NASA Contractor Report 187625

ATAMM Analysis Tool

Robert Jones, John Stoughton,
and Roland Mielke

OLD DOMINION UNIVERSITY RESEARCH FOUNDATION
Norfolk, Virginia

Cooperative Agreement NCC1-136
October 1991

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

LIST OF SYMBOLS

SYMBOL	DESCRIPTION
ADG	Algorithm Directed Graph
ADM	Advanced Development Model
AMG	Algorithm Directed Graph
AMOS	ATAMM Multicomputer Operating System
ATAMM	Algorithm to Architecture Mapping Model
C_i	ith circuit in the CMG
CMG	Computational Marked Graph
DP	Data processed
DR	Data read
E	FDT file event: waiting for the channel in order to read
F	FDT file event: reading
FDT	Fire Data Time
FU	Functional Unit
I	FDT file event: processing
IBM PC	International Business Machines Personal Computer
I.D.	Identification
IE	Input buffer empty

IF	Input buffer full
I/O	Input/Output
$M(C_i)$	Number of tokens in C_i
N	Number of nodes in the AMG
NMG	Node Marked Graph
O	FDT file event: testing
OE	Output buffer empty
OF	Output buffer full
P	FDT file event: waiting for the channel in order to write
P_i	ith path between source and sink in AMG
PR	Process ready
Q	FDT file event: returning FU I.D. to the resource queue
R	FDT file event: idle
S	FDT file event: writing
T	FDT file event: waiting for the channel in order to return the FU I.D. to the resource queue; or a time interval
T1	Time interval for one stay in the idle state of AMOS
T2	Time interval for one stay in the examine state of AMOS
T3	Time interval of the execute state of AMOS
T4	Time interval of the test state of AMOS
T5	Time interval of the update state of AMOS
TB	Broadcast time

TBI	Time Between Inputs
TBIO	Time Between Input and Output
$TBIO_{LB}$	Lower bound limit of TBIO
TBO	Time Between Outputs
TBO_{LB}	Lower bound limit of TBO
TBO_{min}	Minimum TBO due to overhead requirements
TCE	Total Computing Effort
$T(C_i)$	Sum of transition times in C_i
T_e	Evaluation time interval
TE	Time to execute the algorithm operation belonging to the critical node
TG	Channel grab time
TMR	Triple Modular Redundancy
$T(P_i)$	Sum of transitions times in P_i

CHAPTER ONE

Introduction

1.1 Overview

There are increasingly greater demands placed on computer systems today. This is especially true in applications such as real-time signal processing and control systems. Meeting these demands often requires a system with multiple processors working concurrently on the same algorithm. Due to these concurrent activities, a special model is needed to describe the system behavior and predict its performance for real-time applications.

The Algorithm to Architecture Mapping Model (ATAMM) is a new Petri net based model capable of describing the execution of large grain algorithms on data flow architectures. A data flow architecture is one in which instructions are enabled for execution by the arrival of their operands. This is contrary to conventional computer architectures where the execution of instructions are controlled by an instruction counter. Large grain means that the time required to execute data by an algorithm operation is much greater than the time to transfer data between the operations.

The ATAMM model provides a description of the data and control flow necessary to specify the criteria for predictable execution of an algorithm by a

data flow architecture. The ATAMM model also provides the means to investigate different algorithm decompositions without having to consider the hardware. Once the intended hardware is selected, the model can be used to match the algorithm requirements with the hardware capability in order to achieve optimum performance.

Computer diagnostics software is needed to productively analyze the operation of an ATAMM based system. Computer software can provide the means to analyze data reported by the system and portray the system behavior. The software can also extract certain measurements from the reported data in order to evaluate the system performance.

The detailed description of this research is taken from the Master's Thesis by Robert L. Jones entitled "Diagnostics Software for Concurrent Processing Computer Systems." The use of brand names in this report is for completeness and does not imply NASA endorsement.

1.2 Research Objective

The objective of this research is to develop a software diagnostic tool capable of analyzing ATAMM based systems. It will provide the means to examine the overall system behavior and obtain performance measurements. The performance measurements will indicate the computing speed, throughput, concurrency, and resource utilization attained by the system. The software tool also provides measurements associated with system overhead. The software is developed within a window environment in order to provide user-friendly features

that facilitate the analysis process. This software diagnostic tool is able to assist with the development of ATAMM based architectures and the investigation of theories concerning the ATAMM model.

1.3 Report Organization

The ATAMM model is presented in Chapter Two and performance measures are defined. The ATAMM based operating system (AMOS) is also presented. A state diagram description of AMOS is used as a means to discuss its operation. An overhead model associated with AMOS operation is considered. Also, an approach that extends ATAMM to the modeling of a fault tolerant system is presented.

The development of a software tool for analyzing an ATAMM based system is presented in Chapter Three. The program input requirements and output features are discussed.

Experimental results are described in Chapter Four and provide a demonstration of the software developed in Chapter Three. First, the program's overhead evaluation capabilities are demonstrated on a simulated system. Information obtained from the overhead evaluation is then used with the ATAMM model to predict system performance and behavior at two different operating levels. The program is used to evaluate the resulting performance and behavior for comparison with the predictions. Finally, the error reporting features for the analysis of fault tolerant systems is demonstrated.

A summary is given and topics for future research are stated in Chapter Five.

CHAPTER TWO

Theory

2.1 Introduction

This chapter presents the ATAMM model for describing the data and control flow associated with a certain class of algorithms and distributed-processing systems. A brief description of the ATAMM model and its application is presented in Section 2.2. The state diagram description of the ATAMM Multicomputer Operating System (or AMOS) is discussed in Section 2.3. The communication events associated with AMOS are also discussed. These communication events are of particular importance to the software developed in the following chapter. The overhead associated with AMOS is discussed in Section 2.4 by considering the activities other than processing which must occur for correct overall system operation. The extension of ATAMM to incorporate fault detection and correction is discussed in Section 2.5.

2.2 ATAMM Model

There is considerable ongoing research today concerning the modeling of multiprocessor or distributed-processing systems. Of particular interest is the development of parallel architectures composed of identical, special purpose computing elements [1]. The computing elements of a distributed system must

share resources and information. Therefore, there is a need to synchronize and control this sharing in order to obtain correct overall system operation [2].

The ATAMM model incorporates both the data and control flow necessary for the execution of an algorithm decomposition on a special multiprocessor system. The model was designed by Stoughton and Mielke in order to describe the control, communication, and scheduling issues not included in other models [3]. The algorithm graphs applicable to the ATAMM model must be decision free with respect to data flow and contain computationally complex primitive operations. The architecture is intended to be a dedicated distributed-processing system consisting of identical computing elements (functional units). Each functional unit is capable of executing any primitive operation of the algorithm. It is assumed that the number of functional units range from two to twenty.

The ATAMM model is based on a special class of timed Petri nets which lend themselves well to system modeling. A Petri net is a special kind of directed graph capable of describing data and control flow of a system [2]. Petri nets serve as both a graphical and mathematical tool. It is assumed that the reader is already familiar with Petri net theory so a detailed discussion pertaining to the topic will not be provided. The reader unfamiliar with Petri nets should refer to [2] or some other source of information about Petri net theory.

Due to the decision free criteria, the timed Petri net representation is simplified to a class of Petri net called a marked graph. An example of a marked graph is presented in Figure 2.1. Circles represent nodes (transitions) and line

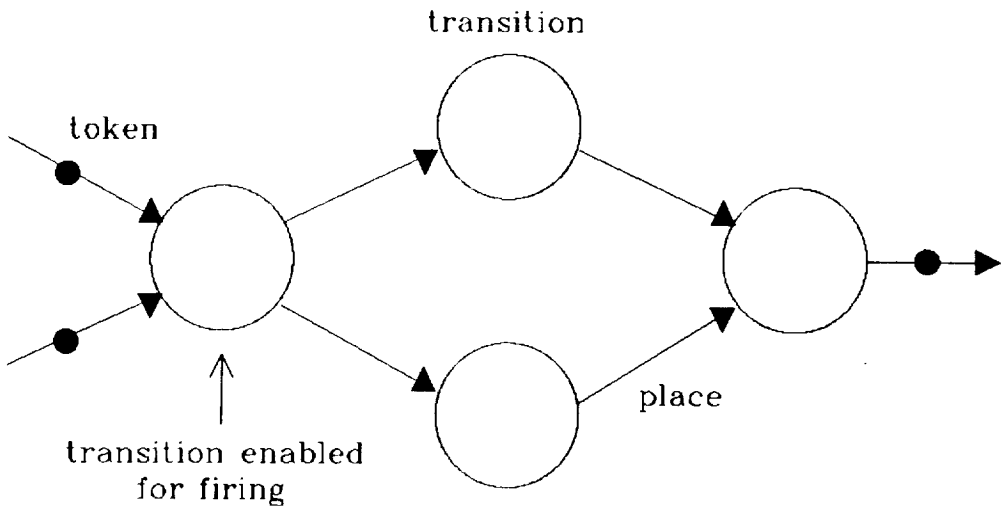


Figure 2.1. Partial Marked Graph.

segments represent edges (places). Tokens representing the availability of signals or data are indicated by black dots on the edges. A node is "enabled" for "firing" by the presence of tokens on all incoming edges. The node "fires" by encumbering all input tokens, delaying for some time interval, and depositing one token on each outgoing edge.

The ATAMM model provides the analytical means to integrate the algorithm data flow with the data flow architecture [1]. The algorithm marked graph (AMG), the node marked graph (NMG), and the computational marked graph (CMG) constitute the three main components of the ATAMM model. A flow diagram portraying the ATAMM modeling steps is presented in Figure 2.2.

Given some algorithm decomposition, the algorithm directed graph (ADG) is used to describe the primitive operation data dependencies. It uses circles (nodes) to represent primitive operations and line segments (edges) to describe the data dependence. Squares are used to indicate sources and sinks of data. An example ADG is provided in Figure 2.3.

The AMG is a marked graph representation of the ADG. The AMG provides a description of the algorithm data flow. The edges represent data containers and tokens are used to indicate the presence of data. The corresponding AMG of the algorithm decomposition of Figure 2.3 is presented in Figure 2.4.

Given some computing environment assumptions, the NMG specifies the functional unit activities which must occur in order to execute a primitive

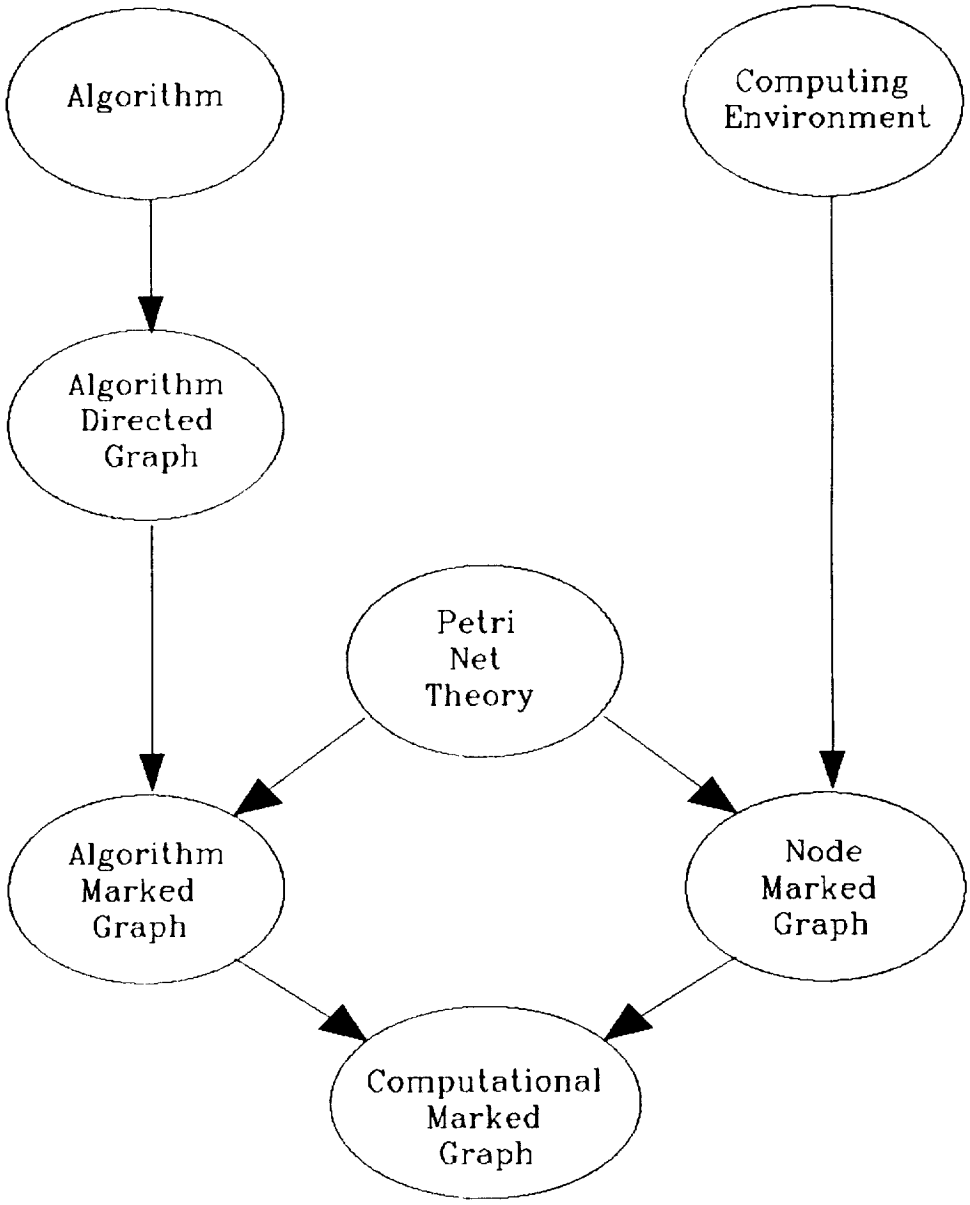


Figure 2.2. ATAMM Model Components.

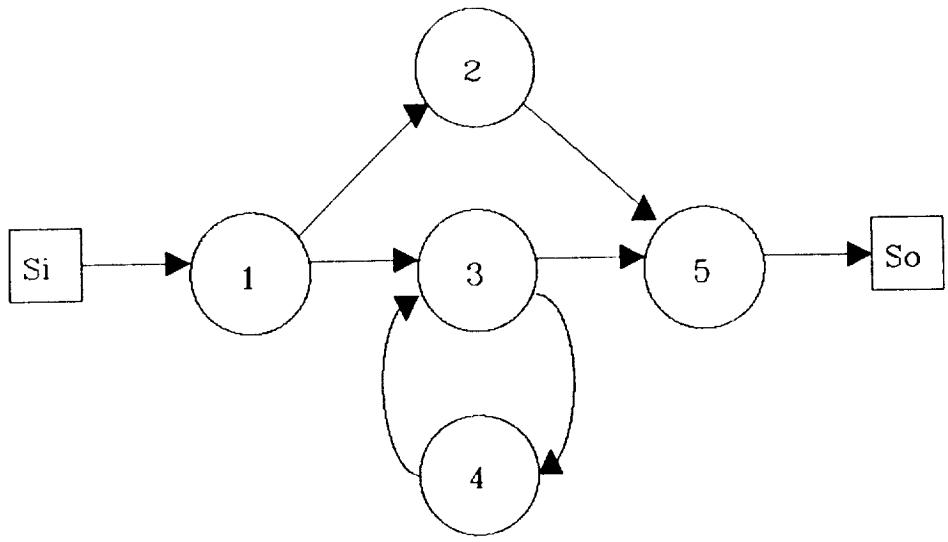


Figure 2.3. Example ADG.

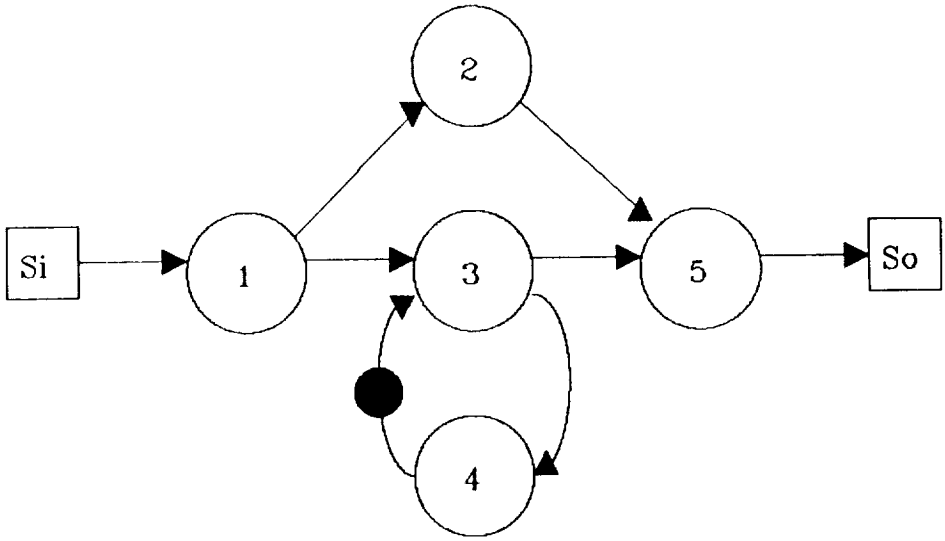


Figure 2.4. Example AMG.

node cannot be "fired" until the process is ready, input is available, and the output has been read by the successor operation. Also, a functional unit must be available to perform these computing activities. Once assigned to "fire" the read transition, the functional unit will remain assigned in order to process and write the data before becoming available once again.

The two modeling steps of ATAMM discussed so far have specified data flow with the AMG, and the functional unit activities and control flow required of each AMG node. The CMG is a marked graph which incorporates the AMG and NMG specifications into one graph. Thus, the CMG displays the data and control flow necessary to implement a decomposed algorithm on a multiprocessor data flow architecture [1]. The CMG is constructed by replacing each AMG node by the NMG. Source and sinks of the AMG are represented the same way in the CMG. Each AMG edge is replaced with one forward directed edge for data flow and one backward directed edge for control flow. The resulting CMG is presented in Figure 2.6.

The CMG of Figure 2.6 has certain characteristics that should be mentioned briefly. Execution of the CMG results in live, reachable, safe, deadlock free and consistent behavior. Liveness indicates that every transition of the graph can be fired from the initial marking [3]. Reachability implies that an output will be produced for every input. The CMG is safe because the backward control edges prevent data from being overwritten. The backward control edge will prevent enablement of a primitive operation until previous output data are

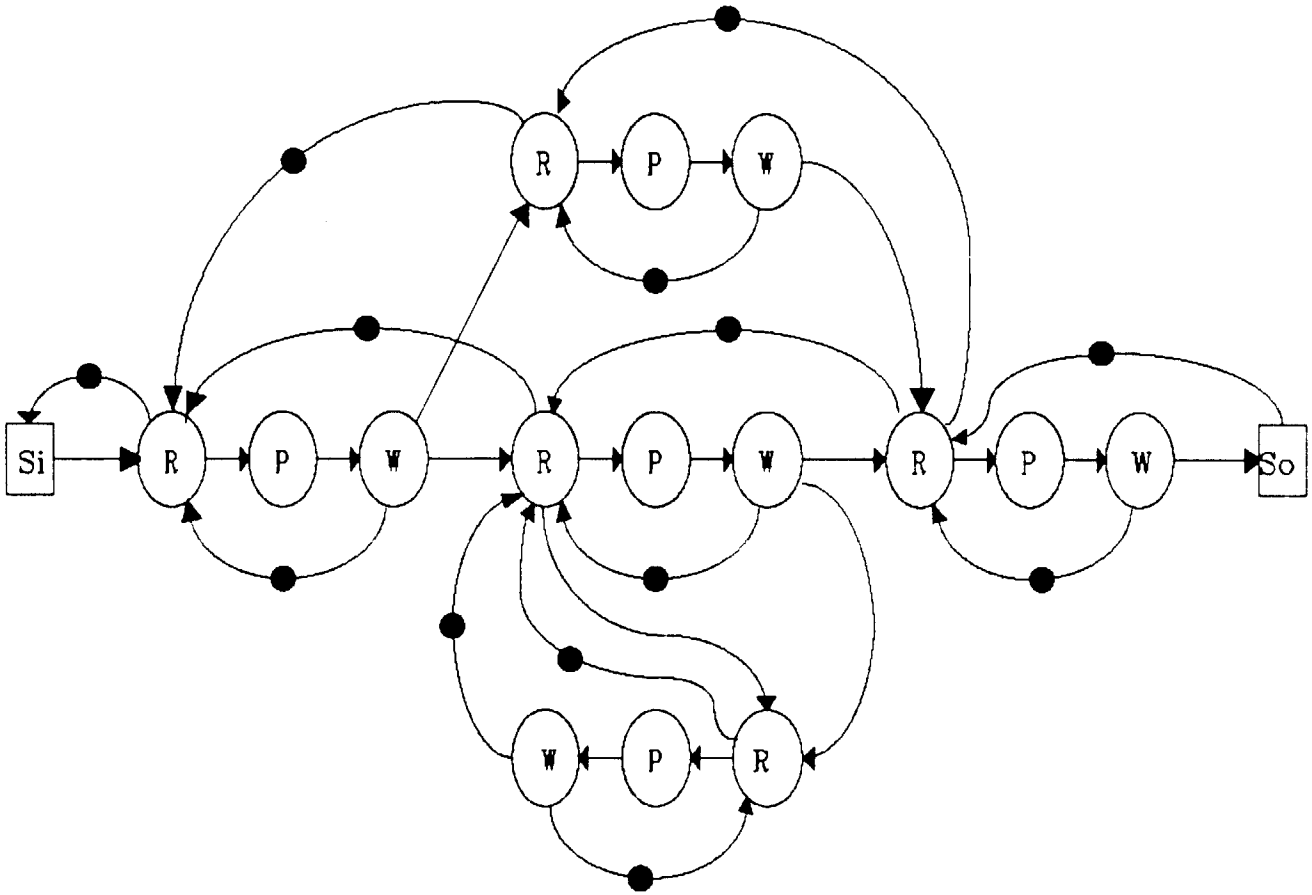


Figure 2.6. Example CMG.

picked up. The CMG is also deadlock free because once assigned to a primitive operation, a functional unit will always be able to complete execution. Consistency implies that the CMG will periodically produce output when input is applied periodically [3]. This also means that primitive operations will also be executed periodically.

There are two types of concurrency possible with executing an algorithm decomposition as specified by the CMG. Primitive operations belonging to the same data set which are independent of each other may be executed simultaneously. This is referred to as parallel concurrency and provides parallelism on a single data set [4]. The amount of parallel concurrency possible depends on the number of parallel paths in the algorithm decomposition and the number of functional units available. As with any data flow computer, new data sets will be accepted for execution before the completion of previous data set computations. This simultaneous processing of different data sets is referred to as pipeline concurrency [4]. The amount of pipeline concurrency possible depends on the ability of the algorithm decomposition to accept new data sets and the number of functional units available.

