THE MODELING AND SYNTHESIS OP BUS SYSTEMS

by

Chia-Jeng Tseng & Daniel P. Sieworelc

DRC-18-42-82

April, 1902

The Modeling and Synthesis of Bus Systems

Chia-Jeng Tseng and Daniel P. Siewiorek
Departments of Electrical Engineering and Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

November 28,1980

This research is supported by the NSF under grant #55898 and by the Department of Electrical Engineering, Carnegie-Mellon University.

Table of Contents

List of Figures

The Modeling and Synthesis of Bus Systems

Chia-Jeng Tseng and Daniel P. Siewiorek
Departments of Electrical Engineering and Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

Bus oriented interconnection of registers and data operators is the dominate mode of design for the data paths of digital systems. A study of ten processor implementations, ranging in size from microprocessor to large mainframes, that spanned almost 20 years in the practice of digital design indicated a strong similiarity. From this study a set of bus style primitives and generic bus models were developed. The generic bus models can be simplified to match each of the ten processors composing the study. An algorithm for generating a bus style design is presented and compared to prior work reported in the literature. The algorithm is also used to generate the data paths of the PDP-11/40 resulting in lower cost and shorter delays than the original implementation. Finally, the paper concludes with a discussion of the bus synthesis algorithm's implementation and its role in the CMU functional-to-hardware Design Automation System.

1. Introduction

Modern digital systems are often divided into a data part and a control part. The concept of design styles was proposed in [Thom77]. A design style is an abstraction of a group of module sets. Furthermore, a module set is a collection of digital devices which is complete in the sense that arithmetic, logic, and memory functions needed to implement a general digital system are available. The design of either the data part or the control part can be implemented in various styles. For the data part the possible design styles are bus, distributed, bit-slice microprocessor, etc. The bus style is one of the most widely used in practice.

This research is a part of the CMU design automation (CMUDA) system [Siew76,Park79]. The CMUDA system is partitioned into global optimization, design style selection, abstract module allocation, physical module binding, and control allocation phases. Bus style allocator is intended to be one of the possible realizations in the abstract module allocation phase. This paper investigates the properties of and proposes a synthesis algorithm for bus style systems.

Section 2 describes the primitive components used in bus style design. In Section 3 some basic models for bus style systems are proposed. Section 4 describes a synthesis algorithm for the design of bus style data paths. Two examples are given to illustrate the algorithm. Section 5 compares bus style design and distributed style design. General conclusions and areas for further research are provided in the last section.

2. Primitives for Bus Style Design

The primitive components of bus style systems are storage elements, functional units, and interconnection units-

Storage elements may be special or general purpose registers. Special registers are those which have dedicated functions such as stack pointer, instruction register, program counter, memory address register, memory data register, etc. The general registers are not restricted to a dedicated function. Functional units are data operators which can be further subdivided into unary and binary operators. Unary operators include complementers, shifters, byte swappers, etc. Binary operators include arithmetic and boolean functions such as adders, multipliers, AND, etc. Binary operators and some of the unary operators are typically combined into a centralized arithmetic logic unit (ALU).

A data transfer is associated with a source and a sink. Depending upon the direction of the data transfer, a storage element or functional unit may act as the source or a sink at a particular moment Taking the physical or logical constraints into account, a bus may have more than one sink active simultaneously, that is, more than one sink can be "opened" to receive the input data at the same time. However, only one source is allowed to be active at a time. Hence, the storage elements and functional units are linked together by various kinds of interconnection primitives.

An interconnection primitive may be a general bus, a multiplexer, a demultiplexer, or a set of simple wires. A general bus is an interconnection primitive which has multiple inputs and multiple outputs. Each input or output is connected to the bus through a gating element. When the gating element is deactivated, the connection is in the passive state, i.e., very high impedance exists between the bus and the data source or sink. A multiplexer has multiple inputs and a single output. Only one input can be active at a time. A demultiplexer, on the other hand, has a single input and multiple outputs. Only one output can be active at a time. Both multiplexer and demultiplexer have a suitable number of address leads to select the active input or output. Either multiplexer or demultiplexer can also have a gating element to control the data transfers. A set of simple wires are associated with one source and one sink. In a general sense all of these interconnection primitives can be regarded as buses.

All of the interconnections mentioned above can be either unidirectional or bidirectional. A bidirectional component is symmetrical with respect to the direction of data transfers. In a general bus, bidirectional gating logic can activate a data transfer into or out of a storage or functional unit which is connected with the bus.

For our purpose we are not interested in the detailed physical structure of these devices. What we are interested in is their cost and performance . Rgure 2-1 depicts the graphic representations of the various primitives which will be used to describe the data paths of digital systems in later sections.

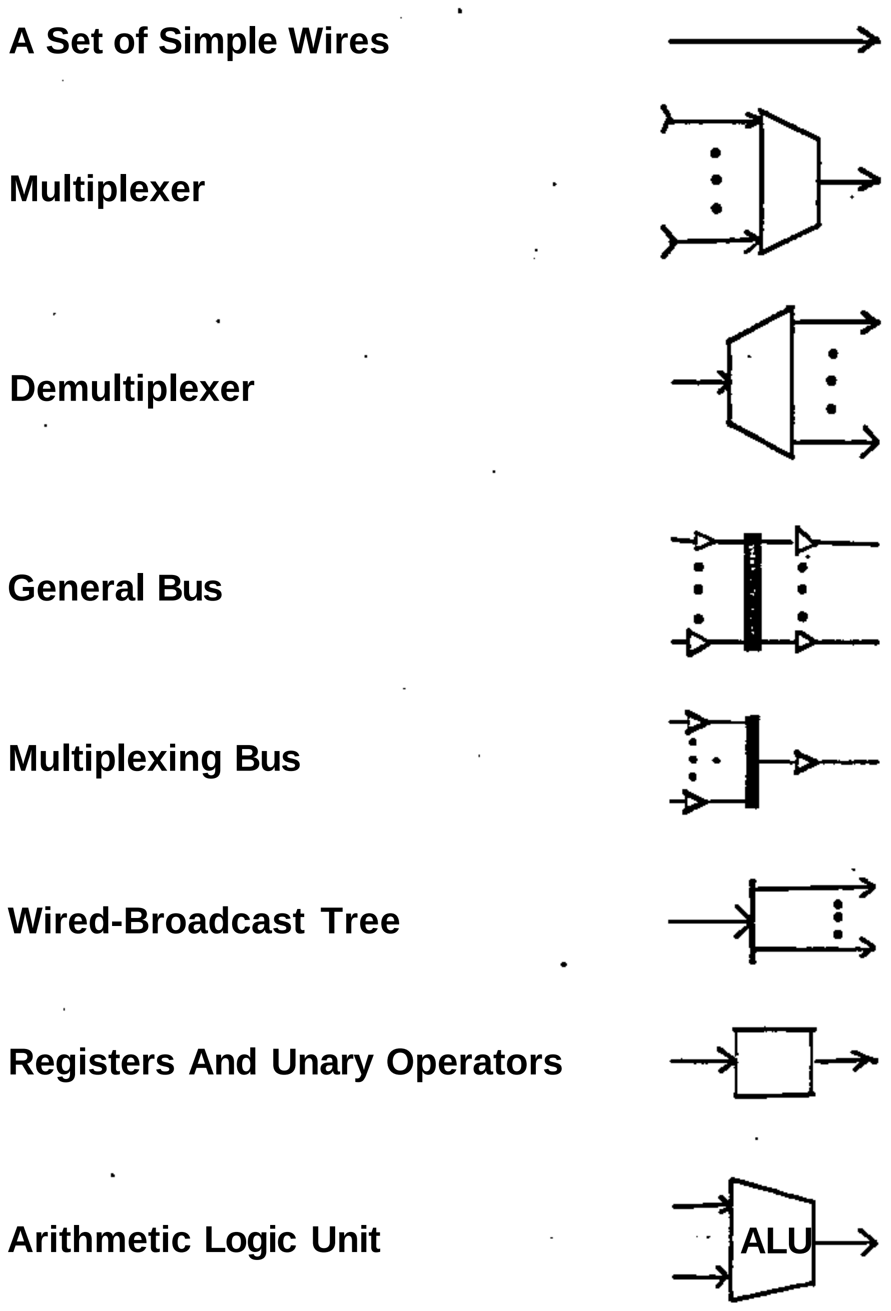


Figure 2-1: Graphic Representations of the Primitives

3. Modeling Bus Style Systems

In order to understand the structure and characteristics of bus style systems in depth, the implementations of a number of existing systems were studied, including the PDP-11, INTEL 8080, PIC 1650, ALTO, AM 2901, AM 2903, TMS 1000, HP 2116, IBM 360/30, and IBM 360/91. These systems were found to have a very strong commonality. Among them, the IBM 360/91 is a system consisting of multiple functional units [Ande67,Toma67]. Each of the other systems has a single centralized ALL). The inputs of the ALU are usually fed from registers which may be accumulators or temporary storage. A register can be allocated at the output of the ALU to store the result of various operations. Unary operators such as byte swappers or complementers can be inserted in the data paths in the vicinity of the ALU to speed up some data operations. Some independent, special purpose storage such as program counter, instruction register, memory address register, memory data register, stack pointer and index register may also be used to speed up operations. For the IBM 360/91, the data paths associated with each functional unit still has a common structure as the other single centralized ALU systems.

The capability for parallel data transfers in a system is directly related to the number of buses. Thus the number of buses can be used to evaluate the degree of parallelism in a bus style system. One design synthesis heuristic would be to select one of the following general bus models based on the concurrency of the algorithm. The design would be completed by simplifying the general model until only essential primitives remained. Rather than use this approach, Section 4 will present a synthesis algorithm.

