Completion and Validation of The Design of a Reconfigurable Image Processing Board.

by

Nitin Deo

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Electrical Engineering

APPROVED:

Dr. Morton Nadler, Chairman

Dr. C. E. Nunnally

Dr. J. C. McKeeman

June, 1985

Blacksburg, Virginia

Completion and Validation of The Design of a
Reconfigurable Image Processing Board.

by

Nitin Deo

Dr. Morton Nadler, Chairman

Electrical Engineering

(ABSTRACT)

Starting in September 1984, the Telesign project is an extensive and complex project proposed and undertaken by Dr. Nadler at Virginia Tech. The emphasis of this project is to enable the members of the deaf community to communicate visually using sign language or lip reading over the telephone network.

The Image Processing Board (IPB) is the 'Brain' of the whole system. The IPB processes a given frame of an image to transmit only selected data. It uses the pseudo-laplacian operator, invented by Dr. Nadler, for edge detection. According to a recent survey of various edge detection algorithms by D.E.Pearson, [1], the pseudo-laplacian operator is the most efficient one and it produces the most natural pictures.

The whole IPB hosts about one hundred LSI/VLSI chips according to the present hardware description. In the case of such a big system, hardware simulation becomes mandatory in order to ensure reliability of the design and to anticipate any kind of logic or timing errors in the design. This thesis describes the modifications to the original design to make it reconfigurable with proper initialization and the Hardware Simulation of the IPB, using General Simulation Program (GSP), including some comments on the simulators available at Virginia Tech and in particular a critique of the simulator used here. Many improvements to the simulator are suggested. Precautions to be taken while preparing the lay-out and wiring of the IPB, suggestions to simplify the design at some points at the cost of a few more chips, and lastly the instructions to run the models to get the required results, are outlined in this thesis.

Attention Patron:

Page __iv__ omitted from numbering

ACKNOWLEDGEMENTS

My deepest gratitude goes to Dr. Nadler, a guide and a friend. It has been an honor to study under him. His help, guidance and constructive criticism will always be treasured.

Thanks are due to Drs. C. E. Nunnally and J. C. McKeeman, my committee members. They have invested much time and support directly and indirectly.

Many others have supported me personally and I am grateful to , , and . I thank all the Telesign Project members for their helpfulness and support. This acknowledgement will not be complete without my thanks to my wife, who stood with me through pleasure and pain.

I dedicate my work to my Parents without whose love and care I would be nothing.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

1.0 INTRODUCTION.

The purpose of this section is to introduce not only the IPB and GSP but also the overall Telesign project as such. Figure 1 on page 2 shows the overall system as it is going to be installed in Gallaudet College, Washington D.C. This section also explains the need for simulation, outlines the simulator used and gives an overview of the IPB structure.

1.1 THE TELESIGN PROJECT

Telesign is designed to offer a means of visual communication over a 56 or 64 kb/sec data network. The purpose is to supply a means of visual telecommunication among the members of the deaf community using sign language or lip reading. The system consists of an edge detector, followed by digital compression coding to meet the channel requirements. Psychometric experiments have shown the need for 25 frames/sec with a minimum definition of 128 x 128 points [2].

The telesign project consists of six different subsystems, inter-related in one way or the other. There are six graduate students working on each subsystem under the guidance of Dr. Nadler.

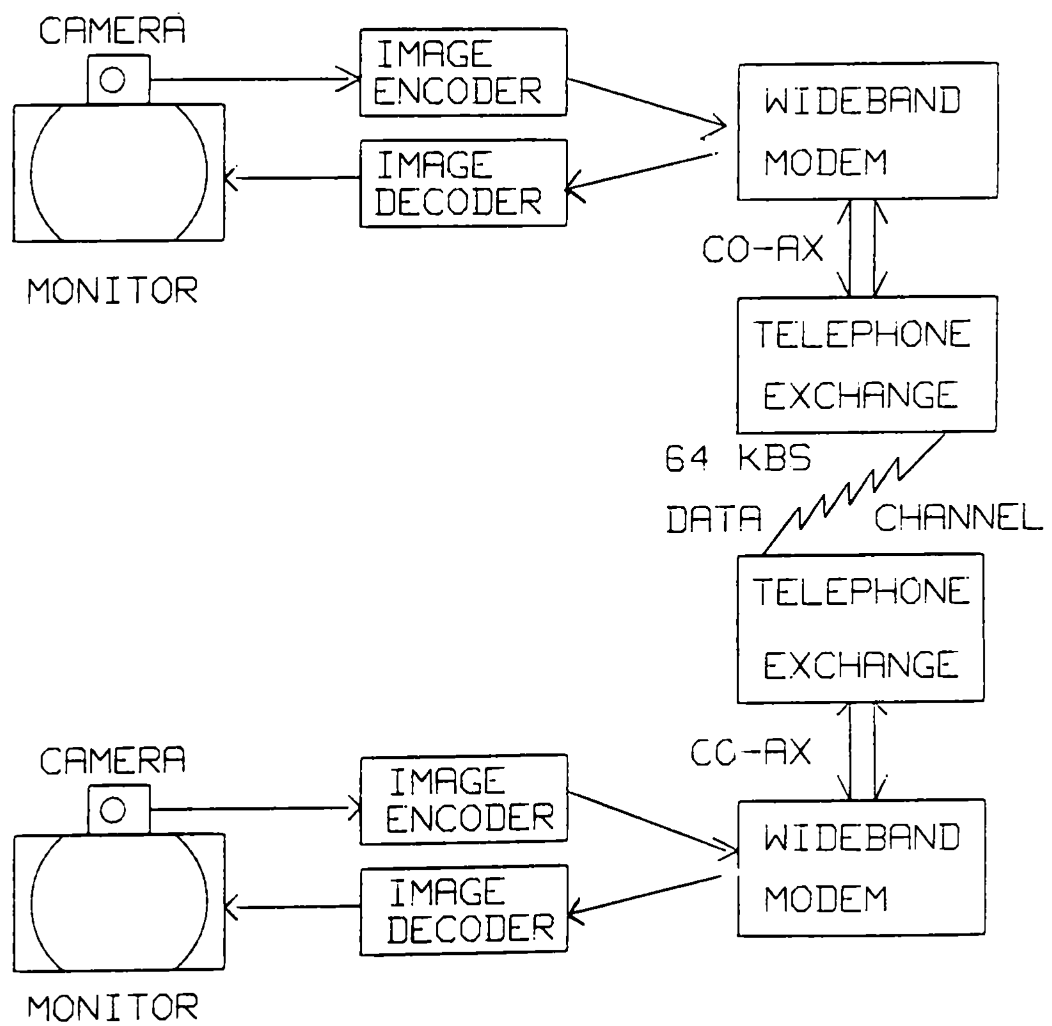


Figure 1. Telesign Project: Visual Telecommunication for Deaf.

1. Software Simulation of the NAD[1] operator : The edge-detection by NAD operator was first written in PASCAL and was verified by Dr. Nadler. NAD was then transfered to GIPSY (General Image Processing SYstem) on VAX 11/785, for convenience. The actual images are processed by this operator and are used as a reference while validating the Hardware Design of the IPB.

2. Hardware Simulation of IPB : Taking the input/output data from the GIPSY program for each module, the IPB was simulated module-by-module.

3. Camera and Data Acquisition System : A video camera is being built, which will create an image of 256 x 256 pixels.

4. Video Processing Board (VPB) : The VPB transfers the data from the camera to the IPB through Link board, and from IPB output displays on the monitor of the other station. It also displays actual image and the processed image from both the stations on a central monitor.

5. Link Board : Link transfers initialization sequence from VAX 11/785 to the IPB, links IPB and VAX 11/785 during

---

[1] The pseudo-laplacian edge detector with blackfill.

debugging of the hardware and it also links all the boards to each other to monitor the data transfer.

6. Smoothing of the Images: At the output of the system, before displaying.

Figure 2 on page 5 shows the interconnections of all these boards in the system.

1.2 WHY SIMULATE ?

The purpose of this section is to discuss a topic which is of crucial importance to all those engaged in designing hardware of big systems, for example image processors, robot controllers, microcomputers etc. Objective tests are used in verification and validation which help the modeller to determine that the model works; and it also contributes to the additional subjective factors which come into play when a third party is to be convinced [3].

Consider first the objective criteria for establishing the credibility of a model. Figure 3 on page 6 depicts a simple representation of the modelling process.

A conceptual model of the real system is produced by making assumptions about variables and system parameters. The conceptual model is converted to a computer model by programming, punching, and program entry.