The AMG and CMG for a given algorithm decomposition can be used to calculate performance measurements. Two important performance measurements are the time between input and output (TBIO) and the time between outputs (TBO). TBIO is directly related to computing speed which indicates the amount

of parallel concurrency attained. TBO is associated with throughput and therefore reflects the amount of pipeline concurrency attained.

Lower bound values for TBIO and TBO can be calculated using the AMG and CMG. Lower bound ($TBIO_{LB}$) can be determined from the AMG by determining the longest path between the input source and the output sink. More formally, let P_i be the ith directed path in the AMG and $T(P_i)$ be the total path time associated with P_i. $TBIO_{LB}$ is then defined as

$$TBIO_{LB} = Max(T(P_i)) \tag{2.1}$$

where the maximum is taken over all paths in the AMG [3]. A proof of this theorem can be found in [5] and is based on critical path theory.

TBO_{LB} is a result of how quickly primitive operations can be repeated periodically. Let C_i be the ith directed circuit in the CMG and $T(C_i)$ denote the total path time associated with C_i. Also, let $M(C_i)$ denote the number of tokens contained in C_i. Then, TBO_{LB} is defined as

$$TBO_{LB} = Max(T(C_i)/M(C_i)) \tag{2.2}$$

where the maximum is taken over all circuits in the CMG [3]. TBO_{LB} is thus the largest time per token of all CMG circuits. The CMG circuits which determine

TBO_{LB} are called critical circuits. A proof of equation 2.2 can be found in [5] and is based on the maximum node firing rate of marked graphs.

Knowing TBO_{LB} is important because it determines the minimum injection interval of graph input. Data may temporarily be accepted within a time interval shorter than TBO_{LB} but at the cost of decreased computing speed (TBIO will increase). However, it is important in real-time applications to have high computing speed as well as high throughput. The ATAMM model provides the means to match the algorithm requirements with resource availability for optimum performance and establishes the criteria for predictable performance. Predictable performance is attained by maintaining an input injection rate within the range determined by ATAMM.

Implementing the ATAMM model requires three logical components. Two of the components are the functional units and global memory already discussed. A third component is needed to assign available functional units to primitive operations as they are enabled. This component, called the graph manager, is responsible for ensuring that the overall system operates according to the ATAMM rules. The graph manager examines the CMG for enabled nodes and assigns functional units (according to priority if more than one node is enabled) from a queue of available functional units. The graph manager uses status information communicated to it by the functional units to update the marking of the CMG. As with global memory, the graph manager can be centralized or distributed [1].

The integration of the graph manager with the hardware's operating system constitutes the ATAMM Multicomputer Operating System (AMOS). The resource queue, global memory, CMG, and the algorithm operations provide the necessary support to AMOS. An AMOS controlled architecture consisting of IBM PC's has been developed and tested to validate the ATAMM rules [6], [7]. A centralized graph manager and centralized global memory were utilized in this testbed. Another testbed, called the advanced development model (ADM), utilizing VHSIC technology 1750A processors is currently being developed [8]. The ADM system is composed of four functional units, utilizing a distributed graph manager and distributed global memory. There is also a 1553B module which performs the source and sink activities and provides an interface with an IBM AT. The IBM AT functions as a system monitor, provides the input data for the graph, and stores the output data. It also stores time-tagged status events generated by the functional units (discussed later) for analysis purposes.

One of the computing environment assumptions is that the same functional unit will complete all NMG activities. Therefore, the internal token marking at the "DP" edge is not important to the graph manager. Also, the "PR" edge provides only redundant information. A simplified NMG providing a description of the functional unit activities of AMOS is presented in Figure 2.7. When input is available, previous output read, and a functional unit is available, the "F" transition can be fired. This event marks the beginning of the primitive operation

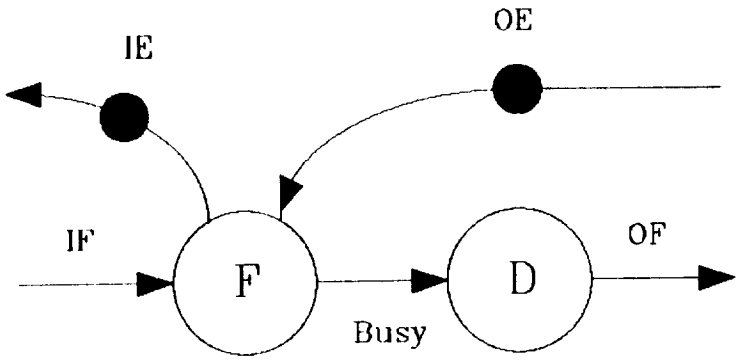


Figure 2.7. Reduced NMG.

execution. The completion of the execution is indicated by the firing of the "D" transition.

2.3 AMOS Communication Events

The state diagram description of the AMOS currently under development is shown in Figure 2.8. It is composed of the five states: idle, examine, execute, test, and update. A functional unit will initially start in the idle state. It will remain idle until it finds its identification number (I.D.) at the top of the resource queue (last in-last out) for available functional units. Upon finding its I.D., the functional unit will progress to the examine state where it will examine the CMG for enabled nodes. It will remain in the examine state until it locates an enabled node. Once a node is found, the functional unit will transition to the execute state and broadcast an "F" command to the other functional units indicating the firing of the node process. This broadcast, as well as the two others discussed next, provides the status information necessary for maintaining the data in global memory, the resource queue, and the status of the CMG. Since the graph manager may be distributed, this communication is especially important to ensure that all individual graph managers contain the same CMG marking. Having completed processing, the functional unit will write the output data to global memory and broadcast the "D" command and the node process output data to the other functional units.

Before returning to the idle state, the functional unit enters the test state where it may perform a self test. This state provides the means to remove a

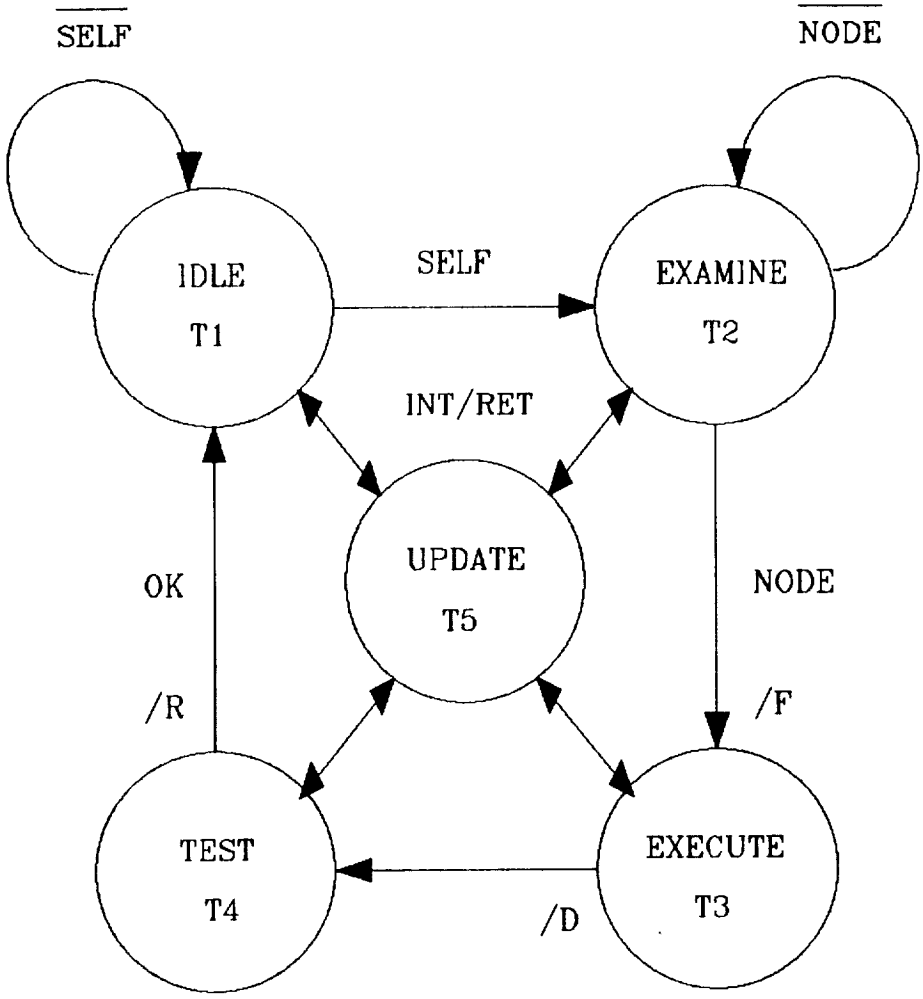


Figure 2.8. AMOS State Diagram.

functional unit from the system for inspection during real-time operation. If the functional unit checks out alright it will broadcast the "R" command announcing the return of its functional unit I.D. to the bottom of the resource queue and transition back to the idle state.

Since the operation of the system is asynchronous, the graph manager must generally be interrupt driven. When a broadcast is received, a functional unit will be interrupted from any of the states just mentioned and enter the update state. It will remain in this state long enough to update the CMG, global data, and the resource queue if necessary.

The "F", "D", and "R" commands not only provide the communication and synchronization necessary for correct overall system operation, but also the means to analyze the system performance. By time tagging and storing information about each broadcast, such as the event (F, D, or R), the node number, and functional unit I.D., the token movement within the CMG as well as functional unit utilization and concurrency may also be extracted. However, such an analysis requires the aid of a computer if it is to be done in a reasonable amount of time.

2.4 Overhead Model

The performance bounds discussed in Section 2.2 are ideal values. Read and write times are the only overhead parameters considered in the CMG. When one considers the AMOS state diagram which controls the functional unit activities, it is apparent that there are other activities besides reading, processing, and writing which must be considered.

Assume that the critical circuit of Figure 2.9 (a) determines TBO_{LB} for an algorithm containing at least two nodes and no recursion circuits. Within a TBO time interval all transitions within the critical circuit must be fired. There are needs associated with AMOS and the communication channel which must be met in order to fire these transitions. These needs require functional unit computing effort not associated with algorithm processing and therefore result in added overhead.

Let TB equal the time to broadcast an "F", "D" or "R" command. Let TG equal the time to grab the channel for the broadcast. It is assumed that it always requires some minimum amount of time to obtain a channel. TG may however be greater than this minimum value if there is contention with one or more other functional units. Referring to Figure 2.8, T2 is the time to examine the graph and T3 is the time to perform an "F" broadcast (TG + TB), execute the primitive operation (TE), and perform a "D" broadcast (TG + TB). All functional units will simultaneously transition to the update state for T5 amount of time following a command broadcast. It is assumed that there are sufficient functional units to fire nodes as soon as they become enabled.

Figure 2.9 (b) displays the time diagram associated with firing the nodes within the critical circuit. Starting with graph marking (1), the time to fire the critical node transitions and progress to graph marking (3) is equal to the summation of T2, T3, and 2T5. The time to fire the "F" transition of the successor node and return to graph marking (1) is the summation of T2, TG, TB,

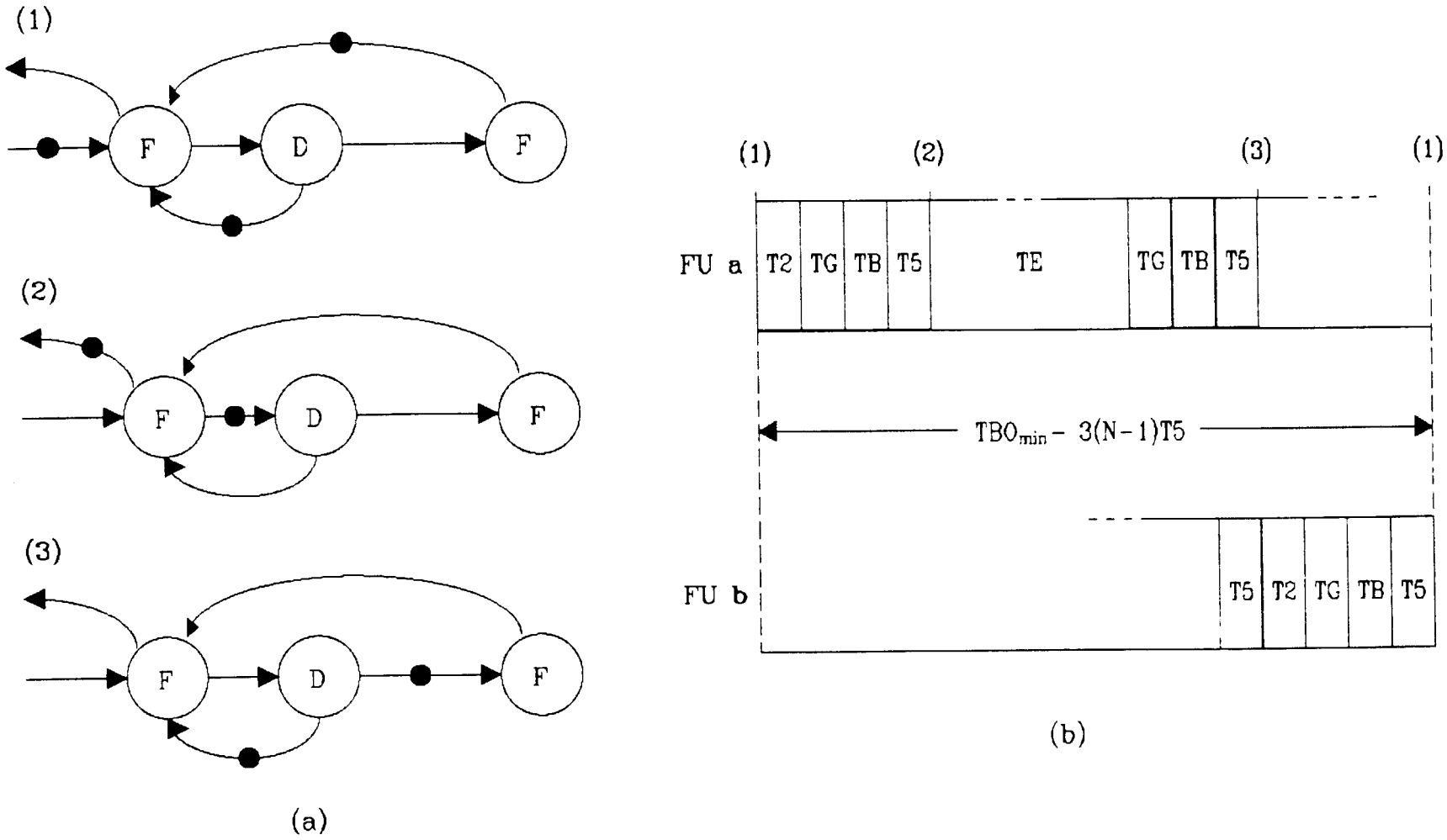


Figure 2.9. (a) Critical Circuit for Overhead Model
(b) Time Diagram for Critical Circuit.

and T5. Neglecting source and sink broadcasts, there will be 3(N-2) broadcast interrupts during the TBO interval that must also be considered where N is the number of AMG nodes. These interrupts occur from the "F", "D", and "R" broadcasts of the other N-2 AMG nodes. Also, there will be the interrupts resulting from the "R" broadcast of the functional unit previously assigned to the critical node and the "D" and "R" broadcasts of the successor node.

Therefore, the minimum theoretical TBO interval possible based on this model is

$$TBO_{min} = T2 + T3 + 2T5 + T2 + TG + TB + T5 + 3(N-2)T5 + 3T5$$

$$TBO_{min} = TE + 2T2 + 3(TG + TB) + 3NT5 \qquad (2.3)$$

This model assumes that there are sufficient resources to fire nodes as enabled. This assumption makes it possible to neglect test time (T4) in the model. If there are limited resources then TBO must be increased even higher. This is because the computing capacity (computer time available) must be greater than or equal to the computing effort (computer time used). Computing capacity for a time interval T is equal to R times T where R is the number of resources [3]. Total computing effort (TCE) is defined as the total amount of computing effort required to execute all AMG transitions one time [3]. Since all AMG transitions will fire within a TBO time period, the following condition must hold [3].

$$R * TBO \geq TCE \quad (2.4)$$

Functional unit test time increases TCE. Therefore, either R must increase as was assumed for the model above, or TBO must increase in order to satisfy equation 2.4. Both Equations 2.3 and 2.4 must be considered when predicting a theoretical lower bound for TBO.

2.5 Fault Tolerance

Many applications such as critical control systems require that output data be reliable. A fault tolerant system is usually created by providing processing and/or data path redundancy within the system. One such method is the use of Triple Modular Redundancy (TMR) for the detection and correction of single errors. The TMR approach implemented in the ADM system triplicates the processing and the data associated with each AMG node [8]. A primitive operation specified by a simplex AMG node (Section 2.2) is now specified by three AMG nodes with color extensions red, green, and blue. The three colored AMG nodes are enabled simultaneously. Execution of the colored AMG nodes is performed simultaneously by three functional units. Each colored node triplicates its output data for each colored successor node. These data are also color referenced. When all data are available as input to a successor node, a majority vote within the successor node execution determines the corrected data for processing. Data are also triplicated at the sink, so a majority vote routine must

be implemented by the sink as well. A description of the AMG transformation from simplex to TMR is shown in Figure 2.10.

A duplex method for the detection of single errors can also be implemented. Duplex is implemented the same way as TMR except that nodes, edges, and functional units are duplicated instead of triplicated. Nodes and edges are referenced by the colors red and green.

Since redundant AMG nodes are enabled simultaneously as well as executed simultaneously, a simplex description of a TMR or duplex system will still suffice for analysis purposes [8].

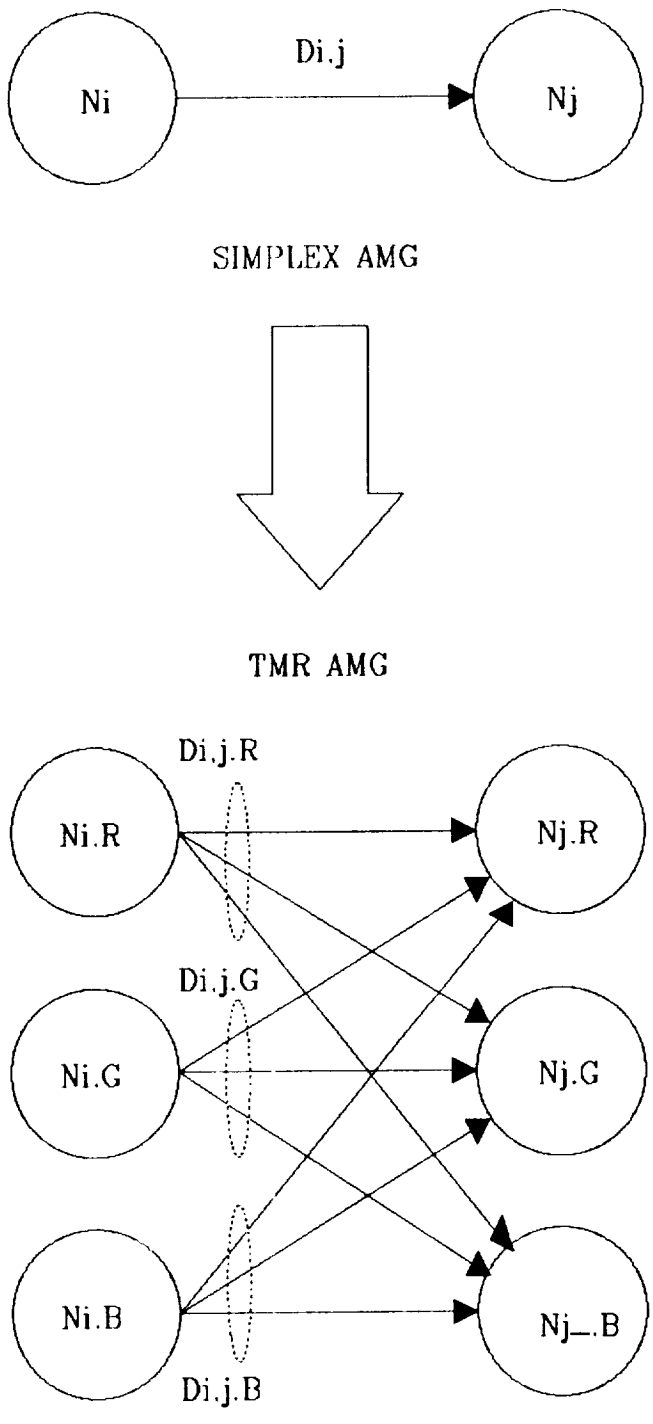


Figure 2.10. Simplex to TMR Transformation.

CHAPTER THREE

Analysis Tool Development

3.1 Introduction

The development of the Analysis Tool program is presented in this chapter. This program allows a performance evaluation to be made on concurrent processing systems based on the ATAMM model. Input to the Analysis Tool is a file containing information about the firing of each node through a list of time-tagged events. This file is generated by collecting the communication events of AMOS, discussed in the previous chapter, or by simulation. The format for this file, called the FDT file, is discussed in Section 3.2. Efficient processing of this information requires that the data be read into memory where it can be quickly accessed. The memory management implemented in storing and accessing data obtained from the FDT file is presented in Section 3.3. Presented in Section 3.4 are the two main activity displays. These displays allow the user to view the node and functional unit activities that occur during the execution of the graph. Measurements derived from the FDT file such as TBI, TBO, TBIO, utilization, and overhead are discussed in Sections 3.5, 3.6, and 3.7, respectively. The overall Analysis Tool

program structure and user-friendly features inherent in the window environment used to develop the program are presented in Section 3.9.