Figure 3-1 depicts different configurations for the data paths in the vicinity of an ALU. In Figure 3-1(a) and Figure 3-1 (b) the inputs of the ALU share a common bus, while in Figure 3-1 (c), Figure 3-1 (d) and Figure 3-1 (e) each input is connected to a dedicated bus. Assume the operands of a binary operation are available on the buses, but not already in the registers in front of the ALU. Let the time delay be the same for each data transfer. In Figure 3-1 (a) two units of time are required for transferring the operands into the registers and one unit of time is needed to feed the operands from the registers to the inputs of the ALU. Thus three units of time are necessary to set up the input data. In Figure 3-1 (b) one of the inputs is directly connected to the common data bus, so only two units of time are required to set up the input data. In Figure 3-1 (c) and Figure 3-1 (d) the inputs of the ALU are fed through two different buses. The transferring of operands from the buses to the registers can occur simultaneously. Thus two units of time are enough to set up the input data. In Figure 3-1 (e) both inputs are directly connected to the buses, the input data can be set up in one time unit. The input data set-up time (IDSUT) is a measure for evaluating the parallelism associated with the ALU.

Some basic models for bus style systems are proposed in this Section. These basic models focus on the single centralized ALU systems. For convienience of comparison, they are classified by the

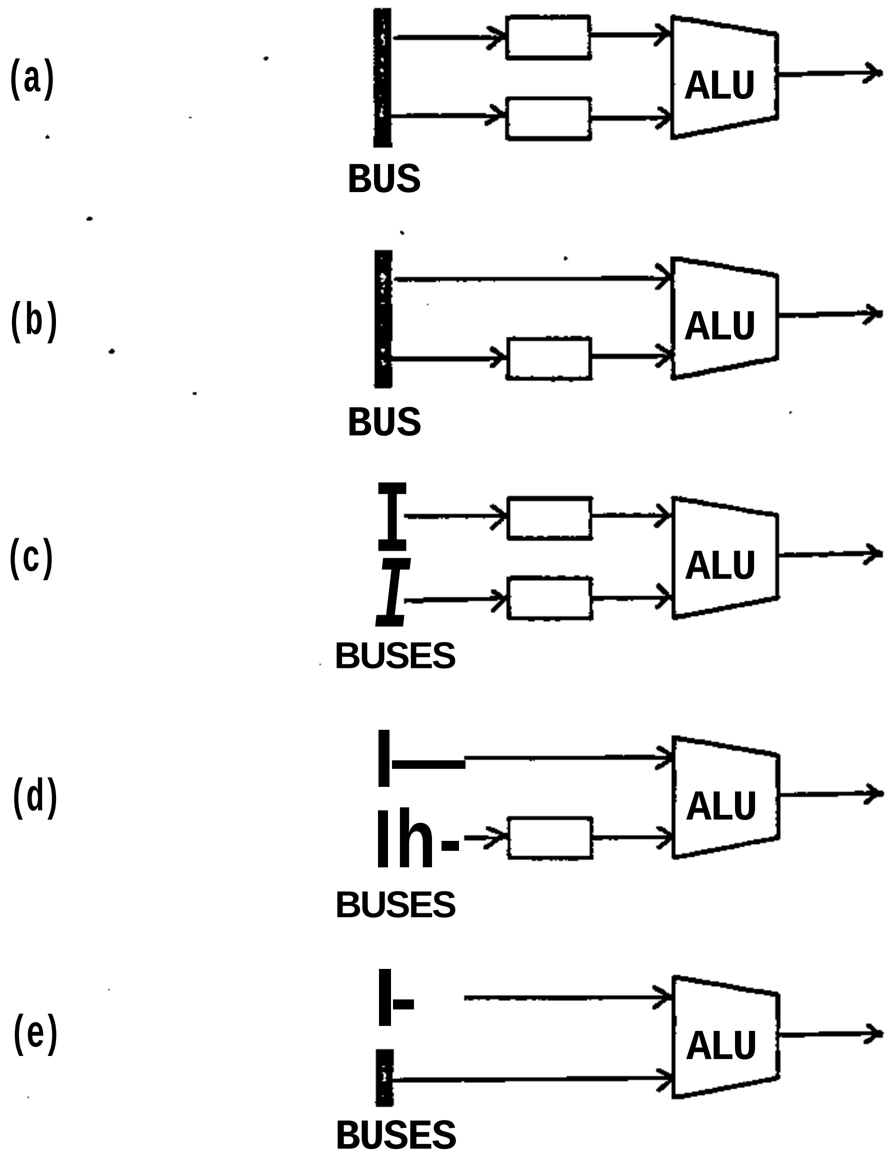


Figure 3-1: Variations of Data Paths in the Vicinity **of** an **ALU**

number of general buses in the systems.

3.1. A Canonical Model of One-Bus Systems

In the simplest system the storage elements and data operators communicate through a common data bus. Figure 3-2 is a canonical model for one-bus systems. Aside from the data transfers which originate from the same source, a one-bus system provides no parallelism for inter-register data transfers through the common data bus. The N square blocks attached to the common data bus are general and special registers. The multiplexer in front of each square block provides the capability of selecting one input from multiple sources. The wired-broadcast network at the output of a square block allows the register to be fed to multiple destinations[1]. A floating arrow front on a wired-broadcast tree may be connected to any floating arrow rear on a multiplexer in front of the other square components. The floating arrow front and the floating arrow rear represent respectively an output and an input of these two square blocks. These floating connections provide variations in routing the data transfers such as bypassing some components or feedback to previous stages.

The total numbers of multiplexers and wired-broadcast trees in the model are $N+5$ and $N+4$ respectively. In general, connecting the output of a square block to its own input is a null operation. Thus the possibility of directly connecting an output of a square block to its own input can be excluded. An output lead on a wired-broadcast tree which is located at the output of a square block can be connected to one of the input leads of the remaining $N+4$ multiplexers. The number of different interconnections which are available is $(N+4)^{N+3}$. On the other hand, a storage element is required to store the ALU output so that it can perform various operations appropriately. Directly connecting the ALU output to its inputs is not allowed. The ALU output can only be connected to the inputs of the other $N+3$ multiplexers in front of the square blocks. Thus the total number of different designs is $(N+3) \bullet (N+4)^{N+3}$.

The one-bus model has been widely adopted in the design of micro- and mini-computers. Some examples are the INTEL 8080, PIC 1650, PDP 11/10, etc. Figure 3-3 depicts the PDP-11/10 data paths [Snow78] as an instantiation of the canonical one-bus model.

3.2. Basic Models of Two-Bus Systems

Additional buses may be included to increase the degree of parallelism. The buses can be assigned to the inputs and/or output of the ALU. Depending on the allocation, the structure may be quite different. Figure 3-4 and Figure 3-5 are two basic models for two-bus systems. In Figure 3-4 the working storage associated with each input of the ALU is assigned a separate bus. Input data transfers can proceed in parallel through these two general buses. In Figure 3-5 the working storage

[1]The input of a wired-broadcast tree can be fed to multiple destinations simultaneously. It is considered to be a more general interconnection mechanism than a demultiplexer in which only one output can be in the active state at a time. Thus, instead of using a demultiplexer, a wired-broadcast tree is used in the model.