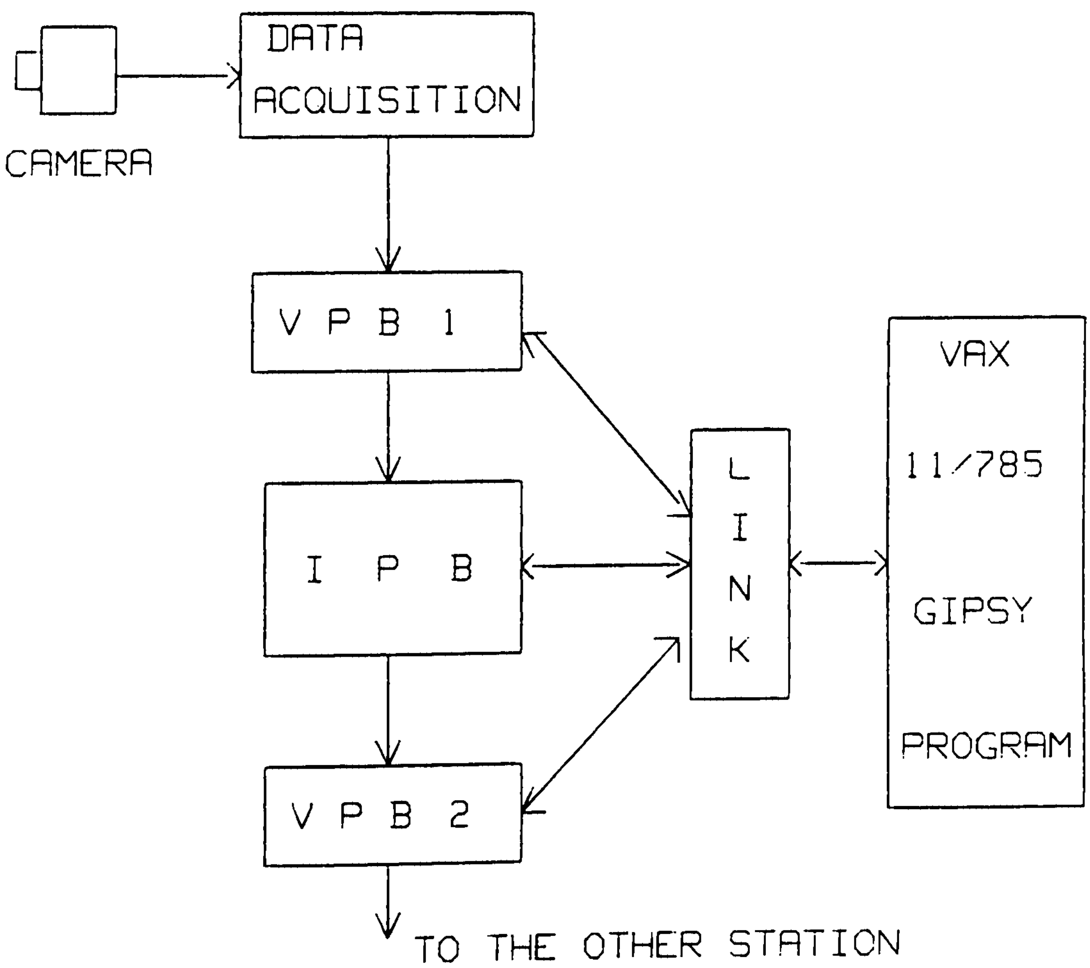


Figure 2. Interconnections in the Hardware of Telesign.

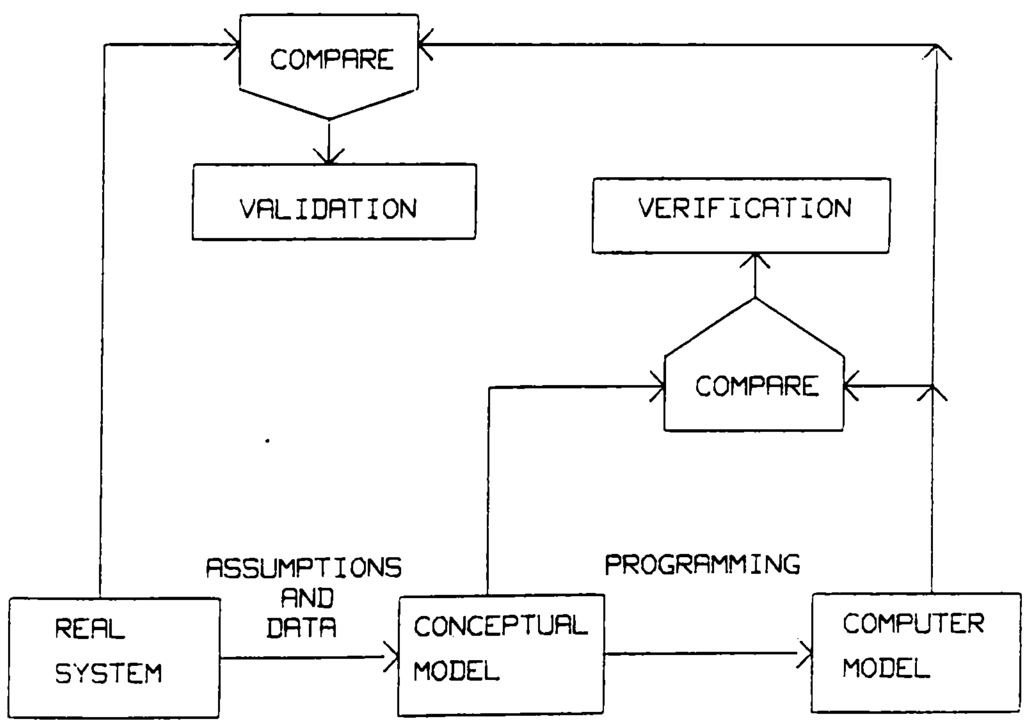


Figure 3. The Modelling Process.

The process of establishing that the conceptual model truly represents the real system is validation, and the correlation of the computer model and the conceptual model is verification. Verification is concerned with the elimination of programming and punching errors and the reduction of numerical approximation errors to an acceptable level. Verification, in short, establishes that the specified equations are solved correctly within the computer model. Validation is a much more fundamental process which asks whether the equations are the right ones and whether the basic assumptions of the conceptual model are justified. In practice it is usual to test the outputs of the verified computer model (and hence conceptual model) with the real system as shown in the figure.

Thoroughly executed verification and validation procedures ought to go a long way toward establishing the credibility of a computer model, and will usually be sufficient to convince any modeller, however sceptical.

1.3 GENERAL SIMULATION PROGRAM [GSP]

The simulator used for this complicated architecture has been developed in Virginia Tech by Dr.J.R.Armstrong. This program provides functional simulation capability as well as the ability to simulate chip interface timing. Chip modeling utilizes chip input/output specifications and timing diagrams. Construction and coding of the model is a process very much akin to assembly language programming. [5].

GSP is a program suitable for the simulation of LSI/VLSI devices as it allows for a manageable amount of detail in model descriptions and it is claimed that it can simulate with an efficiency that is adequate for system validation activities.

Until now all the extensive work done on GSP has been more under fault modeling and fault simulation. The program was developed keeping this in mind. So, some people dealing with hardware simulation may not find (or have not found) this simulator particularly suitable for their application. There are certain constraints or restrictions of GSP. And some of those people have designed their simulators to suit their requirements, rather than circumventing those constraints of GSP. As a result the simulators they have designed are suitable for them and may not be suitable for others. In other words, it is very difficult to build a Universal Simulator. On the other hand, taking into consideration the complexity, the number of chips involved, size of each module in IPB design and the time factor, one would rather circumvent the constraints of GSP than writing a new simulator. After all the main purpose of this work is to simulate the IPB design to validate it. The purpose is not to design the most efficient simulator and then simulate IPB design as an illustration.

This is the first attempt made to simulate an image processing hardware using GSP. This is an ideal illustration of how an engineer can utilize the tools available to get what is desired. Also, it emphasises the need for simulation of hardware design.

The GSP models described in The GSP User's Guide [5] and in [6], are of a single chip each. These models exactly define the I/O pins, behaviour of the chip and timing. While simulating IPB this is not always true. At times a group of chips is treated as a single chip, with input pins of the chips at the top of the model as input pins to the module and similarly output pins of the chips at the end of model as output pins of the module. But the behaviour of each chip in that module is perfectly defined, although it may be transparent to the user.