3.2 FDT File

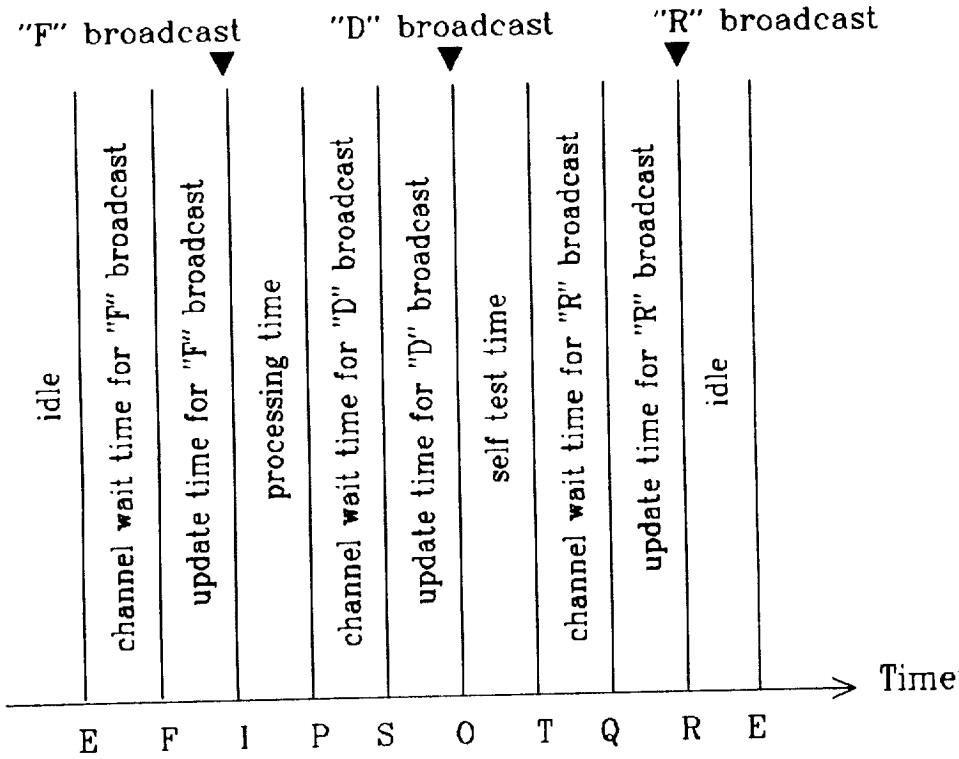
Evaluating the performance of a concurrent processing system based on the ATAMM model requires information concerning the state of each processor and the algorithm with respect to time. The broadcast of events as a processor progresses through the states of AMOS was discussed in Section 2.3. By knowing the graph structure, these events imply information about the movement of tokens within the CMG. Therefore, by recording these events along with the time of occurrence, processor and algorithm activity can be reconstructed.

The FDT (Fire, Data, Time) file contains a list of information pertaining to each AMOS broadcast event, in order of occurrence, which provides a means of evaluating the system performance and graph execution. Basic information in the FDT file includes the time occurrence of the event, name of the event, block number, node color, FU I.D., and the current mode (simplex, duplex, TMR) of the system.

The capability of evaluating overhead is made possible by adding information to each AMOS broadcast event. This information is the time spent waiting for a communication channel and the time spent updating the graph structure for the broadcast. The update time also includes the read and write time associated with processing a node when attached to the respective "F" and "D" broadcast events. The format for the FDT file is presented in Figure 3.1.

T, *time*, M, *mode*, *event*, N, *node*, C, *color*, *resource*

time : time of the event
mode : 1 → simplex; 2 → duplex; 3 → TMR
event : name of the event {E. F. I. P. S. O. T. Q. R}



node : block number
If the value of *node* is less than or equal to zero then the *event* is associated with a source or sink.
"P", "S", "O" *events* are associated with sources.
"E", "F", "I" *events* are associated with sinks.

color: color of the node within the block
1 → red; 2 → green; 3 → blue

resource: identification number of the functional unit processing the node or the device performing the sink or source activity.

Figure 3.1. FDT File Event Format.

The Analysis Tool also provides the option of including information concerning voting errors, fatal errors or increase or decrease of available functional units. When a voting error is detected, a voting error report is included after the "O" event information in the FDT file so the Analysis Tool can single out the node, node color, and functional unit involved in the error. A change-in-resource report is included after an "R" event in order to allow the "available" number of resources within the program data structure to be adjusted in response to the change. The error information formats for voting errors and change-in-resource are presented in Figures 3.2 and 3.3, respectively.

3.3 Data Storage and Access

Efficient processing of information available in the FDT file requires that the data be brought in from the file and stored in memory. Depending on the number of events recorded, this may require considerable memory so efficient use and access of this memory is needed.

Information gathered from the FDT file is stored in a "C" data structure array, called "all", with the following elements.

```
all[ ] = { the_time,
           the_event,
           node,
           color,
           mode,
           resource,
           available,
           working }
```

D, *fatal* , # , *count*, N, *node* , C, *color* , *FUid* , ...

repeated *count* number of times

fatal : 0 → no fatal errors
1 → fatal error

count : the number of voting errors listed

node : the block number that was in error

code : 0 all three colored nodes were in error
1 red node was in error
2 green node was in error
3 blue node was in error

FUid : identification number of the functional unit processing the colored node that was in error

Figure 3.2. FDT File Voting Error Information.

R, *action* , # , *changes*, *FUid* , C, *code* , ...

repeated *changes* number of times

action : action pertaining to first FUid and code in the list
] ⟶ FU installed; R ⟶ FU removed

changes : the number of changes listed

FUid : identification number of the functional unit

code : 0 ⟶ died in queue
1 ⟶ died in process
2 ⟶ died in self test
3 ⟶ told to remove
4 ⟶ told to install
5 ⟶ initial installation

Figure 3.3. FDT File Change-in-Resource Information.

The first six elements of the data structure are obtained directly from the file and stored. "Available" and "working" are elements derived from the data as they are read into memory. "Available" contains the total number of resources available to the graph and is updated based on the change-in-resource report. As for the last element, by incrementing a counter by one with the occurrence of a ON_HOLD_READING event and decrementing by one with the occurrence of an IDLE event, the number of "working" functional units can be determined and stored. This makes it very easy to display the resource envelope and calculate resource utilization discussed later in Section 3.6.

Since one would want to use as little memory as possible, the memory required for this structure is allocated dynamically depending on how much is needed. Instead of having to scan the file twice, once for the number of events and then again for processing and storage, the Analysis Tool requires that the number of events listed in the file be included as the first line in the file. Refer to the appendix for the FDT file header format.

Due to the potentially large amount of information, locating data concerning only a particular node or functional unit can be time consuming. A solution to this problem implemented by the Analysis Tool is the use of link lists. The link list is a collection of indices within the data structure array connected together by head and tail pointers which allows the list to grow dynamically in memory as needed. Indices are used instead of actual address pointers because the link lists are stored in local memory and the data structure is stored in the

global heap which would require the use of far pointers (segment and offset). A link list for one particular functional unit, for example, pointing to entries within the data structure involving the functional unit, allows quick extraction of its activity from the entire structure. See Figure 3.4 for an example. By having a link list for each red, green, and blue node in a TMR block (Section 2.5) as well as each functional unit, the node and FU activities can be easily displayed.

3.4 Node and Functional Unit Activity Displays

Playing back the activities involved in executing an algorithm is important in the performance evaluation of a concurrent processing system. A graphical display of both node and functional unit activities that took place allows one to view a large amount of information brought together in one picture. By having a time axis with cursors capable of measuring time along the axis, transition times such as read, process, and write can be measured for any node or functional unit activity.

3.4.1 Activity Blocks

The Analysis Tool presents both node and functional unit activities by painted rectangles which will be referred to as activity blocks. Depending on the display type, the blocks are painted a color representing either the functional unit identification number, node number, or a transition such as processing. The length of the block along the time axis represents the amount of time spent on the activity.

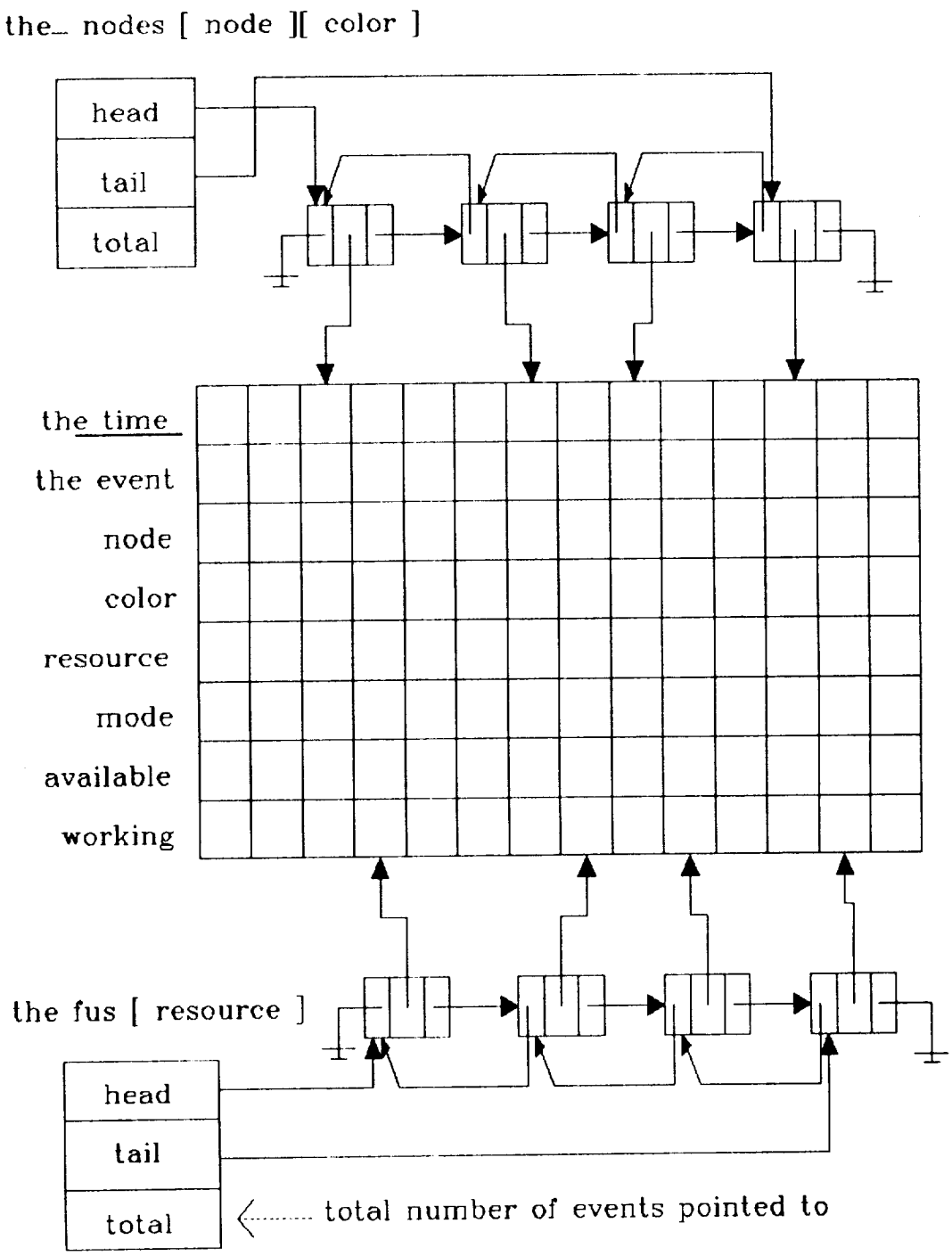


Figure 3.4. Link List Data Access.

The node activity display, called Graph Play, and the functional unit activity display, called FU Activity, have distinctive activity block formats. Graph Play activity blocks depict TMR, duplex, or simplex behavior with overlapped blocks having three different sizes. The smallest activity block displays the red node, the second smallest displays the green node, and the largest displays the blue node activity of a TMR block. When operating in duplex, only red and green node activity blocks will be present. Simplex mode behavior is displayed by red node activity blocks only.

Activity blocks of the FU Activity display depict TMR, duplex, or simplex behavior by including a colored band on top of the blocks. A red, green, or blue band determines the respective nodes within a TMR block with which the activity block is associated. As with the Graph Play format, duplex behavior is displayed with activity blocks having red and green bands and simplex behavior is displayed with blocks having red bands only. The portion of the activity block with the colored band on top is associated with node processing activities. The remaining portion of the activity block without the colored band is associated with functional unit testing.

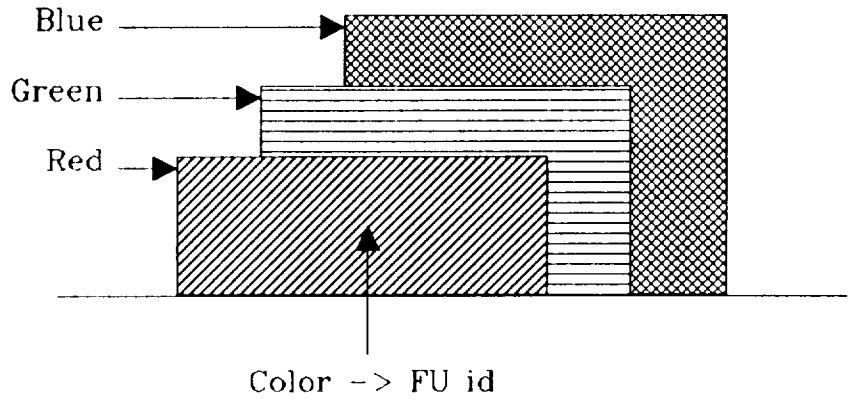
The Graph Play display can provide a view of single or total graph play. Total graph play refers to the node activity of all nodes processing all data packets whereas single graph play refers to the node activity associated with only one user selected data packet. A default data packet number equal to one is used when the Single Graph Play is initially displayed.

A Graph Play activity block without a view of the individual transitions is shown in Figure 3.5 (a) and with the transitions in Figure 3.5 (b). Shown in Figures 3.6 (a) and (b) are the activity blocks for the FU Activity without the individual transitions and with the transitions, respectively.

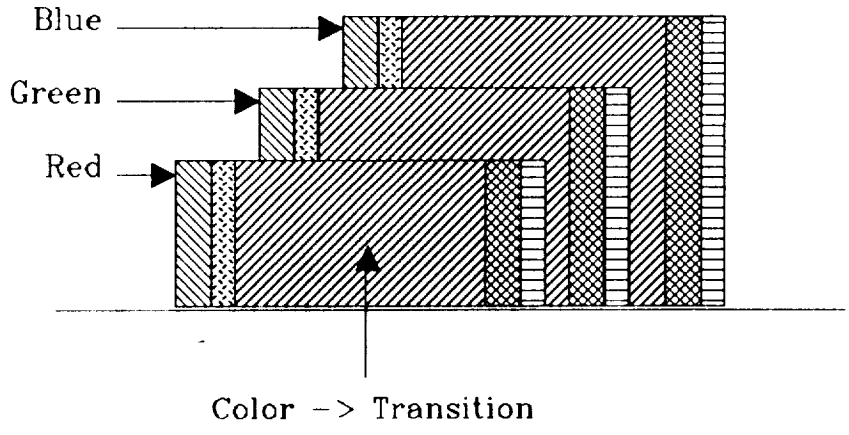
3.4.2 User-Interactive Measurement of TBO and Throughput

The cursors in both the Graph Play and FU Activity displays allow time measurements to be made as was mentioned in the previous section. For example, one could move the cursors to the end of two consecutive write activities of the output node and measure a value for time between output (TBO). However, this measured TBO value may not be identical to the measured TBO values presented in Section 3.5. The potential discrepancy exists in the rule that governs the TBO measurement in Section 3.5 where the measurement is with respect to sink activity instead of the output node activity.

If one is interested in throughput, an average TBO is needed. Throughput is defined as the reciprocal of the average TBO. It is possible in both displays to define an evaluation interval and obtain a measurement of average TBO for that interval of time thereby allowing a calculation of throughput to be made. The measurement is an arithmetic mean of the TBO values measured over the evaluation interval for each data packet based on the definition of measured TBO in Section 3.5. A default sink number equal to the number of the sink providing the first activity in the FDT file is initially used to obtain the measurement; however, the sink number can also be defined by the user. Refer to Figure 3.7

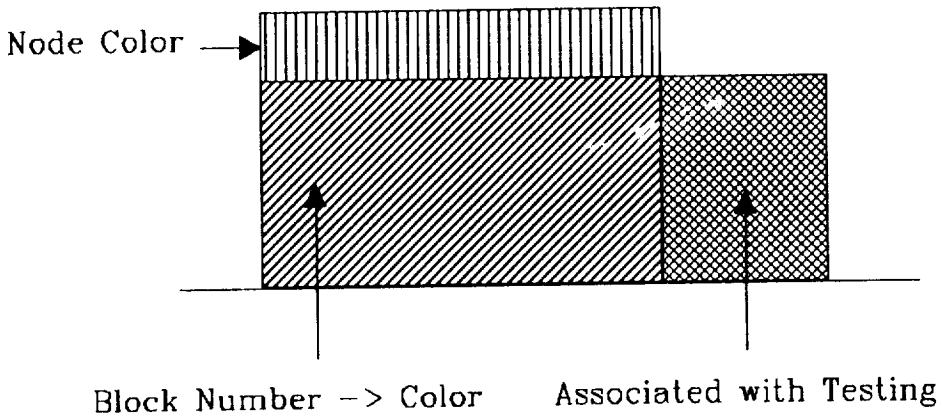


(a)

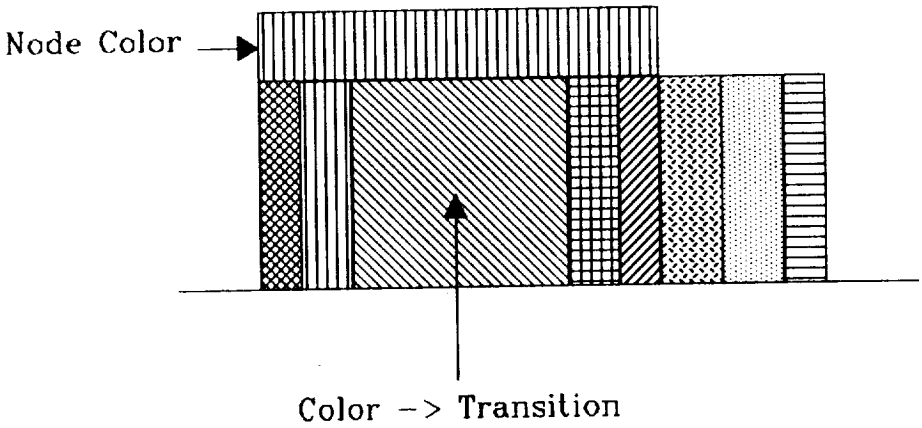


(b)

Figure 3.5. Graph Play Activity Block
(a) Overall Activity Display
(b) Display of Internal Transitions.



(a)



(b)

Figure 3.6. FU Activity Block
(a) Overall Activity Display
(b) Display of Internal Transitions.

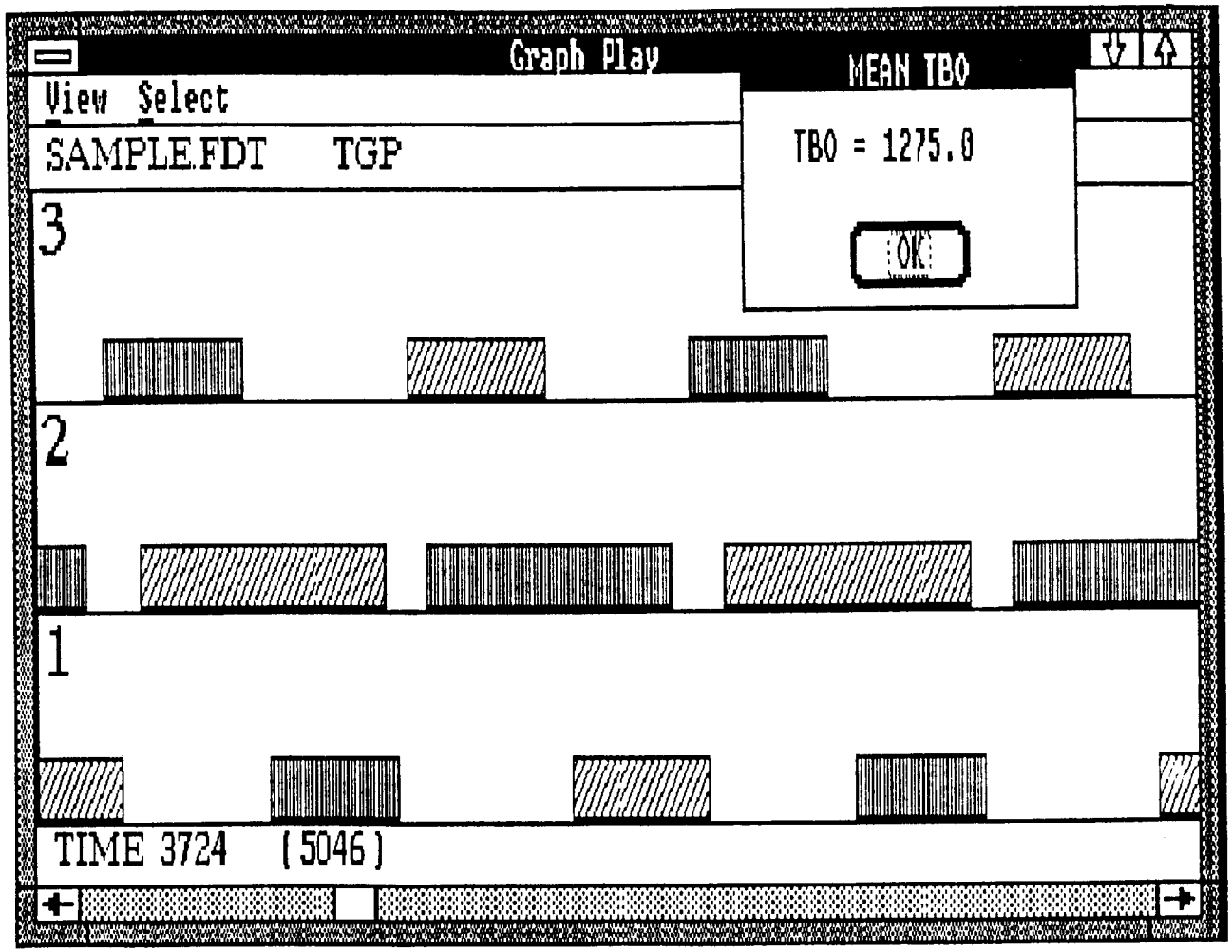


Figure 3.7. Sample Graph Play Display.

for an example of the Graph Play display showing the measurement of mean TBO.

3.4.3 Quick Access List of Data

There are two ways to obtain measurements of time spent performing certain activities associated with nodes or functional units. One way is the use of the cursors mentioned in the previous sections. However, both displays also provide another means of obtaining time measurements along with other identification information. By pointing the cursor to an activity block of interest and performing a key sequence discussed in the appendix, a list of data pertinent to that activity block will be displayed for quick and easy viewing. The data include the node number, color, FU I.D., current mode, data packet number being processed, and the transition times such as waiting, updating, and processing represented by the activity block. Refer to Figure 3.8 for an example of the FU Activity display showing a Quick Access List of information.

3.5 Automatic Measurement of TBI, TBO, and TBIO

Important to any algorithm and system analysis are the measurements associated with computing performance. The Analysis Tool provides time performance measurements of TBI, TBO, and TBIO (Section 2.2) for every data packet that is processed by a predefined output sink.