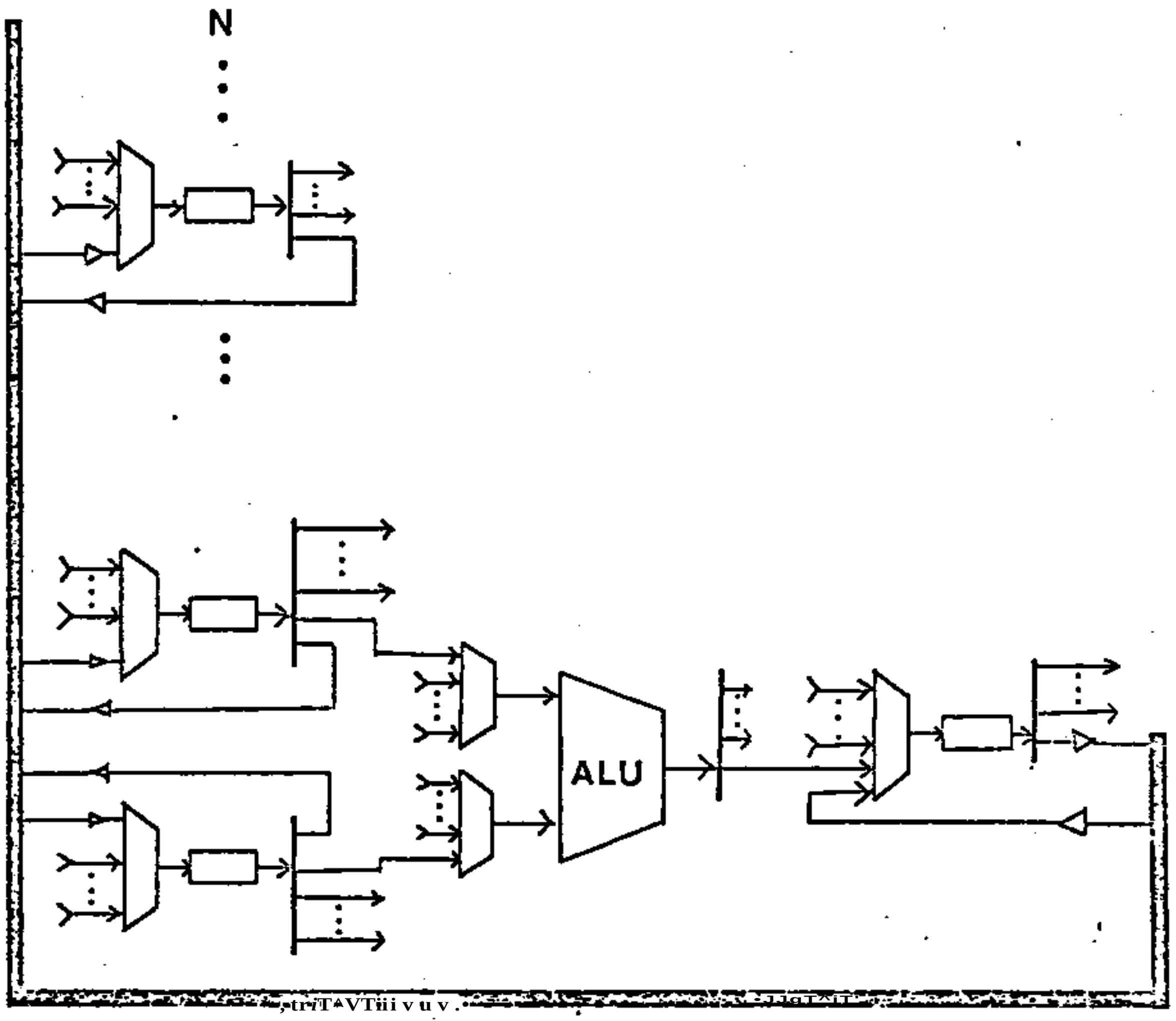
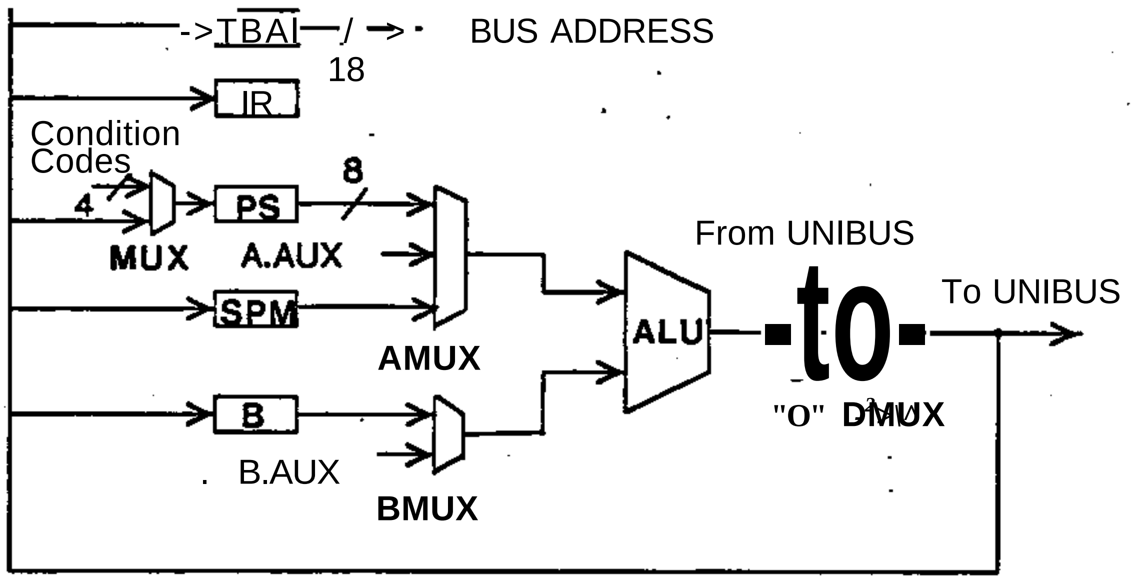


Figure 3-2: **A** Canonical Model of One-Bus System

elements associated with both inputs are connected to the same bus. The other bus is assigned to the output of ALU. Unless the input data come from the same source, only one data transfer is allowed through the common data bus. However, a different kind of intsr-register data transfer can proceed simultaneously through the other bus (OBUS).

The number of wired-broadcast trees is N + 3 in Model II. Each of them can be connected to an input of the other N + 4 multiplexers. The total number of different designs is $(N + 3) \bullet (N + 4)^{N*3}$. **Following the same derivation, the total number of different designs for Model I is $(N1 + N2 + 3) \bullet (N1 + N2 + 4)^{N W N2 + 3}$.** If the number of storage elements and/or unary operators (N1 + N2) in the model is replaced by the integer N, then the result is again $(N + 3) \bullet (N + 4)^{N*3}$.



Note: All data paths are 16 bits wide unless otherwise indicated

Figure 3-3: Data Paths ofThePDP-11/10

3.3. A Canonical Model of Three-Bus Systems

The parallel transfers of input data to an **ALU** is made possible by assigning a dedicated bus to each input of the ALU. If a third bus is assigned to the output, three data transfers can proceed simultaneously. Figure 3-6 is a canonical model for three-bus systems.

As shown in Figure 3-6, there are three sets of storage elements and/or discrete unary operators. Assume the number of components are N1, N2 and N3. Then the number of different designs are $(N1 + N2 + 3) \bullet (N1 + N2 + 4)^{N1+N2*3}$.

3.4. A Canonical Model of Four-Bus Systems

In some special purpose applications such as a hardwired Fast Fourier Transform, extremely intensive inter-register data transfers are required. Additional buses might be essential to improve the performance. Figure 3-7 is a canonical model for four-bus systems. A common data bus which surrounds the three-bus model is included in the model. Again, the number of different designs can be derived as $(N1 + N2 + N3 + 3) * (N1 + N2 + N3 + 4)^{N1+N2+N3+3}$.

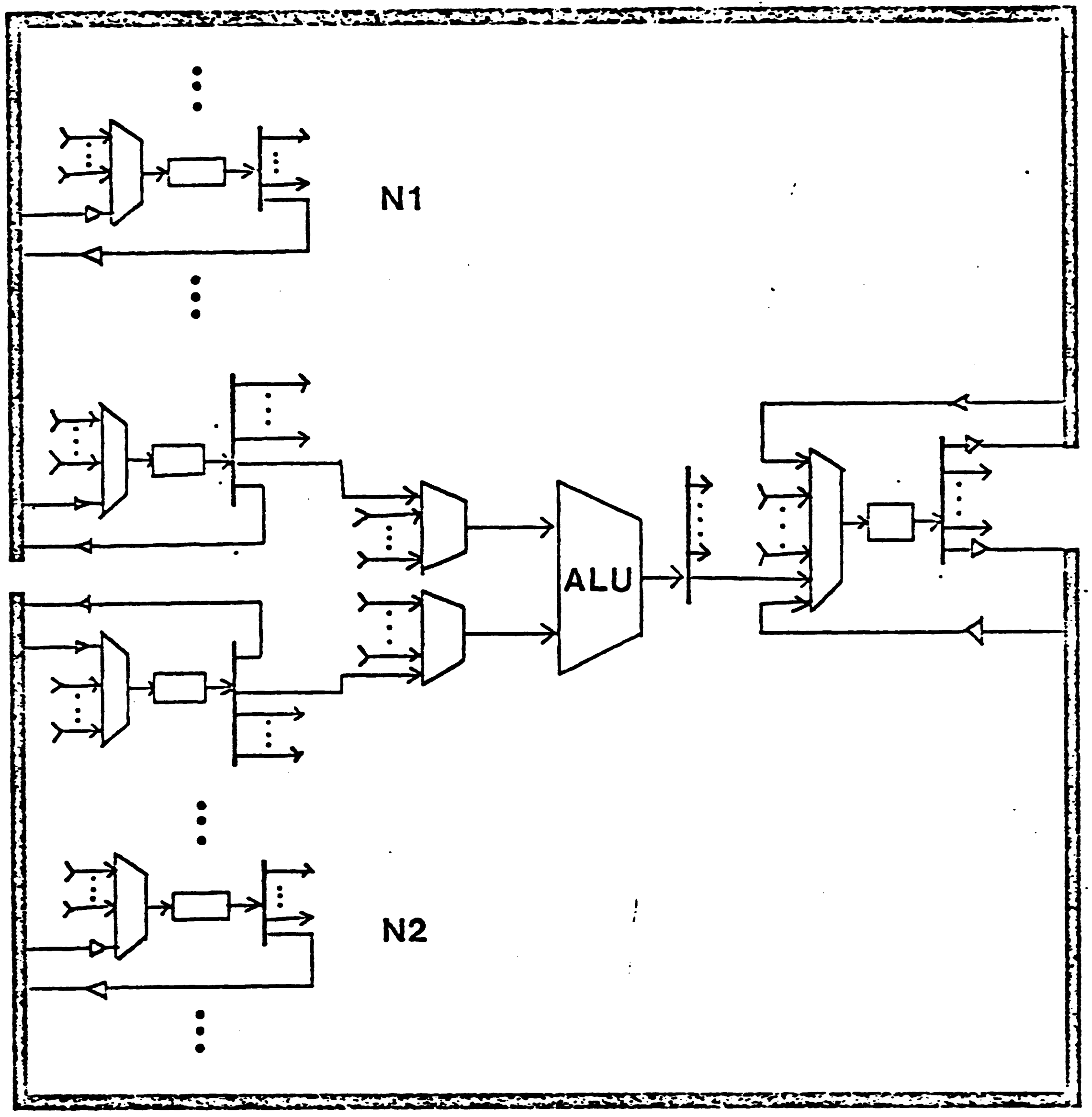


Figure 3-4: Model I of Two-Bus Systems

3.5. Generalization of the Basic Models

Generally speaking, additional number of buses can be included to facilitate more inter-register data transfers. Theoretically, a system may have an arbitrary number of buses. However, due to the limitation on the availability of data operators, extra buses can not significantly improve the performance.

