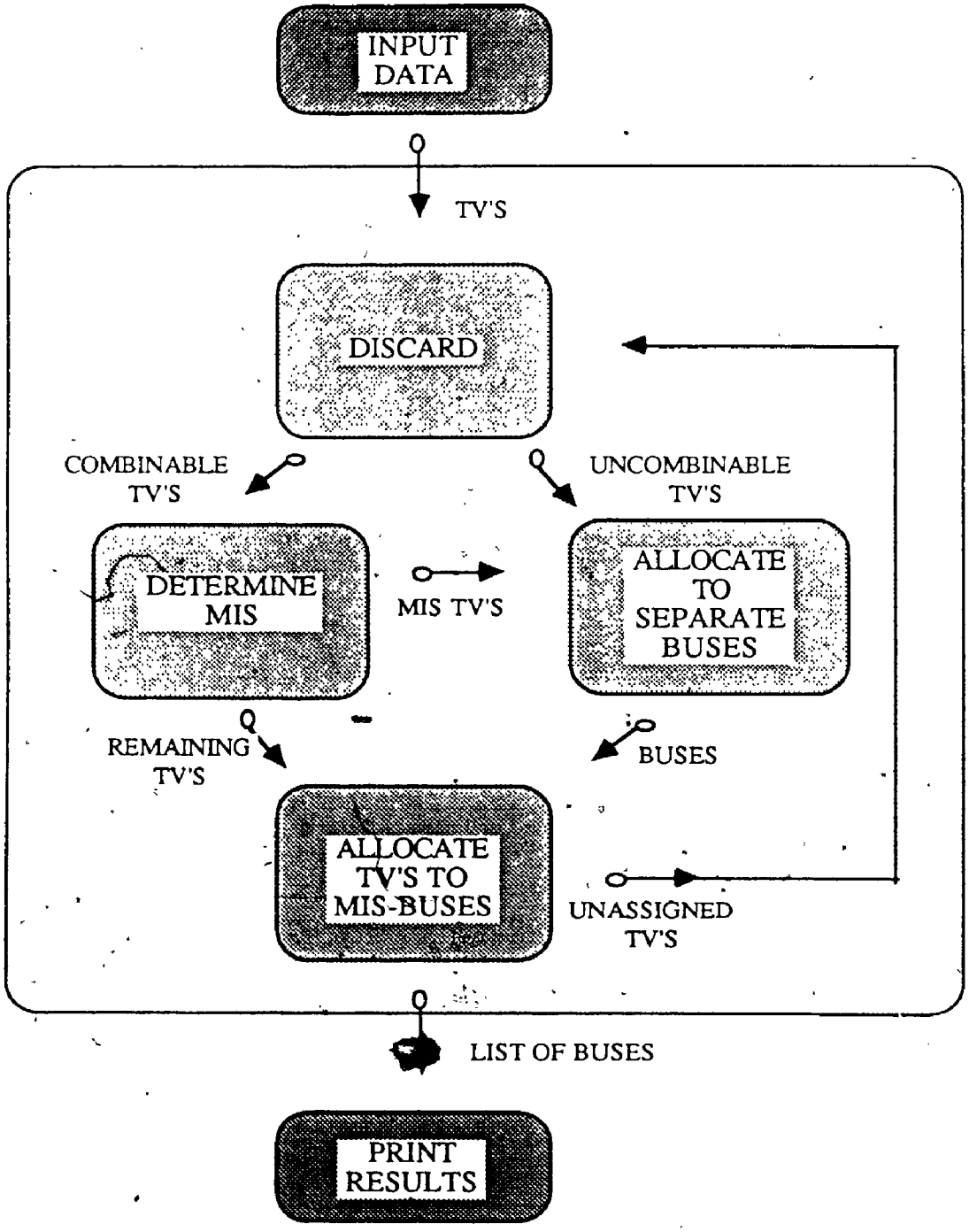
Allocate_To_Separate_Buses: This contains those functions that are necessary to create instances of buses, and to assign the given TV's to those buses.

Figure 4.1.19: **OUTLINE OF BAL FOR COMPLEXITY ANALYSIS (WITHOUT SECONDARY WEIGHTS)**

N = Number of TV's to be assigned,
G = Number of times steps 1-4 must be executed to assign all TVs
T = Number of time steps,
C = Number of combinable TV's,
R = Number of combinable TV's that must be assigned after the MIS TV's have been assigned, R = C - M
M = Number of TV's in the MIS (i.e.- minimum number of buses),
PRIM = Number of TV's having equal and highest primary weight
P = Number of TV's that may be assigned to any one MIS bus.

```
REPEAT
    STEP 1:  FOR I = 1 TO N DO                          } N(N-1)
                 FOR J = I + 1 TO N DO
                 ...
    STEP 2:  FOR I = 1 TO C DO                          } C * T
                 FOR J = 1 TO T DO
                 ...
    STEP 3:  FOR I = 1 TO M DO                          } M
             ...
    STEP 4:  STEP 4.1:  FOR I = 1 TO R DO               } R * M
                            FOR J = 1 TO M DO
                            ...
             REPEAT
               STEP 4.3:  ...
               STEP 4.4:  FOR I = 1 TO M DO             } M
                          ...
               STEP 4.5:  FOR I = 1 TO P DO             } P
                          ...
               STEP 4.6:  FOR I = 1 TO M DO             } M
                          ...
             UNTIL  Number-Of-TVs-With-Non-0-PWeight = 0   } * R
    UNTIL Number_Of_Combinable_TVs = 0                     } * G
```

Figure 4.1.20: BAL STRUCTURE FLOW GRAPH



Allocate_Transfers_To_MIS_Buses: Finally, the functions that are responsible for assigning the remaining combinable TV's to the MIS buses are contained in this package.

4.2 Interconnection-Cost Evaluation Algorithm

This section presents two register-transfer level interconnection cost evaluation algorithms that were developed as part of this thesis. The need for such an algorithm is driven by the questions:

How can the automatically generated interconnect be evaluated without actually performing a full placement, layout, and route of the circuit?

What is the area cost of this interconnection?

Two methods of evaluating **BAL** generated connections were developed, cost-scheme-1 and cost-scheme-2. The first method was implemented and was used to evaluate **BAL**'s results. Later, a scheme that considered b-bit wide buses was developed. It was not implemented, as the former scheme was deemed to be sufficient for comparison purposes.

4.2.1 Cost-Scheme-1

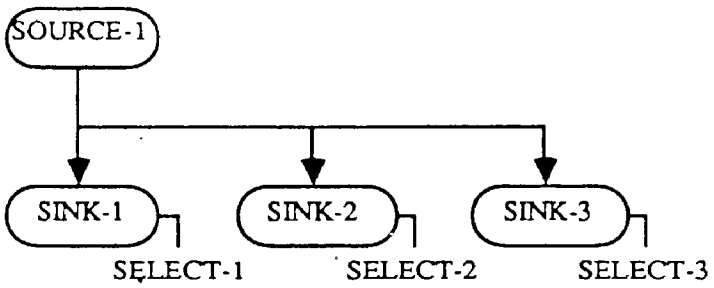
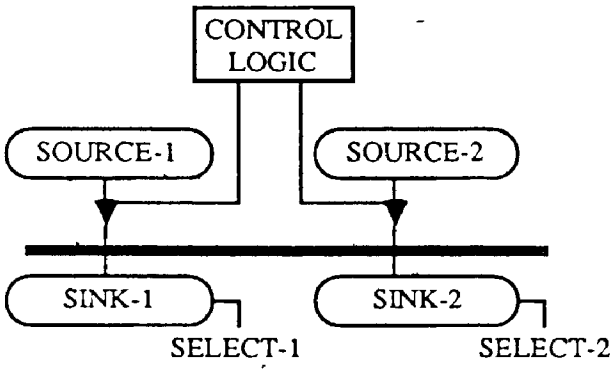
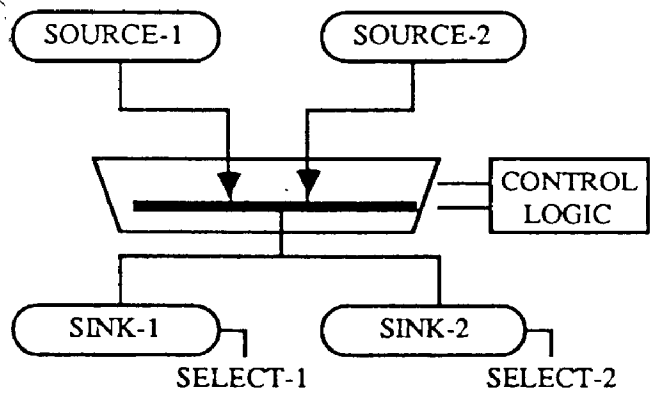
Figure 4.2.1 shows three types of connections that are possible : bus, mux, and leads. In each case, one bit wide transfers are assumed.

Let: L = the cost (area) of one lead,
T = the cost (area) of one pass transistor, or other equivalent type,
B = the cost (area) of one control bit,
n = the number of inputs to a bus or mux,
m = the number of outputs from a bus, mux, or a lead (i.e.- fanout).

Consider the cost of a lead connection (Figure 4.2.1(a)). The cost of connecting registers *Source-1* and *Sink-1* is proportional to the lead area between the two registers, *1*. The cost of connecting *Source-1* and *Sink-2*, once the connection between

Figure 4.2.1: **COST-SCHEME-1 " 1-BIT WIDE TV'S "**

Let: L = Cost (area) of one lead,
T = Cost (area) of one pass transistor, or other type,
B = Cost (area) of one control bit logic,
n = number of inputs to bus or mux,
m = number of outputs from a bus or mux.

CASE	INTERCONNECT	COST
(A) LEADS		L(m+1)/2
(B) BUS		(B)Log(n) + (T)(n) + (L)(n) +L(m+1)/2
(C) MUX		(B)Log(n) + (T)(n) + (L)(n) +L(m+1)/2

Source-1 and *Sink-1* has been established, is proportional to the lead area *2*. Note that some of the wire *1* is used by the *Source-1* to *Sink-2* connection. From this, let it be assumed that the cost of connections, between *Source-1* and each additional destination, will be less than that of the first wire *1*. Furthermore, let this cost, for each additional destination, be half that for the first one. This can be more formally stated as the fanout cost :

$$\text{Lead Cost} = L(m+1)/2 \qquad (4.2-1)$$

Consider the cost of the bus configured interconnection. This cost consists of the control logic area, pass transistor area, and bus lead area. Let:

$$\text{Bus Control Cost} = \begin{pmatrix}\text{Area of control}\\ \text{logic}\end{pmatrix} + \begin{pmatrix}\text{Area of control leads}\\ \text{from control logic to}\\ \text{pass transistors}\end{pmatrix}$$

$$= (B)(\log_2 n) + (L)(n) \qquad (4.2-2)$$

$$\text{Pass Trans. Cost} = \text{Area of pass transistor}$$

$$= (T)(n) \qquad (4.2-3)$$

$$\text{Bus Lead Cost} = \text{Area of bus fanout from (4.2-1)}$$

$$= L(m+1)/2 \qquad (4.2-4)$$

Note that there is no lead area cost associated with the inputs to the bus. This cost has been assumed to be absorbed by the pass transistor cost, and the bus lead cost. Therefore, the bus cost is :

$$\text{Bus Cost} = (B)\log_2 n + (T)(n) + (L)(n) + (L)(m+1)/2 \qquad (4.2-5)$$

The cost of the mux type interconnection (Figure 4.2.1(c)) will consist of the control logic area, the pass transistor area, mux-to-destination lead area, and the

source-to-mux lead area. Note that there is no lead cost associated with the control logic, as this is always part of the mux. Let :

$$\text{Mux Control Cost} = \text{Area of control logic.}$$
$$= (B)(\log_2 n) \qquad (4.2-6)$$

$$\text{Pass Trans. Cost} = \text{Area of pass transistors}$$
$$= (T)(n) \qquad (4.2-7)$$

$$\text{Mux Lead Cost} = \begin{pmatrix}\text{Area of source}\\ \text{to mux leads}\end{pmatrix} + \begin{pmatrix}\text{Area of mux to}\\ \text{destination leads}\end{pmatrix}$$

$$= (L)(n) + (L)(m+1)/2 \qquad (4.2-8)$$

Therefore,

$$\text{Mux Cost} = (B)(\log_2 n) + (T)(n) + (L)(n) + (L)(m+1)/2 \qquad (4.2-9)$$

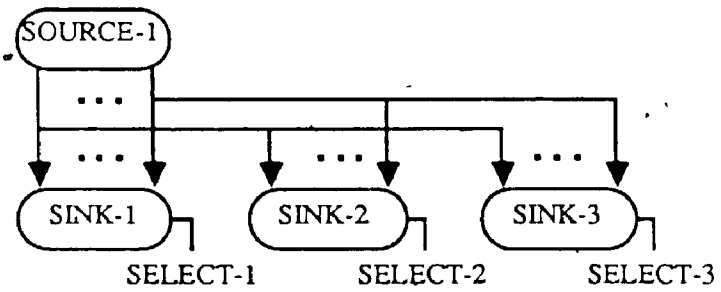
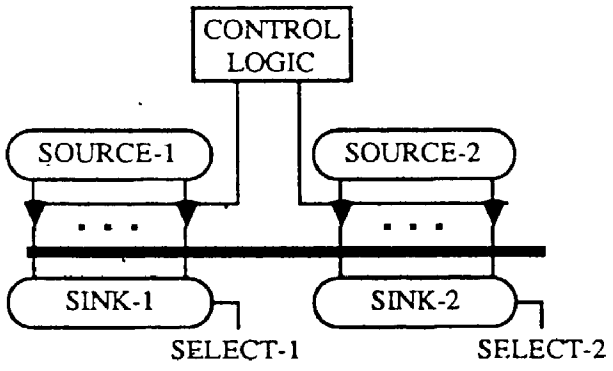
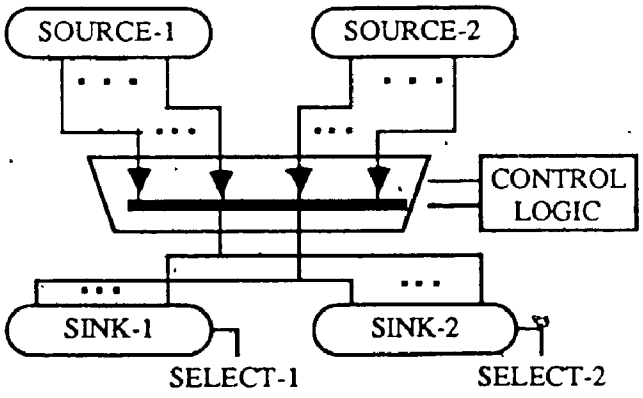
On comparing the mux cost with that of the bus (equations (4.2-9) and (4.2-5), respectively, it can be seen there is no difference between a bus and a mux. This can be explained by the fact that the transfers were assumed to be one bit wide. Any data lead area saved by using the bus was consumed by an additional expense in control lead area. However, this difference will no longer exist when b-bit wide transfers are considered (Section 4.2.2).

4.2.2 Cost-Scheme-2

Cost-scheme-2 is similar to the one presented in Section 4.2.1, cost-scheme-1, except that b-bit wide transfers are considered. Figure 4.2.2 illustrates the three types of interconnection, bus, mux and lead.

Figure 4.2.2: **COST-SCHEME-2 " B-BIT WIDE TV'S "**

Let: L = Cost (area) of one lead,
T = Cost (area) of one pass transistor, or other type,
B = Cost (area) of one control bit logic,
n = number of inputs to bus or mux,
m = number of outputs from a bus or mux
b = number of bits per TV.