TBI, TBO, and TBIO are shown within the Performance display along with the corresponding data packet number and the current number of resources available at the time the data packet was consumed by the sink. An example of

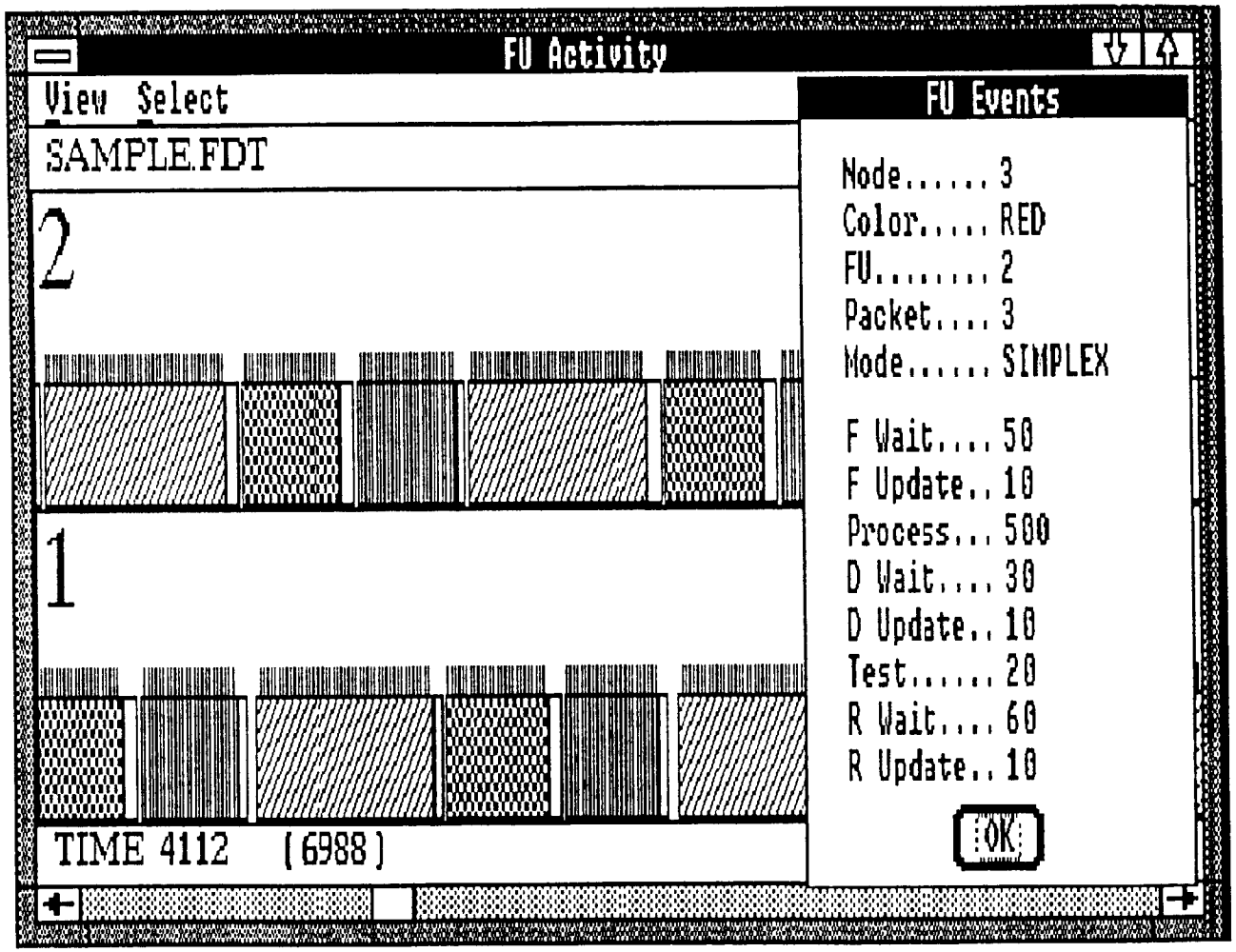


Figure 3.8. Sample FU Activity Display.

the Performance display is shown in Figure 3.9. These time measurements are made with respect to source and/or sink activities by measuring the time between certain source and/or sink events obtained from the FDT file. Rules governing the measurement of TBI, TBO, and TBIO are as follows:

Measured TBI

1. Mark the time the last "O" event associated with the defined source number occurs for some data packet number.
2. Repeat step 1 except with the next data packet.
3. Subtract the time of step 1 from the time of step 2 to obtain the measured TBI between the two data packets.

Measured TBO

1. Mark the time the last "I" event associated with the defined sink number occurs for some data packet number.
2. Repeat step 1 except with the next data packet.
3. Subtract the time of step 1 from the time of step 2 to obtain the measured TBO between the two data packets.

Measured TBIO

1. Mark the time the last "O" event associated with the defined source number occurs for some data packet number.
2. Mark the time the last "I" event associated with the defined sink number occurs for the same data packet number.

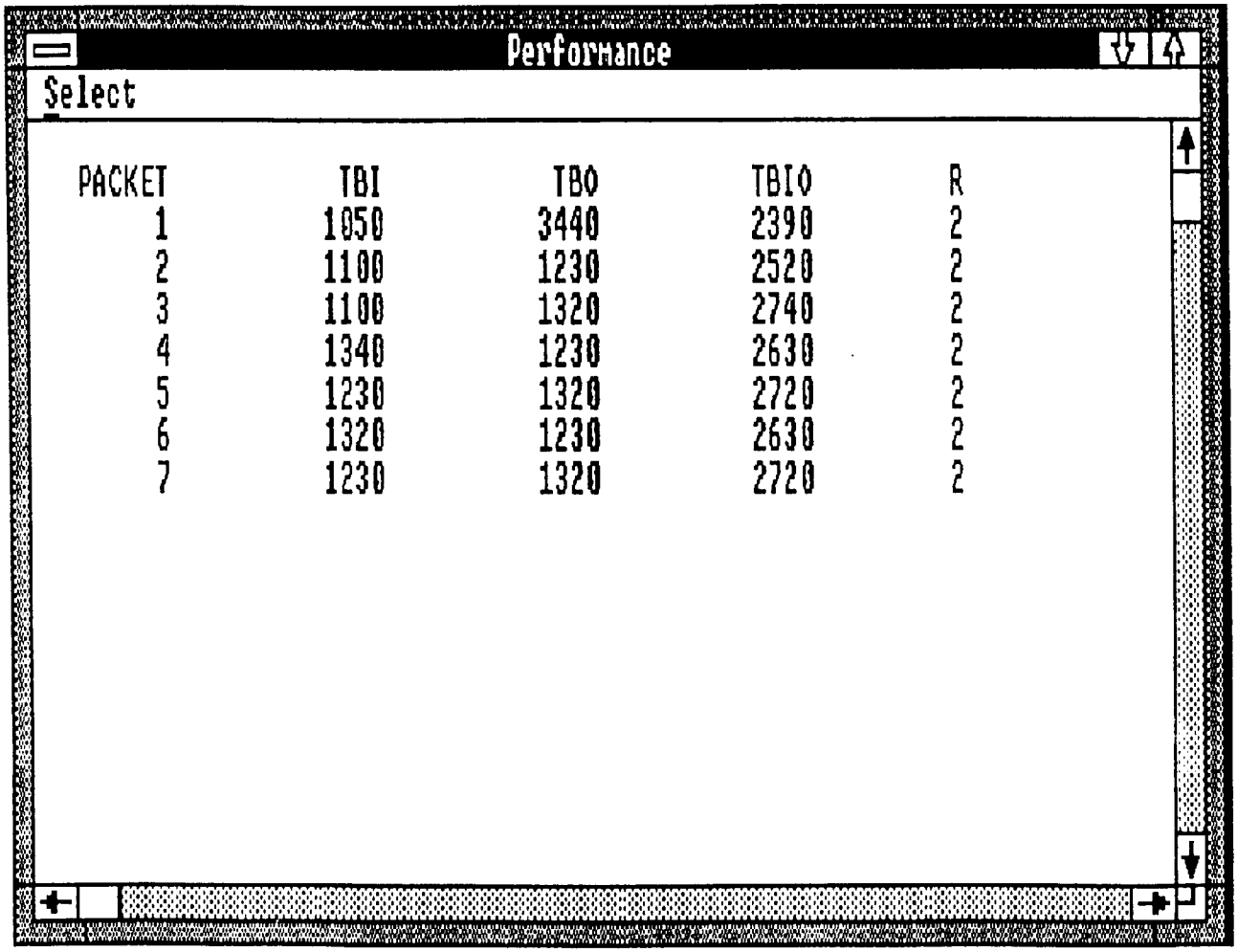


Figure 3.9. Sample Performance Display.

3. Subtract the time of step 1 from the time of step 2 to obtain the measured TBIO for the data packet.

A plot of TBO verses TBIO may be observed from the Operating Point display (Figure 3.10) within the Performance display.

Just as the activities of functional units and nodes are organized in link lists for quick access, the same organization is provided for all sources and sinks. Consequently, arriving at these time measurements is made efficient since there is no need to scan the entire data structure array for the source and sink events.

Since there may be more than one source or sink in the graph, the source or sink number can be selected by the user. The default source and sink numbers are the numbers associated to the first source and sink activities encountered in the FDT file.

3.6 Concurrency Evaluation

The Analysis Tool provides a means of evaluating the concurrent functional unit activity. This evaluation is accomplished through the Resource Envelope display where the number of resources working in parallel at any time can be easily established.

A resource envelope is a graph depicting the number of resources active verses time. A resource is active when it is waiting for a channel, updating, processing, or testing. This graph provides a visual evaluation of the achieved concurrency and resource utilization. The Resource Envelope display provides a

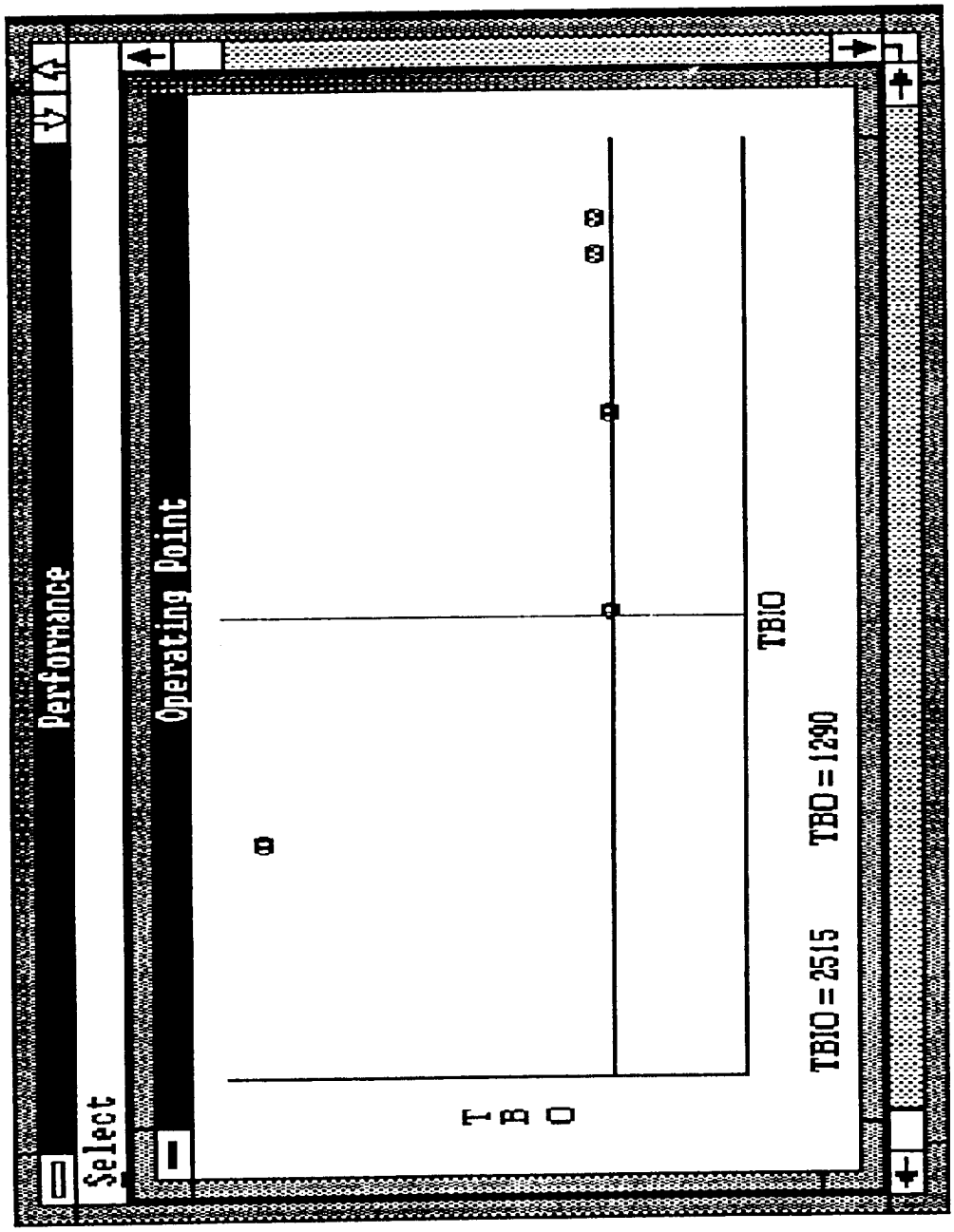


Figure 3.10. Sample Operating Point Display.

average value of examine time can then be used to derive an approximate value for actual idle time when the system is run on the same graph but with two or more resources.

3.8 Error Reporting

Reports of voting errors and changes-in-resource in the FDT file was discussed in Section 3.2. Including these reports permits the Analysis Tool to recognize faults and system changes that may otherwise go undetected and may even invalidate certain measurement calculations.

There is no automatic notification of voting errors that may occur during normal observation of the node and functional unit activities. The Analysis Tool uses the voting error reports to set flags in its data structure to later identify the errors, if desired, in the Graph Play and FU Activity displays. If a report indicates any voting errors while reading in the FDT file, a routine will be invoked which will search back through the data structure array until the node-functional unit combination identified in the report is found for each error. A flag is set by placing a negative sign in the "available" field for the TESTING event (Figure 3.1). When the Graph Play or FU Activity displays are in the error reporting mode and detect a negative "available" value, an "X" is placed next to the activity block to identify the node and functional unit responsible for the voting error. This type of error report functions as a voting error histogram for functional units when viewing the FU Activity display. Since the "X" marks will

accumulate horizontally for each functional unit, the total number of voting errors for each functional unit can be easily observed and counted.

Errors resulting from resources dying in the queue, or processing, or testing can be inherently detected and displayed since a resource's extended stay in any one of these states can be viewed from the FU Activity display. However, the total number of resources available in the system is assumed to remain constant unless the program is told otherwise. Since the Analysis Tool performs calculations based on the total number of resources, a knowledge of when there is a change in this number is necessary if the calculations are to remain valid. When the Analysis Tool is reading the FDT file and storing data, it will adjust the value stored in the "available" field upon encountering a change-in-resource report thereby allowing the calculations based on this value to remain valid.

3.9 Window Environment Features

The Analysis Tool program was developed to be run in a window environment for greater flexibility and user-friendliness. Written in the Microsoft Quick C language, the Analysis Tool also uses Microsoft Windows as its window environment. Since the window program structure is not unlike an object-oriented language structure, the complexity of such a large project as this was greatly reduced once this new programming concept was understood.

Window environment features include the capability to run more than one application in parallel permitting the user to run more than one copy of the Analysis Tool at the same time, thereby providing a means to analyze and

compare two or more FDT files simultaneously. As another example, the Analysis Tool and a simulation program could be running concurrently allowing an easier transition between them. Keeping with this parallelism, all of the Analysis Tool's displays are managed by their own independent window which allows the displays to be viewed at the same time. This provides an analysis capability that would otherwise be lost if it were only possible to view one display at a time.

Due to Microsoft Window's easy management of menus, the Analysis Tool is entirely menu driven within each window. Most of the program-user interaction is through dialogue boxes and mouse I/O so only slight use of the keyboard is required.

CHAPTER FOUR

Experimental Results

4.1 Introduction

A demonstration of the Analysis Tool as a productive means to diagnose ATAMM based systems from their behavior is presented in this chapter. A six node graph is used for the simulation and analysis. First, the Analysis Tool is used to infer information about the overhead that will alter the expected performance. This information is then used to obtain theoretical time performance values from the ideal values. Lastly, the Analysis Tool is used to measure performance of the system at two theoretical operating points for comparison with the expected performance.

The graph used for the demonstration is presented in Section 4.2 along with the corresponding ideal performance estimates. Experimental results for the overhead evaluation are presented in Section 4.3 using several features of the Analysis Tool. The resulting operation at the theoretical lower bounds is demonstrated in Section 4.4 using the remaining Analysis Tool features. A demonstration of the system behavior with reduced resources is presented in Section 4.5. Error reporting is demonstrated in Section 4.6.

4.2 Graph Description and Performance Estimates

The graph used to demonstrate the capabilities of the Analysis Tool contains six nodes and parallel paths. The AMG for the graph is presented in Figure 4.1.

The ideal lower bounds on performance can be calculated using the AMG. Since the graph is without a recursion circuit, the ideal TBO_{LB}, assuming zero read time, is the largest node time (Section 2.2). There are two nodes that have a transition time of 200 ms. which establish a TBO_{LB} of 200 ms. The critical path time of 400 ms. sets $TBIO_{LB}$ equal to 400 ms. Note that these lower bound values are ideal in this case since read, write, and other overhead times have not been considered. Predicted single graph play and total graph play for lower bound operation are presented in Figure 4.2 (a) and (b), respectively. Single graph play shows the node processing and concurrency involved with processing one data packet. Total graph play displays the concurrency that occurs in a TBO interval during steady state operation. The numbers enclosed in parentheses indicate the data packet sequence. Number one indicates processing associated with the present data packet; whereas, number two indicates processing associated with the next data packet. The single resource envelope and total resource envelope drawings are included in Figure 4.3 (a) and (b), respectively. These envelopes determine the resources required by the respective graph plays of Figure 4.2. Equation 3.1 with a computing power equal to 700 ms, TBO equal to 200 ms, and four resources, yields an expected resource utilization of 87.5 percent.

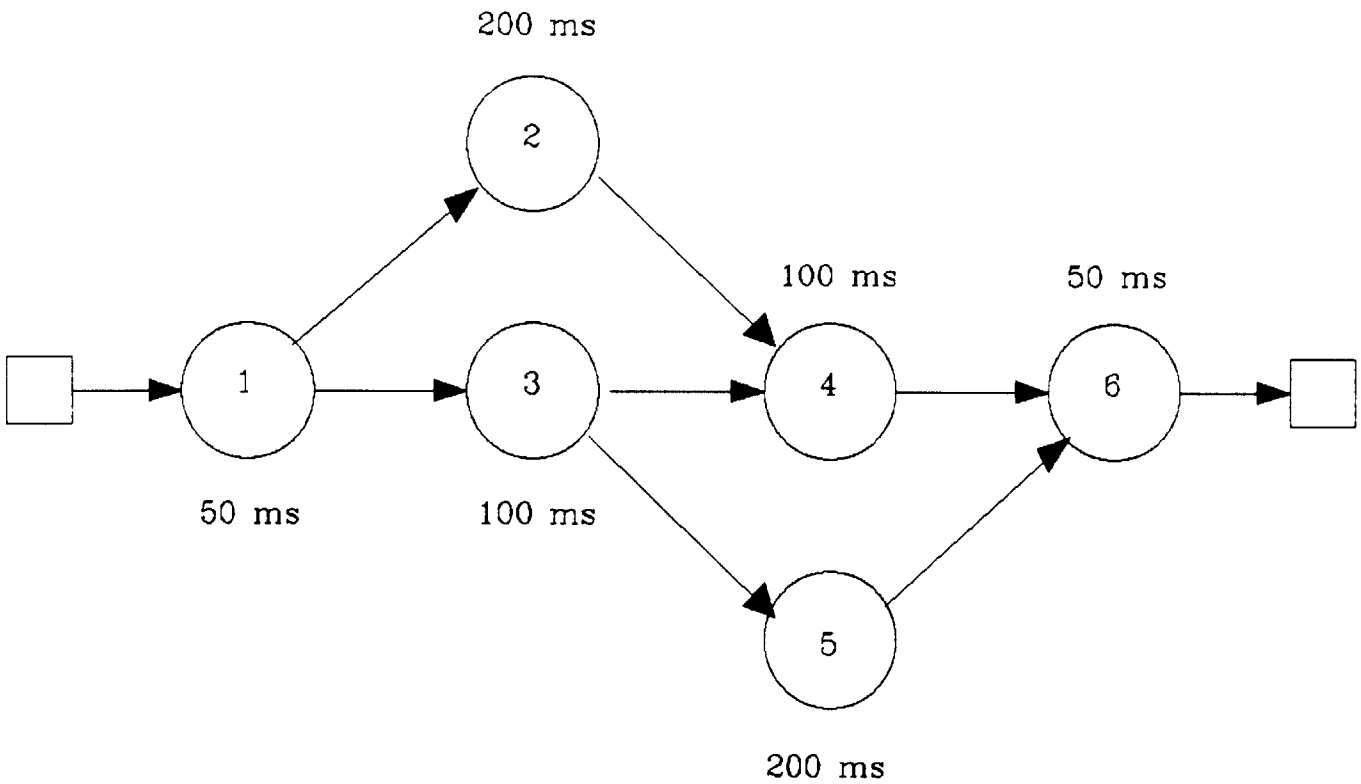
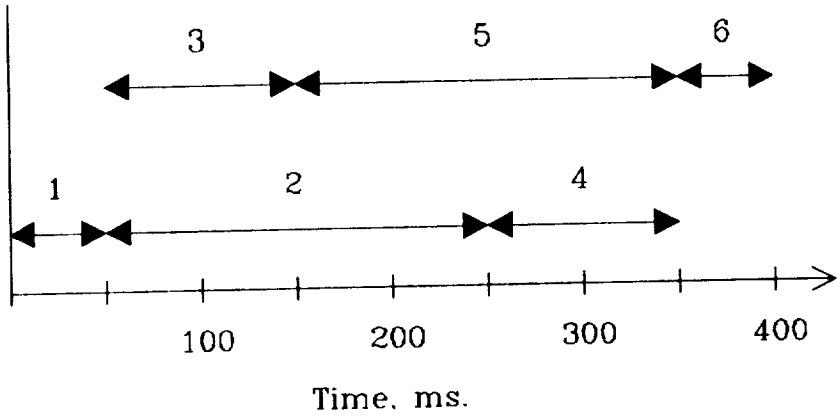
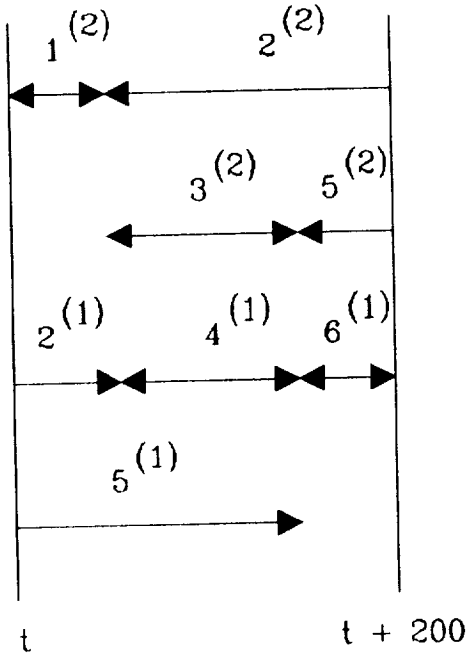


Figure 4.1. AMG of Test Graph.