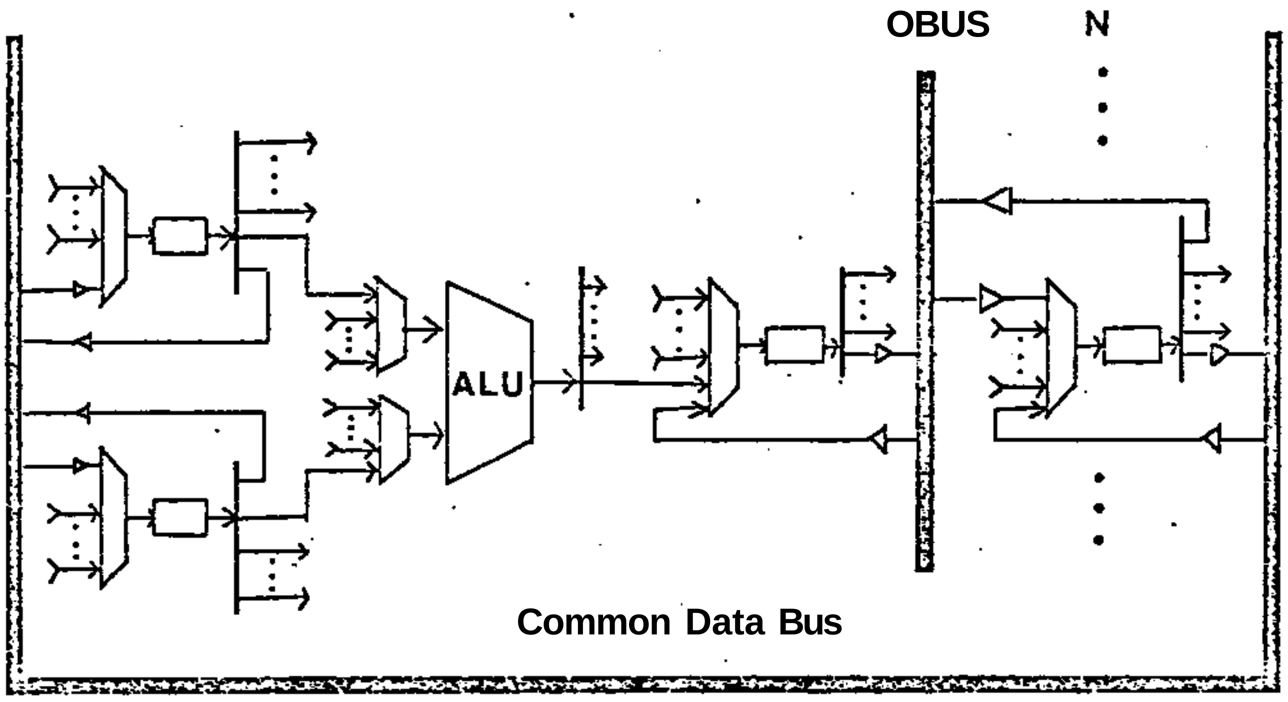


Figure 3-5: Model II of Two-Bus Systems

4. A Synthesis Algorithm for Designing Bus Style Systems

4.1. Related Work

Taking the data transfer variables and concurrency constraints as the inputs, Torng and Wilhelm[Torn77] proposed a formal algorithm for the synthesis of digital systems. They characterized a design according to the number of buses and the number of links.[2] Their algorithm applied the technique of generation and test in dynamic programming to find all the solutions which have the minimal numbers of buses and links. The combinatorial searches require significant computation. Mathialagan and Biswas[Math??] addressed the same problem with the techniques for finding the minimal cover in switching theory. Their algorithm reduces the amount of computation required to **perform** a design. However, finding the minimum link solution is still'a cumbersome process. In addition, both researchers treated every interconnection as a general bus. It is not unusual that interconnections degenerate into the other primitives presented in Section 2: a multiplexer, a wired-broadcast tree, and a direct connection. Thus, interconnections have different costs. The algorithm proposed in this paper is conceptually very simple. The strategy adopted is, without violating the constraints of concurrent data transfers, to assign as many of the data transfer variables to a bus as

[2]A link is named as a gating element in this paper.

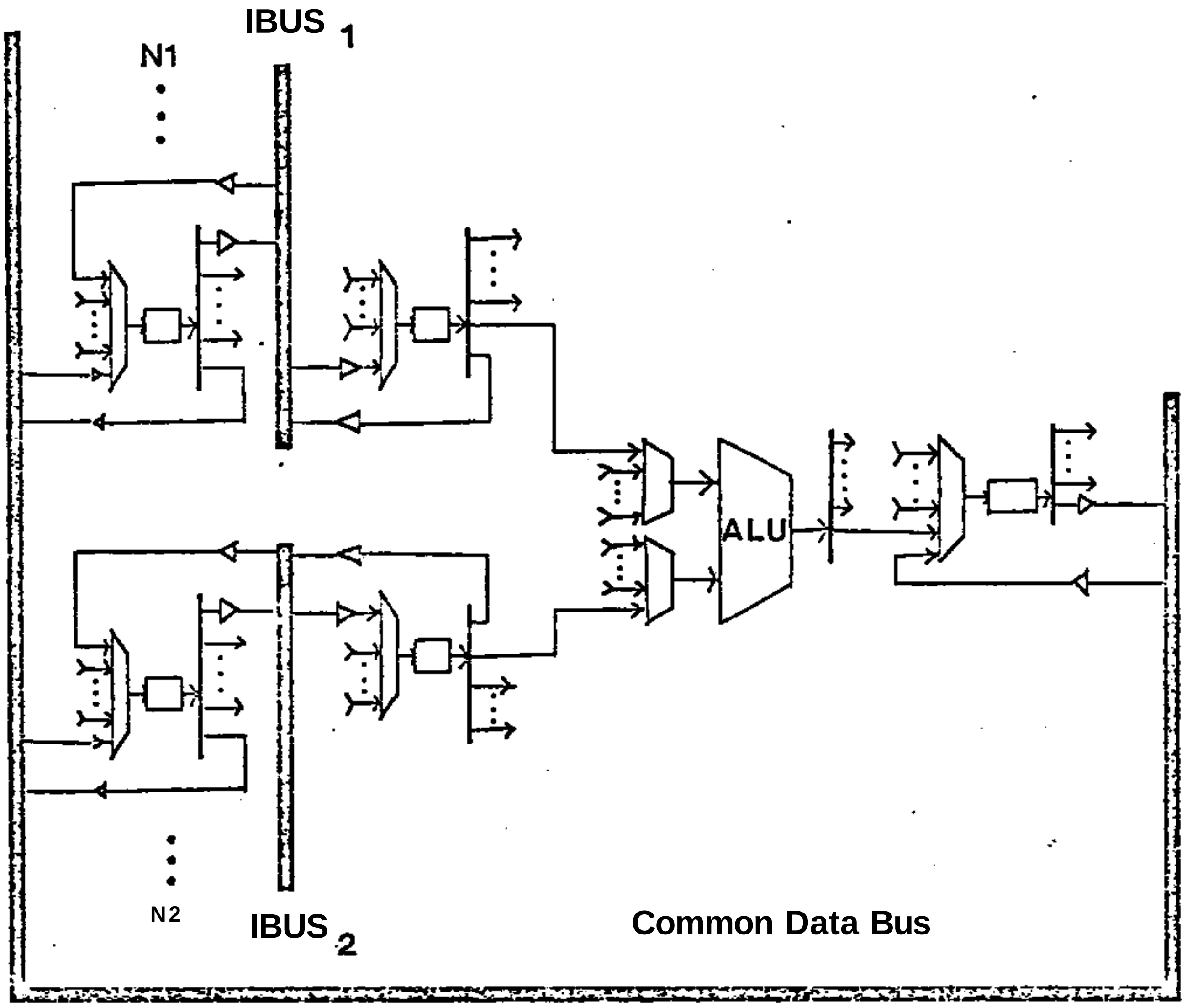


Figure 3-6: A Canonical Model of Three-Bus Systems

possible. The algorithm also attempts to reduce the number of gating elements. The next section describes the representation used in the algorithm.

4.2. Representation

In [Torn77] the data transfers in a system are represented by a transfer matrix. Basically, the same representation is used in the proposed algorithm. The rows and columns in the transfer matrix specify the sources and sinks of data transfers respectively. An entry in the transfer matrix may bo null or filled with concurrency identifiers depending on whether there are data transfers between the specified source and sink or not. A data transfer from a source indexed with i to a sink indexed with j is identified by a variable X(iJ). Those data transfers which occur simultaneously are assigned with