1.4 THE IPB ARCHITECTURE.

The IPB architecture is pipelined byte-sliced. Each byte is a pixel given to IPB by the Video Camera output buffer. In order to have consistent structure, each pixel (a byte) is processed at every stage in the pipeline in 8 clock cycles. Thus, 8 clock cycles make one pixel time. A pipeline is divided into five different 'tubes' or modules. During the first pixel time, the first module operates on the first pixel, hands over the processed byte to the next module, and so on. The five modules are: Find_Diffs, Sum_Diffs, Eval_Diffs, Filters, and Set_Rel_T. Figure 4 on page 11 shows the interconnections between these modules.

The process of edge-detection and black-filling are accomplished by a series of interconnected functional units [7]. The modules can be described in brief as follows:

1. Find_Diffs :- Compute the differences between graytones according to the vector directions supported in the laplacian mask, and then threshold those differences against TDIFF[2], flagging those differences above the threshold.

2. Sum_Diffs :- Generate all of the laplacian vectors from the supplied mask input and count the numbers of black and white edge elements, respectively.

3. Eval_Diffs :- Determine the black and white edge discontinuities in the input image using T1, T2, T3. See footnote[2].

4. Filters :- Remove isolated black and white points using the neighborhood connectivity criterion.

---

[2] These are thresholds, see section 4.1.

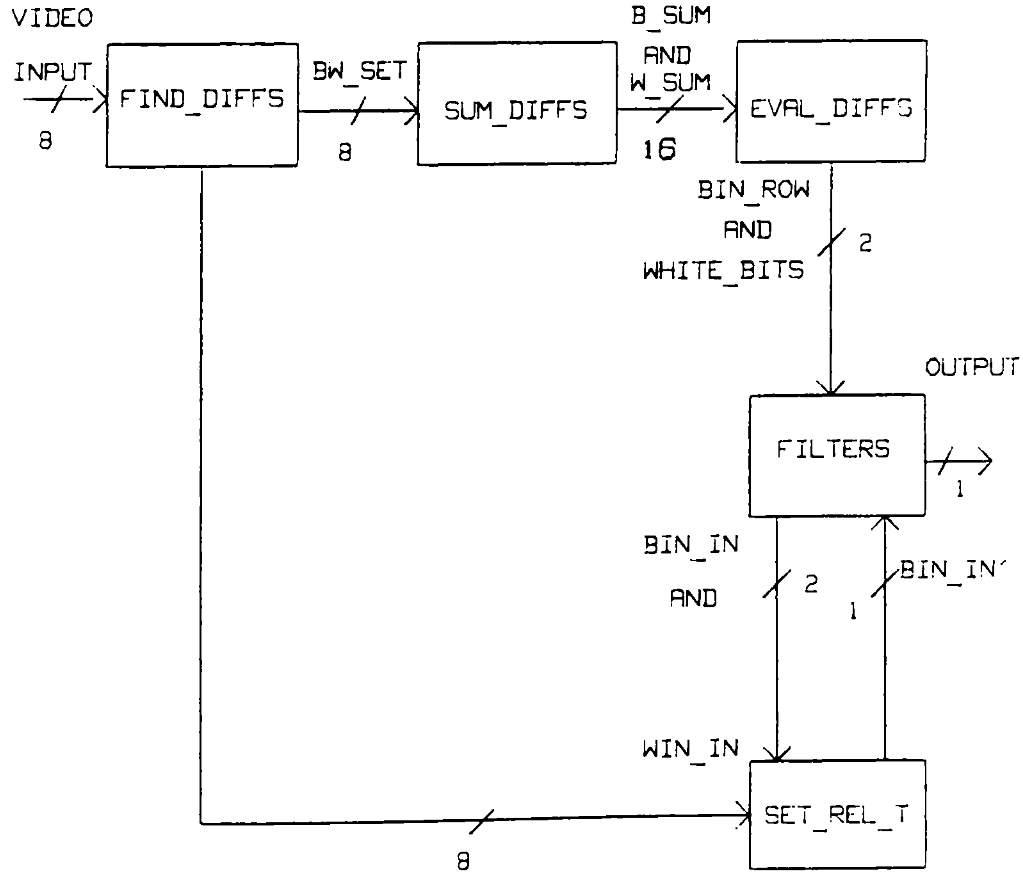


Figure 4. The Interconnections between the modules of the IPB

5. Set_Rel_T :- Blackfill selected regions between the black edges of the image, using the white edge image and TBF, see footnote[2].

The detailed description of each module is given in [4].

The pipelined architecture facilitates module-by-module simulation and validation of the design.

2.0 A CRITIQUE OF VARIOUS SIMULATORS.

Selection of a suitable simulator for the simulation of the IPB was one of the major decisions of this work. Mainly three different types of simulators were reviewed. Various aspects of all these simulators were considered, some of the drawbacks of the simulators were thought over and the possibility of overcoming those drawbacks was considered. The main criterion for the selection of the simulator was the critical timing specifications. It was desired that the simulator should specify the state of the board at every clock cycle. The maximum clock frequency being 32 MHz, minimum clock period becomes approximately 31.7 ns. Thus, the simulator should be able give an output at every 32 ns. The second criterion for the selection of a simulator was its practicability for simulation of about 90 LSI/VLSI chips. The other criteria were simplicity of output interpretation, simplicity of the language, availability of literature and expertise on the simulator, etc.

The simulators under consideration were the ones which were readily available at Virginia Tech, namely, ISPS, TILADS, SPLICE and GSP. There were some drawbacks of each simulator. (These drawbacks mentioned here are only from the point of view of this thesis work, they may not be generalised.) It is not claimed that the selected simulator is totally flawless, but only that the selected simulator has certain advantages over the other from this work's point of view. Some suggestions regarding improvements in the selected simulator are listed in Section 5.1.

2.1 ISPS.

First it was intended that ISPS (Instruction Set Processor Specifications) be used. So, the literature on ISPS was made available for review. ISPS was criticised a few years ago by four scholars and the criticism was presented as a paper in the 4th International Symposium on Computer Hardware Languages [8]. A part of that paper will be reproduced here, with some comments.

"ISPS was invented with the intent of using it for many applications of machine description languages. These applications include description of the behavior of arbitrary register-transfer level circuits: image processors, display processors, video terminals etc. Description of the physical hardware itself, however, has not been one of the goals of ISPS and attempts to apply ISPS in this manner have resulted in a set of pathological examples which illustrate the inadequacies of ISPS for this purpose." Simulation of IPB is definitely categorised as hardware simulation. Thus it was found that ISPS was not practicable for this use. Further discussion will justify the rejection of ISPS for this work.

" Our criticisms of the language could fall into several categories: semantics, syntax, and support software. Each of these in itself is important. We will emphasize, however, issues dealing with semantics because without these the other issues are moot points.

" From the point of view of ISPS as an instruction set representation, the chief weaknesses are: 1) The lack of abstractions to cover certain operations common in hardware systems, and 2) The need for a constrained structure to the description. The latter problem can be remedied through the use of qualifiers identifying sections of the description (the computations, the processor state, the instruction interpretation, etc.); this structure is explicitly not desirable in ISPS for general digital system design. On the other hand the former problem, concerning abstract operations, is shared by general system designers as well. For example:

> "No operator exists for transfering a block of memory into another block, or extracting a field from a variable position in register. As a consequence, it is necessary to describe such instructions indirectly, with loop or shifts and masking. The detection of the higher level functions expressed in terms of these more involved descriptions is difficult for both software and hardware synthesis programs. There are some hardware structures which are now considered to be hardware primitives. These

include FIFO buffers and LIFO stacks, associative memories, and large AND-OR arrays of logic (PALs). Descriptions of these functions in ISPS produce lengthy code; many I/O operations such as code conversion and buffering require these structures.

" Real time processing applications with interrupt procedures that service peripheral (asynchronous) devices, require synchronisation between the producer of data, say the main program, and the consumer of the data, say the device. Catastrophic effects can occur if the proper mutual exclusions and signalling conventions are not used to maintain data integrity."

This means that ISPS cannot describe real time processing and operating systems. This is the biggest drawback of ISPS for the intended use. But, realtime simulation with details down to every 32 ns was one of the main objectives of this work. Hence, ISPS could not be used for the IPB hardware simulation. From this discussion it is obvious that rather than spending time in discovering ways to circumvent all these major flaws in ISPS, it was better to consider another simulator.

2.2 SPLICE AND TILADS.

Another simulator under consideration was SPLICE. This simulator is used quite extensively in VLSI design simulation these days in industries. It was used extensively at Virginia Tech by the author of this thesis for studying an effect of three phase clocking system on VLSI chips, according to the ideas put forward by Dr. Nadler [9]. Although SPLICE is good for simulating the realtime behavior of a system, it is a GATE Level as well as CIRCUIT Level simulator. Since, a CHIP Level or FUNCTIONAL Level simulator was necessary, SPLICE was not chosen.