CASE	INTERCONNECT	COST
(A) LEAD		L(b)(m+1)/2
(B) BUS		(B)Log(n) + (T)(n)(b) + (L)(n) +L(b)(m+1)/2
(C) MUX		(B)Log(n) + (T)(n)(b) + (L)(n)(b) +L(b)(m+1)/2

Let: L = the cost (area) of one lead,

T =the cost (area) of one pass transistor, or other type,

B =the cost (area) of one control bit,

n =the number of inputs to a bus or mux,

m =the number of outputs from a bus, mux, or a lead (i.e.- fanout).

b =the number of bits in the transfer (assumed to be non-varying).

The cost of a b-bit wide fanout lead connection (Figure 4.2.2(a)) will be the same as b 1-bit wide fanout connections. Therefore, from (4.2-1),

$$\text{Lead Cost} = Lb(m+1)/2 \qquad (4.2-10)$$

The cost of a b-bit wide bus configuration (Figure 4.2.2(b)) consists of the leads in the bus itself, including the fanout, the cost of the pass transistor, the cost of the control logic and control leads. Let :

$$\text{Bus Control Cost} = \begin{pmatrix}\text{Area of control}\\ \text{logic}\end{pmatrix} + \begin{pmatrix}\text{Area of control leads}\\ \text{from control logic to}\\ \text{pass transistors}\end{pmatrix}$$

$$= (B)(\log_2 n) + (L)(n) \qquad (4.2-11)$$

$$\text{Pass Trans. Cost} = \text{Area of pass transistor}$$

$$= (T)(n)(b) \qquad (4.2-12)$$

$$\text{Bus Lead Cost} = \text{Area of bus fanout}$$

$$= (L)(b)(m+1)/2 \text{ from (4.2-10)} \qquad (4.2-13)$$

Therefore, the bus cost is :

$$\text{Bus Cost} = (B)(\log_2 n) + (T)(n)(b) + (L)(n) + (L)(b)(m+1)/2 \qquad (4.2-14)$$

The cost of a b-bit wide mux connection (Figure 4.2.2(c)) will consist of the input-to-mux and output-from-mux lead cost, pass transistor cost, and control logic cost. Note that with a mux configuration there is no control-to-mux lead cost associated with the control cost. Therefore,

$$\text{Mux Control Cost} = \text{Area of control logic}$$
$$= (B)(\log_2 n) \qquad (4.2-15)$$

$$\text{Pass Trans. Cost} = \text{Area of pass transistors}$$
$$= (T)(n)(b) \qquad (4.2-16)$$

$$\text{Mux Lead Cost} = \begin{pmatrix}\text{Area of source}\\ \text{to mux leads}\end{pmatrix} + \begin{pmatrix}\text{Area of mux to}\\ \text{destination leads}\end{pmatrix}$$

$$= (L)(n)(b) + (L)(b)(m+1)/2 \qquad (4.2-17)$$

Therefore,

$$\text{Mux Cost} = (B)(\log_2 n) + (T)(n)(b) + (L)(n)(b) + (L)(b)(m+1)/2 \qquad (4.2-18)$$

Comparing the cost of the bus, equation (4.2-14), with that of the mux, equation (4.2-18), yields,

$$\text{Bus Cost} \overset{\text{versus}}{\Longleftrightarrow} \text{Mux Cost}$$
$$B\log_2 n + Tnb + Ln + Lb(m+1)/2 \overset{\text{versus}}{\Longleftrightarrow} B\log_2 n + Tnb + Lnb + Lb(m+1)/2$$
$$Ln \overset{\text{versus}}{\Longleftrightarrow} Lbn$$

Therefore, it can be seen that the mux is $Ln(b-1)$ more expensive than the bus. This difference can be explained by the fact that the Ln term in the bus cost

equation represents the cost of the leads from the control unit to each set of pass transistors - only one lead per source is required. Whereas, the *Lbn* term in the mux cost equation represents the cost of the leads from each source to the mux - there are b leads for each source.

4.2.3 Example

Cost-scheme-1 was used to evaluate the cost of the interconnection that was generated for the example from Section 4.1. The number of inputs and outputs, n and m respectively, were determined, and the appropriate equation selected to produce the cost of a connection. For $n > 1$, equation (4.2-5) is used, while for $n = 1$ (fanout only), equation (4.2-1) is used. The cost of all the connections were summed. Results are shown in Figure 4.2.3. It should be noted that *B-Sum* is the cost of the control logic, $\log_2 n$, per bit. This must be a whole number, hence the *B-Sum* values were rounded up to the nearest whole number.

4.3 Analysis of BAL

This section examines the effects on the performance of **BAL** when the *primary weights* are varied (Section 4.3.1), and when *secondary weights* are not used (Section 4.3.2).

4.3.1 Effect of Varying Primary Weights

In section 4.1.1, a description of the seven cases for selecting a particular TV to be assigned to a particular bus were presented. Each case was given a priority factor, or *primary weight*. A TV that received the highest weight, with respect to an MIS-BUS, of all TV's on all MIS-BUSES, was selected to be assigned to that bus. However, it was not clear which set of weights would give the best results, and the lowest cost overall. Therefore, connections for a number of different circuits were generated with different sets of weights (Figure 4.3.1), and the resulting cost examined. Figure 4.3.2 summarizes these results.

Figure 4.2.3: **COST OF BAL GENERATED INTERCONNECT**

This figure show the cost of BAL generated interconnect (Circuit in Figure 2.7) using cost-scheme-1.

L = Cost (area) of one lead (L = 1),
T = Cost (area) of one pass transistor, or other type, (T = 1)
B = Cost (area) of one control bit logic (B = 0.5),
n = number of inputs to bus or mux,
m = number of outputs from a bus or mux.

BUS	TRANSFER VARIABLES			N	M	COST		
	INDEX	SOURCE	SINK			B SUM	T SUM	L SUM
1	4	R2	OP1.IN1	3	2	2	3	4.5
	19	DX	OP1.IN1					
	6	R3	OP1.IN1					
	7	R3	OP3.IN2					
	5	R2	OP3.IN2					
2	8	R4	OP2.IN2	3	2	2	3	4.5
	20	DX	OP2.IN2					
	21	DX	OP3.IN1					
	11	R5	OP3.IN1					
3	10	R5	OP2.IN1	2	1	1	2	3
	2	R1	OP2.IN1					
4	12	R6	OP5.IN2	1	3	-	-	2
	13	R6	R2					
	14	R6	R3					
5	24	C5	OP1.IN2	3	2	2	3	4.5
	22	C3	OP1.IN2					
	1	R1	OP1.IN2					
	3	R1	OP4.IN1					
6	9	R4	OP4.IN2	1	1	-	-	1
7	15	OP1	R5	1	1	-	-	1
8	16	OP2	R4	1	1	-	-	1
9	17	OP3	R6	1	1	-	-	1
10	18	OP4	R1	1	1	-	-	1
11	23	A	OP5.IN1	1	1	-	-	1

TOTAL COST = 39

Figure 4.3.1: **DIFFERENT SETS OF PRIMARY WEIGHTS USED TO DETERMINE WHICH IS BEST**

CASE NO.	DESC.	WEIGHTS USED TO GENERATE CONN.					
		SET-1	SET-2	SET-3	SET-4	SET-5	SET-6
1	The two transfers occur in the same time step, therefore, they cannot be combined.	NIL	NIL	NIL	NIL	NIL	NIL
2	The two transfers have nothing in common.	0	0	0	0	0	0
3	The two transfers have the same source, but occur at different times.	1	1	2	1	2	1
4	The two transfers have the same source, and occur at the same time.	2	4	4	3	3	2
5	The two transfers have the same sink, and occur at different times.	3	2	1	2	1	3
6	The 3rd transfer has the same source& sink, & occurs at different times.	4	3	3	4	4	4
7	The 3rd transfer has the same source& sink, & occurs at the same time.	5	5	5	5	5	5

Figure 4.3.2: **RESULTS OF RUNNING BAL WITH DIFFERENT SETS OF PRIMARY WEIGHTS**

This table shows the resulting interconnect cost for connections generated by BAL using different sets of primary weights. These weight sets are summarized in Figure 4.3.1. Weight Set-1 produced consistently the best results and is fully described in Figure 4.1.10.

CIRCUIT NAME	COST ** (WITH SECONDARY WEIGHTS)				
	SET-1	SET-2	SET-3	SET-4	SET-5
Ex-6a	91	100.5	116	98.5	116
Ex-6b	93.5	93.5	115	93.5	115
Ex-6c	80	80	89.5	80	89.5
Ex-7a	96	99.5	120	99.5	120
Ex-7b	104.5	103	119	105	118.5
Ex-7c	79	79	100	79	100
Ex-7d	85.5	85.5	100	85.5	100
NUMBER OF TIMES LOWEST COST PRODUCED	6	5	0	4	0

** Cost of interconnect is evaluated using "cost-scheme-1" (section 4.2.1).

Weight-set-1 produced the lowest interconnect cost the most times (6/7) over all the other weight sets. Furthermore, the one time that this set of weights did not produce the lowest costing configuration, it generated a circuit with a cost that was within 1.5 % of the lowest cost circuit. This is the set of weights that was fully described in Section 4.1 (Figure 4.1.10).

It is interesting to note that TV's with common destinations are given a higher weight than those with common sources. This can be explained by the fact that it is better to favour the destinations to avoid having to *mux* outputs from different buses that go to the same destination. For instance, consider the example that was shown in Figure 2.14. There were six TV's that were to be partitioned into groups, with each TV in a group being assigned to the same bus. There were two ways that these TV's could have been be partitioned: *Choice-1*, which favoured destinations, and *Choice-2* which favoured sources.

Choice-2 partitioned the TV's into two groups such that those TV's, with the same destination (*OP1.IN2*), were assigned to different groups. This required an extra mux at this destination (*OP1.IN2*) to be able to transfer the data from the two different groups (Bus1 and Bus2).

Choice-1, which favoured the destinations, partitioned the TV's such that those TV's with common destinations (*OP1.IN2*) were assigned to the same group. Hence each bus could go directly to the destinations without any extra muxes being required.

It should be noted that although these weights are the recommended ones, the user may change them.

4.3.2 Effect of Secondary Weights on BAL Performance

Secondary weights, as explained in Section 4.1.1, are used to decide which TV should be assigned to which MIS-BUS, when more than one TV has the same highest primary weight. Its role was to allow BAL to look ahead to what might happen in the next step if a particular TV was chosen first. However, it was not

clear if the overall cost of the circuit would be reduced by implementing this complex *look-ahead* feature (Secondary weights calculations required an additional N^2 order of complexity, as described in Section 4.1.6).

In order to evaluate the effectiveness of the secondary weight system, interconnections for several large and complicated circuits were generated, with and without the use of these secondary weights. The resultant cost of the connections were then analyzed (Figure 4.3.3). In some cases, there was only a marginal reduction in cost when secondary weights were employed, in other cases, there were actually slight increases. Over all the cases examined, there was only a 0.5% improvement in cost, if the secondary weight system was used. When secondary weights are not included in **BAL**, its complexity is reduced from order N^4 to N^2 (Section 4.1.5 & 6). This represents a significant reduction in execution time. It far outweighs the increase in the cost of interconnections. Hence, the use of secondary weights, in **BAL**, is not recommended.

4.4 Comparison of BAL with Other Methods

This section examines the effectiveness of **BAL**, in generating interconnect, with respect to other methods. All comparisons should not be construed as being statistically accurate, as some were based on only a few small examples, obtained from the literature. **BAL** will be compared with:

- **FACET** (Section 4.4.1),
- **HAL** (Section 4.4.2),
- **CHIPPE** (Section 4.4.3).

4.4.1 FACET Versus BAL

The **FACET** design automation system used a modified clique partitioning algorithm to assign transfers to buses, or muxes (Section 3.2.1). The algorithm was modified to include the notion of *divide and conquer*, and to include the *number of*

Figure 4.3.3: **RESULTS OF RUNNING BAL WITHOUT SECONDARY WEIGHTS**

This table shows the resulting interconnect cost for connections generated by BAL with and without the use of secondary weights. Note that there is not a great deal of improvement when secondary weights are used.

CIRCUIT NAME	COST **		% IMPROVEMENT OF 'WITH' / 'WITHOUT'
	WITH	WITHOUT	
Ex-6a	91	92	1.1 %
Ex-6b	93.5	93	-0.5 %
Ex-6c	80	79.5	- 0.6 %
Ex-7a	96	92.5	- 3.6 %
Ex-7b	104.5	100.5	- 3.8 %
Ex-7c	79	82	3.8 %
Ex-7d	85.5	91.5	7.0 %
AVERAGE % IMPROVEMENT OF USING SECONDARY WEIGHTS			0.48 %

** Cost of interconnect is evaluated using "cost-scheme-1" (section 4.2.1) with interconnect being generated using weight set-1 for primary weights.

neighbours and *transitive* properties. These properties modified the classic clique partitioning algorithm of NP complexity to one of polynomial complexity [TsSi86].

Interconnect for three different circuits were generated by **BAL** and by the **FACET** algorithm. Both methods produce almost the exact same interconnect (Figure 4.4.1). This is probably due to the fact that the example circuits are small. Unfortunately, interconnect for larger circuits could not realistically be generated, by hand, with the **FACET** algorithm. It is probably reasonable to conclude that both methods are quite good. However it is believed that the **BAL** approach is better for the following reasons:

- **BAL** uses a more global allocation scheme in that it examines all combinations (all TV's with each MIS bus) before making an assignment. **FACET**, on the other hand, uses a more greedy technique, as it insists on exhausting a category (i.e.- assigning all the TV's from the same category) before TV's from other categories can be considered.
- **BAL** assigns TV's using a weighting scheme that distinguishes between different situations (i.e.- common source, common destination, common source and destination, etc.), whereas **FACET**'s weighting scheme is based on the total number of these situations (total number of *common neighbours* in a category). In other words, it considers those TV's with common sources, common destinations, common sources and destinations, etc. to all have the same weight. This is not a valid assumption as was shown in Section 4.3.1.
- **BAL** has a complexity of N^2, whereas **FACET**'s must be at least N^3.

It should be noted that the **FACET** system was developed before July, 1986, and hence it is probable that it has since been improved.

4.4.2 BAL Versus HAL

The **HAL** approach to improving interconnections was described in Section 3.2.2. The approach of that algorithm was:

Figure 4.4.1: **'BAL' VERSUS 'FACET'**

DATA SET NO.	NAME	FACET COST	FACET NO. OF 'BUSES'	BAL * COST	BAL * NO. OF 'BUSES'	BAL VS. FACET % IMPROVEMENT
1	Facet-DP	30.5	4	30.5	4	0 %
2	HAL-DP	25	3	25	5	0 %
3	HAL-Ex	37	4	36.5	4	1.4 %

* without secondary weights.

Figure 4.4.2: **'BAL' VERSUS 'HAL'**

DATA SET NO.	NAME	HAL COST	HAL NO. OF 'BUSES'	BAL * COST	BAL * NO. OF 'BUSES'	HAL VS. BAL % IMPROVEMENT
1	Hal-3	35	4	36	4	2.8
2	Ex-6a	88.5	9	98	10	10.7
3	Ex-6b	93	9	93	9	0
4	Ex-6c	73	6	79.5	8	8.9
5	Ex-7a	89	9	92.5	9	3.9
6	Ex-7b	99	10	100.5	10	1.5
7	Ex-7c	79	7	82	8	3.8
8	Ex-7d	86	7	91.5	9	6.4

* without secondary weights.

- to assign one mux for each unique destination,
- evaluate all combinations of merged compatible muxes, that have at least one common input and are disjoint in time,
- to select the set of merges that yielded the lowest cost interconnect.

The **HAL** and **BAL** systems differ in a number of ways. **HAL** uses a more global method, in that it generates many interconnects, which are then evaluated. **BAL**, on the other hand, uses a more greedy technique, in that it assigns the best (highest weighted) transfers first. As well, the complexity of the two approaches differ. **BAL** was determined to be of order N^2, where N was the number of transfer variables. Whereas the **HAL** interconnect algorithm is exponential, where N is the number of unique destinations (muxes). However, its author argues that the number of different combinations of compatible muxes is typically small [Paul87].