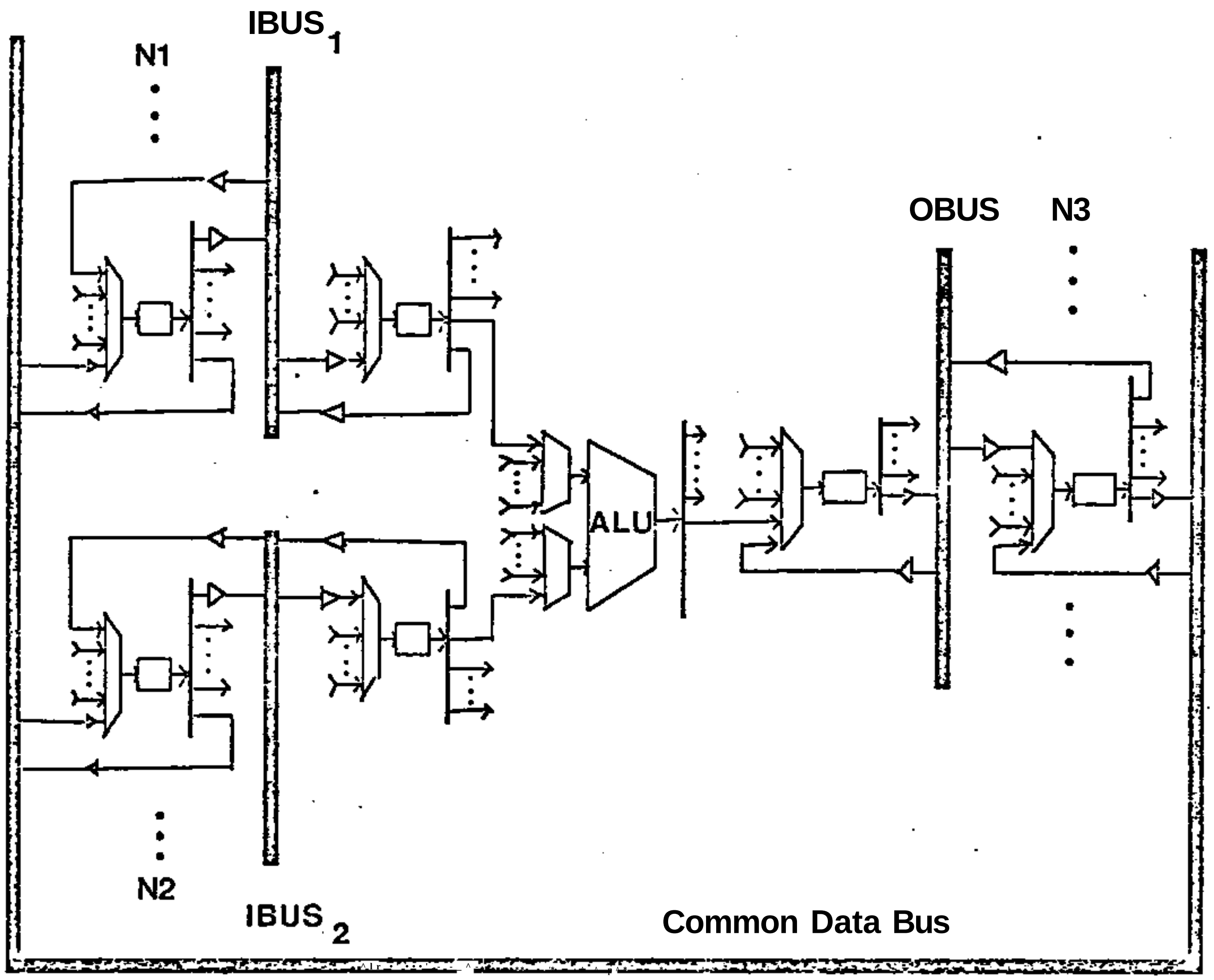


Figure 3-7: A Canonical Model of Four-Bus Systems

the same concurrency identifiers. A null transfer has no concurrency identifier. The concurrency identifier of an isolated data transfer is assigned to be zero. Each group of simultaneous data transfers are identified by a unique nonzero integer. Thus a data transfer variable might have one or more than one concurrency identifiers. The transfer matrix in Figure 4-1 which represents the data transfers in the HP 2116 minicomputer [Torn77] can be used to explain these identifiers. The system consists of registers A, B_t M, P, T, and an ALU which are indexed with integers 1, 2, 3, 4, 5, and 6 respectively. As an example, data transfer variable $X(1_V6)$, $X(5,6)_f$ X(6,1), and $X(6_t5)$ have the same concurrency identifiers, they represent data transfers which may occur at the same time.

The transfer matrix of a system can be derived from, its control specifications. For a microprogramming system, the operations specified by a microinstruction are considered to occur

concurrently. Let op be a binary operation in the ALU and -> mean "store into". The data transfer variables X(1,6), X(5,6), X(6,1), and X(6,5) mentioned in the last paragraph can be derived from a microinstruction which initiates the operation A op T -> A, T.

		1 A	2 B	3 M	4 P	5 T	6 ALU
1	A					0	1
2	B					0	5
3	M				0		
4	P						2 3
5	T	0	0	0	0		145
6	ALU	1	5	2	2	134	

Figure 4-1: The Transfer Matrix Representation of the HP 2116 Computer

4.3. The Algorithm

In order to describe the algorithm, the **following terms are defined:**

- **Compatible Set:** a set of data transfers which are in the same row or are in the same column and do not have the same concurrency identifiers. The data transfer variables in **a** compatible set can share a bus.

- **Maximal Incompatible Set:** the set **of all** concurrent transfers each of which originates **from** a different source constitute a maximal incompatible set (abreviated as MIC). An **MIC** can be defined with reference to the transfer matrix as the set of all data transfer variables with the same concurrency identifier, each of which appears in a different row.

The algorithm generates the buses of a system one by one. In general, a bus can carry a number **of data** transfers. Before a bus is generated, the data transfer variables which are to be assigned to the bus are unknown. A "selected set" is used to contain those data transfer variables which can be realized with the bus being allocated. Initially the selected set is an empty set.

As mentioned in Section 2, a gating element is required to control the input from and/or output to a bus. If the data transfer variables in a compatible set (which have the same source or sink) can share a bus, only one gating element is required for the source or sink. If more than one bus is necessary to carry the data transfers, additional gating elements are required. To minimize the number of gating elements, the number of buses assigned to the data transfer variables in a compatible set must be minimized. Based on this criterion, the algorithm is:

1. Put all of the isolated transfer variables into the selected set and exclude these elements from the transfer matrix by deleting the corresponding concurrency identifiers.

2. Construct all of the possible disjoint compatible sets in the reduced matrix. Those compatible sets which consist of elements in a row can be easily constructed. Taking advantage of the concurrency identifiers, the maximum compatible sets in a column can also be constructed.

a Choose a compatible set which has maximum cardinality from the compatible sets constructed in step 2, named MCS. Put all of the transfer variables in MCS into the selected set.

4. Delete those elements which are already contained in the MCS from the remaining compatible sets.

5. Taking advantage of the MCS just selected and the MIC, delete those elements which can not share the bus being allocated from the compatible sets.

6. Repeat steps 3 to 5 till there are no more data transfer variables that can be included in the selected set.

7. Delete the concurrency identifiers of those variables which have already been selected from the transfer matrix.

Now a set of data transfer variables which can share a bus have been collected. The original transfer matrix is reduced to a smaller one. Any data transfer variable accompanied with a lone concurrency identifier become an isolated element. Its concurrency identifiers are updated to zero. Repeatedly applying the above steps, the transfer matrix will enventually be reduced to a null matrix. Each time we finish steps (1) to (6), a new bus is generated.

In step (3), if more than one compatible set has the same number of elements, perform step (5), compare the number of variables to be excluded, and choose a compatible set which excludes the least number of variables. Theoretically, the lookahead can be further applied. However, if the size of the selected sets are still the same after one stage lookahead, choosing any one of these selections will result in a system with essentially the same cost and performance. If necessary, all of the alternate solutions can be found by repeating the derivation procedures.

Refining the Initial Design:

As depicted in Figure 4-2(a), an initial design might have a "join node" in which more than one bus is connected to a single input. To make the data paths adhere to the basic models in Section 3, a multiplexer is inserted in front of the input. The resulting data paths are shown in Figure 4-2(b). The "join-node" can easily be found by checking the data transfers to an input. If the input data transfers

are fed from more than one bus, then this node needs to be refined.

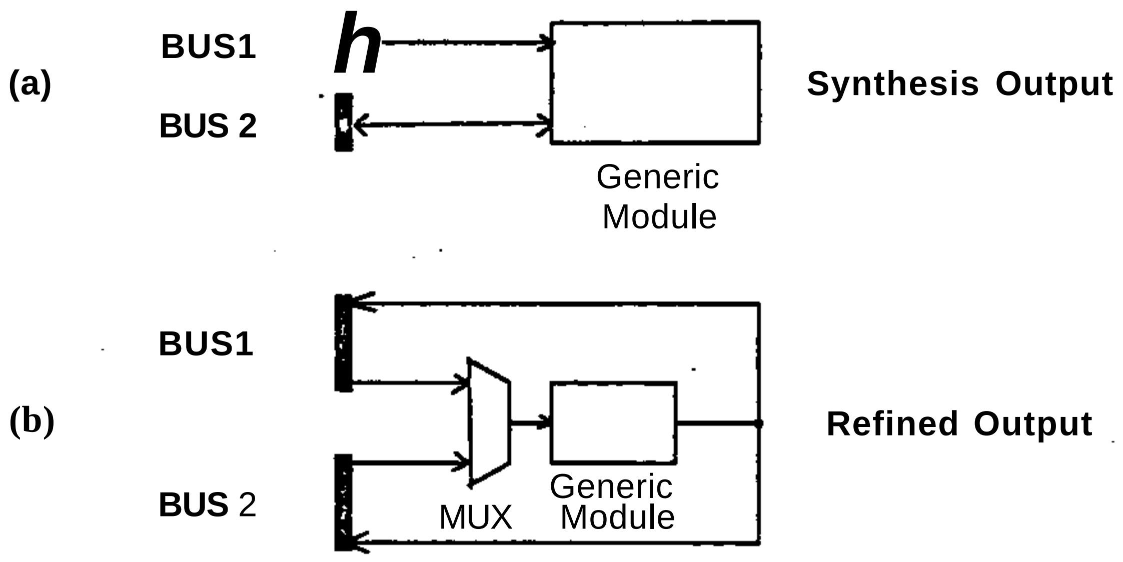


Figure 4-2: Refinement of A ^{H}Join-Node[19]

4.4. Illustrative Examples

For the purpose of illustration, two examples are given. The first example is used to compare the proposed algorithm with others in the literature. The second example is used to compare the POP* 11/40 data paths synthesized by the proposed algorithm and the actual implementation.

Example 1.

In Figure 4-1, a transfer matrix constructed from a description of the Hewlett Packard 2116 computer is given.

The synthesis process is as follows:

- Put isolated transfer variables X(1,5), X(2,5), X(3,4), X(5,1), X(5,2), X(5,3), and X(5,4) into the selected set and delete the concurrency identifiers of these elements from the original transfer matrix.

- Construct the compatible set from the reduced matrix. They are { X(6,1), X(6f2), X(6,3), X(6,4), X(6,5) }, { X(1,6), X(2,6), X(4,6) }, and { X(4,6), X(5,6) }.

- Choose th^compatible set which has the maximal cardinality, { X(6,1), X(6,2), X(6,3), X(6,4), X(6,5) }. Put these transfer variables into the selected set.