There were some other simulators such as TILADS (Texas Instruments Logic And Design Simulator). There are two versions of this particular simulator: Internal and External (to Texas Instruments, obviously!) At Texas Instruments the internal version of TILADS is extensively used. The version available at Virginia Tech was External version. The Internal version of TILADS is much more sophisticated than the external version. Both versions could simulate in real time. They have no restriction on the number of modules they can handle and can transfer one memory bank to another and so on. But lack of simplicity of language was a disadvantage of this simulator. So, TILADS was rejected for this use.

2.3 GSP

As mentioned earlier, the main criterion for selection of a simulator was the critical timing specifications. Out of all the simulators considered, it was found that only GSP is capable of giving the details about the state of the board at every clock cycle. GSP can go as deep as one unit of time, e.g. ps or ns. Once the unit of time is fixed, it is necessary to specify all timing in that unit itself. Since the minimum clock pulse is approximately 32 ns, if the unit is fixed to ns, GSP can give the state of board at every ns if and when desired. 'State of board' is given by indicating the state of each input/output pin, contents of index registers and/or contents of desired registers. Hence with respect to this criterion GSP was the ideal simulator.

The second criterion was the practicability of the simulator for such a massive simulation. GSP can link as many as 16 modules at a time. Each module can consist of any number of chip-models. A chip-model need not model a single chip, it may model a group of chips by merging the individual chip-models. In fact that is the methodology adapted for this simulation. It is explained in detail in the next chapter.

Then came the question of availability of literature and expertise. Since GSP was developed at Virginia Tech itself, there was absolutely no difficulty in obtaining the required literature. It will be very clear later how with the help of an expert in GSP a major problem was solved very fast. Although, most of the work in progress at Virginia Tech using GSP falls under the category of fault modeling and fault simulation, the knowledge of the structure of GSP and its applications was the best help one could ever get. Secondly, as will be explained in the next section, the simulation language of GSP is very much akin to any other assembly level language. That was an added advantage of GSP.

Taking into consideration all these points, the selection of GSP for this large scale simulation is quite justified. And with that GSP has become the first simulator to simulate a pipelined, byte-sliced, microprogram-contolled image processing hardware design.

2.4 GSP: THE SIMULATION LANGUAGE.

The functional chip models were prepared using GSP (General Simulation Program) [5,6,10]. GSP is a general purpose, two-valued (1,0) simulation language, developed at Virginia Tech specifically to perform the simulation of VLSI devices at chip level [10]. Its most useful application is the modeling and simulation of complicated VLSI circuits and microprocessors. The language has been used extensively for modeling functional-level faults in simple and complex VLSI devices. It also has the capability to model such interface timing specification as setup time, hold time and minimum pulse width.

Modeling in GSP is done in an assembly language with special instructions for hardware description. The GSP manual, [5], contains a detailed explanation on the instruction set and the utilisation of each instruction.

Figure 5 on page 21 shows the GSP simulation system structure. Each module description file is assembled to obtain a microcode file. The microcode files are merged together with the states into the LINK file. The DATA file has the information on module interconnections, initializations and inputs. The simulator reads the data file at the beginning of simulation and executes the microcode during simulation, generating the outputs.

2.4.1 METHODS OF FUNCTIONAL MODELING

The modeling process involves :

1. Detailed examination of manufacturer's specifications,

2. Generation of the model flow-chart,

3. Coding of the model, and

4. Model checkout to verify the correctness of the model.

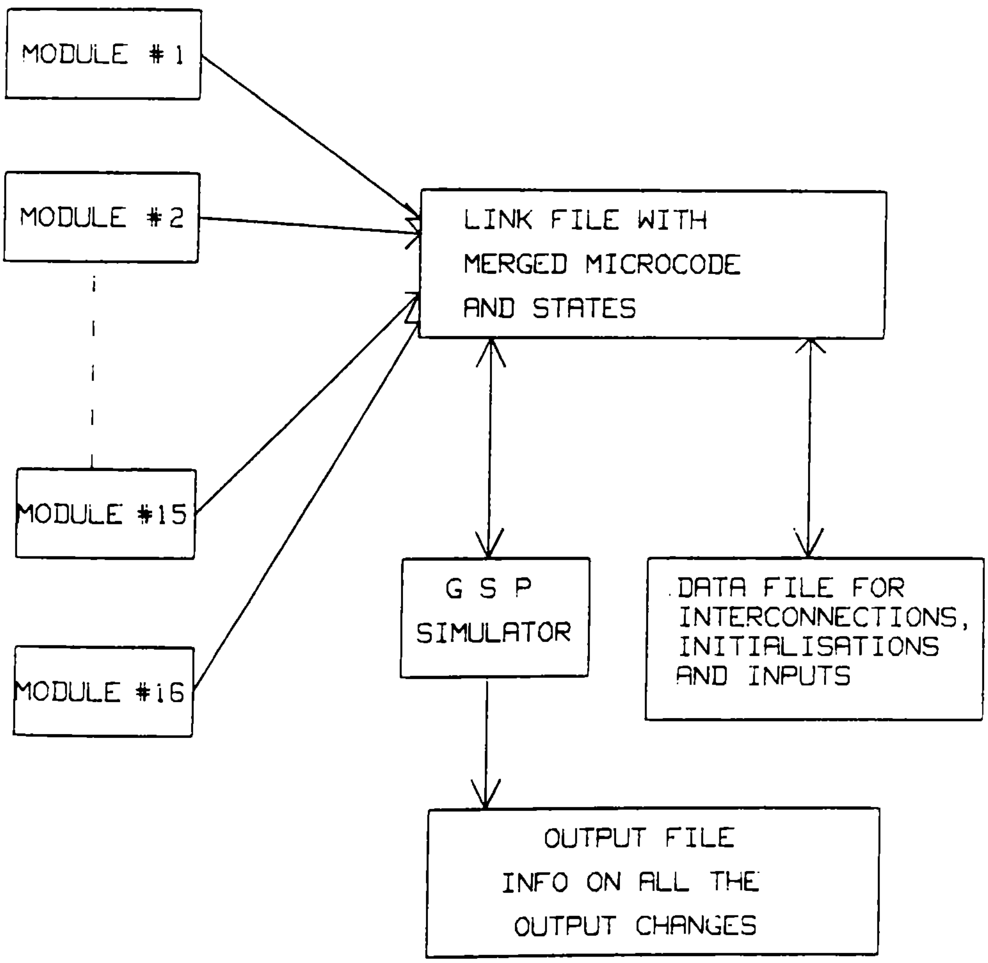


Figure 5. GSP Simulation Structure.

The functional-level models can be prepared in different ways. Two general methods for modeling digital devices at the functional-level are shown below :

FUNCTIONAL LEVEL MODELING

LOOK-UP TABLE MODEL MICRO-OPERATION MODEL

2.4.1.1 Look-up Table Model

In this method, the functional unit is represented in the form of a truthtable (combinational logic) or a state table (sequential logic). The discussion here pertains to combinational logic. In order to access a particular value in the truth-table, the inputs to the functional unit are decoded to point to the location containing that value in the 'Look-up Table'. Figure 6 on page 24 shows the hypothetical similarity between the actual function and its look-up table model. As evident, this is a very simple approach to modeling. Several such truth-tables for the different functions are put together to form the model for the whole device. Also, the functional units that are repeatedly used in the device can be made into subroutines and 'called' whenever needed, during the data flow of the device.

In GSP, the decoding constructs are used to perform this operation. The example in Figure 7 on page 25 describes the 'look-up table' model for an And-Or-Invert function of three inputs, F(x1,x2,x3) = (x1x2 + x2x3 + x3x1)'. As can be seen from the figure, the number of bits of the input register that are to be decoded, are moved into one of the index registers (index register 1, in the example). The index register is used as the pointer to the locations of a table (table AOI, in the example), and the value contained in the location pointed to by the contents of the index register is then moved out to the destination (pin OUT) after a delay of 40 ns. (DEL1).

2.4.1.2 Micro-operation Model

In this approach, the functional unit is defined as a sequence of model micro-operations, using the constructs of the modeling language. The functional model can be viewed as a nodal graph with two kinds of edges interconnecting the nodes. Each node is a set of model micro-operations and control and data get passed from one node to another along the edges. The dashed lines in Figure 8 on page 26 indicate control transfer while the solid lines indicate the passage of variables from one node to another.