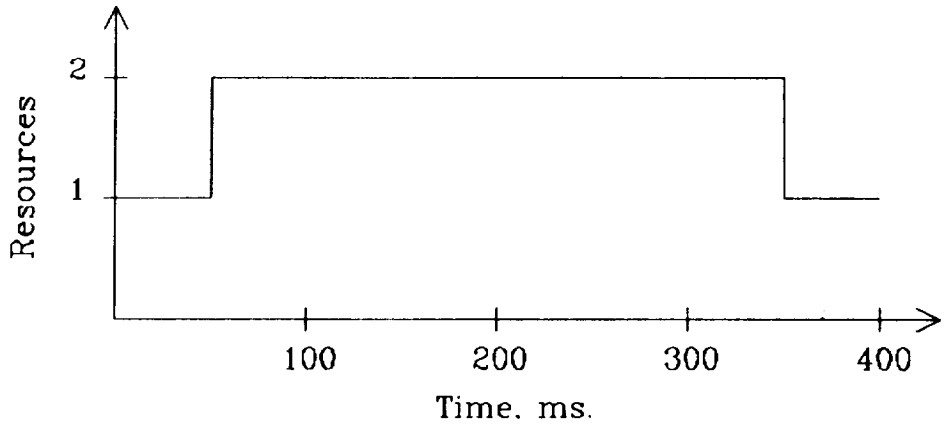


(a)

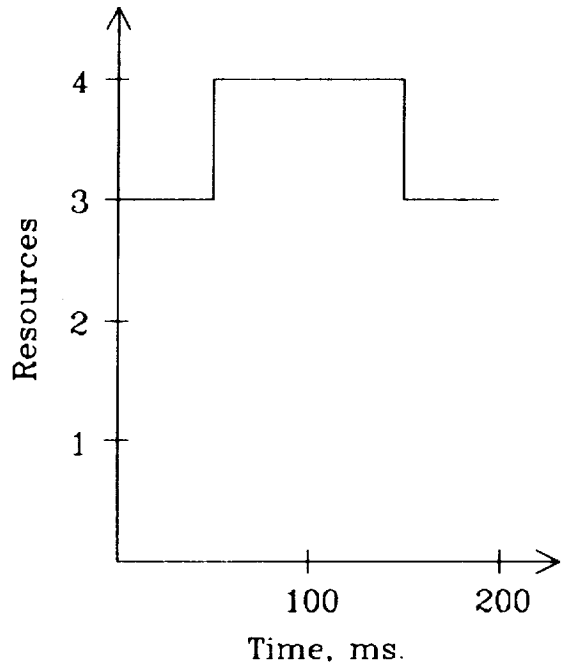


(b)

Figure 4.2. (a) Single Graph Play. (b) Total Graph Play.



(a)



(b)

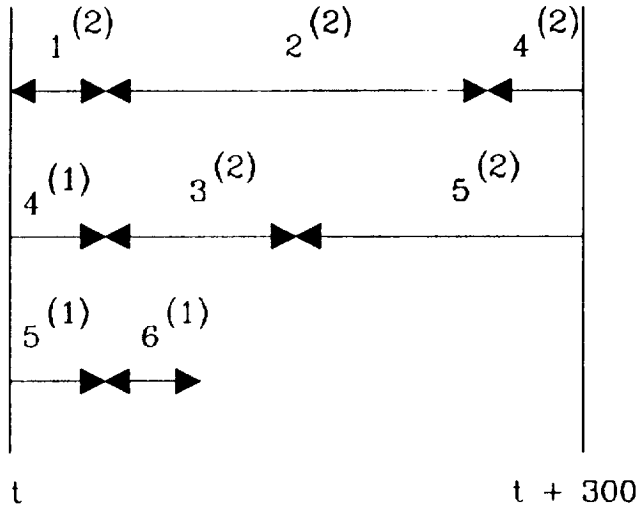
Figure 4.3. (a) Single Resource Envelope
(b) Total Resource Envelope.

Ideally, simplex operation at these lower bounds is possible with four or more resources. If, however, the system has less than four resources, predictable performance is still possible. The ATAMM model also predicts that the graph can be executed with three resources (in simplex) at the same $TBIO_{LB}$ value but with TBO equal to 300 ms. The method used to obtain this operating point will not be discussed since it is beyond the scope of this thesis. Again, using Equation 3.1 with TBO equal to 300 ms. and three resources, the expected resource utilization is equal to 77.8 percent. The total graph play and total resource envelope for this operating point are presented in Figure 4.4 (a) and (b), respectively.

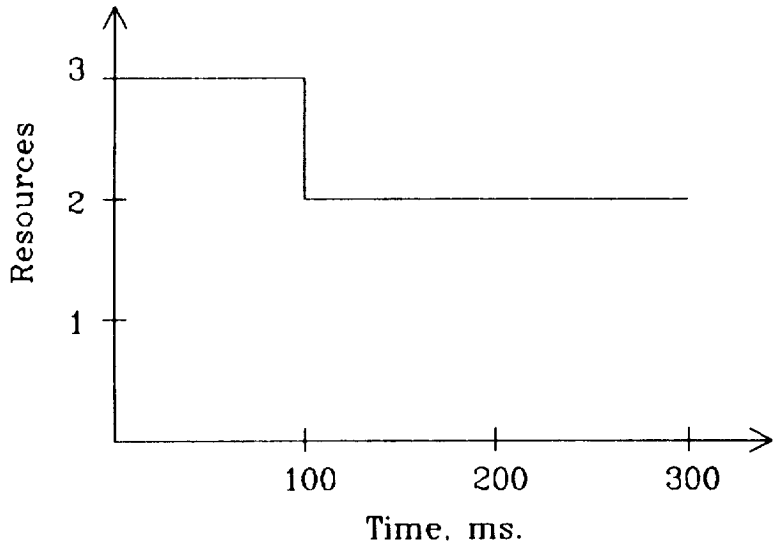
4.3 Overhead Evaluation Results

The performance estimates stated in the previous section are based on the ideal case where overhead is not considered. Since the real world is not ideal, an estimated overhead requirement is needed in order to predict theoretical performance. A demonstration of the overhead evaluation features of the Analysis Tool is presented in this section using an ATAMM based simulator in TMR mode with assumed overhead parameters. The evaluation will utilize the FU Activity, Performance, and Overhead displays of the Analysis Tool.

The overhead model discussed in Section 2.4 included all activities required to execute the critical circuit of a graph in a TBO interval. It can be used to estimate a worst case overhead requirement in order to establish a minimum injection interval for predictable behavior. The model is useful to



(a)



(b)

Figure 4.4. (a) TGP, (b) TRE for TBO = 300.

establish an initial injection interval until the actual system is run and analyzed. The Analysis Tool can then provide an overhead estimate that is more exact, thereby giving support to a more realistic design.

The evaluation of the simulated system was done by performing four test simulations with twelve resources (due to TMR) at different injection intervals. Each system simulation assumed the following overhead parameters:

Channel Grab Time = 200 microseconds

Update of Graph and Data = 100 microseconds

Self Test Time = 100 microseconds

Each functional unit will experience a wait time of at least the channel grab time when attempting to broadcast. Even though these base times are known in advance, the total wait time that results and how self testing effects resource availability are not known. The simulator also assumes that there are three independent devices performing the source and sink activities (one for each red, green, and blue TMR edge). These devices also have the same channel grab time and update time as the functional units. Therefore, the resulting TBI will differ from the specified injection interval due to this added overhead.

The first test involved simulating the system with a zero injection interval. The inherent nature of the data flow architecture will cause input to be taken as fast as the edges and resource availability will permit at the expense of low

computing speed. This determines the minimum TBO that is possible as a result of the system overhead and available resources. The results of this test are presented in the FU Activity, Performance, and Overhead displays in Figures 4.5, 4.6 and 4.7, respectively. All of the displayed times have a ten microsecond resolution. The unpredictable behavior of functional unit activity for this constant supply of input can be observed in Figure 4.5. It is observed from Figure 4.6 that TBO reaches a steady state value of 204.9 ms. with TBIO reaching a steady state value much greater than the lower bound due to the low computing speed, as expected. A time interval for the overhead calculations was defined over four TBO intervals using the FU Activity display. The resulting measurements are presented in the Overhead display of Figure 4.7. Summing the wait, update, and test percentages shown determines that the overall overhead for the evaluation interval is 2.2 percent. Adding this value to the 85.4 percent for processing determines that the resource utilization is equal to 87.6 percent.

The second test involved injecting at the ideal TBO value of 200 ms. Since this is below the minimum TBO due to overhead, TBIO should still be above its minimum value. Referring to the Performance display of Figure 4.8, it is observed that TBO reaches a steady state value of 204.9 ms. which is presumed to be close to the actual TBO_{LB}. An evaluation interval for the Overhead display is again defined as in the previous test. The results of the Overhead display in Figure 4.9 indicates that overhead and resource utilization have not changed appreciably.

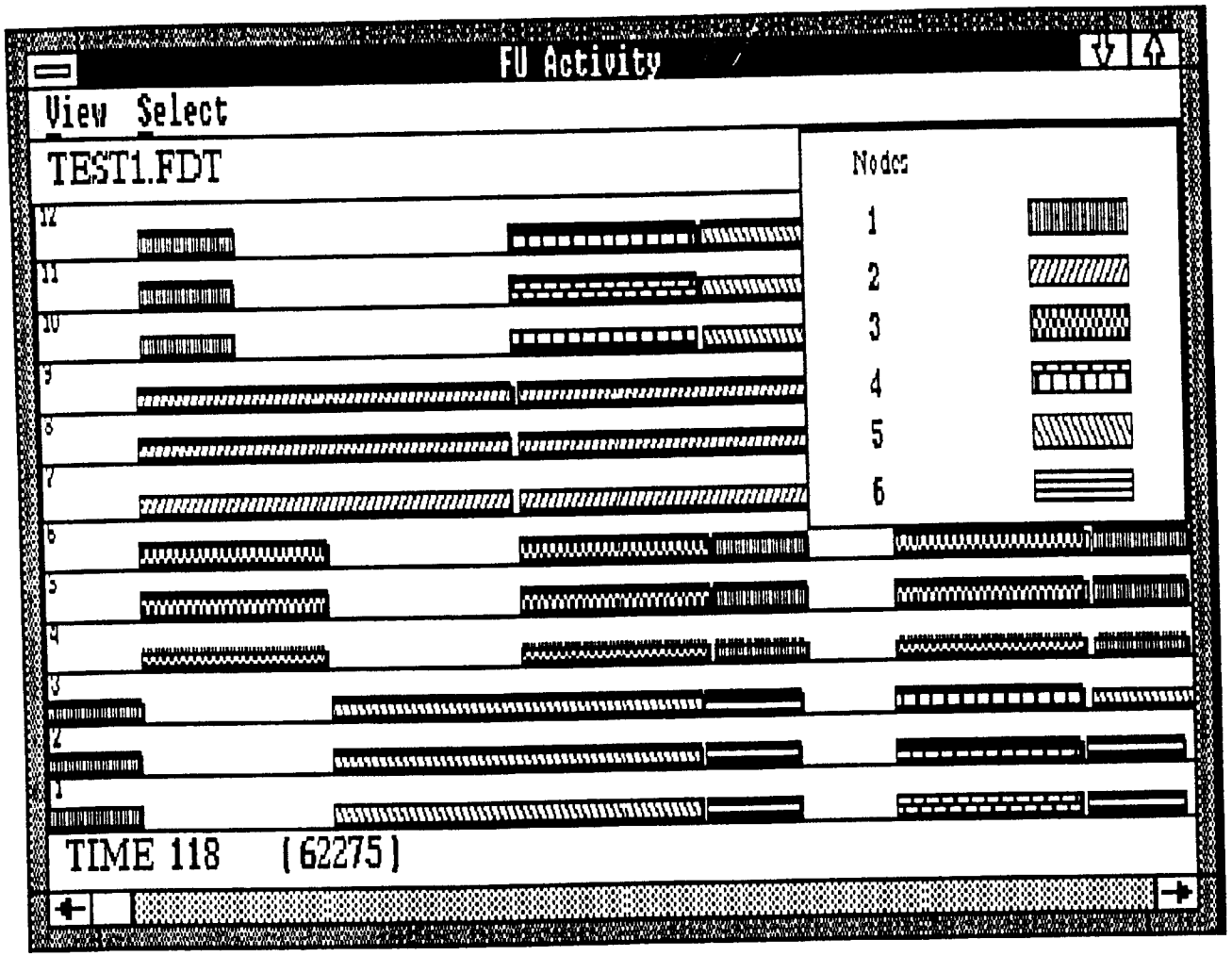


Figure 4.5. FU Activity Display for Test One.

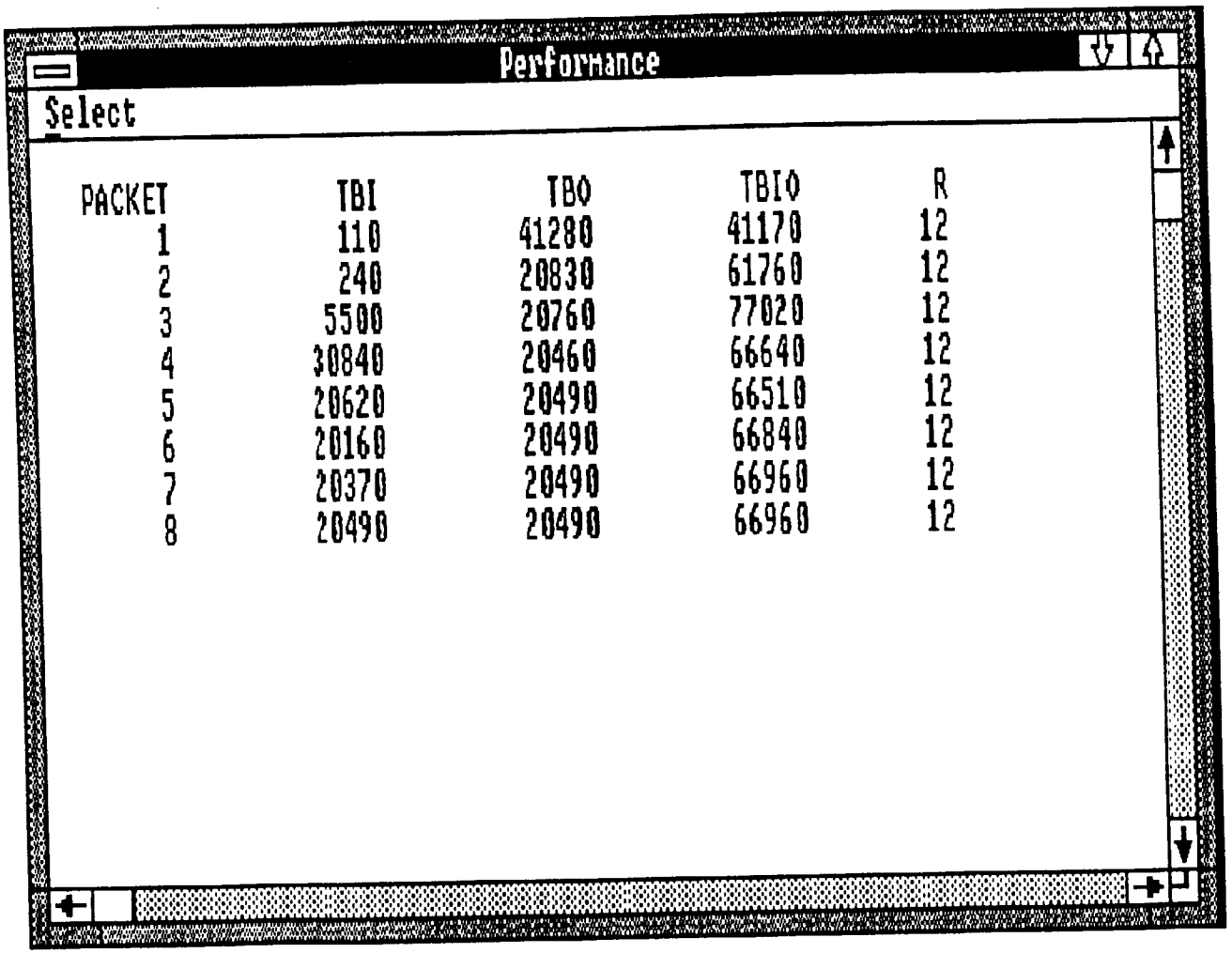


Figure 4.6. Performance Display for Test One.

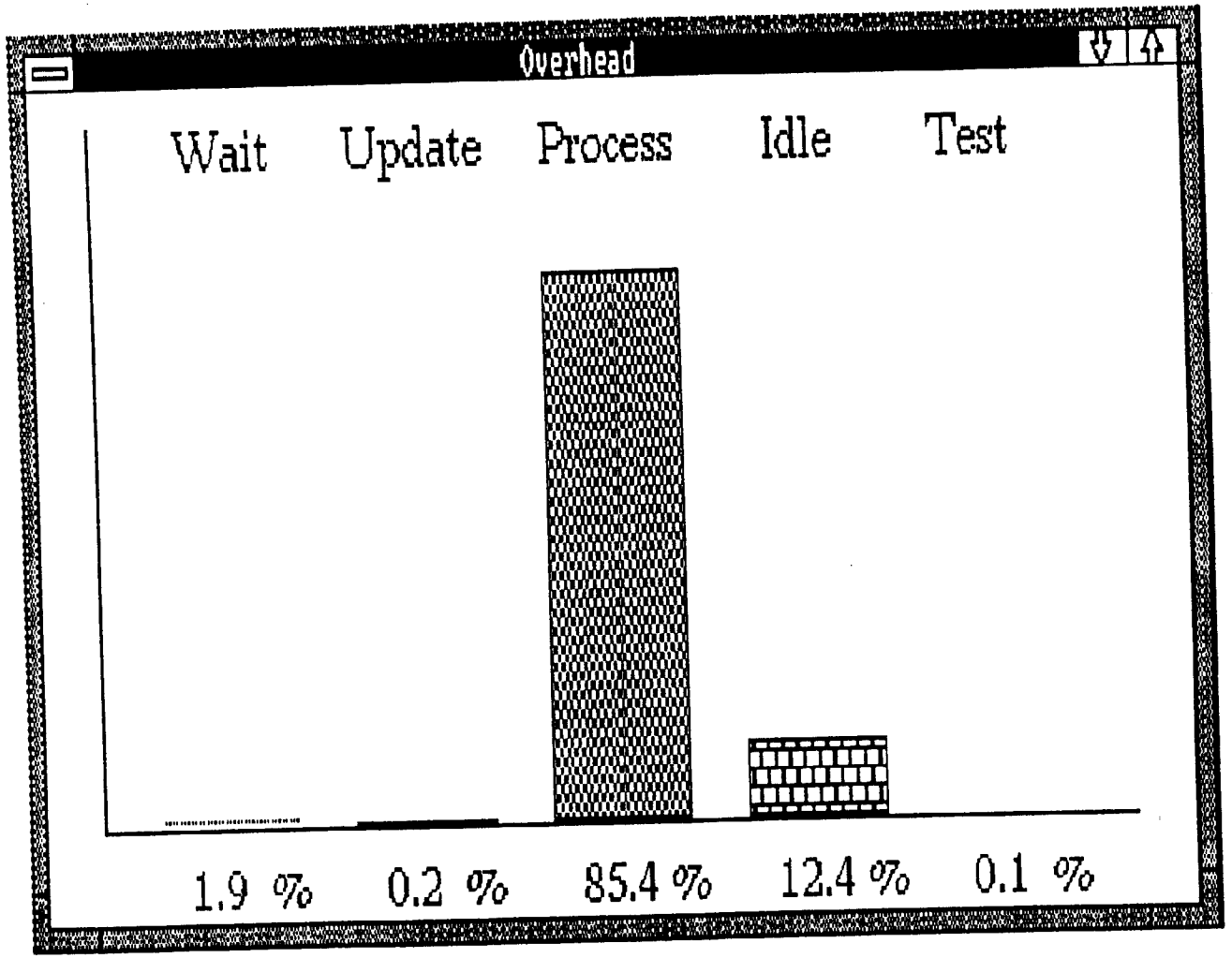


Figure 4.7. Overhead Display for Test One.

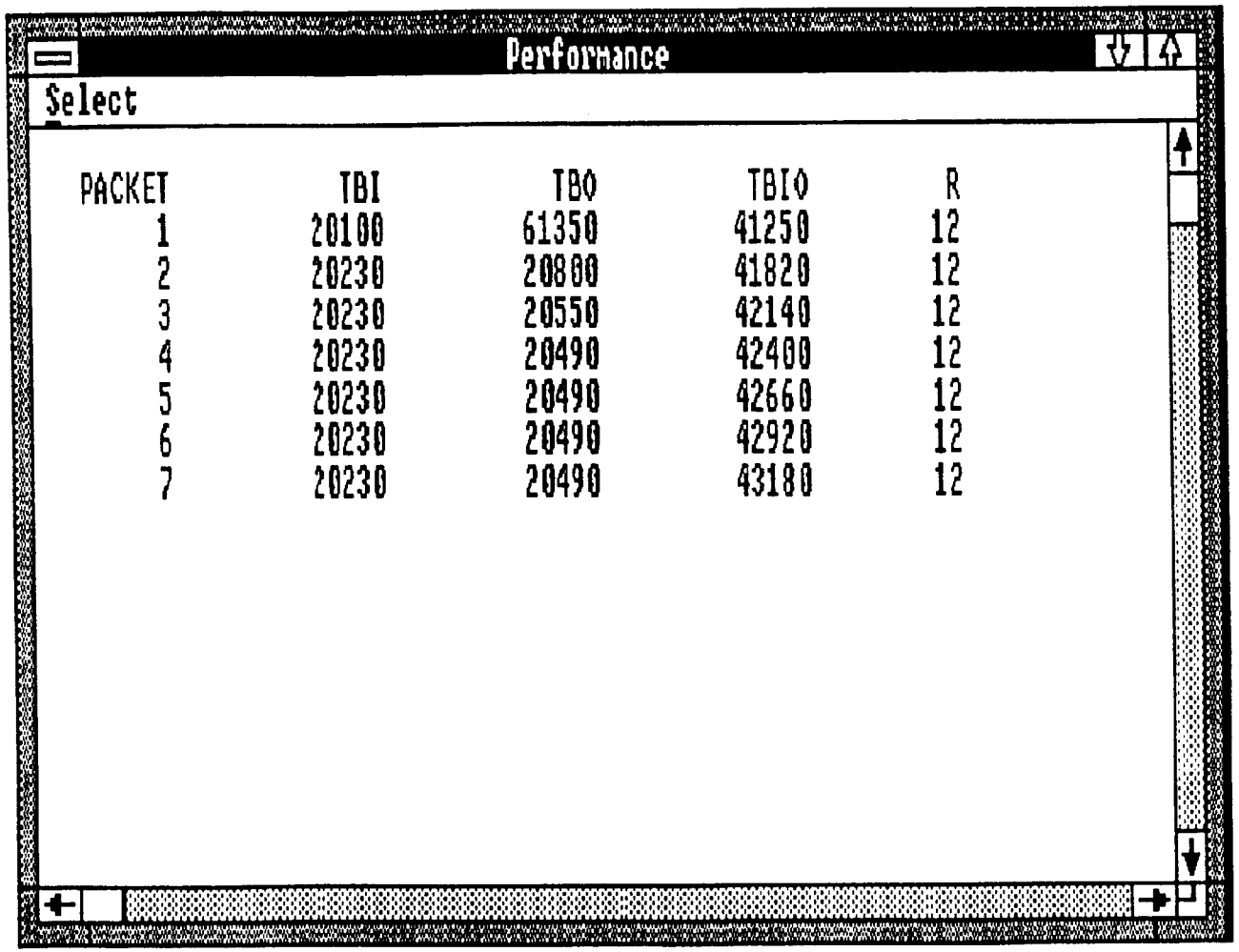


Figure 4.8. Performance Display for Test Two.