- Since none of the elements in the selected compatible set is contained in the remaining compatible sets, step (4) in the algorithm is skipped.

- The remaining compatible sets are { X(1.6), X(2,6), X(4,6) } and { X(4,6), X(5,6) }. None of these data transfer variables can share the bus being allocated. All of them are deleted.

- Now, no more data transfer variables can share the bus being allocated.

- Delete the concurrency identifiers of those elements in the selected set from the transfer matrix. Update the concurrency identifiers of each isolated data transfer in the transfer matrix to zero. The reduced matrix is shown in Figure 4-3.

Repeat these procedures for the reduced matrix. Finally we find that the data transfer variables are partitioned into three sets. They are { X(1,5), X(2,5), X(3,4), X(5,1), X(5,2), X(5,3), X(5,4), X(6,1), X(6,2), X(6,3), X(6,4)f X(6,5) }, { X(1,6), X(2,6), X(4,6) }, and { X(5,6) }. The data paths are depicted in Figure 4-4.[3]

The data paths drawn in Figure 4-5 and Figure 4-6 were synthesized by Torng and Mathialagan, et al. Assume all the data paths are n bits wide. If every interconnection is regarded as a general bus, Figure 4-5 and Figure 4-6 require 15*n gating elements. Figure 4-4 needs 16*n gating elements. In practice, one gating element is enough to control the data flow through a bus which has a single input and output Taking this special case into account, Figure 4-4, Figure 4-5, and Figure 4-6 require 15*n, 14*n and 15*n gating elements respectively. Furthermore, in Figure 4-4 the bus which has three inputs and one output can be replaced by a built-in multiplexer. The external connections are reduced.

According to the above explannation, we see the cost of these systems are comparable. However, the amount of computation required by the proposed algorithm is much less than that required by the algorithms proposed by Torng and Mathialagan, et al.

Example 2.

Figure 4-7 is a transfer matrix constructed from a microprogram of the PDP-11/40 [Full76]. SPM, D, DPY, B, BA, IR, PS are registers. B.AUX represents constants, sign extender, and byte swapper. CC represents the condition codes. UBUS represents a system bus, the UNIBUS. Applying the proposed algorithm to this transfer matrix, the synthesis process can be summarized as follows:

- Put the isolated data transfer variables X(1,4), X(1,8), and X(10,8) into the selected set.

- There are two compatible sets with the same cardinality. They are { X(10,1), X(10,3), X(10,4), X(10,7), X(10,8) } and { X(2f1), X(2,3), X(2,4), X(2,8) }. If the first set is chosen,

[3]For clarity, instead of using unidirectional interconnections, bidirectional gating elements are used in Figure 4-4, Figure 4-5, and Figure 4-6.

		1 A	2 B	3 M	4 P	5 T	6 ALU
1	A						1
2	B						5
3	M						
4	P						0
5	T						1 5
6	ALU						

Figure 4-3: A Reduced Transfer Matrix for the HP 2116 Computer

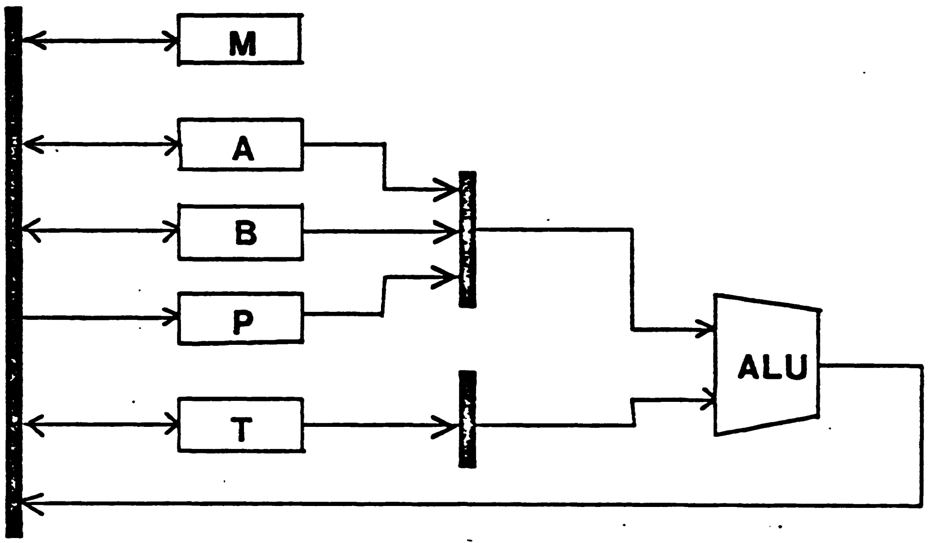


Figure 4-4: Data Paths of the HP 2116 Computer -- Proposed Algorithm

data transfer variables X(1,11), X(5,11), X(11,2), and X(11,6) must be excluded. If the second set is chosen, then the data transfer variables to be excluded are X(1,6), X(1,11), X(4,11), X(5,11), X(8,11), X(11,2), and X(11,6). The first compatible set excludes fewer data transfer variables. Thus data transfer variables X(10,1), X(10,3), X(10,4), X(10,7), and X(10,8) are put in the selected set.

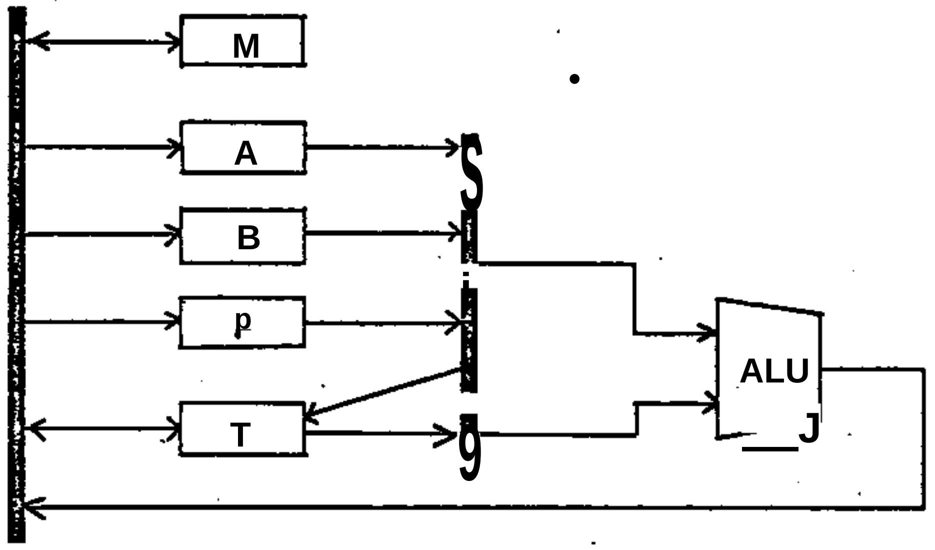


Figure 4-5: Data Paths of the HP 2116 Computer - [Torn77] and [Math??]

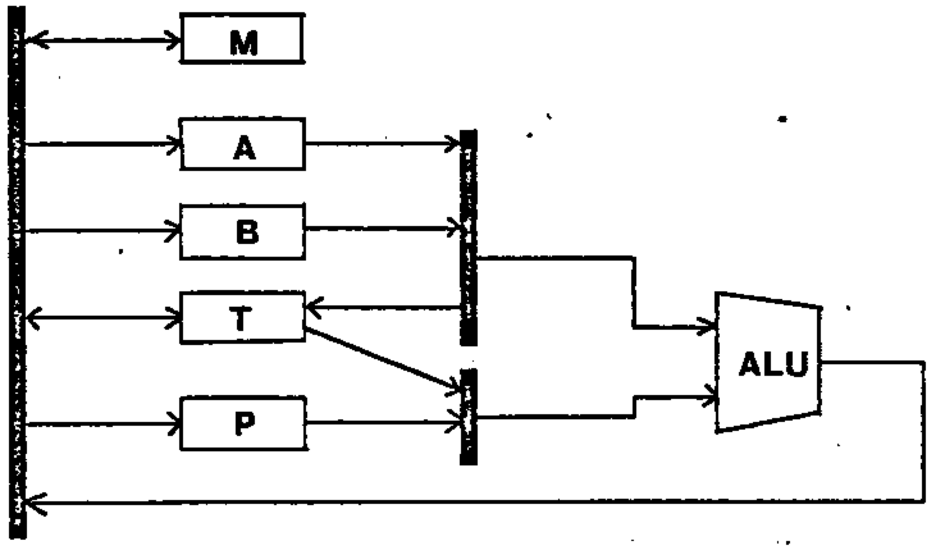


Figure 4-6: Data Paths of the HP 2116 Computer - [Torn77] and [Math??]

- Now the MCS { X(2,1), X(2,3), X(2,4), X(2,8) } has the maximum cardinality. These data transfer variables are put in the selected set.

- Deleting those data transfer variables which can not be included in the selected set, only X(1,3) remains in the compatible sets. Put it in the selected set.

- The selected set consists of data transfer variables X(1,4), X(1,8), X(10,8), X(10,1) X(10,3).

X(10,4), X(107), X(10,8), X(2,1), X(2,3), X(2,4), X(2,8), and X(1,3). They share the first bus- Deleting these data transfer variables and updating the concurrency identifiers, the original transfer matrix can be reduced to the form of Figure 4-8.

- Repeatedly applying the synthesis process, the following selected sets will be constructed. Each of them corresponds to a bus.

Selected Sets:

Bus 1	{ X(l,4). X(l,8). X(10,8), X(10tl), X(10,3), X(10,4), X(10.7), X(2,l), X(2,3), X(2,4), X(2,8)f X(l,3) }
Bus 2	{ X(l,6), X(lfll), X(8fll) }
Bus 3	{ X(11.2), X(ll,6) }
Bus 4	{ X(4,ll), X(5fll) }
Bus 5	{ X(9,8) }

Inspecting these selected sets, it is found that primitive modules BA and PS are fed by more than one bus with different concurrency identifiers. BA is fed by Bus 2 and Bus 3 while PS is fed by Bus 1 and Bus 5. To refine the data paths, a multiplexer is inserted in front of each of these two primitive modules. The resulting data paths are drawn in Rgure 4*9.