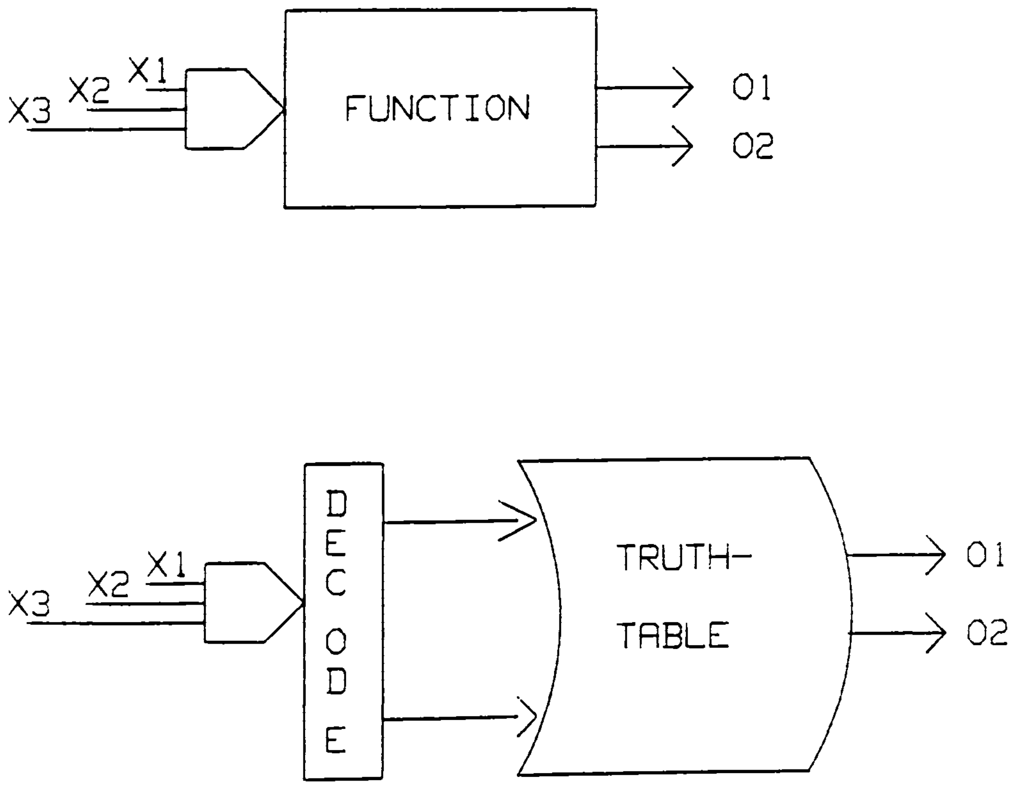


Figure 6. Actual Function vs. Look-up Table Model

AND-OR-INVERT (AOI)

```
; registers for the model
;
REG(3)    OLDX
;
; pins for the module  X1,X2,X3 : 1,2,3 ; AOI : 4
;
PIN       X1X3(1,3),OUT(4)
;
; delays for the module
;
EVW       DEL1(40)
;
; module description
;
     BNE   X1X3,OLDX,PRO   ; START PROCEDURE IF DATA
     EXR                   ; IS CHANGED
;
PRO: MOV   X1X3,OLDX       ; STORE FOR NEXT CHECK
     IDX   OLDX(0),3,1     ; STARTING WITH OTH BIT,
                           ; MOVE 3 BITS INTO INDEX REG.1
     MOV(DEL1) AOI@1,OUT   ; MOV THE CONTENTS OF LOCATION
                           ; POINTED BY INDEX REG.1 TO
                           ; THE OUTPUT, AOI, AFTER DEL1.
     EXR
;
;LOCATIONS 0  1  2  3  4  5  6  7
;
AOI : BYT #1,#1,#1,#0,#1,#0,#0,#0
;
END
```

Figure 7. "Look-up Table" model

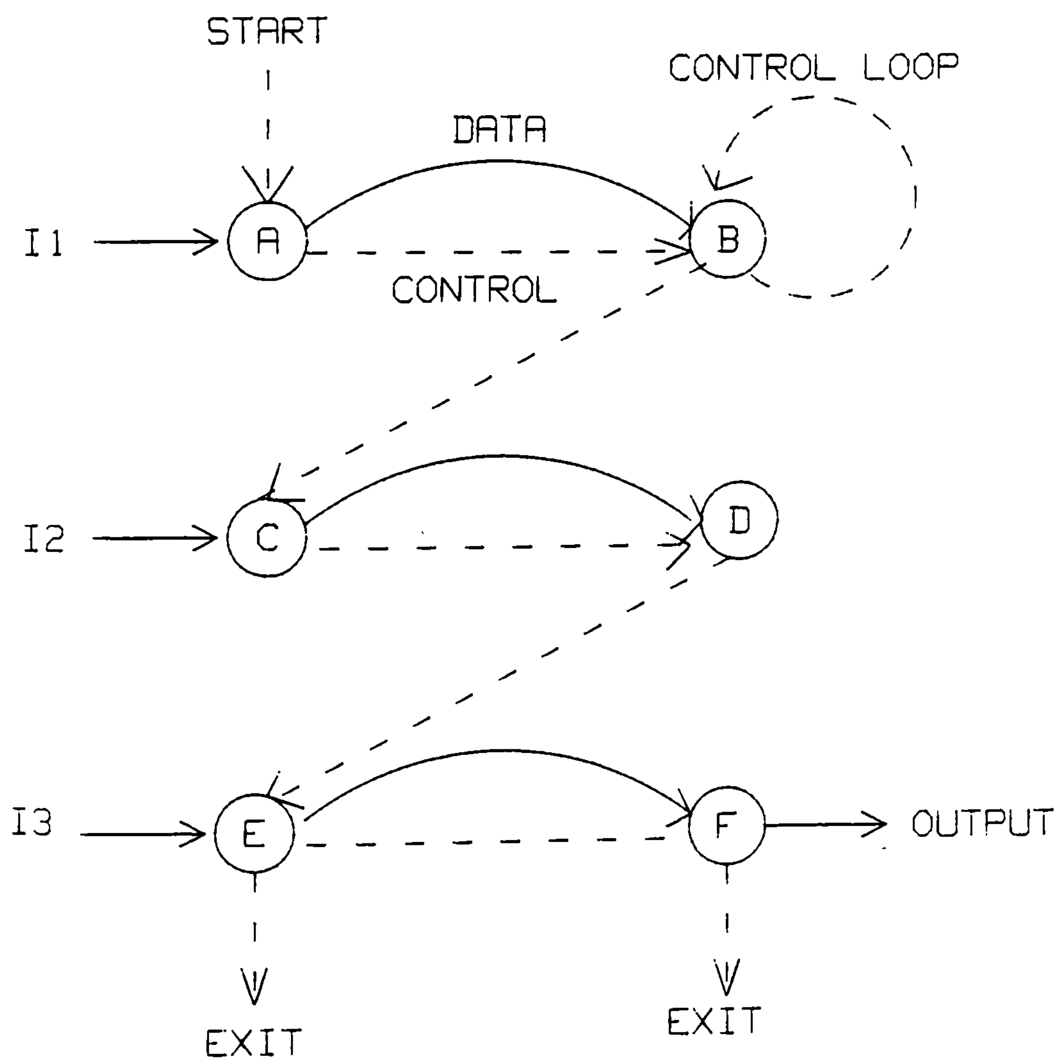


Figure 8. Graphical representation of Micro-operation model

The functional unit is not described in terms of the inputs and the truth table as in the previous case; instead, modeling language constructs are used to manipulate the data and obtain the resultant output. The example in Figure 9 on page 28 describes the 'micro-operation' model for an And-Or-Invert function of three inputs, similar to the one in Figure 7 on page 25. In GSP, modeling constructs such as AND, OR, and NOT are used in a sequence of micro-operations which yield the final output.