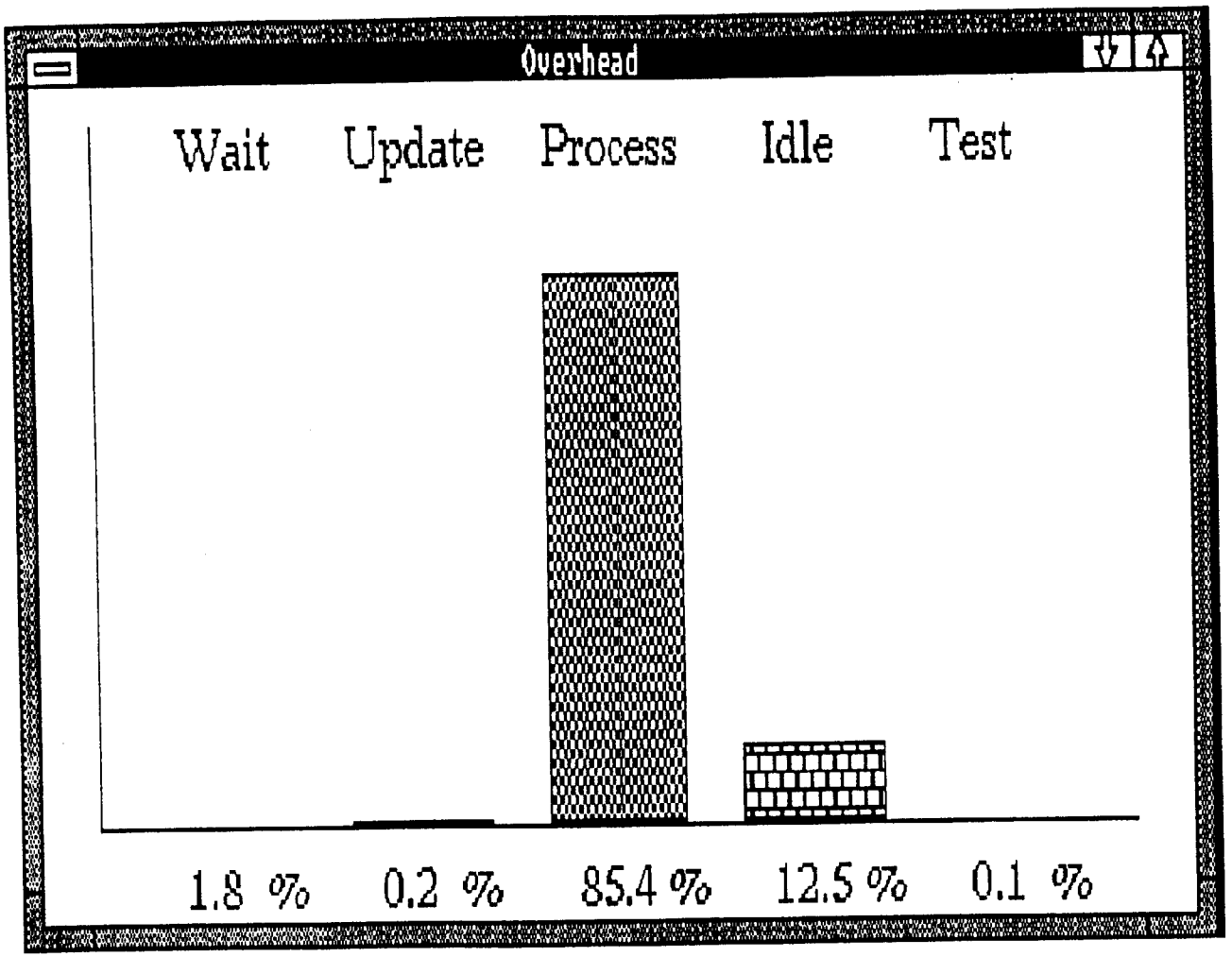


Figure 4.9. Overhead Display for Test Two.

A useful measurement that can help in establishing a theoretical TBO_{LB} is the processing time percentage associated with executing the critical circuit. This value should be maximum for injection intervals below TBO_{LB} and decrease for injection intervals above TBO_{LB}. The decrease is due to the added idle time that is introduced when TBO is greater than TBO_{LB}. The FU Activity display can be used to select only the functional units processing nodes two and four which comprise one of the critical circuits. When the viewing window is expanded, an interval containing just the critical circuit activities can be defined as shown in Figure 4.10. One can observe the percentage of time spent processing the critical node versus the time spent on overhead and in idle. It is determined from Figure 4.10 that the critical processing is equal to 97.5 percent. The results of this measurement for the other three tests as well as this test can be inspected in Table 4.1.

Information included in the Performance displays of the previous tests assisted with the injection interval choice of the remaining tests. When a TBI of 200 ms. was desired in test two, a 202.3 ms., TBI value resulted due to device overhead. Therefore, it was assumed that the specified injection interval for other TBI values should be the desired TBI minus 2.3. The previous tests also showed that the minimum TBO is equal to 204.9 ms. Therefore, a TBI value equal to 204.9 ms. was used for the third test. The added 4.9 ms. results in 2.4 percent added overhead which is close to the measured overhead percentage. TBI for the final test was chosen be 210.0 ms. to allow twice the overhead (4.8 percent).

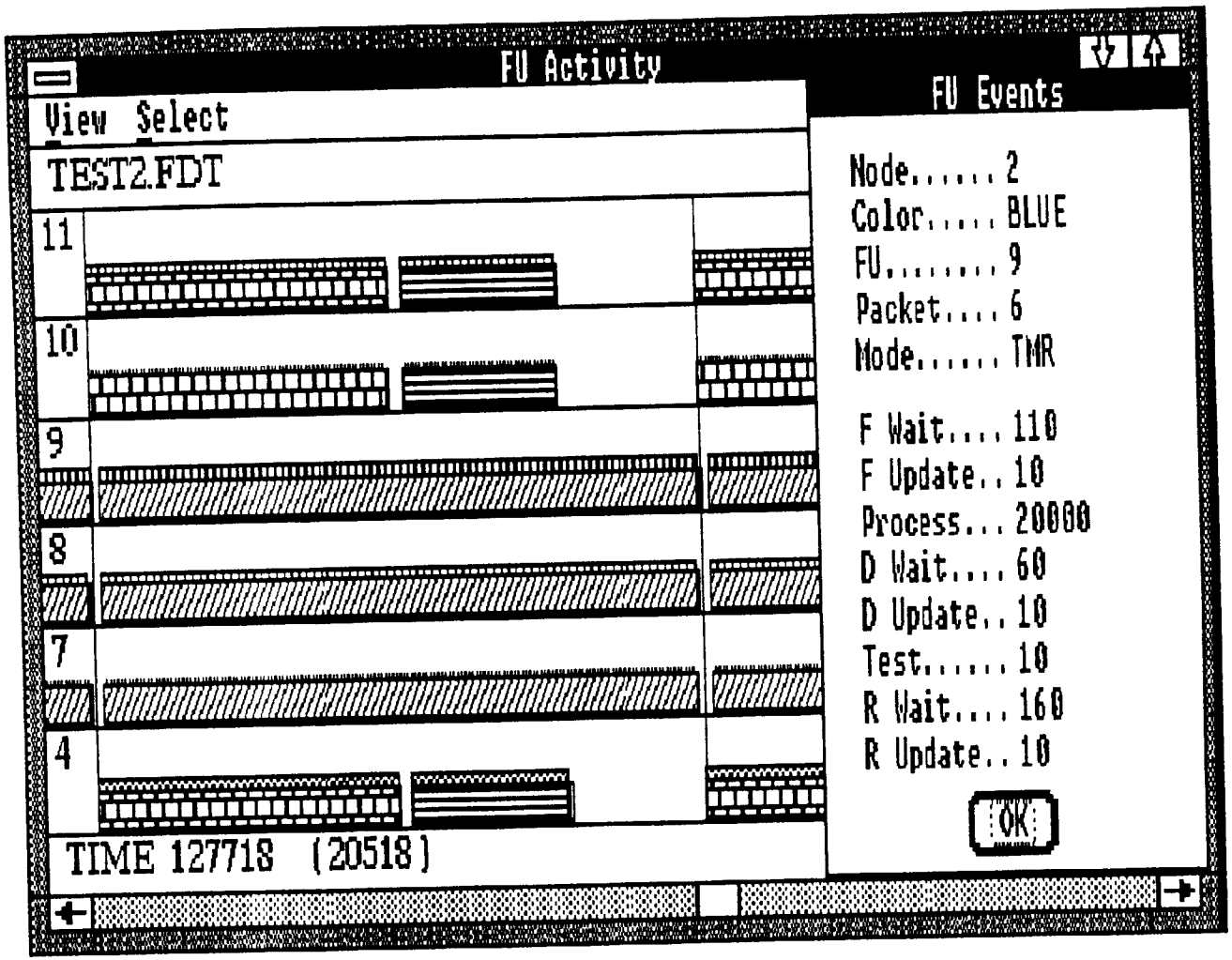


Figure 4.10. FU Activity Display for Test Two.

Table 4.1. Test Results for Section 4.3

TEST	Mean TBI	Mean TBO	Mean TBIO	Overhead	Resource Utilization	Critical Processing
1	204.1	204.9	668.2	2.2 %	87.6 %	97.5 %
2	202.3	204.9	427.9	2.1 %	87.5 %	97.5 %
3	204.9	204.9	418.8	2.1 %	87.5 %	97.8 %
4	210.0	210.0	412.1	1.9 %	85.2 %	95.1 %

(TBI, TBO, and TBIO are in milliseconds)

However, an injection interval of 202.6 ms. was used for test three and an injection interval of 207.7 ms. was used for test four to account for the added 2.3 ms. in device overhead. The results of these two tests, using the same Analysis Tool features, along with the previous test results are included in Table 4.1.

Table 4.1 indicates that the TBIO of 412.1 ms. that resulted in test four was less than the TBIO of test three. The decrease in the critical processing time percentage between test three and test four indicates that a TBO of 210.0 ms. resulted in increased idle time. Therefore, theoretical TBO_{LB} for the simulated system must be between 204.9 ms. and 210.0 ms. It was assumed heuristically that a TBO equal to 207.5 ms. would provide minimum TBIO and minimum idle time for the critical circuit.

4.4 Graph Operation at TBO_{LB} and $TBIO_{LB}$

The ideal lower bound values for TBO and TBIO were predicted in Section 4.2. An evaluation was made in Section 4.3 in order to establish theoretical lower bound TBO and TBIO values in consideration of system overhead. This section will now demonstrate the remaining features of the Analysis Tool while displaying the graph operation at the theoretical TBO_{LB} and $TBIO_{LB}$. The observed behavior should be comparable to the predictions included in Figure 4.2.

It is inferred from the previous section that an added 3.75 percent for overhead should be added to the ideal TBO. Therefore, the injection interval

should be 205.2 ms. in order to obtain a TBI equal to 207.5 ms. Theoretical $TBIO_{LB}$ is presumed to be 412.1 ms.

Simulation results of the graph with twelve resources at a TBI equal to 207.5 ms. are included in Figures 4.11 through 4.15. The Single Graph Play and Total Graph Play displays of Figures 4.11 and 4.12, respectively, can be compared to Figure 4.2. Note the defined time interval marked in the Single Graph Play display determines the approximate TBIO. The mean TBO can be determined from the TBO box shown within the Total Graph Play display. Included in Figures 4.13 and 4.14 are the Single Resource Envelope and Total Resource Envelope displays, respectively, which show the functional unit utilization. Measurements of resource utilization are included in the Utilization display also shown in Figure 4.14. The resulting operating point can be observed by referring to Figure 4.15 which shows the plot of TBO verses TBIO along with the performance times associated with each packet. Observe that TBO quickly reaches a steady state value of 207.5 ms. with a TBIO value of 412.1 ms., as expected. The measured utilization of 86.2 percent results in only a 1.5 percent difference from the ideal prediction.

4.5 Graph Operation at $TBIO_{LB}$ and TBO for Reduced Resources

A demonstration of graph operation with fewer resources is presented in this section. Predictions were made in Section 4.2 on the behavior and performance of the system for three resources (nine for TMR). The results

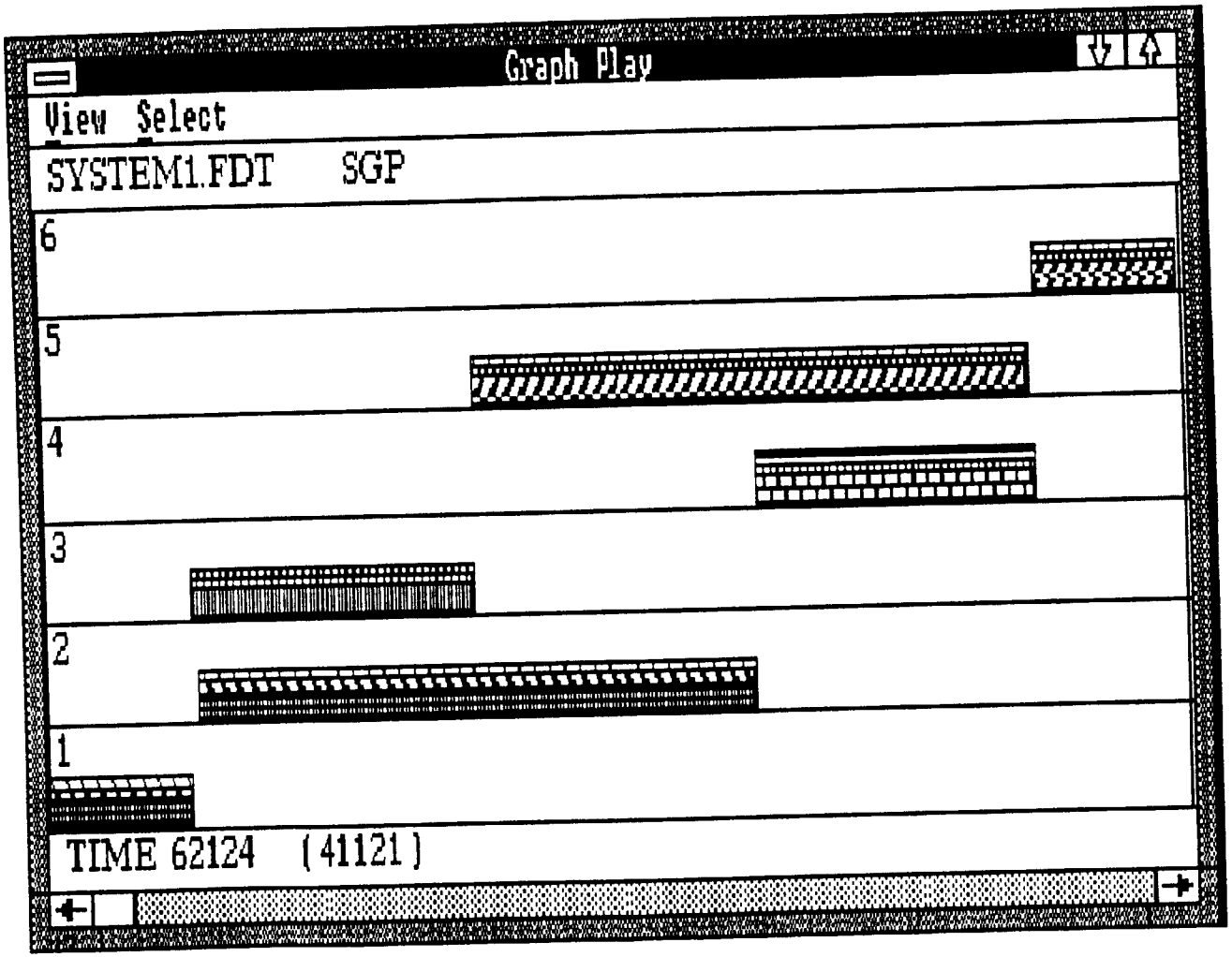


Figure 4.11. Single Graph Play Display for Section 4.4.

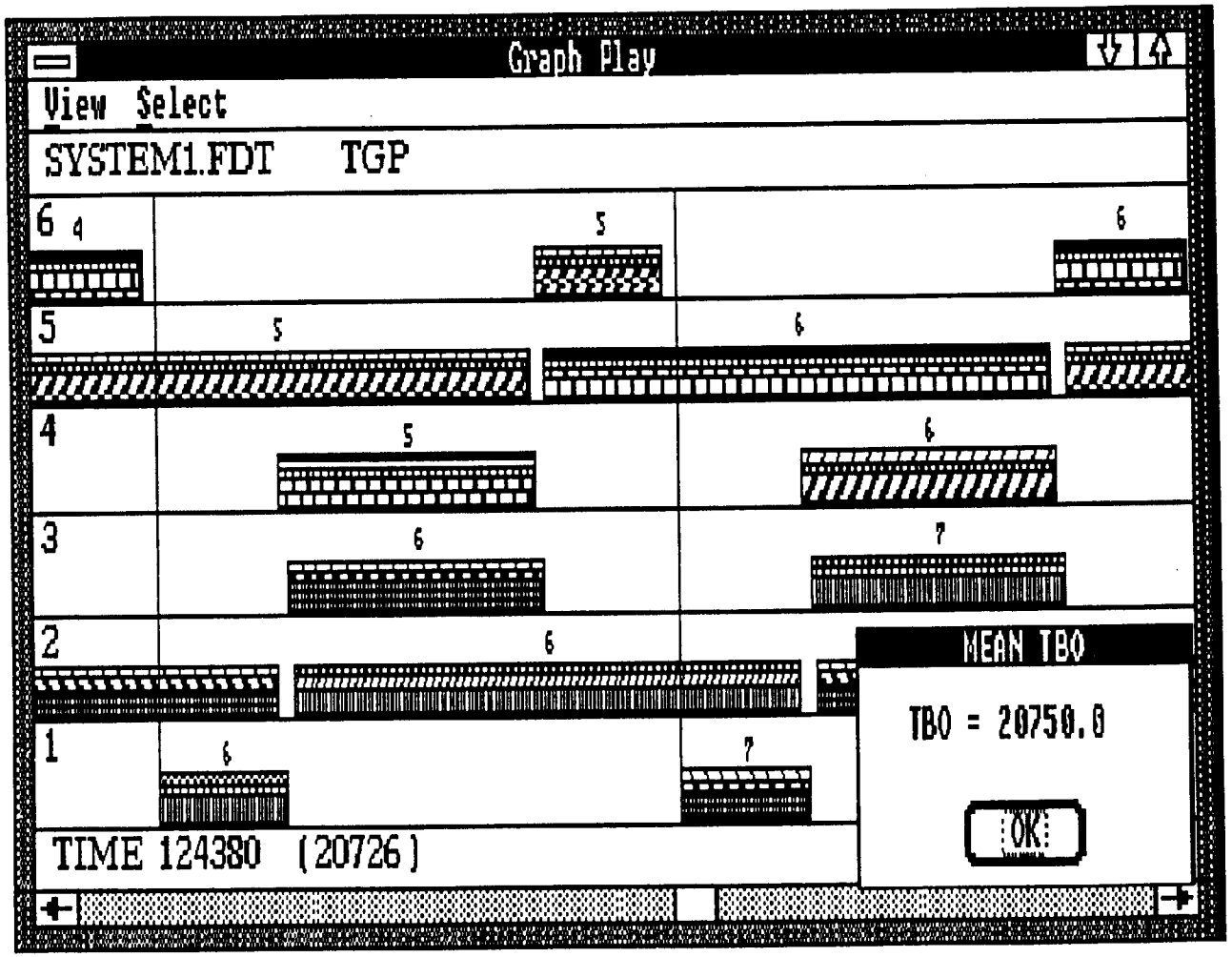


Figure 4.12. Total Graph Play Display for Section 4.4.

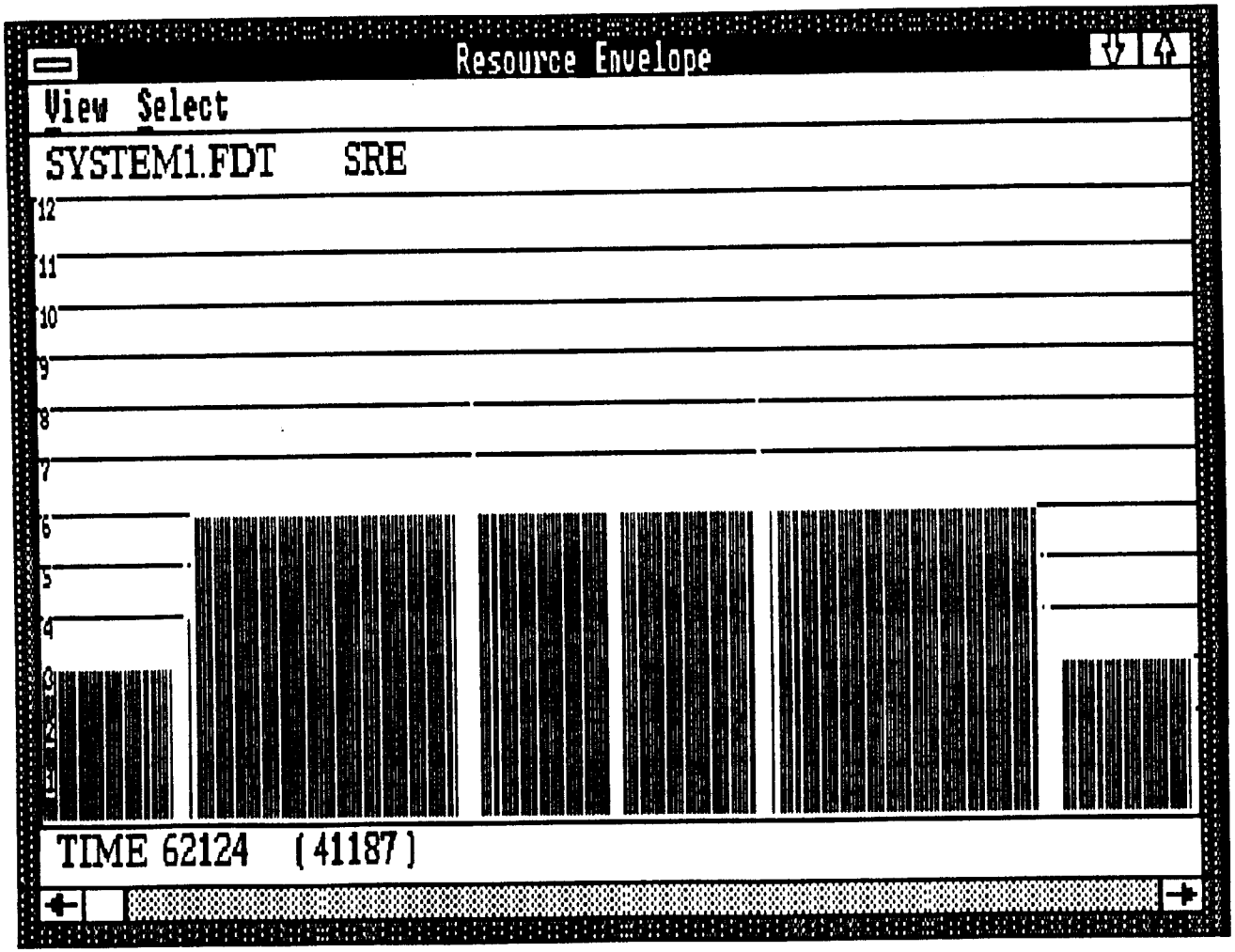


Figure 4.13. Single Resource Envelope Display for Section 4.4.