A study of the performance of these two approaches was conducted by having **HAL** and **BAL** produce connections for eight fairly complex circuits. The result of comparing the generated interconnections is summarized in Figure 4.4.2. It was found that, on average, **BAL** produced interconnect that was within 5% of the cost of that generated by **HAL**. Thus **BAL** performs quite favourably with respect to **HAL**, as the latter system uses considerably more search.

4.4.3 CHIPPE Versus BAL

The **CHIPPE** system used a knowledge based approach to automatically generate circuits (Section 3.2.3). The Slicer and Splicer modules were responsible for creating the initial design, and any refinements to that design, were made under the direction of the *design critic*. This design critic also determined whether or not the circuit had met the requirements. Interconnection was optimized only as the refined circuit approached performance and area constraints.

Two examples of circuits with **CHIPPE** optimized interconnect were taken from [BrGa87], one of which is shown in Figure 3.2.21. The same circuit with **BAL** generated connections is shown in Figure 4.4.3. **BAL** produced interconnect that

was 30% less expensive, for the first circuit, and 15% less costly for the second circuit (Figure 4.4.4). Based on these two examples, **BAL** has a better method of optimization. However, it is not clear whether the **CHIPPE** system performs more poorly because of its rule based approach, or simply because its rules were not refined enough.

Figure 4.4.3: 'BAL' GENERATED INTERCONNECT (DATA SET "GAJSKI-3")

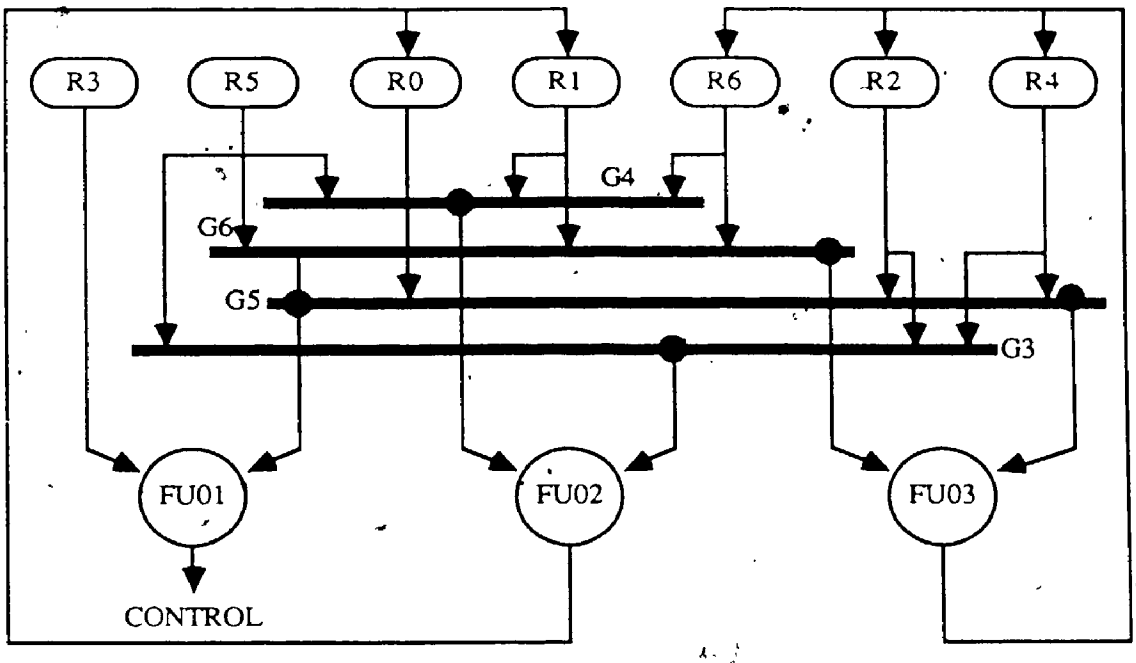


Figure 4.4.4: 'BAL' VERSUS 'CHIPPE'

DATA SET NO.	NAME	CHIPPE COST	CHIPPE NO. OF 'BUSES'	BAL * COST	BAL * NO. OF 'BUSES'	BAL VS. CHIPPE % IMPROVEMENT
1	Gajski-3	48	6	37	4	29.7
2	Gajski-4	52.5	6	45.5	5	15.4

* without secondary weights.

Average improvement of BAL over CHIPPE : 22.6%

Chapter 5
RAL - A Register Allocation Algorithm

This chapter presents an algorithm (**RAL**) to perform register allocation. Register allocation is that task which assigns the variables that are generated by the operators (adders, multipliers, etc) to be stored in specific registers (see Section 2.3.2.4). Finding the set of assignments that yields the fewest registers, and least amount of area cost is an *NP complete* problem and every solution must be determined before the best can be selected.

5.1 Register Allocation - Problem Definition

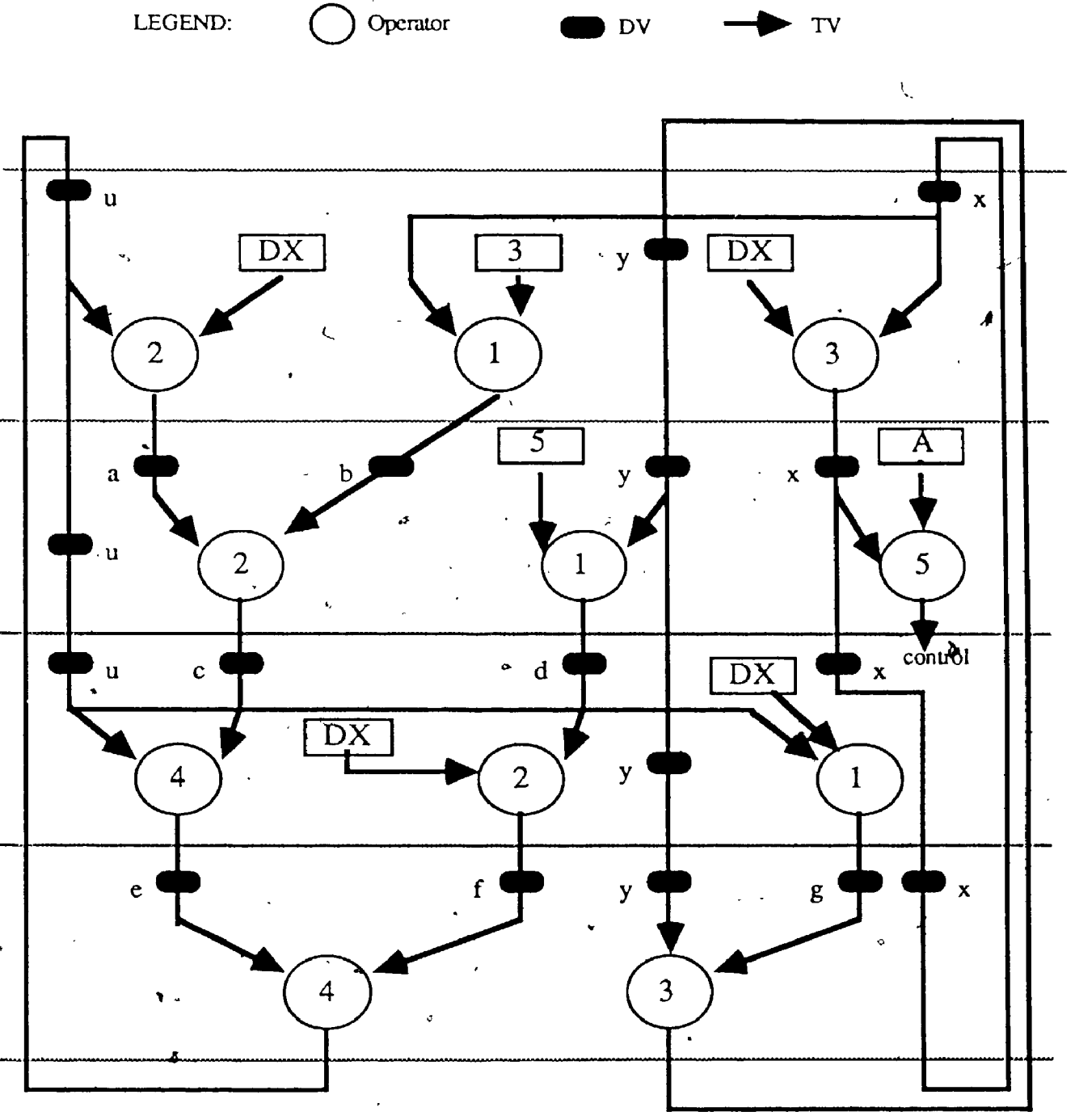
Register allocation is quite similar to that of assigning transfer variables to buses (i.e.- bus allocation). In both instances, a minimum number of resources, buses and registers, are to be shared by a maximum number of objects, transfers and data, respectively. The algorithm that was developed for the bus allocation problem (**BAL**) can be applied in a similar fashion to that of register allocation with a few modifications. These modifications are based on the differences between transfer variables (TV's) and data variables (DV's):

1) TV's represent the transfers of data that occur in the circuit. They can be seen as edges between nodes and operators on the data flow graph (Figure 5.1). DV's, on the other hand, represent the data, itself, which must be stored. They can be seen as the heavy black nodes in the DFG (Figure 5.1).

2) Each TV, by definition, has only one source and one destination (i.e.- the transfer of data is from one unique point to another). On the other hand, a DV has one source, but may have more than one destination (i.e.- a variable is generated by only one operator, although that variable

Figure 5.1: **DFG (PRE REGISTER, ALLOCATION)**

This figure illustrates a DFG after the operations have been assigned to operators (ie- adders multipliers). The data variables (DV's) generated by the operators must be assigned to registers. These DV's are also listed in the DV table in Figure 5.2.



may subsequently be used by more than one operator). For instance, in Figure 5.2, the register node u has only one input, from operator-4, but it feeds more than one destination, operator-2, operator-4, and operator-1.

3) Two TV's may share a bus if they are disjoint in time or if they both have the same source. However, two DV's may only share a register if they are disjoint in time.

4) The decision to allow specific TV's to share a bus is directed by a *profit weight* that reflects the area cost of such a sharing (Section 4.3.1). In the same manner, there will also be certain profits associated with sharing a register among a particular set of DV's. However, this decision will be based on a different set of criteria. Figure 5.3 illustrates six situations, each with a unique weight, or *profit factor*, for which it would be beneficial to have two, or more, DV's share a register. The weight assigned to each case is based on the change in circuit cost. The greater the saving, the greater the weight.

Case 1: This illustrates the situation in which there are two DV's, each with differing sources and destinations. Merging these would require a mux, but would save one register and some lead. The lowest weight has been assigned to this situation, as the sources and destinations involved could be physically far apart. Thus, increasing the likelihood of the resultant circuit having a large lead cost.

Case 2: This situation reflects the case in which the source of one of the DV's is the same as the destination of the other DV. Although, this has the same net cost, as the first case, with respect to registers, muxes and leads, it has been assigned a higher weight. This is to reflect the possibility of a smaller lead cost due to there being only three operators (S1, Sx/Dx, and D2).

Case 3: This reflects the situation in which two candidate DV's have destina-

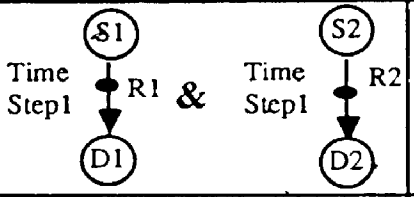
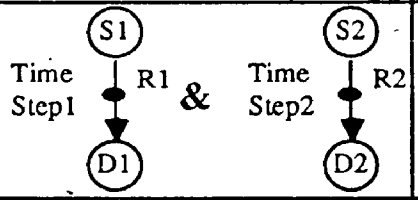
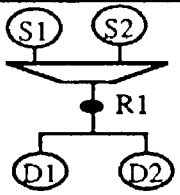
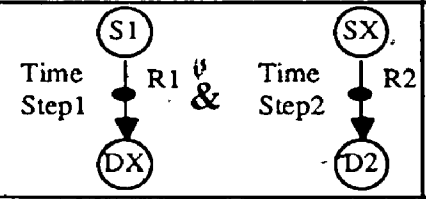
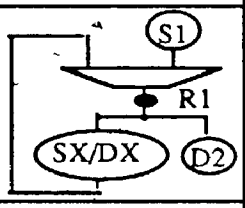
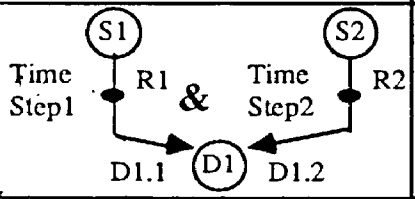
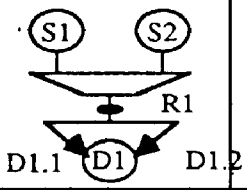
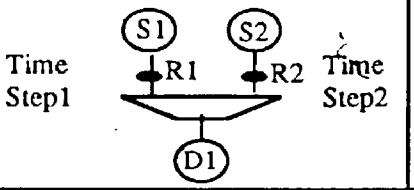
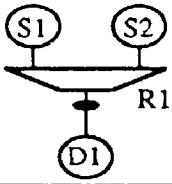
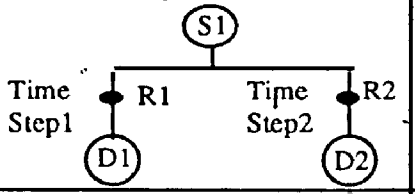
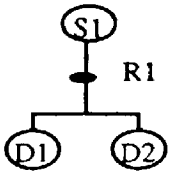
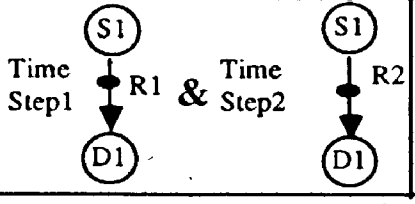
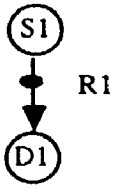
Figure 5.2 **DATA VARIABLE (DV) LIST**

Shown below are the DV's obtained from the DFG of Figure 5.1. The main difference between the DV list and a TV list is that in the former there may be more than one destination, while in the latter there will only be one destination.

For instance DV 'u' has source 'OP4' but feeds destinations 'Op2.1', 'Op1.2', and 'Op4.1'.

NAME	SOURCE	DESTINATION(S)	TIME STEP
a	OP2	OP2.2	2
b	OP1	OP2.1	2
c	OP2	OP4.2	3
d	OP1	OP2.1	3
e	OP4	OP4.1	4
f	OP2	OP4.2	4
g	OP1	OP3.1	4
u	OP4	OP2.1, OP1.2, OP4.1	1, 2, 3
x	OP3	OP5.2, OP3.2, OP1.1	1, 2, 3, 4
y	OP3	OP1.1, OP3.2	1, 2, 3, 4

Figure 5.3 **WEIGHTS FOR COMBINING REGISTERS USED BY RAL**

CASE NO.	DESC.	BEFORE	AFTER	△ COST	WGT
0	The two DV's occur in the same time step, therefore, they cannot be combined.				NIL
1	The two DV'S have nothing in common.			- 1 Register + 1 Mux	0
2	The two DV'S have the same source-destination			- 1 Register + 1 Mux - 0.5 Leads	1
3	The two DV's have the same sink, but different ports.			- 1 Register + 1 Mux - 0.5 Leads	1.5
4	The two DV's have the same destination.			- 1 Register - 0.5 Leads	2
5	The two DV's have the same source, but different sinks. (destinations)			- 1 Register - 1 Leads	3
6	The two DV's have the same source & sink.			- 1 Register - 2 Leads	4

tions in the same locality, i.e.- different ports on the same operator. Again, the net saving with respect to muxes, registers, and lead, is the same as that of the the first two cases. However, merging these DV's would probably result in a less costly circuit, since their destinations will be quite close together in the layout.