AND-OR-INVERT (AOI)

```
; registers
REG(1)      OLDX1,OLDX2,OLDX3
REG(1)      AND12,AND23,AND31
REG(1)      ORBUF
; pins
PIN         X1(1),X2(2),X3(3),OUT(4)
; delays
EVW         DEL1(40)
;
; description
;
      BNE  X1,OLDX1,PROC   ; BRANCH IF VALUE CHANGED.
      BNE  X2,OLDX2,PROC
      BNE  X3,OLDX3,PROC
      EXR
;
PROC: MOV  X1,OLDX1        ; FOR COMPARISON ON NEXT
      MOV  X2,OLDX2        ; SIGNAL CHANGE OF X1, X2, X3.
      MOV  X3,OLDX3        ;
;
      AND  X1,X2,AND12     ; (AND12) = (X1) * (X2)
      AND  X2,X3,AND23
      AND  X3,X1,AND31
      OR   AND12,AND23,ORBUF
      OR   AND31,ORBUF,ORBUF ; DESTINATION =ORBUF ITSELF
      MOV(DEL1) ORBUF, OUT
      EXR
END
```

Figure 9. "Micro Operation" model

3.0 THE STRUCTURE OF IPB.

3.1 PIPELINE ARCHITECTURE.

Computer architects have long resorted to a series of design techniques that are classified under the general term of 'concurrent operation', where at any instant the hardware is simultaneously processing more than one basic operation. Within this general category are two well-recognised techniques, parallelism and pipelining. High performance is attained by having all structures execute simultaneously on different parts of the problem to be solved.

Pipelining generally takes the approach of splitting the function to be performed into subfunctions and allocating separate hardware to each subfunction, termed a stage or a module or a "tube". The pipeline has separate logic for each of the subfunctions, with staging latches positioned between each set of logic to hold the output of the stage for processing by the next stage. Every operation follows the same path through the stages. Further, the time required by each stage to do its subfunction is about equal, and the transitions from stage to stage are rigidly controlled by an external timing source [11].

Pipelines are classified both according to their capabilities and according to how they are actually used. A unifunction pipeline is one that is capable of only one basic kind of function evaluation. The pipeline performs the same operations on every set of inputs given to it with no variations. A multifunction pipeline is one that is capable of several different kinds of function evaluations. Thus in addition to the data inputs, there is some kind of control input directing the pipeline's activity. The pipeline of IPB can be classified as multifunction.

The subfunctions of the main function of the pipeline have following properties:

1. Evaluation of the basic function is equivalent to some sequential evaluation of the subfunctions.

2. The inputs for one subfunction come totally from the outputs of previous subfunctions in the evaluation sequence. (This is not true in the case of pipelines with feedback.)

3. Other than the exchange of inputs and outputs, there are no inter-relationships between the subfunctions.

4. Hardware may be developed to execute each subfunction.

5. The times required for these hardware units to perform their individual evaluations are usually approximately equal.

There are a few precautions to be taken, before a pipeline is constructed. A major concern is the delay introduced by wiring, including the wiring between individual components in a board, and between boards (particularly through connectors). These delays are due to the limited propagation speed of electrical signals in the wiring and cannot be avoided. Intercomponent wiring delays approximately match the logic delays. Special care can be taken as to what logic is placed on what boards. The system can be split up by partitioning logic components onto boards or modules. There is one board for each stage. Consequently, the time for each stage must include an interboard delay. Although proper placement of boards can minimize this delay, and proper selection of wire lengths equalises the effects, in general the clock pulse rate must be slower than the intrinsic logic would otherwise allow.

It is felt necessary at this point to explain parallelism because, although IPB is essentially a pipeline, it is partly parallel also.

Parallelism is to allow similar processing of different data to occur simultaneously or to allow different hardware to handle distinctly different parts of the problem. Both parallelism and pipelining have the same origins and are hard to separate in practice. Both techniques attempt to increase the performance of some function by increasing the number of simultaneously operating hardware modules. For a conventionally designed module to do some generic function, either technique can be used to drive a new design, running up to N times faster. A mixture of the two techniques results in an overlapped or a systolic design. As will be explained later in this chapter, the architecture of the IPB can be correctly described as "overlapped" or systolic.

3.2 MANAGEMENT BY MICROPROGRAMMING.

Microprogramming is too vast a topic to be covered in a page or two. Actually, the microprogram of IPB is so clear and neatly arranged that the theoretical explanation of microprogramming may become more complicated than the microprogram itself. The essential theoretical aspects will be covered in this section.

The most explicit definition of microprogramming is given by Daly, [12], "Microprogramming is a technique for designing and implementing the control function of a data processing system as a sequence of control signals, to interpret fixed or dynamically changeable data processing functions. These control signals, organized on a word basis and stored in a fixed or dynamically changeable control memory, represent the states of the signals which control the flow of information between the executing functions and the orderly transition between these signal states."

Microprogramming puts the control functions into a regular structure, isolating them from the data flow. An appropriate design flexibilty can be retained, if 'a functional perspective' is assumed to be the essence of microprogramming. Thus, microprogramming is a design philosophy and organizational method not limited by the implementation technology. It is the process of producing microprograms. A microprogram is a stored-program that explicitly and directly controls the major logic devices of a digital system. It is a substitute for a sequential-logic control network. At this point one should be able to distinguish between programming and microprogramming. There are two significant differences between the two: parallelism, a microinstruction can potentially cause many parallel events to occur; and a higher degree of asynchronous operations.

There are some substitutes for a microprogram control: random-logic, sequential-logic and hardwired control, i.e. one can design a logic circuit to perform the same function. If there are a few control signals to be generated, then probably microprogramming will be more costly and/or slower [13]. The advantages of micrprogramming for the designs such as the IPB are as follows:

1. A more orderly and uniform way of design.

2. Ease of change.

3. Cheaper for large systems.

4. More suited to LSI/VLSI environment.

5. Better diagnostic capability.

6. Higher system reliabilty.

The basis of bit-sliced (or byte-sliced) logic is the microprogramming. In fact bit-slice logic was designed with microprogrammed control in mind. From the design of IPB and its microprogram one should be able to see how Byte-slice logic and Microprogram blend so well!

3.3 PIPELINED, BYTE-SLICED, MICROPROGRAMMED IPB.

The design of IPB is a unique combination of pipeline, byte-sliced architecture with microprogrammed control. Actually, if one carefully observes the design of the IPB, one will come to know that it has some properties of a parallel architecture also. So, it can be said that the IPB has an Overlapped architecture. As a general rule, special characteristics increase the complexity of the design. That is the case with IPB also. One point is noteworthy; although we understand some complex designs, sometimes we are not able to explain it to others (especially to a layman) so well. It is said that a good scientist is rarely a good teacher. He can invent or develop incredible things, but may not be able to convey the idea to others properly. Similarly, from the technical background created until now, it appears very easy to explain the design of the IPB, but it is a very difficult task. And if it is so difficult to explain it to an intelligent reader, it is more difficult to explain it to a 'dumb' computer through a restricted language.

That was a rather philosophical image of the task of explaining the design of IPB and its simulation. It is felt essential at this point that an explanation be given about the edge detection algorithm used by the IPB - "pseudo-laplacian" or "NAD" operator.

3.3.1 PSEUDO-LAPLACIAN OPERATOR.

This edge detection algorithm has been tested thoroughly in software at INRIA, Rocquencourt, France. It was originally written in PASCAL by Dr. Nadler [14]. Consider the configuration of arrows in Figure 10 on page 37. Each of these arrows joins two pixels and represents the difference between the video intensities at the head and the foot of the arrow.

Each difference is compared in absolute value to a certain threshold T1, and if it is superior to that value, the 'sign' of the difference is retained. The positive and negative signs are then counted separately. The edge decision is taken if the following conditions are satisfied:

$$C(+) > T1$$

or

$$C(+) > T2 \quad \text{and} \quad C(+) - C(-) > T3$$

where $C(+)$ and $C(-)$ are the counts of positive and negative signs, respectively; and T1, T2 and T3 are the thresholds.

A configuration of differences with central symmetry signifies that only second-order finite differences appear. The sum of these differences thus corresponds to a finite-difference approximation to the laplacian. By the same token, we have a configuration with central symmetry of first-order finite differences. The difference $C(+) - C(-)$ consists in summing the unit differnces obtained in the preceding step, and comparing the difference in counts to T3; the condition $C(+) - C(-) > T3$ is analogous to thresholding in the laplacian.

Blackfill is a technique incorporated with the pseudo-laplacian operator to fill the space between two edges with maximum black. Based on blackfill, 'grayfill' and 'colorfill' are also suggested by Dr. Nadler for future enhancements in the system.