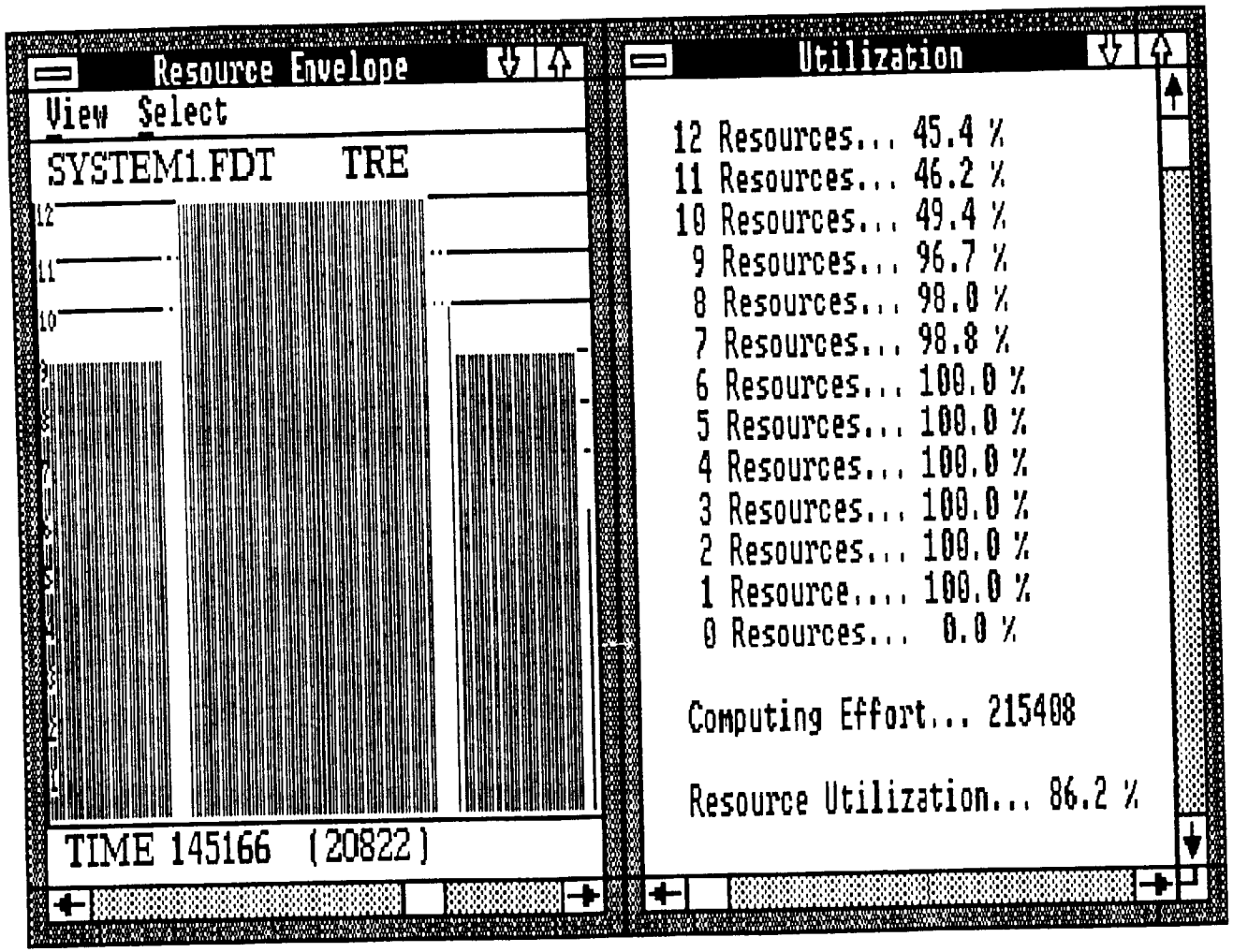


Figure 4.14. Total Resource Envelope and Utilization Displays for Section 4.4.

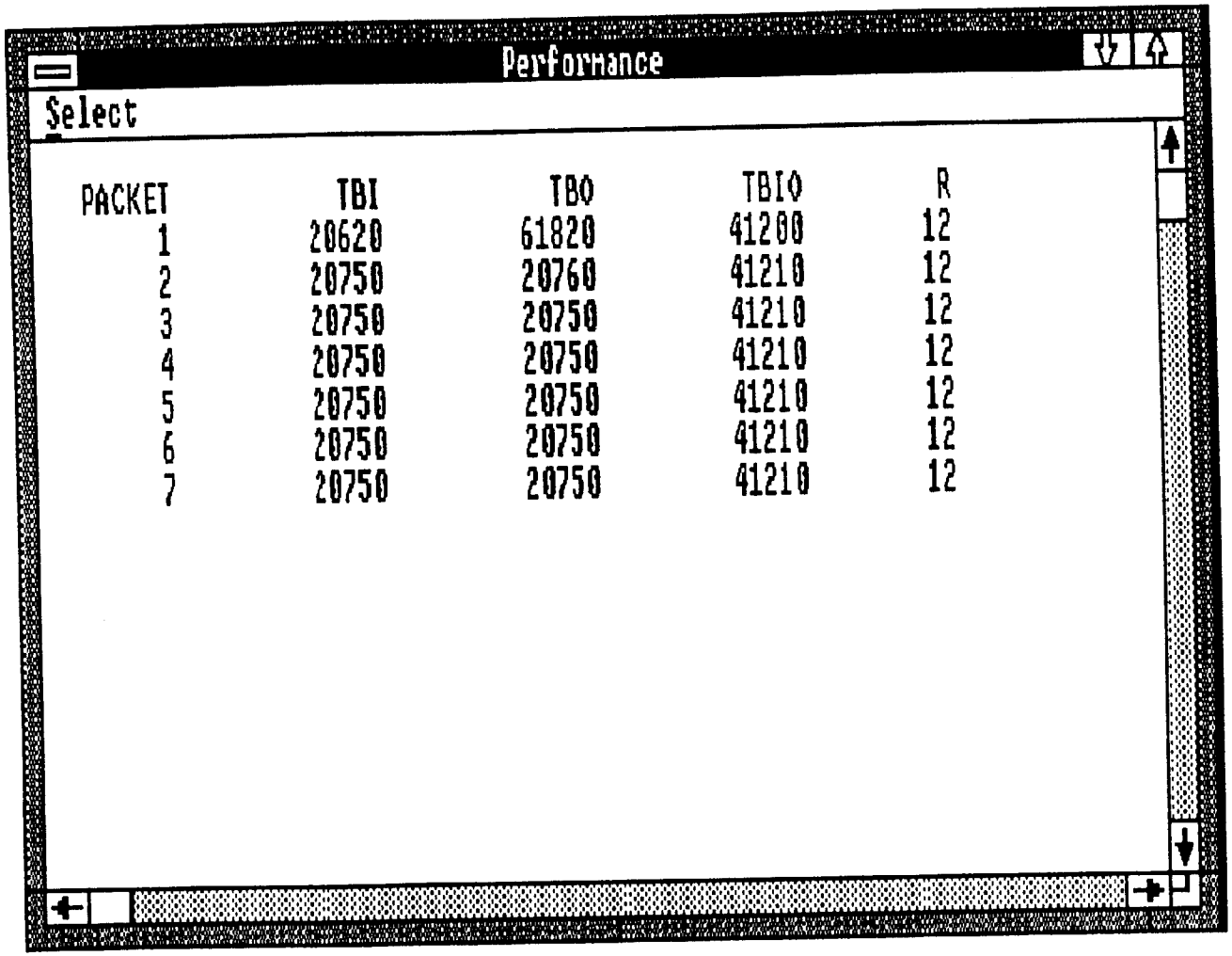


Figure 4.15. Performance Display for Section 4.4.

shown here using the Graph Play, Resource Envelope, Utilization, and Performance should again be comparable with the predictions.

The results of the overhead evaluation will again be used for establishing theoretical TBO and TBIO values. As in the previous section, it is presumed that theoretical $TBIO_{LB}$ is 412.1 ms. Also, it is assumed that the 3.75 percent overhead requirement is independent of TBO. Therefore, TBO is increased by 3.75 percent in order to attain this minimum TBIO. Therefore, the system was simulated for nine resources with a 309 ms. injection rate (for a TBI equal to 311.3 ms).

The Total Graph Play display that resulted is displayed in Figure 4.16. Included in Figure 4.17 are the Total Resource Envelope and Utilization displays. Inspection of these displays shows that the resulting behavior is in agreement with the predicted behavior. The Performance display of Figure 4.18 presents a graphical plot of the operating point along with the individual packet performance times. The Performance display indicates that the system reaches steady state TBO and TBIO values of 311.3 ms. and 412.1 ms., respectively. The measured resource utilization of 76.6 percent results in only a 1.5 percent difference from the ideal prediction.

4.6 Error Reporting Demonstration

The only Analysis Tool feature not demonstrated thus far is that of fault detection and reporting. This feature is useful when analyzing a real system but not useful for a simulated system. Since an FDT file from a real ATAMM based

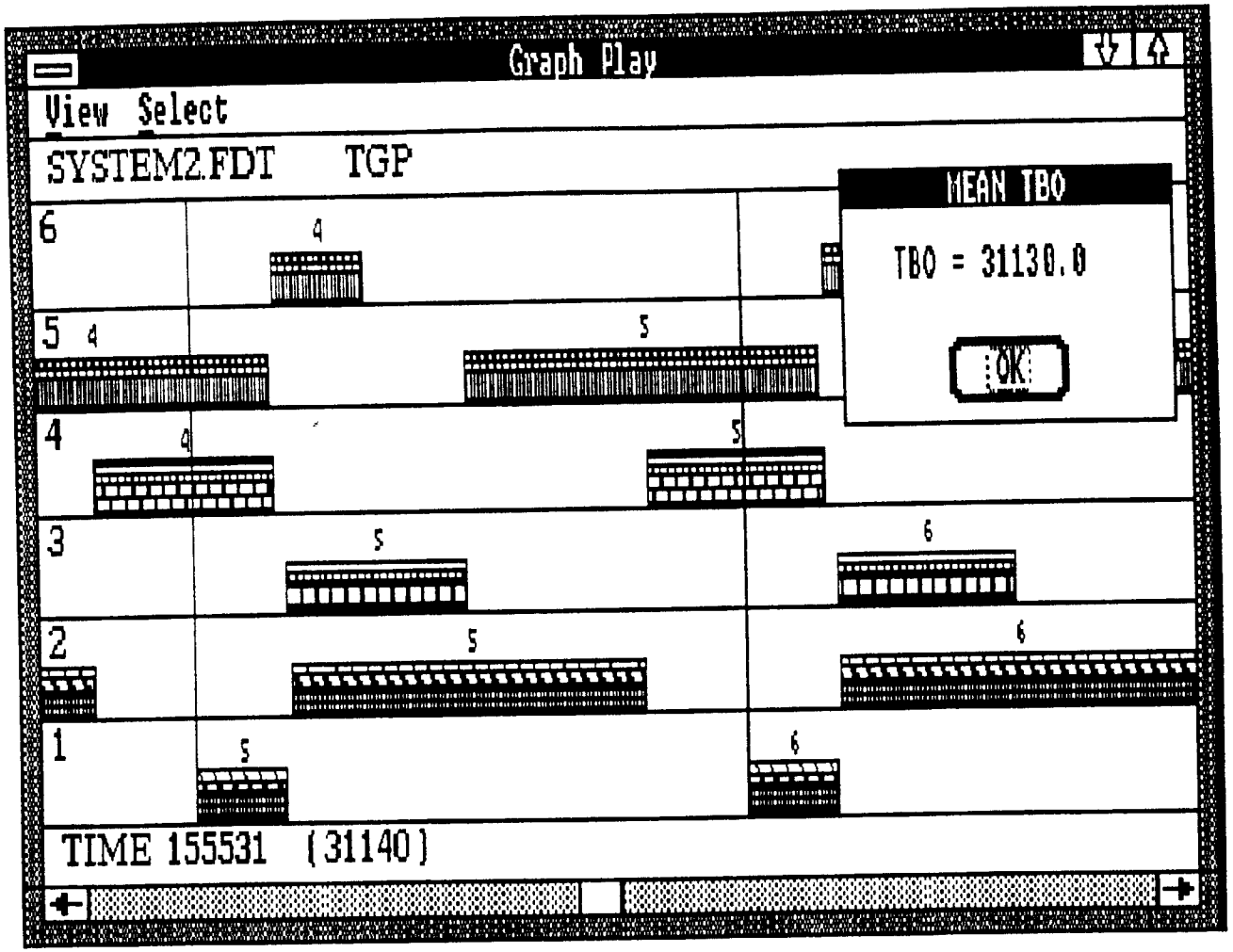


Figure 4.16. Total Graph Play Display for Section 4.5.

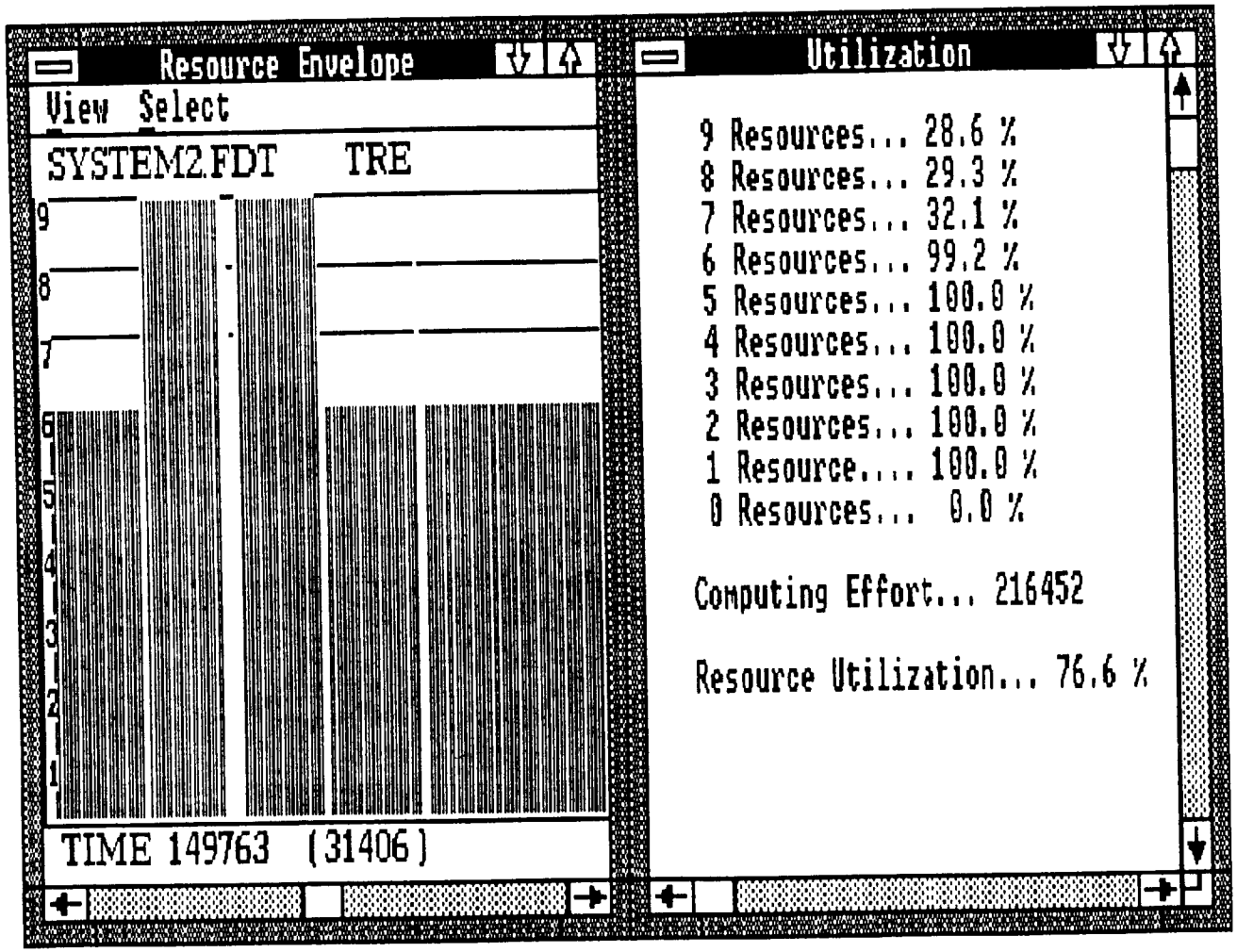


Figure 4.17. Total Resource Envelope and Utilization Display for Section 4.5.

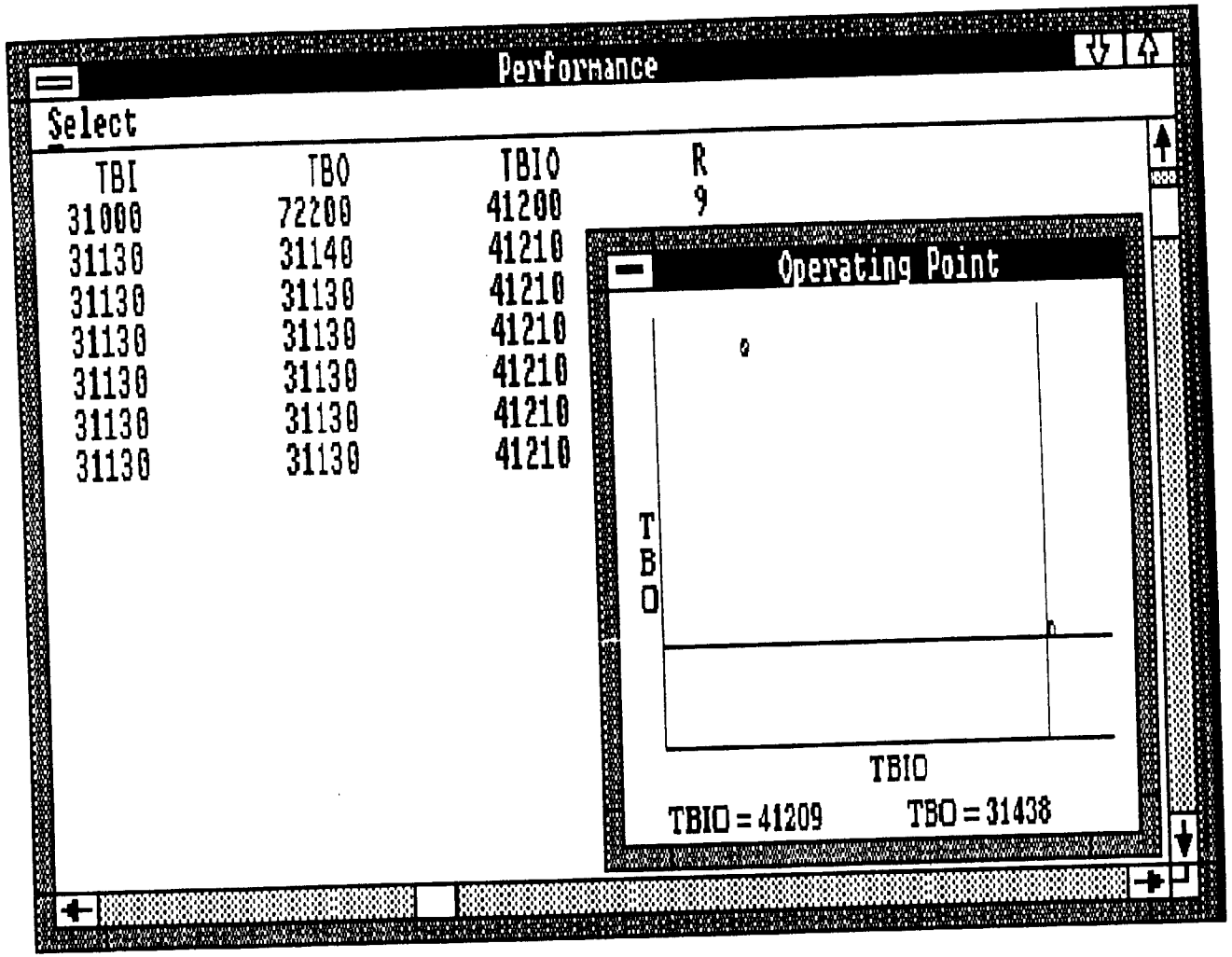


Figure 4.18. Performance Display for Section 4.5.

system was not available during the preparation of this thesis, some improvisation will be necessary.

The FDT file used in Section 4.5 was altered so it would appear to be an FDT file generated by a real system. Faults were inserted at the following places in the FDT file using the voting error reports discussed in Section 3.2.

Packet	Node	Color	F. U.
2	1	Green	8
2	2	Blue	9
2	4	Red	4
3	4	Red	4
3	5	Blue	3

When the error displaying mode is selected within the Graph Play display or the FU Activity display, detected errors will be indicated by "X" marks next to the corresponding activity blocks. The "X" mark will be above the activity block of the FU Activity display and to the right adjacent side of the Graph Play activity block. Inspection of the Graph Play display (Figure 4.19) and the FU Activity display (Figures 4.20) of the modified FDT file indicates errors at the specified locations as expected.