Case 4: The two DV's have the same destination, but have different sources. Merging them, results in the same net cost as in the first three cases. However, the resultant circuit would cost less than the circuits generated by any of the first three types of merge. Hence, it has been assigned a marginally heavier weight.

Case 5: This illustrates what it would cost if two DV's with the same source were merged. The resulting circuit would cost even less than the circuits generated by the first four situations, as no mux is required. Therefore, a heavier weight, than the first four, has been assigned.

Case 6: This reflects the least costly situation. Both DV's have the same source and destination. It has been assigned the largest weight.

5.2 RAL Description

The RAL algorithm is as follows:

Step 1: Determine the minimum number of registers required for each time step (Figure 5.4). This is done by examining either the DV list (Figure 5.2) or the DFG (Figure 5.1) and counting the number of unique variables in each time step. The time step that requires the most registers is then selected for Step 2. The data-variables, that exist in this time step form the maximally incompatible set (MIS). If there is more than one time step with the same maximum number of unique variables, then one is chosen arbitrarily.

Step 2: Assign the MIS variables, determined in Step 1, to separate registers. The *profit weights* of the remaining variables with respect to each of the MIS variables

Figure 5.4 **STEP 1: DETERMINE MINIMUM NUMBER OF REGISTERS REQUIRED PER TIME STEP**

This figure shows the minimum number of registers that are required to store the variables in each time step. This can be determined by examining the DV list (Figure 5.2) or the DFG (Figure 5.1) and counting the number of unique variables in each time step.

The time step that requires the most registers, the MIS time step, is then selected for 'Step 2'. The variables in this time step then become the MIS variables. However since there is more than one MIS time step in this example, one of them has been chosen arbitrarily.

TIME STEP	MINIMUM NUMBER OF REGISTERS
1	3
2	5 ← Choose
3	5
4	5

should then be calculated for all disjoint combinations (i.e.- if there is no lifetime conflict), as shown in Figure 5.5. The *profit weights* are those weights described earlier (Figure 5.3).

Step 3: REPEAT

Step 3.1: Assign the variable with the highest weight to the corresponding MIS register. If there are more than one variable with the same highest weight, then one is selected arbitrarily (Figure 5.6).

Step 3.2: Update the weights of the remaining variables to reflect the fact that one of these was just assigned to a register in Step 3.1. This will only involve recalculating the weights of those variables that could also have been assigned to the same register (Figure 5.6).

UNTIL There are no more variables remaining to be assigned. (Figure 5.6(b-d))

5.3 Results

The DFG with allocated registers is shown in Figure 5.7. From this, the TV's can be obtained (Figure 5.8), and assigned to buses by **BAL**. The resulting circuit is shown Figure 5.9. Also shown is the design produced by **HAL** (Figure 5.9), and by **CHIPPE** [PaGa87] (Figure 5.10) for the same scheduled data flow graph (i.e.- the same problem).

The **RAL/BAL** design produces the exact same register and interconnect allocation as the **HAL** generated design. This is an interesting result as **HAL** uses a bounded clique-partitioning approach (order NP) to assign DV's to registers and TV's to muxes (see Section 3.2.4), while the **RAL/BAL** algorithms are of order N^2.

Figure 5.11 summarizes a comparison between the **CHIPPE** produced circuit and that generated by **RAL/BAL**. The **RAL/BAL** design has one less register, which leads to having fewer muxes, mux inputs, wires, and a lower interconnect cost. However, this is only one example of circuits that can be generated by the

Figure 5.5 **STEP 2: DETERMINE THE WEIGHTS OF THE REMAINING VARIABLES WITH RESPECT TO 'MIS' VARIABLES**

The MIS variables (a, b, u, x, y), determined in Step 1 (Figure 5.4), are assigned to different registers, as shown below. The weights of the remaining variables (c, d, e, f, g) with respect to each of the MIS variables is calculated for time-disjoint combinations.

The variables with the highest weight is then selected to be assigned to the corresponding MIS register (see Step 3). Since there are two DV's with same highest weight (d->Reg2 and e->Reg3), one will be chosen arbitrarily.

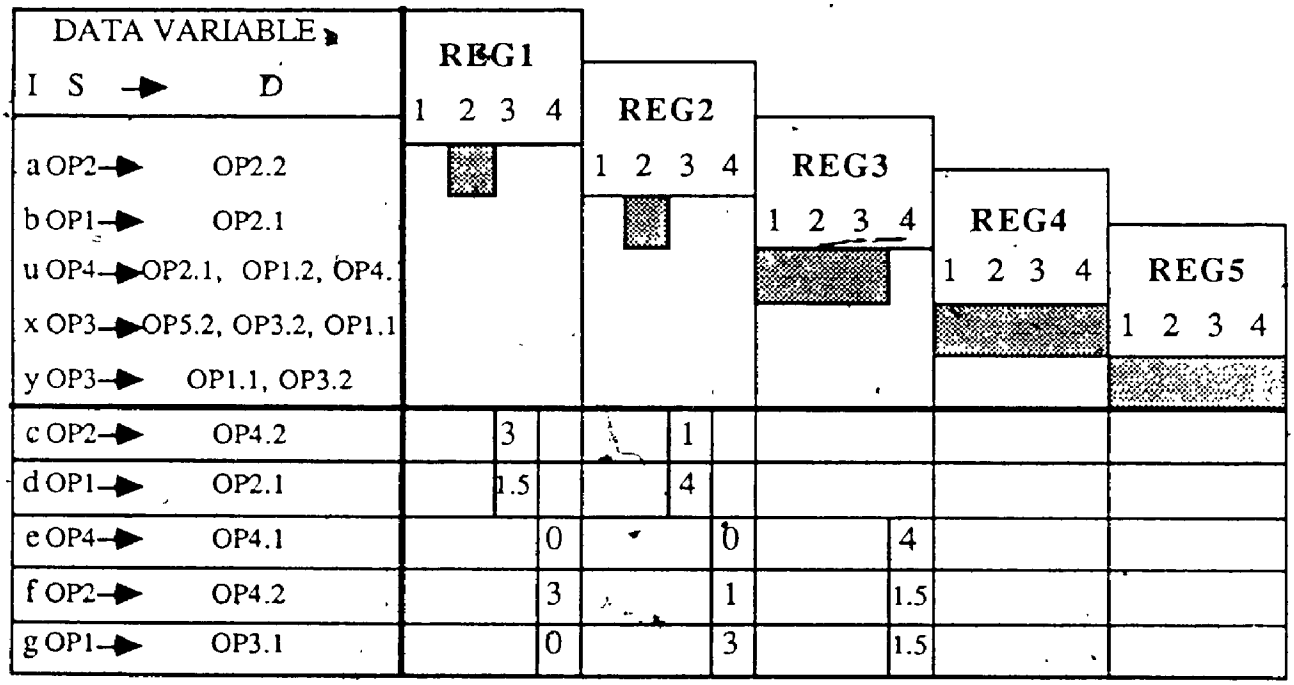
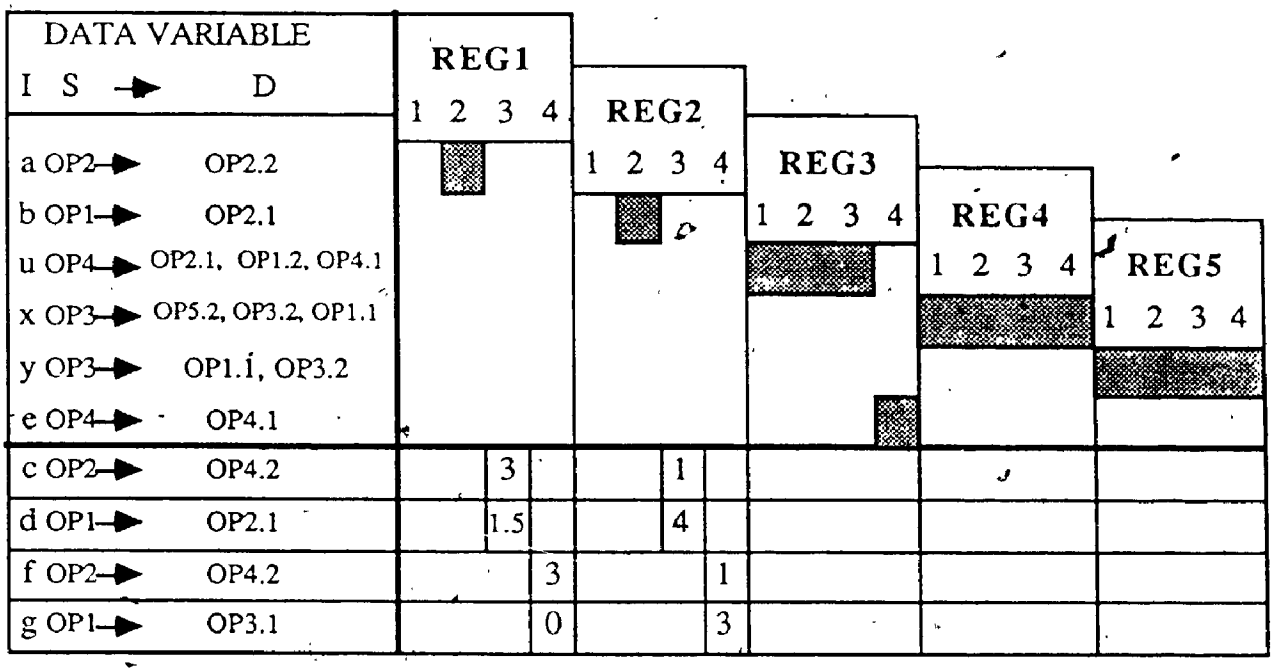


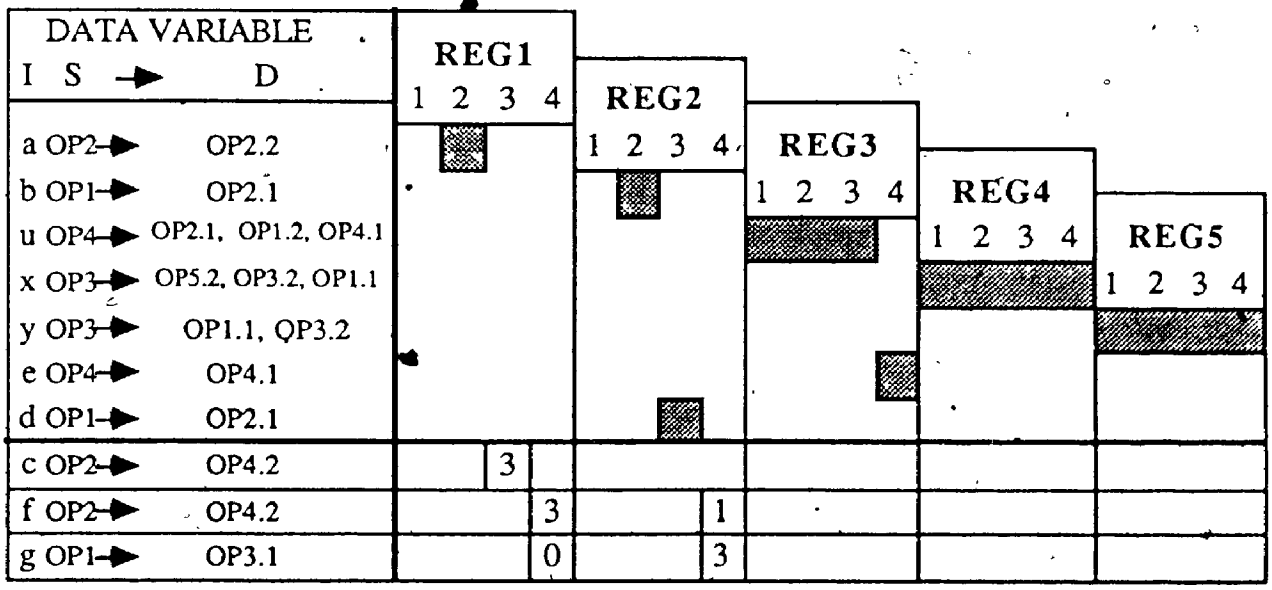
Figure 5.6 **STEP3: REPEAT**
ASSIGN THE VARIABLE WITH THE HIGHEST WEIGHT TO THE CORRESPONDING MIS REGISTER, UPDATE THE OTHER WEIGHTS UNTIL {NO VARIABLES REMAIN TO BE ASSIGNED}

Variable 'e' is assigned to register-3, since it had the highest weight (see Step 2 - Figure 5.5). The remaining variables that could have been assigned to register-3 are updated. Note that no more variables can be assigned to Register-3, hence that column is empty (a).
Variables are assigned in this same fashion until no more variables remain (b-d).

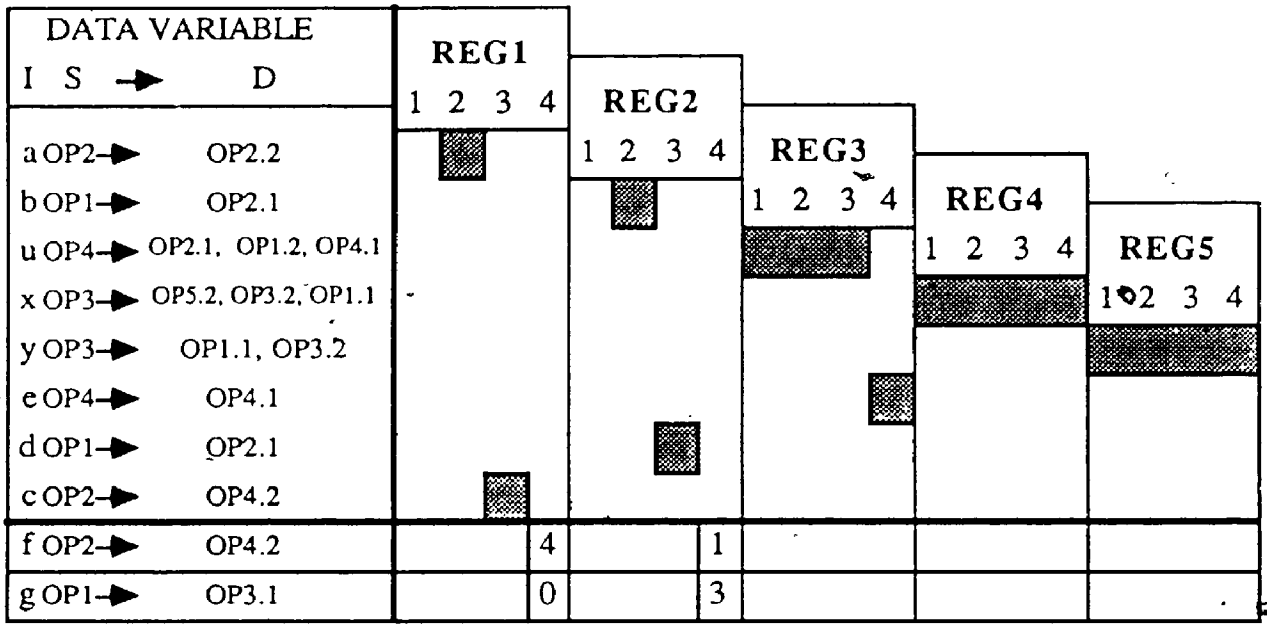
(a) ASSIGN 'e' TO REG3:



(b) ASSIGN 'd' TO REG2:



(c) ASSIGN 'c' TO REG1



(d) ASSIGN 'f' TO REG1 & 'g' TO REG2

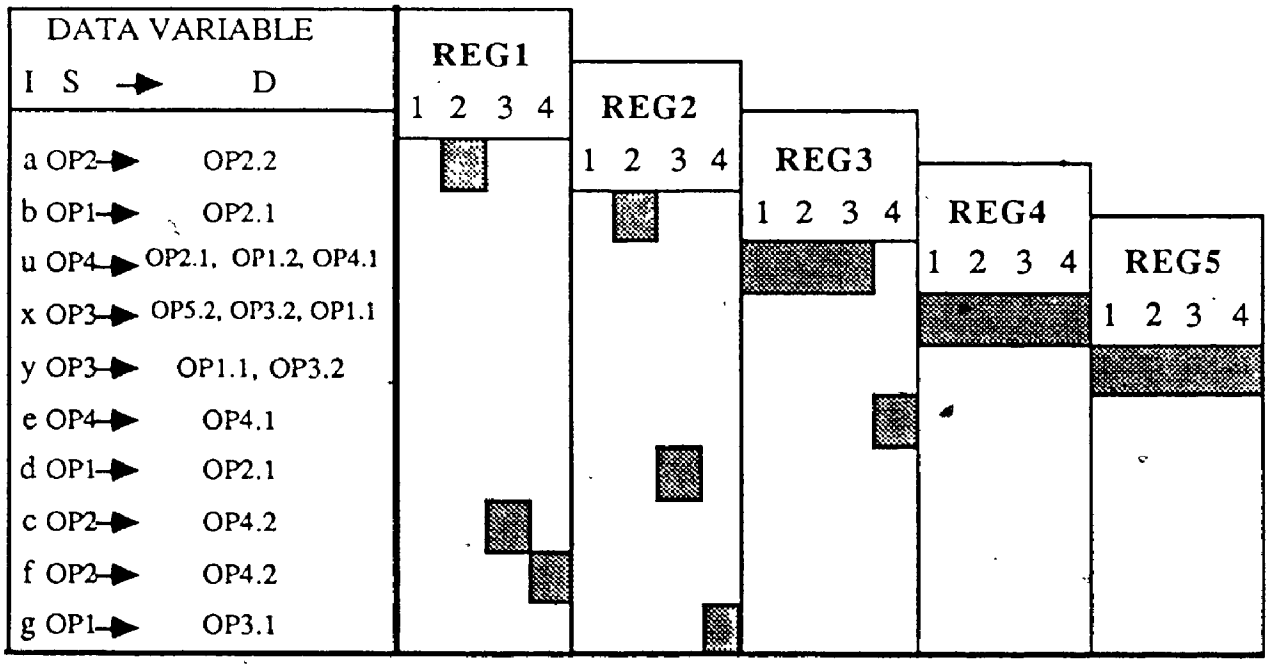


Figure 5.7 **DFG WITH 'RAL' ALLOCATED REGISTERS**

This figure shows the DFG, from Figure 5.1, with the registers that have been allocated by RAL.

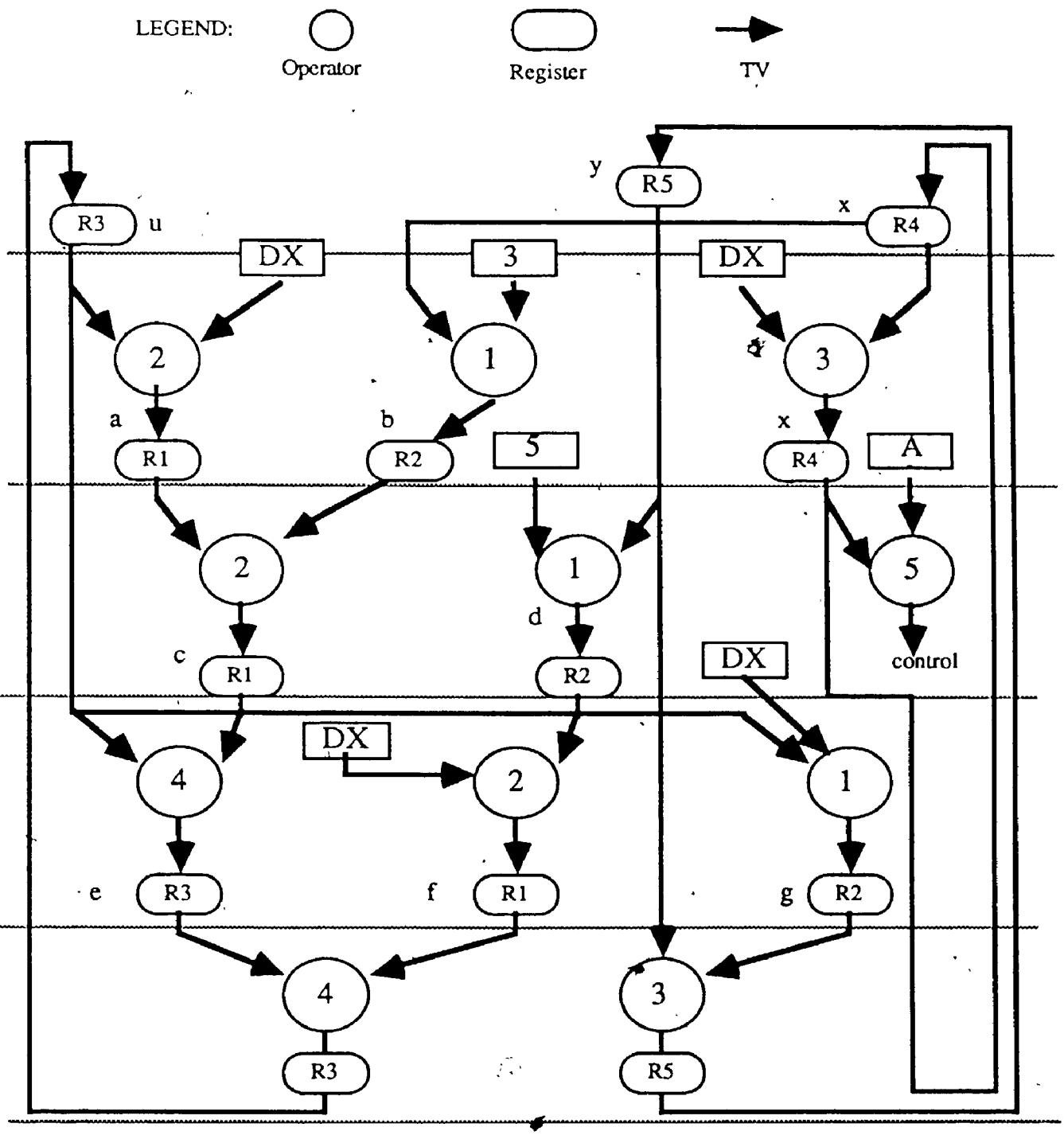


Figure 5.8 **LIST OF TRANSFERS TO BE ASSIGNED TO BUSES**

(Obtained from the DFG shown in Figure 5.7)

TRANSFER VARIABLES			
INDEX	SOURCE	DESTINATION	CYCLES
1	R1	OP2.2	2
2	R1	OP4.2	3, 4
3	OP2	R1	1, 2, 3
4	R2	OP2.1	2, 3
5	R2	OP3.1	4
6	OP1	R2	1, 2, 3
7	R3	OP2.1	1
8	R3	OP4.1	3, 4
9	R3	OP1.2	3
10	OP4	R3	3, 4
11	R4	OP5.2	2
12	R4	OP3.2	1
13	R4	OP1.1	1
14	OP3	R4	1
15	R5	OP1.1	2
16	R5	OP3.2	4
17	OP3	R6	4
18	DX	OP2.2	1, 3
19	DX	OP3.1	1
20	DX	OP1.1	3
21	C5	OP1.2	1
22	C3	OP1.2	2
23	A	OP5.1	2

Figure 5.9 **RAL-BAL GENERATED CIRCUIT FOR EXAMPLE (ALSO 'HAL' GENERATED CIRCUIT)**

This is the circuit, for the example, with RAL generated registers, and BAL generated interconnect. HAL generated the same register and interconnect allocation [Paul87].

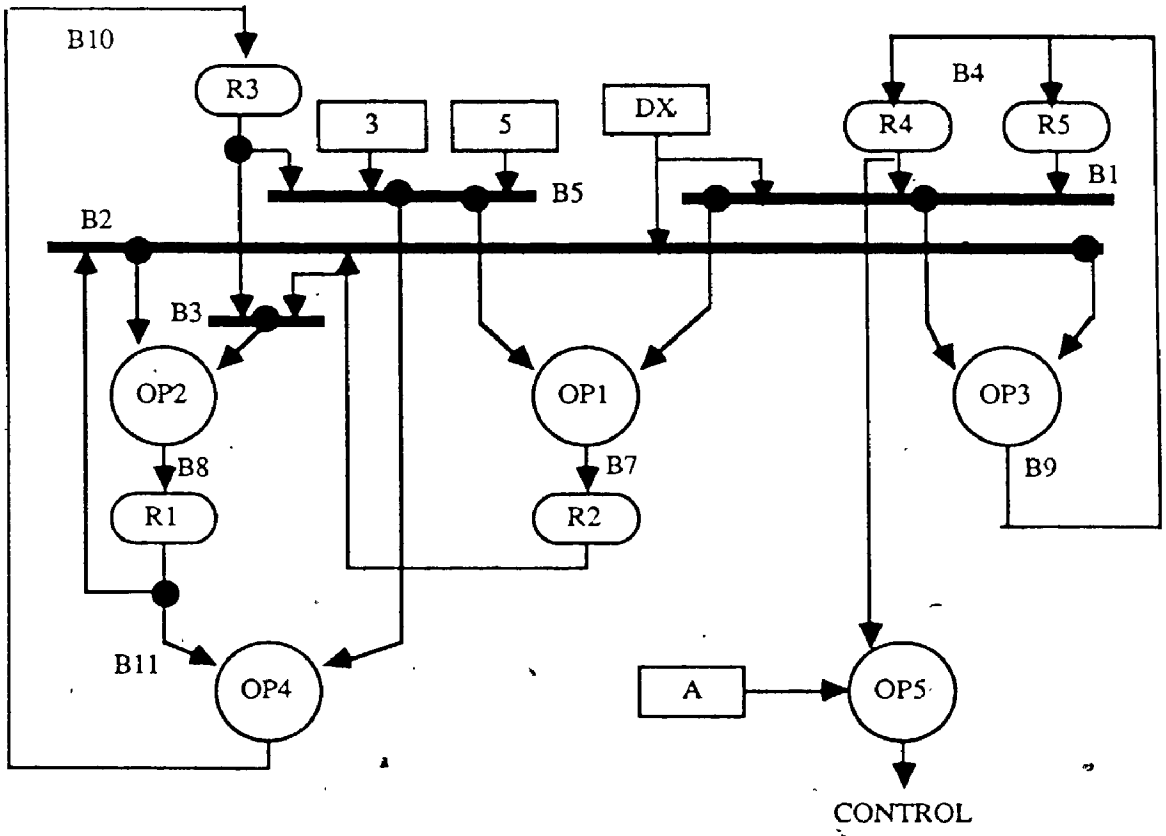


Figure 5.10 **'CHIPPE' GENERATED CIRCUIT FOR EXAMPLE [PaGa87]**

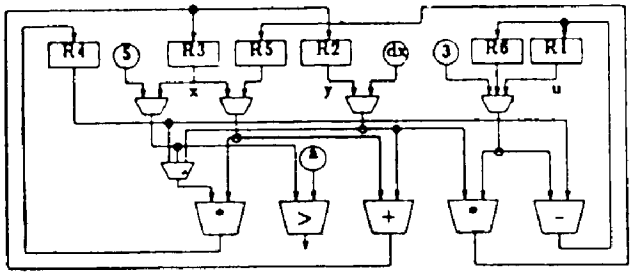


Figure 5.11 **'RAL-BAL' VERSUS 'CHIPPE'**

This figure shows the results of comparing the register and interconnect allocation generated by the 'RAL-BAL' method and the 'CHIPPE' system.

ITEM NAME	RAL-BAL METHOD	CHIPPE SYSTEM	% IMPROVEMENT OF RAL-BAL OVER CHIPPE
Registers	5	6	20 %
Muxes	4	5	25 %
Mux Inputs	11	12	9 %
Leads	26	28	8 %
Connect Cost	38.5	41.5	8 %

CHIPPE system. Moreover, register allocation is only one small task performed by this system.

Chapter 6
LAYBAL - Layout Bus Allocation Algorithm

The synthesis process, as described in Section 2.3, generates an RTL structural description of the circuit. This description captures the design in terms of components (registers, adders, multipliers, etc.) and connections between these components (leads and buses). Note that these connections have been defined before the placement of the components is known. This may cause unnecessary interconnect, as connections from different regions of the chip may inappropriately share the same bus. More efficient interconnect could be generated if the components' placement could be considered during the bus allocation phase of data path synthesis. This section will present a novel bus allocation scheme (**LAYBAL**) that does this.

6.1 Modification of Data Path Synthesis

The data path synthesis process, as defined in Section 2.3, will have to be modified (Figure 6.1) to include a crude placement of the components. Although it is beyond the scope of this thesis to automatically generate such a placement, two general tasks that would have to be performed can be identified:

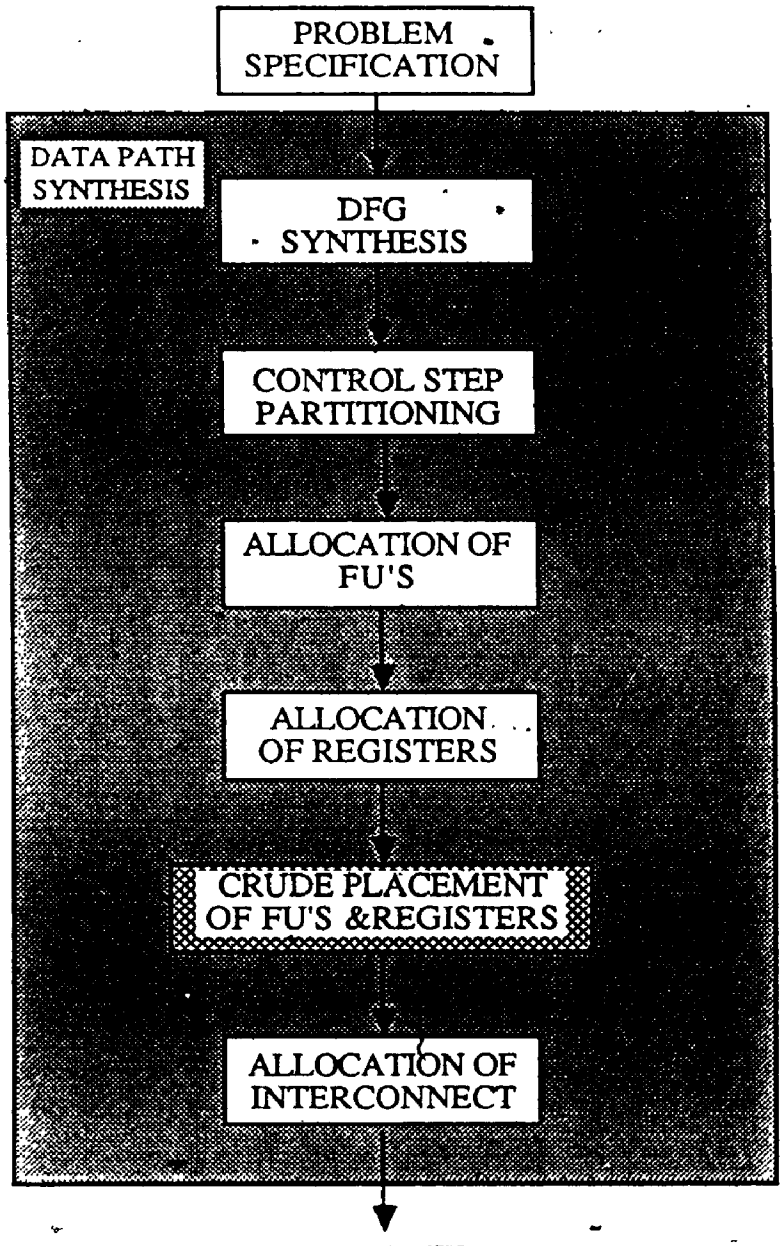
Task 1: To partition the components into modules such that:

- there are approximately equal numbers of components in each module, or each module occupies approximately the same amount of area,
- the number of transfers across partition boundaries are minimized.