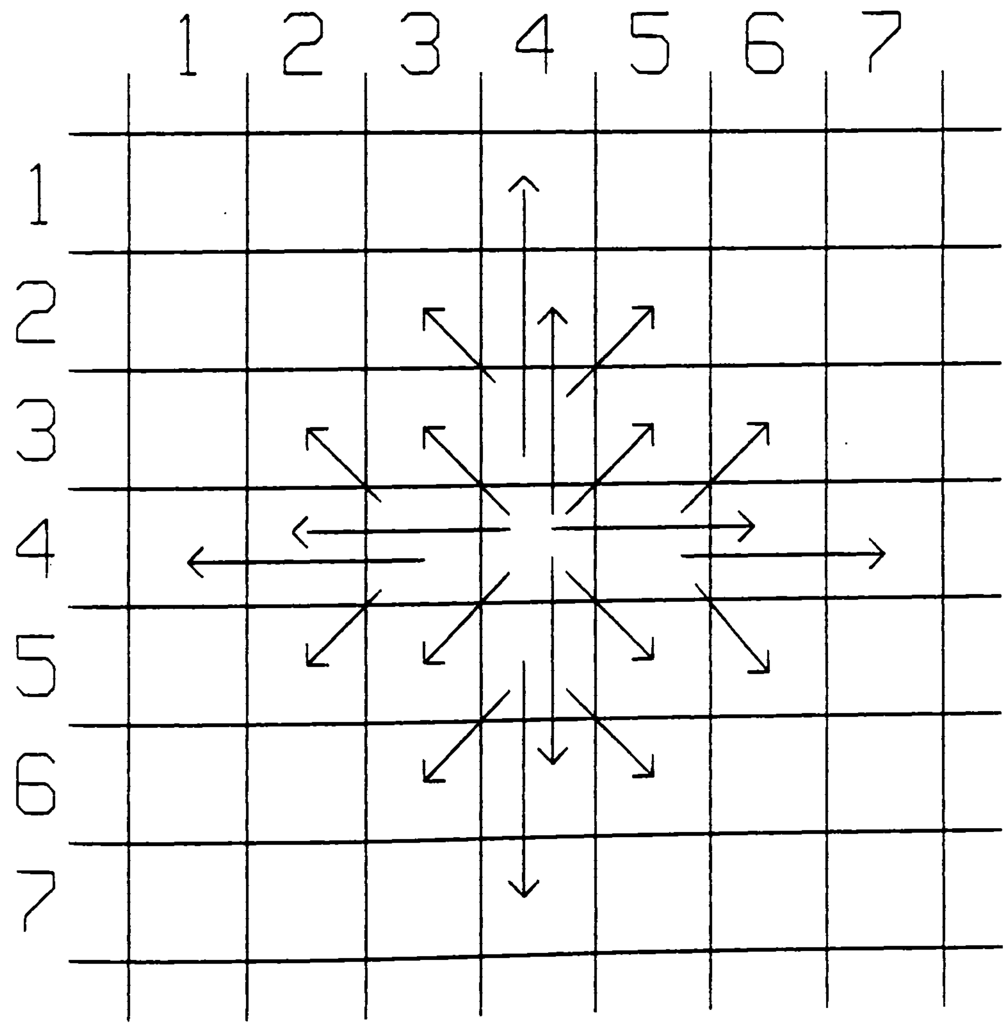


Figure 10. Vectors in 7 x 7 window.

In the following sections each of the modules of hardware implementation of the pseudo-laplacian operator with blackfill (i.e. the design of the IPB) is explained, giving details of the simulation methodology and errors found in and by verification and validation processes. For more detailed description of each of the modules including operations taking place at every clock cycle, the microprogram etc.; please refer to [4].

3.4 FIND_DIFFS.

Find_Diffs basically means find differences. There are two major subfunctions implemented in this module:

- Storage of digital video in one of three memory banks and permutation of memory banks as each row is completed;
- Computation of the elementary differences required to decide black and white edge points.

The computations are executed in microprogrammable byte-slice pipelined processors (Am 29501) in such a way that the carry outputs give the required data directly, inasmuch as they correspond to the sign bit in 2's complement arithmetic. The differences are between the pixels shown by arrows in Figure 12 on page 41. In this figure, X_0 is the pixel at reference and X_S is the pixel which is the newest data given to this module. X_W is the pixel on West of X_0, X_S is the pixel on South of X_0, X_{SW} is on the South-West and X_{SE} is on South-East of X_0. Thus, for a pixel X_0 there are four types of differences X_W, X_S, X_{SW} and X_{SE}. There are two bits given for each difference, one bit indicating if X_0 is on white side and second bit indicating if it is on black side, with respect to the pixel in that direction. These four directions are called the compass directions for the pixel in reference.

The input to this module is the video data and output of this module is a byte called BW_set (Black and White set). The 8 bits in BW_set are as shown in Figure 11 on page 40.

Before actual simulation was begun, a few fairly simple chip-models were developed and tested. One model of just three latches and three memory banks was developed and the input to this model was given to see if latching, writing into and reading from the memory is proper or not. A test pattern of a checkerboard of 9 squares with each square of 10 x 10 pixels, was supplied by a GIPSY expert. When only the first three rows of this pattern were given to the model, after a certain number of inputs, the simulation used to crash giving a message: "Time queue too long." Then the problem was given to a GSP expert. It was found that GSP is capable of handling only 100 events in the time queue. So he changed all those local variables into global variables and made GSP capable of handling 2000 events. If in future this

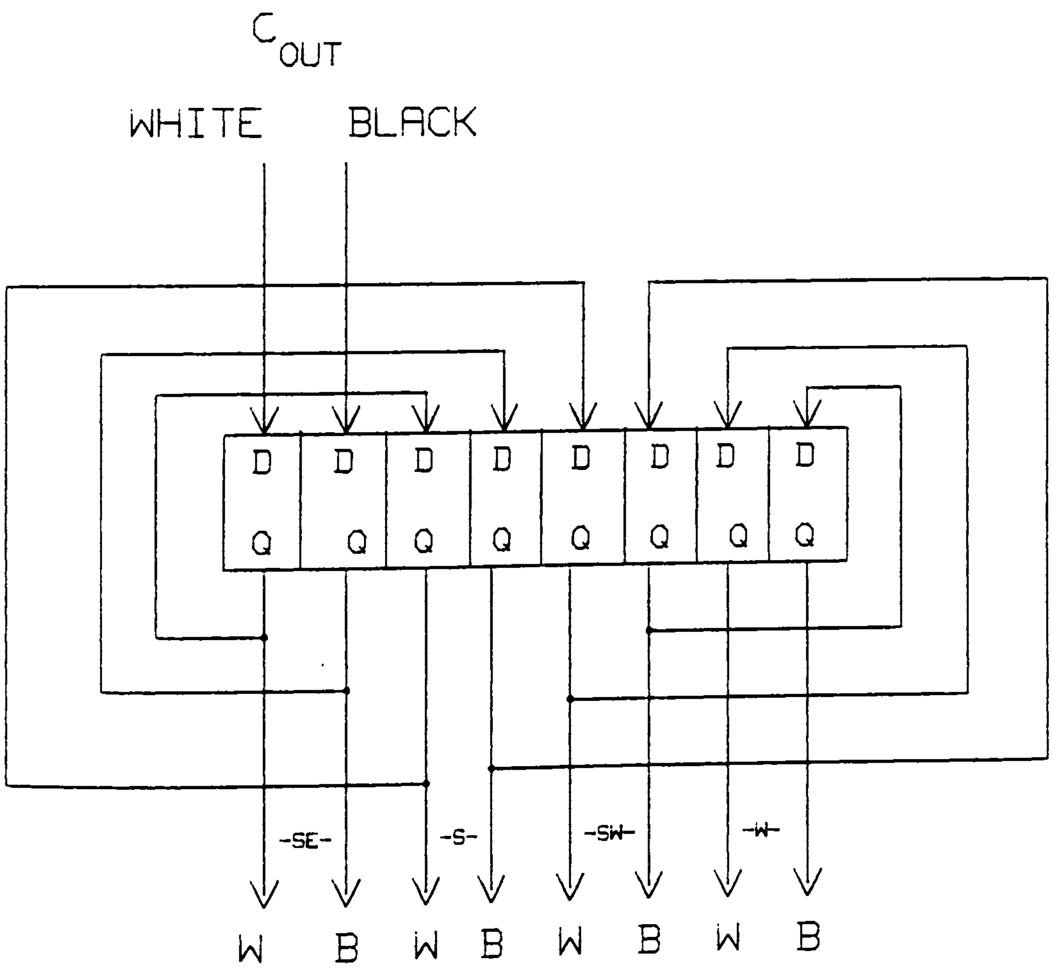


Figure 11. The BW_set latch.