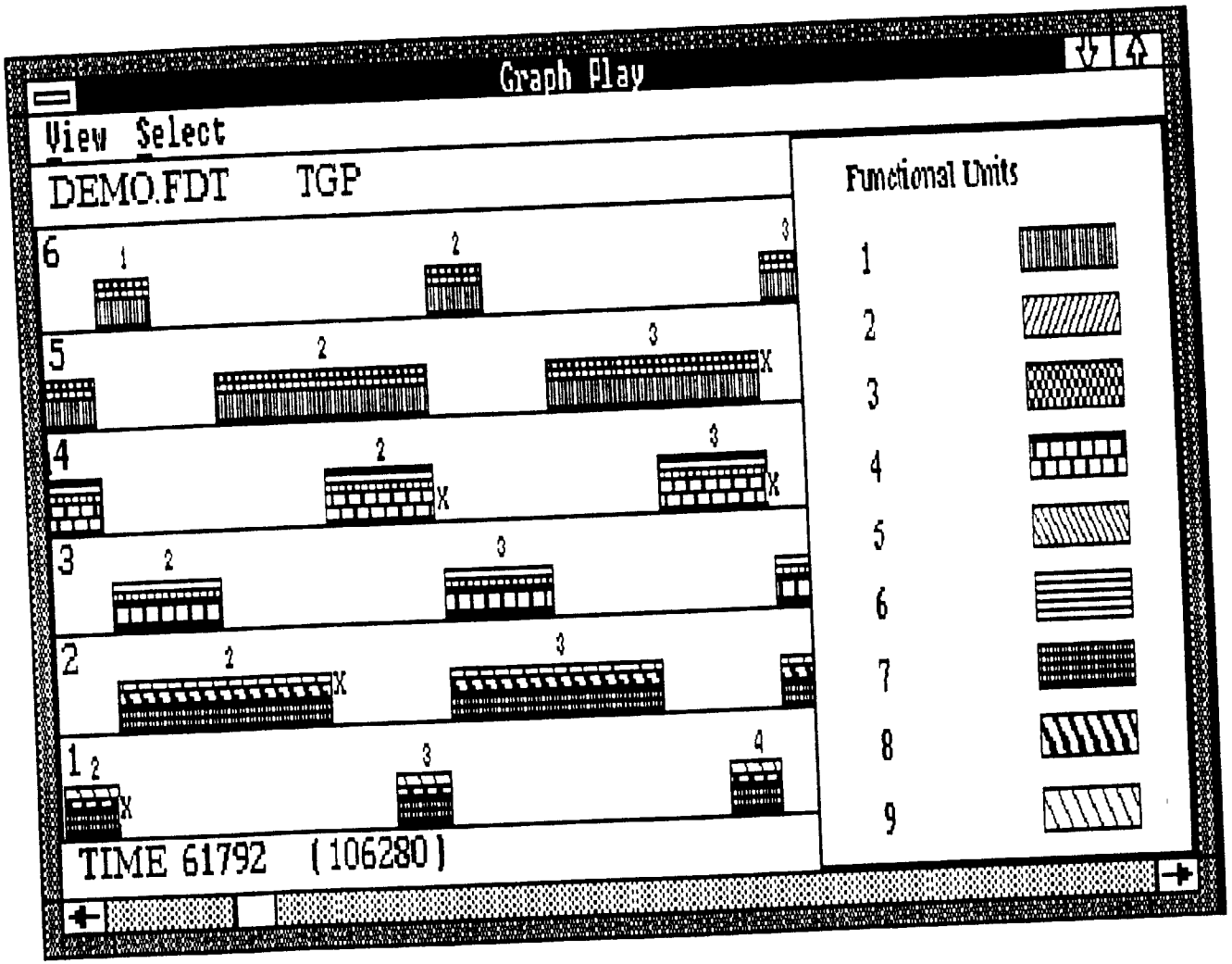


Figure 4.19. Graph Play Display for Section 4.6.

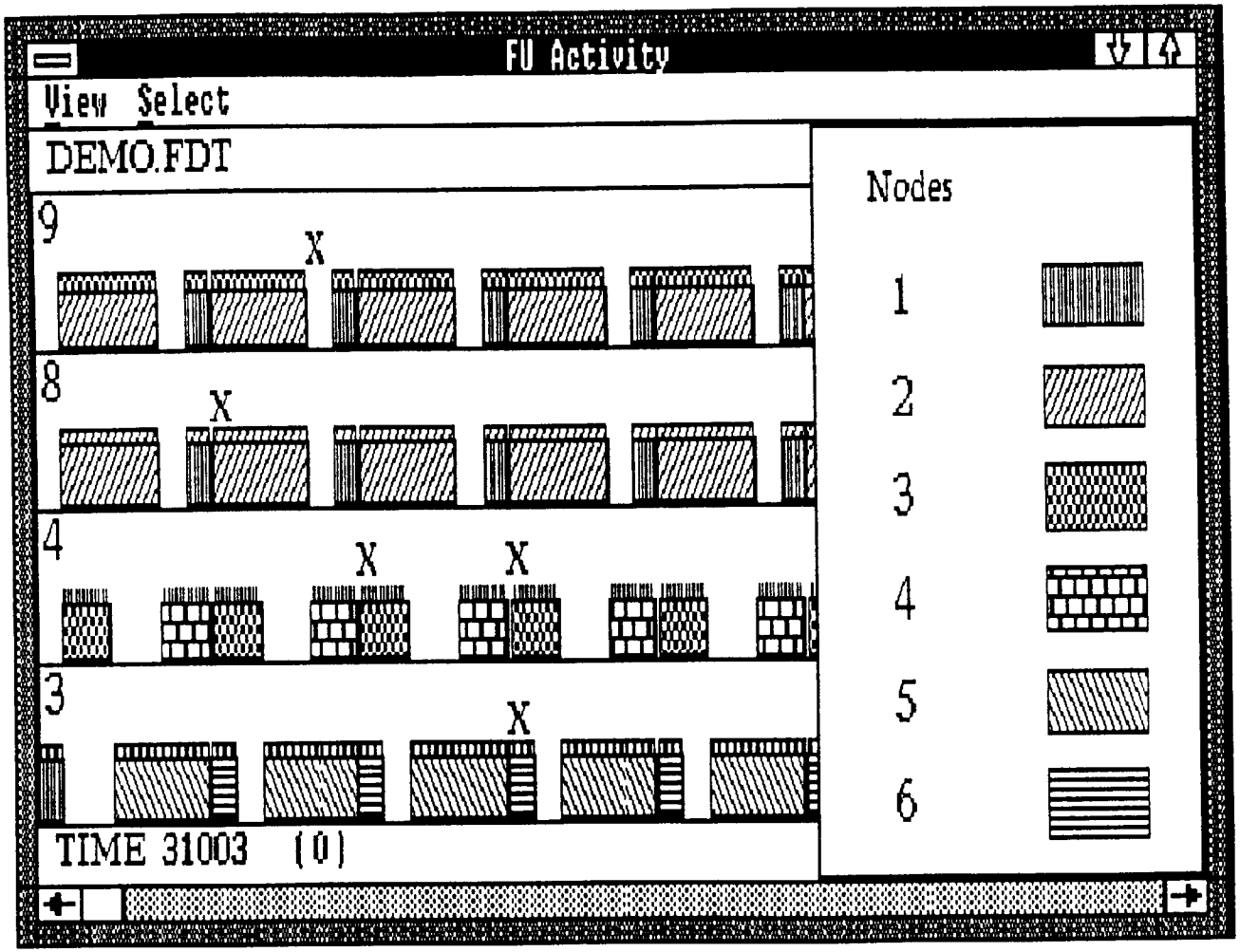


Figure 4.20. FU Activity Display for Section 4.6.

CHAPTER FIVE

Conclusion

5.1 Summary

Diagnostics software is developed in this thesis in order to productively analyze an ATAMM based distributed-processing system. The software is referred to as the Analysis Tool. The Analysis Tool provides a graphical portrayal of the processor activities and node activities resulting from the execution of an algorithm graph. The Analysis Tool also provides automatic and user-interactive time measurements. These time measurements make it possible to evaluate the system's performance and overhead requirements.

The Analysis Tool is developed in the Microsoft Windows environment for increased capability and user-friendliness. Utilizing a window environment permits the Analysis Tool to run concurrently with other software applications (such as a simulator). The window environment also permits one to view all of the Analysis Tool displays simultaneously.

Experiments were performed that demonstrated the overhead evaluation capabilities of the Analysis Tool. The Analysis Tool was used to determine the overhead requirement of a simulated system. The theoretical lower bound performance of the system was determined based on the overhead requirement.

The Analysis Tool also showed the effect of overhead on input injection. The correct input injection rate for a desired TBI was determined from the observed effect. Without the Analysis Tool, such investigation of system overhead would be difficult.

The system activity displays and performance measurement capabilities of the Analysis Tool were also demonstrated. These Analysis Tool features are essential in order to productively study an ATAMM based system. The simulated system was analyzed at two different operating levels. The Analysis Tool provided the means to compare the resulting node and functional unit behavior with the predicted behavior. Measured TBI, TBO, and TBIO values indicated that system performance matched the expected performance. Resource utilization measurements were comparable to the ideal calculations. The ideal calculations neglected the effect of system overhead. Therefore, the discrepancy in the measured and calculated resource utilization values are primarily attributed to the added overhead determined in the earlier tests.

5.2 Topics for Future Research

Future research involves the inclusion of multiple graphs and multiple communication channels to the problem domain of ATAMM. Enhancements to ATAMM to include multiple functional unit types should also be investigated. Example applications include the execution of multiple graphs with priority assignments by a set of homogenous functional units or the execution of a

partitioned graph by sets of functional units having different capabilities. These model enhancements will thus require future modifications to the Analysis Tool.

The goal of the current memory management implementation is to optimize the speed at which pertinent data can be accessed. However, this caused inefficient use of memory. The added information required with the inclusion of multiple graphs and channels will require more memory for storage. Therefore, a different management of memory may be required in the future.

There is continuing interest in the investigation of reliability and fault tolerant issues. The ongoing research on fault tolerance and real-time graph optimization with the ADM system will provide possible refinements to the ATAMM model. Any ATAMM refinements will no doubt spawn other meaningful research topics for future consideration.

An ATAMM design tool and simulation software is currently under development. It is intended that these tools working in conjunction with the Analysis Tool will aid in the research of all these topics.

REFERENCES

[1] John W. Stoughton and Roland R. Mielke, "Strategies for Concurrent Processing of Complex Algorithms in Data Driven Architectures," NASA Technical Paper 181657, Grant NAG1-683, February 1988.

[2] Tadao Murata, "Petri Nets: Properties, Analysis and Applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541-580, April 1989.

[3] Roland R. Mielke, John W. Stoughton and Sukhamoy Som, "Modeling and Optimum Time Performance for Concurrent Processing," NASA Contractor Report 4167, Grant NAG1-683, August 1988.

[4] S. Som, B. Mandala, R. R. Mielke and J. W. Stoughton, "A Design Tool for Computations in Large Grain Real-Time Data Flow Architectures," *Proceedings of the IEEE Southeastcon '90*, New Orleans, Louisiana, April 1990.

[5] Sukhamoy Som, "Performance Modeling and Enhancement for the ATAMM Data Flow Architectures," Ph. D. Dissertation, Old Dominion University, Norfolk, Virginia, May 1989.

[6] S. Som, J. W. Stoughton and R. R. Mielke, "Performance Prediction, Simulation, and Measurement for Real-Time Computing in a Class of Data Flow Architectures," Technical Paper Presented at the ISMM International Conference on Computer Applications in Design, Simulation and Analysis, New Orleans, Louisiana, March 1990.

[7] W. R. Tymchyshyn, "ATAMM Multicomputer System Design," Master's Thesis, Old Dominion University, Norfolk, Virginia, August 1988.

[8] M. Malekpour, R. Obando, R. R. Mielke, and J. W. Stoughton, "ATAMM Simulation Tool for Data Flow Architectures," Modeling and Simulation Conference, Pittsburgh, Pennsylvania, May 1990.

[9] Rodrigo A. Obando, "Software Tools for Performance Evaluation of Concurrent Processing," Master's Thesis, Old Dominion University, Norfolk, Virginia, October 1987.

[10] Charles Petzold, Programming Windows, Redmond, Washington, Microsoft Press, 1988.

[11] M. Waite, S. Prata, B. Costales, H. Henderson, Microsoft QuickC Programming, Redmond, Washington, Microsoft Press, 1988.

[12] Carl Townsend, Understanding C, Indianapolis, Indiana, Howard W. Sams and Company, 1988.

APPENDIX

ANALYSIS TOOL USER'S MANUAL

TABLE OF CONTENTS

A.1 Overview

The Analysis Tool is a program that provides an efficient means to evaluate the performance of a concurrent processing system that is based on the ATAMM rules. Its input comes from data stored in an FDT file. Performance evaluation is made possible by providing a display of node and functional unit activity along with functional unit concurrency. Also, measurements of time performance, functional unit utilization, and overhead is provided.

A.2 System Requirements

The Analysis Tool program requires an IBM PC or compatible, the Microsoft Windows 286/386 multitasking environment, and a mouse. The Analysis Tool system is divided into several modules each occupying its own code segment. The largest code segments are moveable and discardable. This means that even though the entire system requires approximately 100k bytes, the system's code will manage with much less memory. However, the Analysis Tool utilizes dynamic allocation of global and local heap space for storage and management of FDT file data. Therefore, the total amount of memory the Analysis Tool requires depends on the size of the FDT file.

Since the Microsoft Windows environment manages all aspects involved with software/hardware interaction, the type of monitor and printer attached to the computer is of no concern to the Analysis Tool. Peripheral devices need only be supported by the Microsoft Windows environment.

A.3 FDT File

The Analysis Tool uses the FDT file as input in order to obtain the information needed to graphically reproduce the node and functional unit activity and obtain performance measurements. The event information format for this file is discussed in Section 3.2 of the Analysis Tool Development chapter.

Information concerning the number of events and the initial number of resources must be included at the beginning of the file. The first line of the FDT file must state the number of events listed in the file and have the following form:

EVENTS = *number of events*

If voting error reports and change-in-resource reports are not included in the FDT file the second line of the file must state the number of resources present in the system as follows:

R = *resources*

If, however, voting error and change-in-resource reports are included the resource declaration statement must be omitted. Instead, "T", "Q", and "R" events followed by a change-in-resource report announcing the "initial installation" of the functional unit must be present for all functional units in the system. These

reports replace the "R = *resource*" statement in order to initialize the Analysis Tool's "available" data structure element.

A.4 Using the Analysis Tool

When the Analysis Tool is running, a window (called the Analysis Tool) provides a means to open FDT files, create the primary display windows for viewing, and obtain help on any of these windows. You may exit the Analysis Tool program by selecting the "Exit" menu item or closing the Analysis Tool window from its system menu.

A.4.1 Window Management

Each display is devoted to its own window which can overlap other windows. This allows the viewing of more than one display simultaneously. Consult a Microsoft Windows user's manual for detailed information concerning the use of windows.

A.4.2 Opening an FDT File

Select the "Open" option in the Analysis Tool window's menu. A dialogue box will appear allowing you to select an FDT file to open. Choose "OK" to open the file shown within the listbox or "double click" the left mouse button on the file name that you want opened. After opening an FDT file, the Analysis Tool will read and store the file data and display the file name, number of nodes in the graph, number of resources, and the number of events listed in the file within the Analysis Tool window. Select the "Open" menu option again to open and analyze

another FDT file. The Analysis Tool will close all of its display windows that are presently open and reinitialize the system for the new FDT file.

A.4.3 Viewing Node and Functional Unit Activity

You can view the node and functional unit activity encoded in the FDT file by selecting the "Graph Play" and "FU Activity" options, respectively, from the Analysis Tool window menu. Selecting the "Assigned FU's" menu item from the Graph Play window displays a window showing the color-to-functional unit mapping associated with the node activities. Selecting the "Node Assignment" menu item from the FU Activity window displays a window showing the color-to-node mapping associated with the functional unit activities.

The Graph Play window provides a view of either the "Single Graph Play" or "Total Graph Play" depending on which respective menu item is selected. A default data packet number equal to one is used for the single graph play. You can define another packet number by selecting the "Packet Number" menu item and entering a number within the dialogue box that appears.

Selecting the "Display Nodes" menu item allows you to select the node activities you desire to view. Delete the "X" marks, within the dialogue box, next to the node number that you want deleted from the display. Placing an "X" mark in the "All" box causes all nodes to be displayed. The "Display FU's" menu item provides the same feature for the functional unit activities.

The Graph Play window and the FU Activity window can be made icons or closed from their system menus.

A.4.3.1 Viewing Internal Transitions

When the "Transitions" menu item is selected within the Graph Play or FU Activity window, the activity blocks are partitioned in order to expose the transitions that comprise the activity. Select the "Transition Assignment" menu option, when in this mode, in order to inspect the color-to-transition mapping.

A.4.3.2 Quick List of Data

A list of information pertaining to an activity block can be displayed by placing the arrow cursor within the boundaries of the block and clicking the left mouse button while holding down the SHIFT key. The listed information includes the node number, node color, functional unit I.D., mode, data packet number processed, and the transition times associated with the activity.

A.4.3.3 User-Interactive Measurement of TBO

A measurement of average TBO can be obtained by defining a time interval using the two time cursors (see Section 3.4. Controlling the Cursors) and selecting the "TBO" menu item. If the calculation can be made with respect to the defined time interval a box displaying the average TBO for that interval will appear. If not, you will hear a "beep".

A.4.4 Controlling the Cursors

Selecting the "Split Cursor" menu item creates two time cursors that can be used to measure time differentials and define evaluation intervals. The time difference between the two cursors is displayed in parentheses at the bottom of the window. Placing the arrow cursor at a point of interest and pressing the left

mouse button moves the left most time cursor to this new location. The right most time cursor can be controlled in the same way using the right mouse button. Select this menu item again in order to rejoin the cursors.

A.4.5 Viewing a Slice of the Whole Picture

When there are two time cursors within the Graph Play, FU Activity, or Resource Envelope window, an enlarged view of the activity bounded between the cursors can be obtained by selecting the "Slice" menu item. Using the scroll bar controls at the bottom of the window allows you to slide this "slice" left or right in order to view activity outside of the viewing window. The step increment used when the slice is stepped left or right (using the scroll bar arrows) can be user defined. Selecting the "Increment Factor" menu item will display a dialogue box for the input of a new increment factor. The default step increment is equal to 100. Selecting the "Whole" menu item returns a view of the whole picture.

A.4.6 Automatic Measurement of TBI, TBO, and TBIO

A window listing TBI, TBO, and TBIO performance measurements can be created from the Analysis Tool window by selecting its "Performance" menu item. Measurement values are listed for every data packet processed by the predefined sink. Also listed are the associated data packet numbers and the number of resources in the system when the data packet was consumed by the sink.

Selecting the "Source" or "Sink" menu item allows you to define the source or sink number, respectively, used to obtain these measurements. The default source and sink numbers are associated with the first source and sink activities

occurring in the FDT file, respectively. The Performance window can be made an icon or closed from its system menu.

A plot of TBO verses TBIO can be viewed by selecting the "Operating Point" menu item. The intersection of a vertical and horizontal line represents the TBIO and TBO axis values. Placing the arrow cursor at a point of interest within the Operating Point window and pressing the left mouse button causes the vertical and horizontal lines to intersect at the point. The (TBIO, TBO) coordinate value associated with the intersection of the two lines is displayed at the bottom of the window. The window can be closed from its system menu.

A.4.7 Measuring Concurrency

A resource utilization envelope can be displayed for concurrency measurements by selecting the "Resource Envelope" menu item from the Analysis Tool window. Either a "Single Resource Envelope" or a "Total Resource Envelope" can be viewed by choosing the respective menu item. The data packet number that the single graph play is based on can be defined by selecting the "Packet Number" menu item. Enter the desired packet number when the dialogue box appears. A default packet number equal to one is initially used.

Selecting the "Utilization" menu item creates the Utilization window. This window provides functional unit utilization measurements. The measurements are based on a time interval defined by the Resource Envelope window's cursor positions. Each time the "Utilization" menu item is selected, the utilization measurements are updated with respect to the present cursor positions. If the

time between the cursors is zero a default evaluation interval is used. The default evaluation interval is equal to the time between the first node activity and the last event in the FDT file.

Both the Resource Envelope window and the Utilization window can be made icons or closed from their respective system menu.

A.4.8 Measuring Overhead

A bar graph providing overhead measurements can be viewed from the Overhead window by selecting the "Overhead" menu item from the Analysis Tool's window menu. When the Overhead window is first created, the measurements are based on a time interval equal to the time between the first node activity and the last event in the FDT file. A new evaluation interval can be defined from either the Graph Play, FU Activity, or Resource Envelope window by using the time cursors and selecting the "Overhead Interval" menu item within the respective window. The Overhead window will automatically update its measurements with respect to the new evaluation interval. The Overhead window can be made an icon or closed from the system menu.

A.4.9 Getting Help

A help window providing help on the Analysis Tool, Graph Play, FU Activity, Performance, Resource Envelope, or Overhead window can be displayed. Select the desired Help window from the "Help" menu within the Analysis Tool window. The Help window can be made an icon or closed from the system menu.

A.4.10 Selecting Colors or Patterns

A print-out of the Analysis Tool displays is possible with the use of a screen dump program. It is possible to switch from colors to patterns for the purpose of printing. Selecting the "Pattern" menu item within the Analysis Tool window "paints" the graphic images with patterns instead of colors. Selecting this menu item again allows the use of colors once again. A Microsoft Window screen dump program is included with the Analysis Tool package.

NASA

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-187625		

4. Title and Subtitle	5. Report Date
ATAMM Analysis Tool	October 1991
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Robert Jones, John Stoughton, Roland Mielke	10. Work Unit No. 590-32-31-01

9. Performing Organization Name and Address	11. Contract or Grant No.
Old Dominion University Research Foundation P. O. Box 6369 Norfolk, Virginia 23508-0369	NCC1-136
	13. Type of Report and Period Covered Contractor Report 6/1/89-5/31/90
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Langley Research Center Hampton, Virginia 23665	14. Sponsoring Agency Code

15. Supplementary Notes

Technical Monitor: Paul J. Hayes, ISD-Information Processing Technology Branch
Robert Jones is currently employed at Langley Research Center.

16. Abstract

Diagnostics software for analyzing ATAMM (Algorithm to Architecture Mapping Model) based concurrent processing systems is presented in this report and is a new model developed by researchers at Old Dominion University and the NASA Langley Research Center. ATAMM is capable of modeling the execution of large grain algorithms on distributed data flow architectures. The tool graphically portrays algorithm activities and processor activities for evaluation of the behavior and performance of an ATAMM based system. The tool's measurement capabilities indicate computing speed, throughput, concurrency, resource utilization, and overhead. Evaluations are performed on a simulated system using the software tool. The tool is used to estimate theoretical lower bound performance. Analysis results are shown to be comparable to the predictions.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Diagnostic software Dataflow architecture Multicomputer operating system Petri nets Concurrent processing	Unclassified - Unlimited Subject Category 33

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
Unclassified	Unclassified	106	A06