Task 2: To place the modules such that those modules, with large amounts of intercommunication, are close together.

Figure 6.1: MODIFICATION TO DATA PATH SYNTHESIS

In order to consider the placement of FU's and registers during the interconnect allocation phase, a crude placement of these components will be required.



The components in a pre-interconnect allocation DFG (Figure 2.13) (Differential equation example from Section 4.1) have been hand partitioned into four modules and placed in a square formation, using these guidelines (Figure 6.2).

6.2 Including Placement Information

The placement information can help determine which transfer variables (TV's) should be grouped together. One of the criteria for a grouping would be the amount of wire that could be shared. For instance, consider the two situations in Figure 6.3, where the layout is known. The transfer R2→ALU has been assigned to the MIS bus. The decision must be made to assign either transfer-1 (R4→ALU) or transfer-2 (R3→ALU) to this bus, both have the same primary weight. In both cases, transfer-1 should be chosen, as the lowest overall wire cost would be produced. In other words, the largest amount of wire would be shared.

The placement information can be included in the TV information, by defining an area through which the transfer data would pass. This area would be represented by a rectangle formed between the source and destination, and be defined by row and column numbers. The list of TV's, with placement information, is shown in Figure 6.4.

6.3 LAYBAL Description

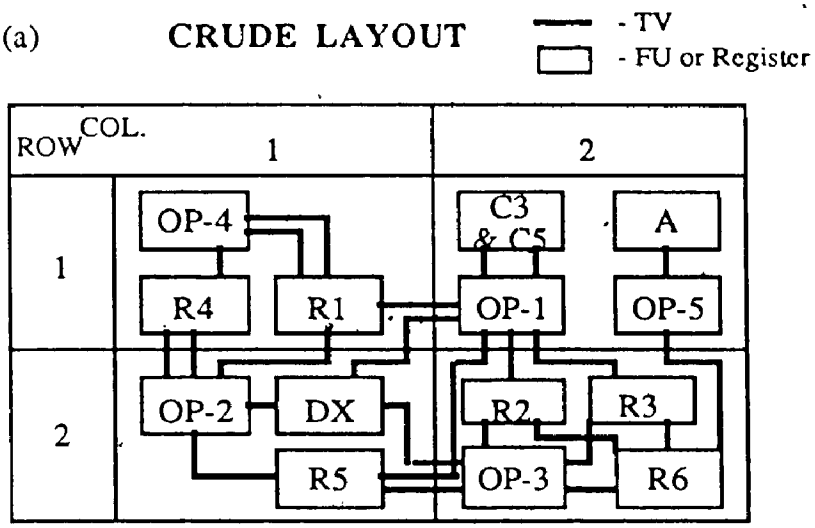
LAYBAL is an algorithm that can perform bus allocation with the circuit layout taken into consideration. It uses the same approach as **BAL**, except that the weighting scheme has been slightly modified to include placement. **LAYBAL** determines the weight of a TV with respect to an MIS bus, by determining both its primary weight (weight of shared sources, sinks, and time steps) and its shared wire weight, as follows :

$$\text{Weight} = \text{Primary\_Weight} + \text{Shared\_Wire\_Weight} \qquad (6-1)$$

Figure 6.2 **CRUDE PARTITION OF CIRCUIT**

The components from the pre-interconnect allocation DFG (Figure 2.13) have been partitioned into four modules, and placed in a square formation (a). Each component occupies a precise location in terms of a "row" and a "column" (b).

(a) **CRUDE LAYOUT**



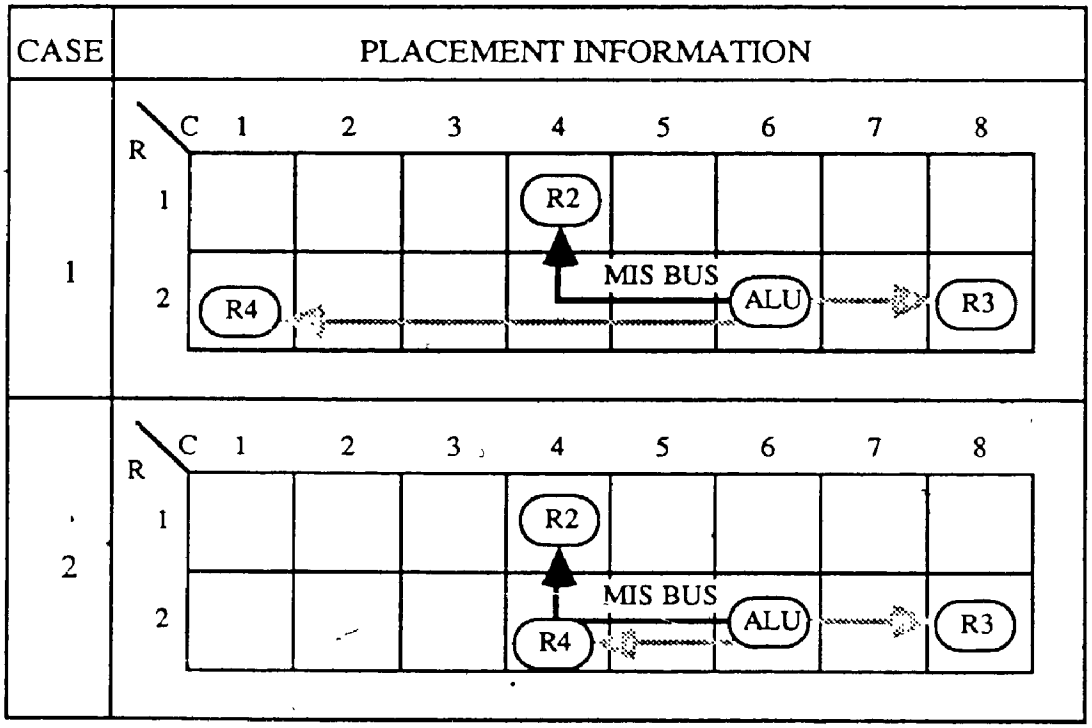
(b) **COMPONENT LOCATION**

COMPONENT NAME	ROW	COLUMN
R1	1	1
R2	2	2
R3	2	2
R4	1	1
R5	2	1
R6	2	2
C3	1	2
C5	1	2
A	1	2
DX	2	1
OP-1	1	2
OP-2	2	1
OP-3	2	2
OP-4	1	1
OP-5	1	2

Figure 6.3: **BUS ALLOCATION CRITERIA**

This figure illustrates the type of decisions that must be made during the bus allocation phase when placement information is available. In both cases, 'ALU -> R2' is the MIS transfer - the decision must be made to either assign 'ALU -> R4' or 'ALU -> R3' to the MIS bus (a). The former choice is the best as it requires the least amount of interconnect overall, as it results in the most sharing of wire (b).

(a)



(b)

CASE	TOT. AMT. OF CONNECT		AMT. OF SHARED LEAD	
	R4 ON MIS	R3 ON MIS	R4 ON MIS	R3 ON MIS
1	5	7	2	0
2	3	5	2	0

Figure 6.4: **LIST OF TV'S WITH PLACEMENT INFORMATION**

The placement information, from Figure 6.2, has been included in this list of TV's. Each TV has a source, a destination, timesteps, and a certain area through which it must pass defined by the row and column.

TRANSFER VARIABLES					
IND.	SRC.	DEST.	TIME STEPS	ROW	COLUMN
1	R1	OP1.IN2	3	1	1, 2
2	R1	OP2.IN1	1	1, 2	1
3	R1	OP4.IN1	3,4	1	1
4	R2	OP1.IN1	2	1, 2	2
5	R2	OP3.IN2	4	2	2
6	R3	OP1.IN1	1	1, 2	2
7	R3	OP3.IN2	1	2	2
8	R4	OP2.IN2	2	1, 2	1
9	R4	OP4.IN2	3,4	1	1
10	R5	OP2.IN1	2,3	2	1
11	R5	OP3.IN1	4	2	1, 2
12	R6	OP5.IN2	2	1, 2	2
13	R6	R2	1	2	2
14	R6	R3	2	2	2
15	OP1	R5	1,2,3	1, 2	1, 2
16	OP2	R4	1,2,3	1, 2	1
17	OP3	R6	1,4	2	2
18	OP4	R1	3,4	1	1
19	DX	OP1.IN1	3	1, 2	1, 2
20	DX	OP2.IN2	1,3	2	1
21	DX	OP3.IN1	1	2	1, 2
22	C3	OP1.IN2	1	1	2
23	C5	OP1.IN2	2	1	2
24	A	OP5.IN1	2	1	2

The primary weight is calculated in the same manner as in **BAL** (Figure 4.1.10). It reflects the profit of allocating a particular transfer to an MIS bus, based on the sources and destinations of other TV's that have already been allocated to that bus. In **LAYBAL**, if the primary weight is greater than zero, then the shared_wire weight will be determined by:

$$\text{Shared\_Wire\_Weight} = (\text{Amount of Wire Shared})^2 \qquad (6-2)$$

The amount of wire that would be shared, between the TV and the MIS-bus, is squared to provide a greater emphasis on sharing long lengths of wire.

The **LAYBAL** algorithm is:

Step 1: Determine the TV's that are combinable (i.e.- have a sink or source in common with another), as shown in Figure 6.5).

Step 2: Calculate the number of unique-source combinable transfers for each time step (Figure 6.6), and determine the MIS (maximally incompatible set).

Step 3: Assign the TV's in the MIS to separate buses (Figure 6.7). Each MIS bus will inherit the row and column position of the TV's assigned to it, thereby acquiring a physical location in the layout. This bus can be thought of as a wire that stretches along one half of the rectangle formed between the TV's source and destination.

Step 4: Determine the weights of the remaining combinable TV's with respect to each of the MIS buses (Figure 6.7) using the following guide:

```
IF [Primary-Weight] > 0 THEN
  Weight = Primary-Weight + (Shared-Wire)**2
ELSE
  Weight = Primary-Weight
END-IF
```

Figure 6.5: **STEP 1: DETERMINE THE COMBINABLE TRANSFERS**

A combinable transfer is one in that has either a source in common with another TV's source, or a destination in common with another TV's destination.

IND.	TRANSFER VARIABLES SRC.	DEST.	TIME STEPS	ROW	COLUMN	COM-BIN-ABLE ?
1	R1	OP1.IN2	3	1	1, 2	YES
2	R1	OP2.IN1	1	1, 2	1	YES
3	R1	OP4.IN1	3,4	1	1	YES
4	R2	OP1.IN1	2	1, 2	2	YES
5	R2	OP3.IN2	4	2	2	YES
6	R3	OP1.IN1	1	1, 2	2	YES
7	R3	OP3.IN2	1	2	2	YES
8	R4	OP2.IN2	2	1, 2	1	YES
9	R4	OP4.IN2	3,4	1	1	YES
10	R5	OP2.IN1	2,3	2	1	YES
11	R5	OP3.IN1	4	2	1, 2	YES
12	R6	OP5.IN2	2	1, 2	2	YES
13	R6	R2	1	2	2	YES
14	R6	R3	2	2	2	YES
15	OP1	R5	1,2,3	1, 2	1, 2	NO
16	OP2	R4	1,2,3	1, 2	1	NO
17	OP3	R6	1,4	2	2	NO
18	OP4	R1	3,4	1	1	NO
19	DX	OP1.IN1	3	1, 2	1, 2	YES
20	DX	OP2.IN2	1,3	2	1	YES
21	DX	OP3.IN1	1	2	1, 2	YES
22	C3	OP1.IN2	1	1	2	YES
23	C5	OP1.IN2	2	1	2	YES
24	A	OP5.IN1	2	1	2	NO

Figure 6.6: **STEP 2: CALCULATE THE NUMBER OF UNIQUE SOURCE TV'S PER TIME STEP**

The time step with the most unique source transfers is the "MIS" (Maximally Incompatible Set) time step. If there is more than one time step with the same maximum number of unique source transfers, then one is selected randomly.

STEP NO.	INDEX NO.	SOURCE OF TRANSFER	NO. OF UNIQUE-SOURCE TRANSFERS
1	2	R1	5
	6	R3	
	7	R3	
	13	R6	
	20	DX	
	21	DX	
	22	C3	
2	4	R2	5
	8	R4	
	10	R5	
	12	R6	
	14	R6	
	23	C5	
3	1	R1	4
	3	R1	
	9	R4	
	10	R5	
	19	DX	
	20	DX	
4	3	R1	4
	5	R2	
	9	R4	
	11	R5	

Figure 6.7: **STEP 3: ASSIGN THE MIS TV'S TO SEPARATE BUS**

During this step, each of the MIS TV's get assigned to separate buses - note that each MIS bus inherits the row and column position of the transfer, thereby acquiring a physical location in the layout.

STEP 4: DETERMINE THE WEIGHTS OF THE REMAINING TV'S WITH RESPECT TO EACH MIS BUS.

If the primary weight > 0 Then
    Weight = Primary Weight + (Amount of wire shared)**2
Else
    Weight = Primary Weight = 0

For instance, TV-6 has a primary weight of 3 w.r.t. Bus-1, since TV-6 & TV-4 have the same sink. TV-6 & Bus-1 have rows 1, 2 & column 2 in common, therfore the shared wire amount is 3, & the Wire Weight is 9. Therfore the total weight of TV-6 w.r.t. Bus-1 is 12.

TRANSFER					BUS1				BUS2				BUS3				BUS4				BUS5			
I	S	D	R	C	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
4	R2	OP1.IN1	1, 2	2																				
8	R4	OP2.IN2	1, 2	1																				
10	R5	OP2.IN1	2	-1																				
12	R6	OP5.IN2	1, 2	2																				
22	C5	OP1.IN2	1	2																				
1	R1	OP1.IN2	1	1, 2			0				0								0				7	
2	R1	OP2.IN1	1, 2	1	0				0				7				0							
3	R1	OP4.IN1	1	1			0				0								0				0	
5	R2	OP3.IN2	2	2				5				0				0				0				0
6	R3	OP1.IN1	1, 2	2	12				0				0				0							
7	R3	OP3.IN2	2	2	0				0				0				0							
9	R4	OP4.IN2	1	1			0				5								0				0	
11	R5	OP3.IN1	2	1, 2				0				0				5				0				0
13	R6	R2	2	2	0				0				0				5							
14	R6	R3	2	2														6						
19	DX	OP1.IN1	1, 2	1, 2			12				0								0				0	
20	DX	OP2.IN2	2	1	0		0		7		7						0		0					
21	DX	OP3.IN1	2	1, 2	0				0				0				0							
23	C3	OP1.IN2	1	2	0				0				0				0				7			

Step 5: The TV with the highest weight is assigned to the corresponding MIS bus. If there is more than one TV with the same highest weight, then one is chosen randomly (Figure 6.8). Once the transfer is assigned , then the other TV's that could have been assigned to that MIS bus, are updated to reflect this new assignment.

Step 6: All the remaining TV's with primary weights greater than zero are assigned in the same manner (Figure 6.9)

The interconnect generated for this example is shown in Figure 6.10.

6.4 Complexity of LAYBAL

The calculation of the weights in **LAYBAL** (step 4) is the only step that differs from **BAL**. However, this step has the same complexity as that of **BAL**. Thus the complexity of **LAYBAL**, is also of order N^2. This does not include the complexity of partitioning and placing the components.

Figure 6.8: **STEP 5: ASSIGN THE HIGHEST WEIGHTED TV & UPDATE THE REST OF THE TV'S.**

This figure shows that TV-19 was chosen to be assigned to BUS-1. The weights of the remaining TV's that could have been assigned to BUS-1 are updated to reflect this new addition to the bus. Also, note that the bus has inherited the placement locations of any TV's on that bus.