Rgure 4-10 depicts the actual implementation of the PDP-11/40. Comparing Figure 4-9 and Rgure 4-10, a slight difference is found in these two data graphs. The DMUX and the wired-broadcast tree following it are replaced by a general bus (Bus 1). The AMUX and the wired-broadcast tree following it are replaced by Bus 2.

		1	2	3	4	5	6	7	8	9	10	11
		SPM	D	DPY	B	B.AUX	BA	IR	PS	CC	UBUS	ALU
1	SPM			13 16	0		1 12		0			1,2,3,4 11,12,14
2	D	1,2 3 17		6,7 12 18	4,9 10,11 17				5			
3	DPY											
4	B											2,6
5	B.AUX											1,3,4,5 7,8,9 10,13
6	BA											
7	IR											
8	PS											5,6
9	CC								5,9,10 11,16 17,18			
10	UBUS	14 15		8	15			15	0			
11	ALU		1,2,3,4 5,6,7 8,9,10 11,12: 13,14				2 3					

Note: SPM a scratchpad memory UBUS = unibus
DPY = display register CC = condition codes

Figure 4-7: The Transfer Matrix Representation of the PDP-11/40 Computer

		1 SPM	2 D	3 DPY	4 B	5 B.AUX	6 BA	7 IR	8 PS	9 CC	10 UBUS	11 ALU
1	SPM						1 12					1,2,3,4 11,12,14
2	D											
3	DPY											
4	B											2,6
5	B.AUX											1,3,4,5 7,8,9 10,13
6	BA											
7	IR											
8	PS											5,6
9	CC								5,9 10,11*			
10	UBUS											
11	ALU		1,2,3,4 5,6,7 8,9,10 11,12 13,14				2 3					

Note: SPM = scratchpad memory UBUS = unibus
DPY = display register CC = condition codes

Figure 4-8: A Reduced Transfer Matrix for the PDP-11/40 Computer

Table 4-1 lists the number of gates required for the corresponding counterparts of these two data graphs. The number of gates required for Figure 4-9 and Figure 4-10 are 96 and 144 respectively. Even if the cost of a tri-state buffer is one and a half times that of a general logic gate, the cost of Figure 4-9 is less than that of Figure 4-10. In fact the sixteen drivers and sixteen receivers which are used as buffers between BUS 1 and the SPM can be replaced by sixteen transceivers, further

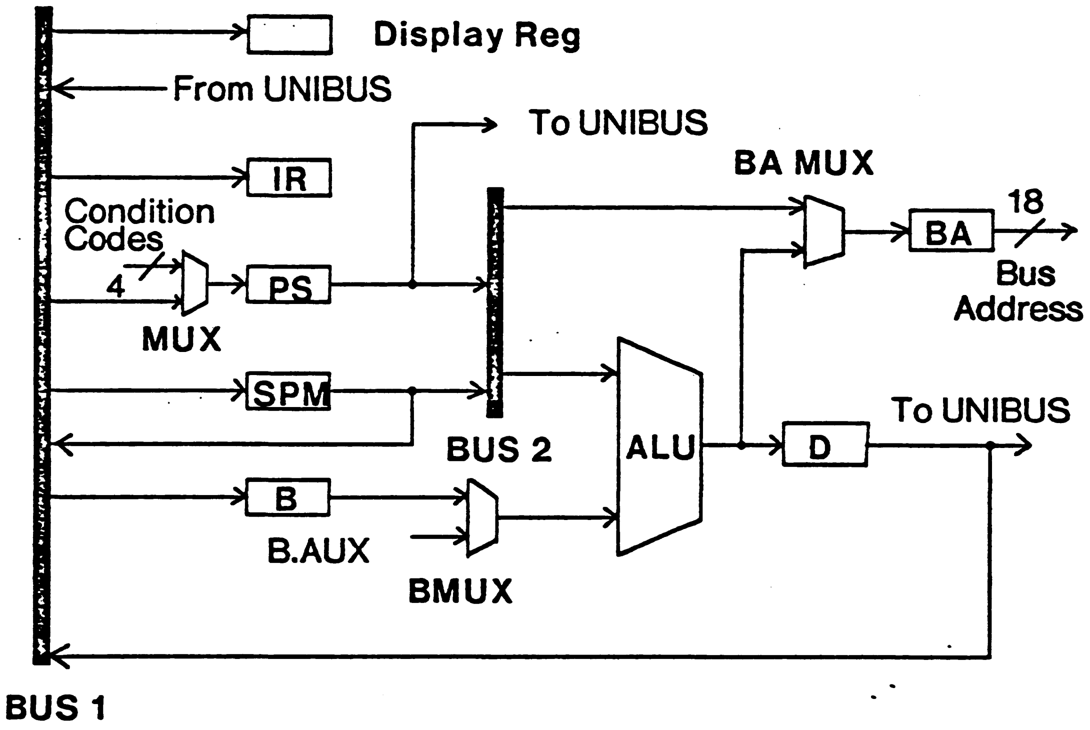


Figure 4-9: Data Paths of the PDP-11/40 Computer -- Proposed Allocation

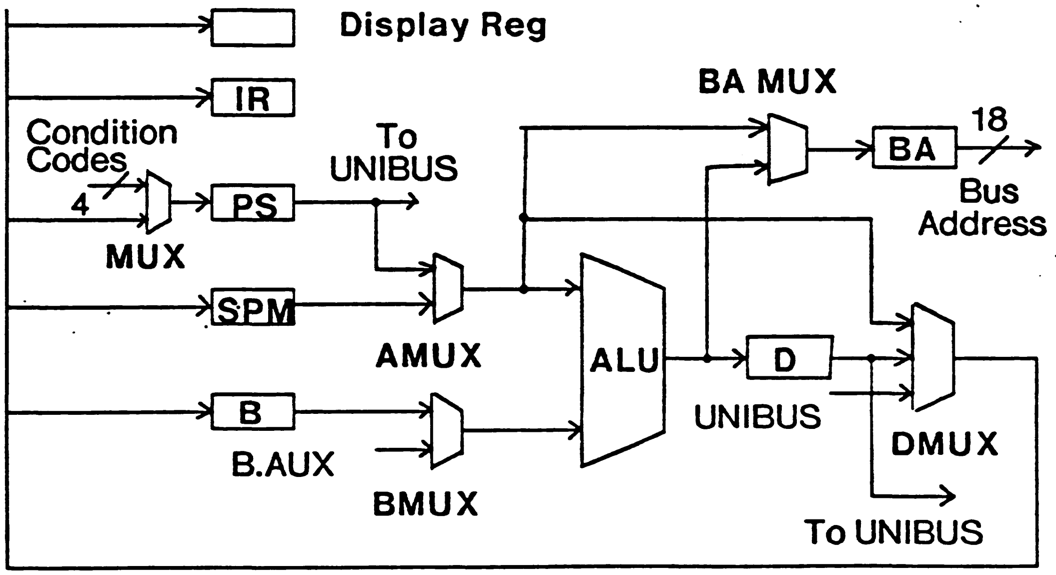


Figure 4-10: Data Paths of the PDP 11/40 Computer -- Actual Implementation

decreasing the cost of the implementation in Figure 4-9.

Table 4-2 lists the delays in selected data transfers. Again, the results show that Figure 4-9 is a better design than Hgure 4-10.

Table 4-1

Figure 4-9 module name	Figure 4-9 gate count	Figure 4-10 module name	Figure 4-10 gate count
Between SPM and BUS 1	16	AMUX	16
Between SPM and BUS 2	16	DMUX	128
Between PS and BUS 2	16		
Between BUS 2 and BAMUX	16		
Between D and BUS 1	16		
total	96		144

Note: AMUX consists of six SN7412 3-input positive NAND gate with open-collector outputs.
DMUX consists of eight SN74153 multiplexers.
The modules used for Figure 4-10 are tri-state buffers (e.g. SN74125 or SN74126).

Table 4-2 Delays for Selected Data Transfers

	Figure 4-9	Figure 4-10
From SPM or PS to ALU	20 ns	22 ns
From SPM or PS to BAMUX	20 ns	22 ns
From SPM to B, PS. or IR	20 ns	39 ns

Note: The typical delay times of SN7412, SN74153, SN74125. SN74126 are 22 ns, 17 ns, 10 ns, 10 ns respectively.
The delay time from SPM to B, PS. or IR consists of delays of SN7412 and SN74153.
The delay of the data transfers mentioned for Figure 4-9 consists of the delays of a driver and a receiver.

5. Comparing Bus Style Design and Distributed Style Design

Currently a distributed style allocator is functioning at CMU. Both the bus style design and the distributed style design assume that abstract modules are used for storage and data operations. Their differences are in the interconnections. These interconnection differences provide the foundation of comparing these two design styles.

5.1. Comparing the Synthesis Algorithms

Conceptually, the distributed style design used a bottom-up approach while the bus style design adopted a top-down approach. The distributed style allocator designed by Hafer [Hafe77] assumed the availability of point-to-point data paths for data transfers. The data paths are allocated only when they are required. A number of optimization criteria were proposed in [Hafe79]. If a storage element or data operator is fed by multiple sources, a multiplexer is used to select the active source. If a data source is feeding multiple destinations, a wired-broadcast tree is used to distribute the data. As illustrated in Section 4, the algorithm proposed in this paper pays attention to structuring the interconnections into a number of buses. A bus is usually associated with multiple sources and multiple sinks. A multiplexer or wired-broadcast tree is a degenerate form of a bus which has a single source or sink. Thus the bus style design is considered to be more general than the distributed design.

5.2. Comparing the Structure of a Bus and its Distributed Counterpart

What is the distributed counterpart of a bus? Functionally, a general bus can be regarded as a multiplexer followed by a wired-broadcast tree. In pratice, depending on the sophistication of the distributed data paths, the distributed counterpart of a bus may consist of the following entries: simple interconnections, multiplexers, wired-broadcast trees, and/or a set of basic units which are composed of a multiplexer followed by a wired-broadcast tree.

5.3. Comparing the Delay and the Cost

Since the cost of each individual component inflates with time, it is very difficult to estimate the total cost of a system. However, a brief discussion might be helpful. Assume the cost of the electric transmission medium is negligible. Then the cost associated with a bus may be counted by the number of gating elements adhering to the bus. The number of gating elements required in a bus style design depends on the direction of data transfers. If a primitive module only consists of an input or output data transfer with respect to a bus, then one gating element is enough. If data transfers in both directions are needed, then depending on a bidirectional or two unidirectional gating elements are used, one or two gating elements are required. The cost of the distributed counterpart is the sum of the cost of the gating elements and that of the multiplexers. If a large number of multiplexers are required, the distributed style design would be more expensive than the bus style design.

Assume the delay time of the electric transmission medium is negligible. Considering the delay, a data transfer through a bus consists of delays of two gating elements. If the corresponding distributed data paths do not consist of a multiplexer, the delay is the time spanned by one gating element. If a multiplexer is in the counterpart, then the delay of the multiplexer must be considered. The delay of a multiplexer depends upon the number of levels in the circuit. The number of levels in a multiplexer circuit is a function of the number of inputs. When a multiplexer consists of a large number of inputs, its delay is then significant.

5.4. Adaptivity of these Two Design Styles

In a distributed system an interconnection mechanism directly connects two primitive modules. As the number of interconnections increases, its complexity also increases rapidly. In a bus system, instead of directly connecting to the primitive modules, the interconnections are added to the buses. Thus the interconnection complexity of a bus style design is better than that of a distributed style design for a large system.

As mentioned in Subsection 5.3, the cost and delays of multiplexers in a distributed system play an important role in the cost and performance of the system. A bus is usually shared by a great number of data transfers in a large system. Thus the distributed counterpart of the bus generally consists of a large number of multiplexers. Based on these justifications, the bus style design is more adaptive to the design of large systems.

6. Conclusions and Future Work

The basic structure of bus style systems has been identified. Two criteria, the degree of parallelism (the number of buses) and the input data set-up time were presented to evaluate a bus system. Taking a transfer matrix as the input, a conceptually simple algorithm for the synthesis of bus style systems has been proposed. Finally, the differences between the distributed and bus style designs have also been compared and contrasted.

The input of the allocator is an intermediate form [McFa78,Snow78] of the original ISPS behavior description [Barb78,Bell71 J. Many parameters in the ISPS, which govern the design decision, need to be specified. For instance, an array storage may be treated as a bulk memory or as a register file. Which choice is better? The criteria (cost and speed) which influence the decision must be justified. Information provided by ISPS may be important or irrelevant to the design of bus style systems. What kinds of information are essential to designing a bus style system? Can they be expressed in ISPS descriptions? How can we generate this information which can not be expressed in an ISPS description? On the other hand, the variations and tradeoffs between cost and performance of an initial design are also worthwhile to investigate. For instance, it is probable that an operation can be realized with different kinds of implementation. Sharing a common bus with other operations might be more cost-effective than using dedicated data paths. However, the latter option can generally improve the performance. Which strategy should be adopted?

As depicted in Figure 6-1, the bus style data-memory allocator is partitioned into the mapping, the synthesis and the control specification modules. The mapping module reads in the VT file [Snow78,McFa78] and converts it into a transfer matrix representation. The function of the mapping module is basically scanning the VT file and assigning semantics. Two approaches may be applied for the design of the scanner. One is to treat it as a finite state automaton recognizer. The other is to apply table look-up techniques. Though it is not clear to us at this time, it is believed that a lot of optimization criteria are inhered in the semantic phase. For example, the issues of efficient register allocation in compiler design might be applicable. To search for and identify these optimization criteria is essential in this phase. The second module is called the synthesis module. Using the algorithm presented in Section 4, the synthesis module will construct a data-path graph for the transfer matrix. The data-path graph is a "static" structure. In order to coordinate with the control allocator, the operation sequences must be specified. The control specification module will specify the operation sequences with respect to the data paths.

The software depicted in Figure 6-1 is nearing completion.

Acknowledgements

The CMUDA system is the result of many individuals. This study could not have been completed without the previous work of the other CMUDA members and their assistance. Especially, helpful conversations with Dr. Steve W. Director, Dr. Donald E. Thomas, Dr. Alice C. Parker, Jin H. Kim, Lou Hafer, Michael McFarland, and John Nestor are gratefully appreciated.

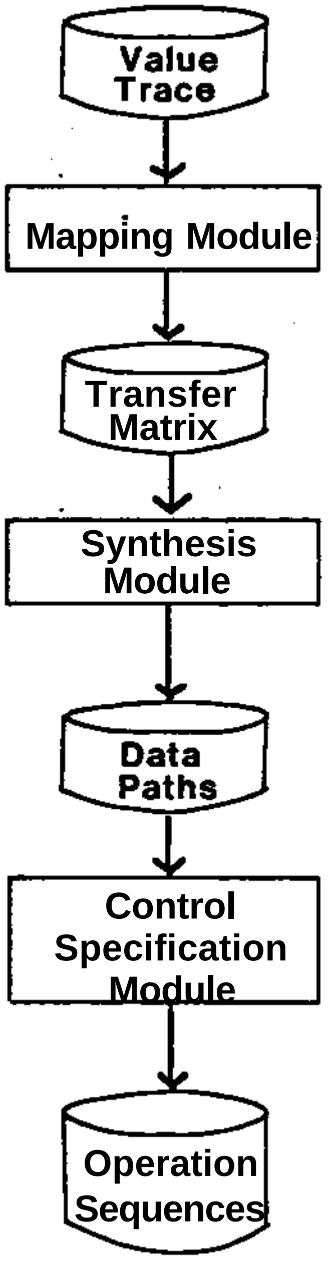


Figure 6-1: The Global Structure of a Bus Style Data-Memory Allocator

Bibliography

[Ande67] D. W. Anderson, F. J. Sparacio, and R. M. Tomasuio.
The IBM System/360 Model 91: Machine Philosophy and Instruction-Hand!ing,
IBM Journal of Research and Development, Vol. 11, No. 1, pp. 8-24, January 1967.

[Barb78] Mario R. Barbacci, Gary E. Barnes, Roderic G. Cattell, and Daniel P. Siewiorek.
The Symbolic Manipulation of Computer Descriptions: The ISPS Computer Description Language.
Technical Report, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania, March 1978.

[Bell71] C. Gordon Bell and Allen **Newell.**
Computer Structures: Readings and Examples.
McGraw-Hill, New York, 1971.

tFull76] S. H. Fuller, G. T. Almes, W. H. Broadley, C. L. Forgy, P. L. Karlton, V, R. Lesser, and J. R. Teter.
PDP-11/40E Microprogramming Reference Manual.
Technical Report, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania, January 16, 1976.

[Hafe77] Lou Hafer.
Data-Memory Allocation in the Distributed Logic Design Style,
M.S. Project Report, Department of Electrical Engineering, Carnegie-Mellon University, Pittsburgh, Pennsylvania, December 1977.

[Hafe79] Lou Hafer.
Automated Data-Memory Synthesis.
Ph.D. Thesis Proposal, Department of Electrical Engineering, Carnegie-Mellon University, Pittsburgh, Pennsylvania, March 6, 1979.

[Math??] A. Mathialagan and Nripendra N. Biswas.
The Optimal Interconnections in Microprocessor and Digital Systems.
Department of Electrical Communication Engineering, Indian Institute of Science, Bangalore, India.

[McFa78] Michael McFarland.
The Value Trace: A Data Base for Automated Digital Design.
Master Thesis, Department of Electrical Engineering, Carnegie-MelIon Universityt Pittsburgh, Pennsylvania, December 1978.

[Park79] A. C. Parker, D. E. Thomas, D. P. Siewiorek, M. R. Barbacci.

L. Hafer, G. Leive, and J. Kim.
The CMU Design Automation System: An Example of Automated Data Path Design.
In Proceedings of The Sixteenth Design Automation Conference, ACM SIGDA and IEEE Computer Society DATC, June 1979.

[Siew76] Daniel P. Siewiorek and Mario R. Barbacci.
The CMU RT-CAD System: An Innovative Approach to Computer Aided Design.
In American Federation of Information Processing Societies Conference Proceedings, Vol. 45, June 1976.

[Snow78] E. Snow and Daniel P. Siewiorek.
Impact of Implementation, Design Tradeoffs on Performance: The PDP-11, A Case Study.
In Computer Engineering: A DEC View of Hardware Systems Design by C. Gordon Bell, J. Craig Mudge, and John E. McNamara.

[Thom77] Donald E. Thomas.
The Design and Analysis of An Automated Design Style Selector.
Ph.D. Dissertation, Department of Electrical Engineering, Carnegie-Mellon University, 1977.

[TI76] Texas Instruments Incorporated.
The TTL Data Book for Design Engineers.
1976.

[Toma67] R. M. Tomasuio.
An Efficient Algorithm for Exploiting Multiple Arithmetic Units.
IBM Journal of Research and Development, Vol. 11, No. 1, pp. 25-33, January 1967.

[Torn77] H. C. Torng and Neil C. Wilhelm.
The Optimal Interconnection of Circuit Modules in Microprocessor and Digital System Design.
IEEE Transactions on Computers, Vol. C-26, Mo. 5, pp.450-457, **May** 1977.