TRANSFER					BUS1 R: 1,2 C: 1,2				BUS2 R: 1,2 C: 1				BUS3 R: 2, C: 1				BUS4 R: 1,2 C: 1				BUS5 R: 1 C: 2			
I	S	D	R	C	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
4	R2	OP1.IN1	1, 2	2																				
8	R4	OP2.IN2	1, 2	1																				
10.	R5	OP2.IN1	2																					
12	R6	OP5.IN2	1, 2	2																				
22	C5	OP1.IN2	1	2																				
19	DX	OP1.IN1	1, 2	1, 2																				
1	R1	OP1.IN2	1	1, 2						0								0				7		
2	R1	OP2.IN1	1, 2	1	0				0				7				0							
3	R1	OP4.IN1	1	1						0								0				0		
5	R2	OP3:IN2	2	2			5				0				0				0				0	
6	R3	OP1.IN1	1, 2	2	12				0				0				0							
7	R3	OP3.IN2	2	2	0				0				0				0							
9	R4	OP4.IN2	1	1						5								0				0		
11	R5	OP3.IN1	2	1, 2			0				0				5				0				0	
13	R6	R2	2	2	0				0				0				5							
14	R6	R3	2	2														6						
20	DX	OP2.IN2	2	1	5		5		7		7						0		0					
21	DX	OP3.IN1	2	1, 2	10				0				0				0							
23	C3	OP1.IN2	1	2	0				0				0				0				7			

Figure 6.9: RESULT OF LAYBAL

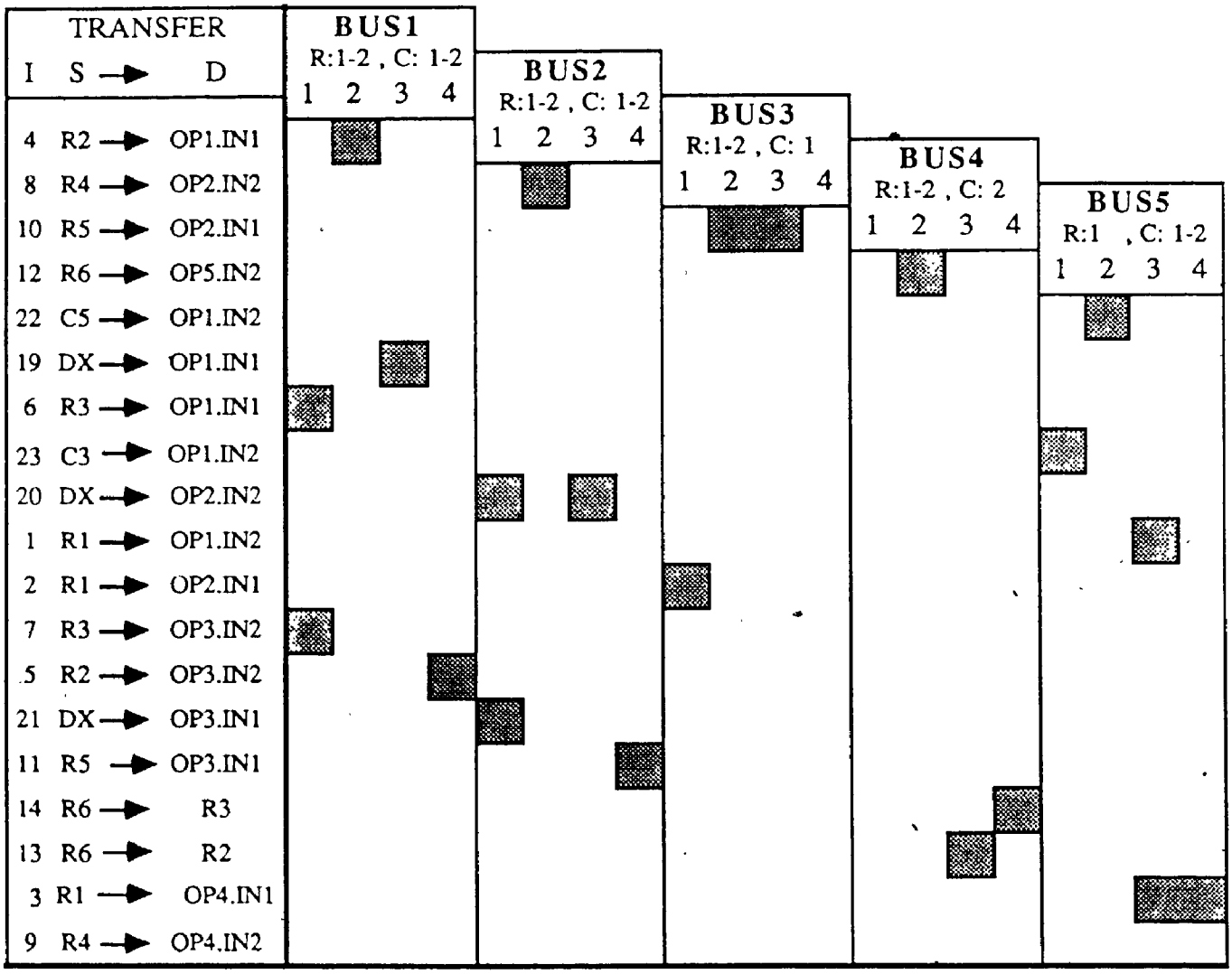
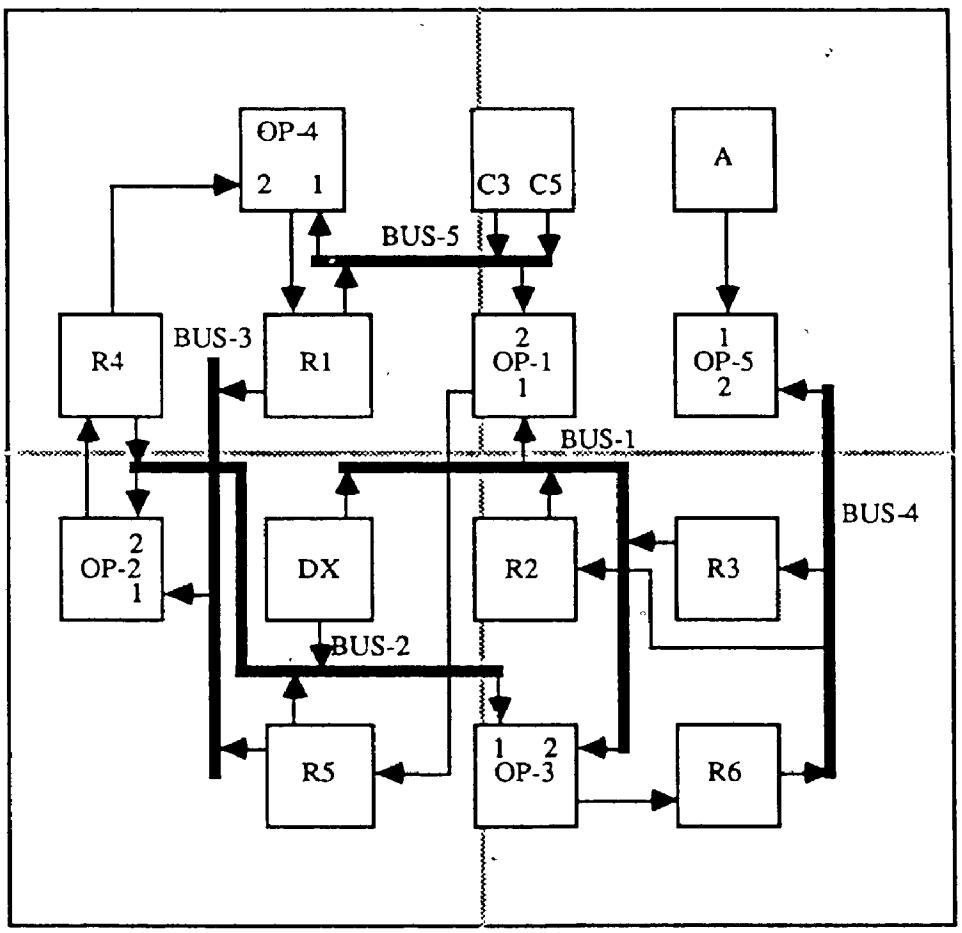


Figure 6.10: **LAYBAL GENERATED INTERCONNECT FOR THE EXAMPLE**



Chapter 7
Future Work, Summary and Conclusions

This chapter presents some possible extensions, or improvements to the three algorithms, **BAL**, **RAL**, and **LAYBAL**, that were developed as part of this thesis. As well, a summary and conclusion of this thesis are provided.

7.1 Additional Features for BAL

This section presents several features that would improve the bus allocation algorithm, **BAL**. In particular, **BAL** could be improved by:

- being able to cope with variable width buses (7.1.1),
- being able to change the time constraints of TV's (7.1.2),
- using a better method of *look-ahead* than that of secondary weights (7.1.3),
- taking into consideration conditional branches and pipelined data paths (7.1.4 and 7.1.5).

7.1.1 Variable Width Buses

BAL assumes that all transfers of data have the same number of bits. This assumption is not always valid, and may lead to circuits that waste area. For instance, an 8-bit wide bus may be used during one time step to transfer only a 4-bit wide data. If data widths are not considered, then this 4-bit wide data transfer could block another 4-bit transfer from using the remaining four free lines during that time step.

BAL could be modified to consider the data transfer widths, by expanding the set of primary weights. Some work would be required to determine the best ratio between data widths and the sharing of sources and destinations.

7.1.2 Time-Flexible Data Transfers

Time-Flexible data transfers are those that may occur in a range of time steps. For instance, a register may be ready to transfer its contents in time step 3, but may have until time step 6 to do so. Thus, this data transfer could occur anytime from time steps 3 to 6. If this information were known, and used, by a bus allocator then it could assign such transfers in a more efficient manner. In other words, it would assign the transfer variable to a bus during one of the time steps in its range, with the least amount of area cost. **BAL** would certainly be improved if it were able to consider these *time-flexible* data transfers.

7.1.3 Global Look-Ahead Feature

The secondary weights used in **BAL** were designed to provide a *look-ahead* facility. They were to help select the most appropriate TV (transfer variable) from among TV's with equal and non-zero primary weights. These secondary weights, described in 4.1.4, attempted to indicate the future assignments that might be blocked if a particular TV were assigned to a particular bus. In other words, it would indicate how other TV's could still be assigned to that bus if the TV in question were assigned. However, this secondary weight scheme did not produce significantly better results than if it were not used (Section 4.3.1).

A more global *look-ahead* approach would be to:

- determine what TV's are being displaced from a bus, when another TV is assigned to it,
- determine if these displaced TV's can efficiently be assigned to other buses,
- to assign the TV that displaces the fewest of those TV's that cannot be assigned efficiently to other buses.(This is similar to the technique of

using the *number of edges deleted* in Tseng and Siewiorik's method of interconnect allocation, described in Section 3.2.1.)

This would provide a less greedy solution. However, it would add significantly to the complexity of the allocation algorithm.

7.1.4 Conditional Branches and Transfers

An example of a conditional branch is given in Figure 7.1. As a result of a conditional evaluation in time step 1, only one of the add or subtract operations will be performed during time step 2. Therefore, either the transfers to and from the add operator, TV's 1-3, or those to and from the subtract operator, TV's 4-6, will actually be active during time step 2. An efficient bus allocator should be able to consider this type of situation when assigning TV's to buses.

BAL can assign such conditional transfers efficiently to buses by considering them to be disjoint - i.e.) not occurring in the same time step. However, such transfers must be so indicated, by assigning them unique time step names. For instance, the time steps of TV's 1-3 should be labelled as *2a*, while those of TV's 4-6 should be labelled as *2b*. Therefore **BAL** need not be modified, just its input.

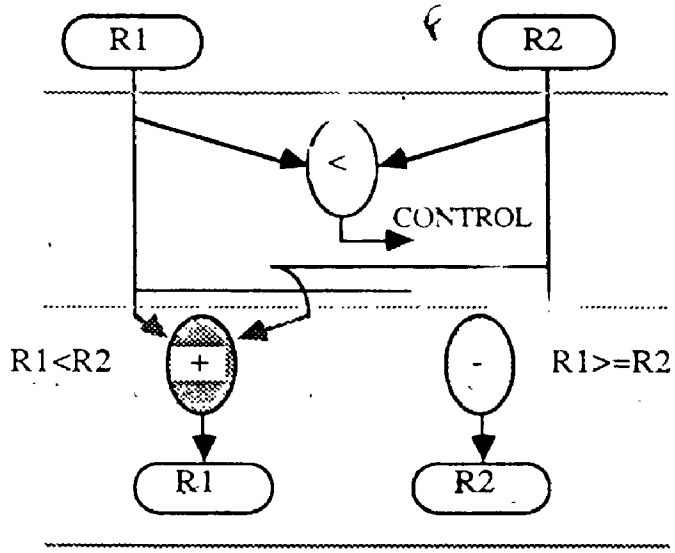
7.1.5 Pipelined Data Paths

Occasionally data paths may be pipelined. A pipelined data path is one in which new data is accepted for processing before the results of computations of previous data is available. The time steps will be overlapped. For instance, the data path shown in Figure 7.2 has a *latency* of three. This means that three time steps must occur before new data can be accepted. Thus time steps 1,4 and 7 are executing concurrently, but on three different sets of data. This type of situation creates unique problems for a bus allocator. Although the operations in different time steps appear to be disjoint, they may actually be executing simultaneously. This implies that the corresponding TV's may also appear to be disjoint when they are not. In order that **BAL** be able to allocate TV's from pipelined data paths,

Figure 7.1: **CONDITIONAL BRANCHES AND TV'S**

A high level description (pseudocode), and a data flow graph (DFG), for a conditional branch is shown in (a). Only one of the operations, add or subtract, will be performed in time step 2.
The transfer variables (TV's) for the DFG are shown in (b). Although they are all in time step 2, only 3 of them will be active at a time, either TV's 1-3 or TV's 4-6. Therefore, these two sets of TV's have been labelled to reflect the fact that they are disjoint in time.

(a) **HIGH LEVEL DESCRIPTION**

DFG	PSEUDOCODE FOR TIME STEP 2
	IF { R1 < R2 } THEN R1 = R1 + R2 ELSE { R1>= R2 } R2 = R1 - R2 END-IF

(b) **TRANSFER VARIABLES (TV'S) FOR TIME STEP 2**

INDEX	SOURCE	SINK	TIME STEP	NEW TIME STEP NAME
1	R1	ADD1.1	2	2a
2	R2	ADD1.2	2	2a
3	ADD	R1	2	2a
4	R1	SUB1.1	2	2b
5	R2	SUB1.2	2	2b
6	SUB	R2	2	2b

those TV's, that are not really disjoint, must be so marked. Again, **BAL** need not be modified, only its data input.

7.2 Future Work for LAYBAL

Some optimization work remains with **LAYBAL**. In particular, three possible areas for improvement are:

- weight scheme (7.2.1),
- starting point (7.2.2),
- cost evaluation scheme (7.2.3).

7.2.1 Weight Scheme

Recall from Chapter 6 that the criterion for assigning a particular transfer variable (TV) to a particular bus was the *weight* of that TV with respect to that bus. This weight was determined by the *wire weight* (square of the shared wire) and the *primary weight* (weight of shared sources and destinations). However, the following questions remain to be answered:

What is the best ratio between the shared wire weight and the primary weight?

Perhaps the shared wire weight should only be used to break ties between TV's with equal primary weights?

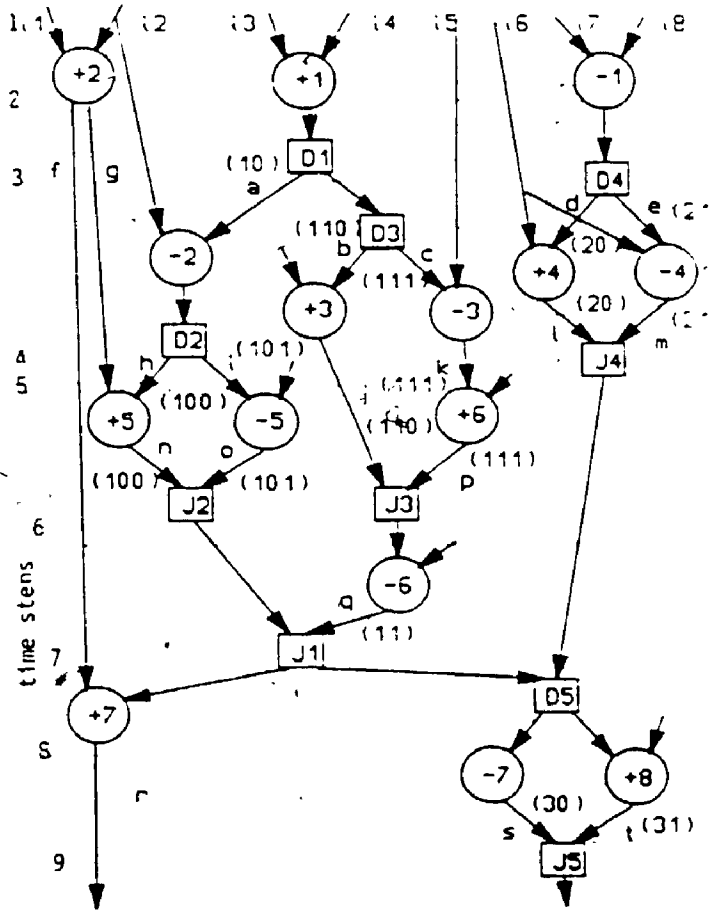
When should incompatible TV's (TV's that share neither source nor destination with any other TV's) be included in the assignment process?

7.2.2 Starting Point

In both **BAL** and **LAYBAL** the *starting point* (step during which the weight scheme can be used to assign TV's) was obtained by determining the MIS TV's (the set of TV's in that time step with the most unique sources). These MIS TV's were then assigned to separate buses (MIS-BUSES), and the remaining TV's were

Figure 7.2: **A DATA FLOW GRAPH WITH A LATENCY OF THREE [KuPa87]**

This DFG has a latency of three. Therfore time steps 1, 4, and 7 are actually executing concurrently, but on three different sets of data.



then assigned to these buses according to the weight scheme. However, perhaps **LAYBAL** should include the wire lengths when determining the MIS TV's in order to get the best *starting point*. This should be explored further.

7.2.3 Cost Scheme for LAYBAL

Before more effort can be put into improving **LAYBAL**, some method of evaluating its generated interconnect will be required. Because the placement of the various components is known, a cost evaluation scheme should include:

- amount of physical area consumed by all leads and pass transistors (drivers),
- amount of power consumed by all drivers,
- time required to transfer data (parasitic effects).

7.3 BAL, RAL, LAYBAL as Usable Programs

BAL, **RAL**, and **LAYBAL** may be used as part of a synthesis package, or by designers wishing to optimize their circuits. If the latter use is desired, then a graphics, or other type, interface should be developed to accept register transfer level descriptions of the user's circuit. As well, regardless of the use, **RAL** should be written, and **BAL** and **LAYBAL** re-written, in an efficient compiled language such as C.

7.4 Summary of Thesis

Two types of *silicon compilers* (programs that automatically generate the circuits) were described:

- structural,
- behavioral.

Structural silicon compilers transform a structural definition of a circuit (components and connections between those components) to a physical, or geometric, description (placement of those components).

Behavioral silicon compilers transform a functional, or behavioral, description of a circuit (desired functions) to a physical description.

The *synthesis process*, as part of behavioral compilation, was described as that process which transforms a functional description of a circuit to a structural one. This thesis addressed the problems of register and interconnect allocation during data path synthesis of register transfer level (RTL) circuits.

RTL circuits were defined in terms of:

- components, or *functional units* (FU's) such as multipliers, adders, comparators,
- registers,
- interconnect between these components such as buses, muxes, leads,
- control.

Data path synthesis was defined as the automatic generation of that part of the circuit which performed operations on the data (i.e.- not control). One method of performing data path synthesis was described, in which each of the following steps was sequential (completed before the next begins) :

1) Specification of the circuit to be generated (really pre-data path synthesis),

2) Data Flow Graph (DFG) synthesis,

3) Scheduling of the DFG (Allocation of operations to time steps),

4) Allocation of functional units (FU's),

5) Allocation of registers,

6) Allocation of interconnect.

This thesis examined in detail the requirements of a register allocator and an interconnect allocator. Original techniques for register allocation (**RAL**), and interconnect allocation (**BAL** and **LAYBAL**) were developed.

7.4.1 Summary of Work Done for Register Allocation

The register allocation problem was defined as an assignment one. *Data variables* (DV's) were used to represent the data that had to be assigned (stored) to registers. These DV's were defined to:

- have a single source (an FU),
- have one or more destinations (FU's),
- exist in one or more continuous time steps.

The problem was to assign the DV's to be stored in specific registers such that the minimum number of registers, and the least amount of interconnect, was required. This was an NP complete problem.

An original technique, **RAL**, was developed, but not implemented, to perform this task. It can be summarized as follows:

- it assumed the input was a scheduled DFG with FU's allocated,
- it had a complexity of N^2, where N was the number of DV's to be assigned to registers,
- it compared favourably with two other techniques of performing this task,
- it was not programmed, but it could easily be done by modifying an

existing program (**BAL**).

7.4.2 Summary of Work Done for Interconnect Allocation

The interconnect allocation problem was also defined as a partitioning and assignment one. *Transfer Variables* (TV's) were used to represent the transfers of data between FU's and registers. These TV's were defined to:

- have a single source (FU or register),
- have a single destination (FU or register),
- exist in one, or more, time steps(continuously or discontinuously),
- and in the case of **LAYBAL**, occupy a specific area in terms of rows and columns.

As well, three interconnection primitives were defined:

- straight leads,
- wired-broadcast tree,
- bus or mux (buses and muxes were shown to be the same at the register transfer level).

The interconnect allocation problem was to partition the TV's into groups, with each TV in a group being assigned to the same interconnection primitive, such that the overall area cost was minimized. This was an NP complete problem. Two novel techniques were developed to perform this task, **BAL** and **LAYBAL**.

Summary of BAL

BAL can be summarized as follows:

- It assumes the input is a scheduled DFG with FU's and registers allocated,
- It assigns TV's to groups based on a criteria, developed for this thesis, of sharing sources and destinations - *primary weights*. The TV with the highest weight is assigned first.

- another technique was also developed to break ties when there was more than one TV with equal highest *primary weight*. This technique is called *secondary weights*,
- However, the use of these *secondary weights* added another order of complexity to **BAL**, and did not significantly improve results. Hence, it was discarded.
- The complexity of **BAL** (without *secondary weight* calculations) is of order N^2, where N is the number of TV's.
- **BAL** was programmed in VAX LISP.
- **BAL** was found to work very well, often producing the best results, or results close to it.
- **BAL** compared favourably with three other techniques of performing the same task.

Summary of LAYBAL

LAYBAL can be summarized as follows:

- It assumed that its input was a scheduled DFG with FU's and registers allocated, and that these components had been crudely partitioned and placed. This assumption required a slight modification to the data path synthesis process:
  1) Specification of the circuit to be generated,
  2) Data Flow Graph (DFG) synthesis,
  3) Scheduling of DFG,
  4) Allocation of functional units (FU's),
  5) Allocation of registers,
  6) Crude placement of FU's and registers,
  7) Allocation of interconnect.
- TV's were modified to capture the placement of the components. This

placement was represented by *rows* and *columns* defined by the TV's source and destination locations.

- TV's were assigned to groups based on two criteria:
  - the *primary weight* (weight of shared sources and destinations),
  - the *wire weight* (square of the potential shared wire).
- **LAYBAL** was programmed in VAX LISP.
- It had a complexity of N^2 where N was the number of TV's.
- More work is required to determine the best ratio between the two assignment criteria (*primary weight* and *wire weight*).
- There are no other known systems that perform this type of task (i.e.- interconnect allocation after crude placement of components), hence no comparison is possible.
- **LAYBAL** seemed to work well. However, the interconnect for only a few simple circuits was generated.

Summary of Other Work for Interconnect Allocation

An original technique for evaluating the cost of structural domain (i.e.- pre-layout) interconnect was developed. It assumed 1-bit wide transfers (*Cost-Scheme-1*) and b-bit wide transfers (*Cost-Scheme-2*). *Cost-Scheme-1* was used to compare interconnect generated by **BAL** with other methods, and to determine the best set of *primary weights*.

Also, the possibility of using the *left-edge* algorithm for interconnect allocation was explored. However, it was found to be unsuitable, because of the discontinuous nature of a TV's lifetime.

7.5 Contributions to Automatic Circuit Design

The following summarizes the useful contributions that this thesis has made in the area of automatic circuit design:

1) A weight-directed approach to interconnect and register assignment of order N^2:
   It was found that the weight-directed approach produced results better than un-weighted approaches (modified clique partitioning and left-edge). As well, this weight-directed approach is less complex (order N^2) than other known techniques.

2) A criterion (weights) for interconnect allocation:
   The set of weights was established by generating interconnect for many different circuits.

3) It is better to merge outputs, over inputs:
   It was determined that any bus allocation scheme should favour merging transfers with common destinations over those with common sources.

4) A comprehensive interconnect costing scheme:
   The costing scheme that was developed provides a gauge on the quality of generated interconnect, without having to do a layout of the circuit.

5) A technique of combining layout information with interconnect allocation was introduced. As far as the author knows, there have been no other techniques to do this.

7.6 Conclusions

BAL and **RAL** are good techniques for allocating interconnect and registers, respectively, in a sequential data path synthesis system. These two algorithms could also be used as a tool for digital circuit designers interested in optimizing their circuits, although, an interface that could accept a register transfer level structural

description of circuits would be necessary. **BAL** and **RAL** were found to work well - they are able to perform optimizations that would normally be overlooked by humans.

LAYBAL is a promising technique for allocating interconnect once a crude placement of components is known. However, if it were to be used in a data path synthesis system, the synthesis process would need to be modified to include an automatic partitioning and placement step. This requirement does not necessarily detract from **LAYBAL** as the components will need to be partitioned and placed, eventually, as part of the overall VLSI design process. In other words, this step need only be performed earlier in an automatic design system.

LAYBAL could also be used by designers wishing to reduce the interconnect in their layout circuits. Again, an appropriate interface would be required.

The automatic design of circuits seems to be a viable technique for the development of digital circuits. However, considerable research remains before *silicon compilers* will be accepted as alternatives to experienced human designers.

Appendix A:References

[BrGa87] F. Brewer and D. Gajski "Knowledge Based Control in Micro-Architecture Design," *Proc. 24th Design Automation Conference*, 1987, pp.203-209.

[Dona79] W.E. Donath "Placement and Average Interconnection Lengths of Computer Logic," IEEE Trans. Circuits Syst., Vol. CAS-26, April 1979, pp.272-277.

[Fuku88] M. Fukuse "Alternatives Abound for IC Designers," Electronics Times, Vol. 3, No. 1 (Jan. 4, 1988), p. 23.

[Gajs85] D. Gajski "Silicon Compilation," VLSI Systems Design, November 1985, pp.48-64.

[HoSa78] E. Horowitz and S. Sahni,*Fundamentals of Computer Algorithms*, Computer Science Press Inc., Rockville, Maryland, 1978.

[Kala86] P. Kalab "Master's Thesis: Automated Microcontroller Synthesis for Application Specific Integrated Circuits," Dept. of Systems and Computer Engineering, Carleton University, December 1986.

[KeLi70] B.W. Kernighan and S. Lin "An Efficient Heuristic Procedure for Partitioning Graphs," Bell Syst. Tech. J., Vol. 49, No. 2 (Feb. 1970), pp. 291-307.

[Klin82] E. Klingman *Microprocessor Systems Design Volume 2*, Prentice-Hall Inc., Englewood Cliffs, N.J. 07632, 1982.

[KuPa87] F. Kurdahi and A. Parker "REAL: A Program for Register Allocation," *Proc. 24th Design Automation Conference*, 1987, pp. 210-215.

[KuPa87

[HaSt71]] A. Hashimoto and J. Stevens "Wire Routing by Optimizing Channel Assignment Within Large Apertures," *Proc. 8th Design Automation Workshop*, 1971, pp.155-169.

[KoTh85] T.J. Kowalski and D.E. Thomas "The VLSI Design Automation Assistant: What's in a Knowledge Base," *Proc. 22nd Design Automation Conference*, 1985, pp. 252-258.

[KoTh84] T.J. Kowalski and D.E. Thomas "The VLSI Design Automation Assistant: An IBM System/370 Design," IEEE Design and Test, February 1984, pp. 60-69.

[KnPa86] D. Knapp and A. Parker "A Design Utility Manager: The ADAM Planning Engine," *Proc. 23rd Design Automation Conference*, 1986, pp.48-54.

[PanGa87] B. Pangrle and D. Gajski "Design Tools for Intelligent Silicon Compilation," IEEE Transactions on Computer-Aided Design, Vol. CAD-6, No. 6 (November 1987), pp.1098-1112.

[Park84] A.C. Parker "Automated Synthesis of Digital Systems," IEEE Design and Test, November 1984, pp. 75-81.

[PaKn87a] P.G. Paulin and J.P. Knight "Force-Directed Scheduling in Automatic Data Path Synthesis," *Proc. 24th Design Automation Conference*, July 1987, pp. 195-202.

[PaKG86] P.G. Paulin, J.P. Knight, E.F. Girczyc "HAL: A Multi- Paradigm Approach to Automatic Data Path Synthesis," *Proc. 23rd Design Automation Conference*, July 1986, pp. 263-270.

[Paul86] P.G. Paulin "Ph.D. Proposal: A Multi-Paradigm Approach to Automatic Data Path Synthesis," Dept. of Electronics, Carleton University, January 13, 1986.

[Paul87] P.G. Paulin, Private Communication, Oct. 30, 1987.

[PaKn87b] P.G. Paulin and J.P. Knight "Scheduling and Allocation for Behavioral Synthesis of Pipelined ASIC's," *Proc. Canadian Conference on VLSI*, Winnipeg, Canada, Oct. 25-27, 1987, pp.229-233.

[ScKe72] D.G. Schweitzer and B.W. Kernighan "A Proper Model for the Partitioning of Electrical Circuits," *Proc. 9th Annual Design Automation Workshop Under Joint Sponsorship of the ACM and IEEE*, 1972, pp. 57-62.

[Subr86] P.A. Subrahamanyan "Synapse: An Expert System for VLSI Design," Computer, Vol.19, No.-7 (July 1986), pp.78-89.

[TsSi81] Chia-Jeng Tseng and Daniel P. Siewiorik "The Modeling and Synthesis of Bu Systems," *Proc. 18th Design Automation Workshop*, 1981, pp.471-478.

[TsSi83] Chia-Jeng Tseng and Daniel P. Siewiorik "Facet: A Procedure for the Automated Synthesis of Digital Systems," *Proc. 20th Design Automation Conference*, 1983, pp. 490-496.

[TsSi86] Chia-Jeng Tseng and Daniel P. Siewiorik "Automatic Synthesis of Data Paths in Digital Systems," IEEE Transactions on Computer-Aided Design, Vol. CAD-5, No. 3 (July 1986), pp.379-395.

[WeEs85] N. Weste and K. Eshragian *Principles of CMOS VLSI Design: A Systems Perspective*, Addison-Wesley Publishing Co., Reading, Massachusetts, 1985.

END

08-08-88

FIN