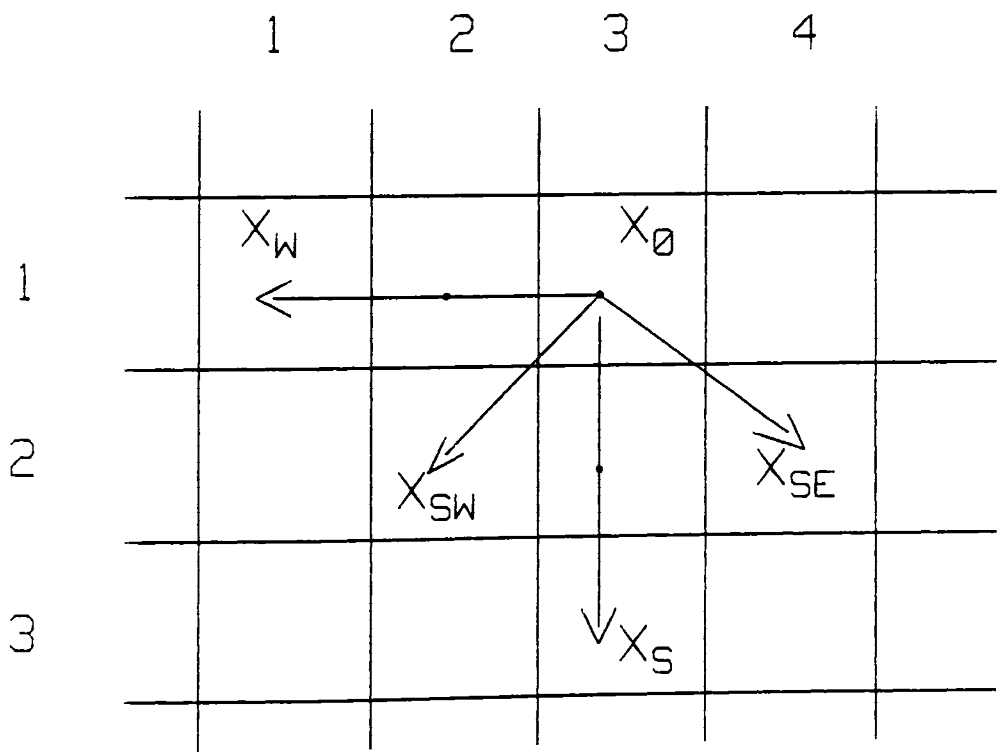


Figure 12. The directions of differences.

length of time queue needs to be changed, one can do it referring to the list of variables given in Appendix F.

Then the behavior of each chip was carefully studied. [15, 16, 17, 18]. There are all together about ten types of chips, and the total number of chips is 23. But that does not mean that there are 23 or 10 modules in the simulation. As explained earlier, some chips are merged into a single module and then the modules are linked. Find_Diffs was simulated with three modules:

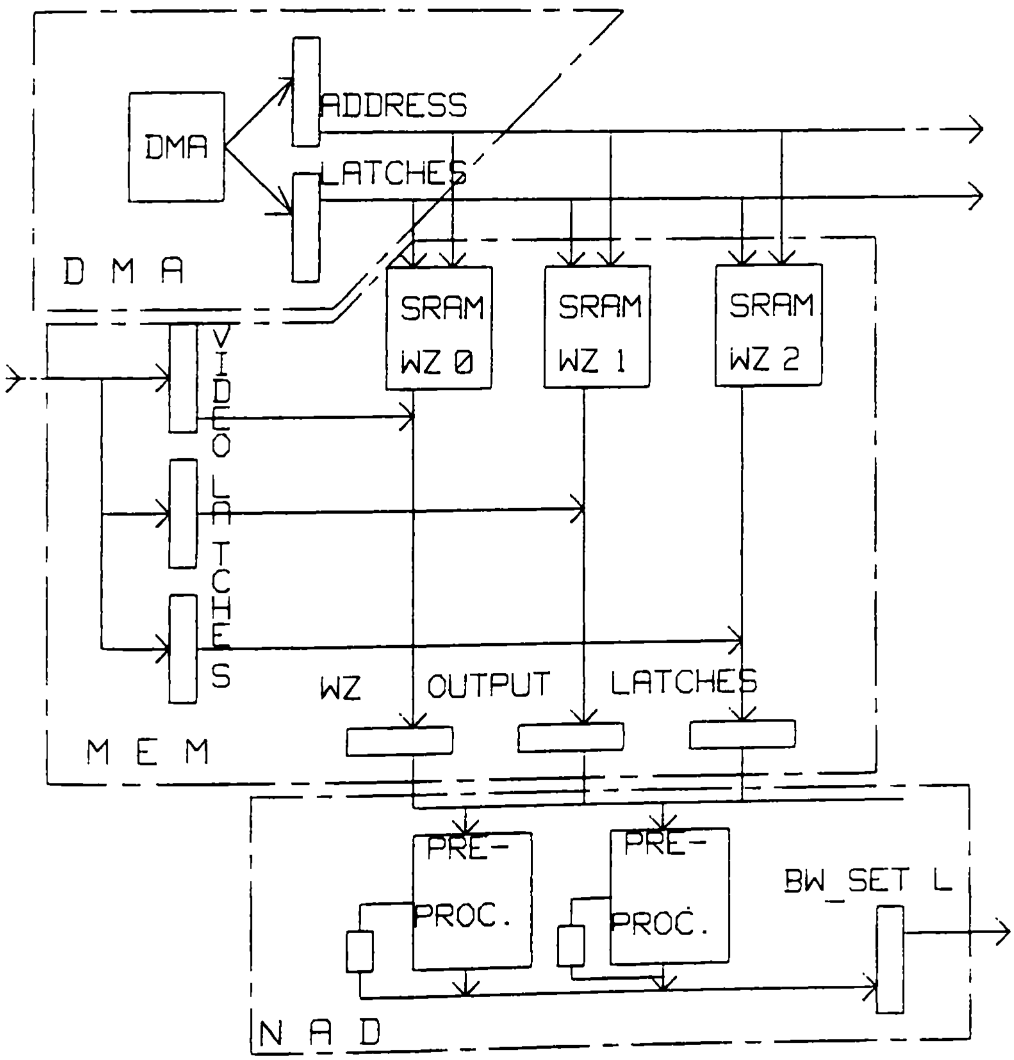
1. DMA: DMA address generator with two latches,

2. MEM: 3 video input latches and three memory banks.

3. NAD: The main computing part of Find-Diffs.

Figure 13 on page 43 shows how this arrangement is done.
The microprogramming control unit was merged with all the three modules according to the need of microprogram to control the chip-models.

In this simulation it was found that it was possible to split the design (of Find_diffs) into three parts, because the intermediate results were known. But it is not always possible to do so as will be seen in the simulation of the next module. The first two models were run and after a little bit of verification the design containing DMA address gener-



Find_Diffs split in three parts.

Figure 13. Example of splitting design for simulation.

ator and latches, and video input latches and memory banks was validated. In the verification of the third module, which simulates the preprocessors and latches the following observations were made:

- It is necessary to know the behaviour of a chip in detail, before it is simulated; knowing just the behaviour used in the design is not enough. From this experience it is highly recommended that the chip used in design be simulated separately and then included in the simulation.

- It is necessary to test the simulation of the chip for different types of data, before it is declared complete and before it is included in the main simulation program.

In the validation, after successful verification of the last module, it was found that there were errors in the design. Signed-arithmetic was expected from an ALU with unsigned-arithmetic. So, various combinations of 1's complement and 2's complement arithmetic were tried on the model, until the exact desired output was obtained from the present design. The necessary changes were made and then after the final simulation run the design was validated.

3.5 SUM_DIFFS.

Sum_diffs means Sum Differences. This module executes three functions:

1. Storage and permutation of rows.

2. Fetching the compass bits required to compute the B_sum (Black sum) and W_sum (White sum) for each pixel.

3. Counting those bits to give final B_sum and W_sum.

This module counts the elementary differences relating to a given pixel to obtain B_sum and W_sum corresponding to the elementary differences indicative of black or white decisions. The four compass directions (West, South-West, South-East and South) are stored in four scratchpad memories, consisting of 2 64-bit RAM files, so that all four directions corresponding to the reference pixel can be accessed in one clock cycle. One register file is for 'Read' and the other for 'Write' in one clock cycle. (This is done to avoid using two-port memory.) The address sequence for these reads and writes is stored in PROM. However, the address cycle does not repeat after 32 cycles (PROM length) but rather the repetition period is 16 major cycles ('pixel cycles') of 8 clock cycles, i.e., 128 minor cycles. The input to this module is a byte corresponding to each pixel of image, from Find_Diffs: BW_set. The output of this module is 5 bits of each B_sum and W_sum.

As explained earlier, it was not possible to know the intermediate results and data. So, it was necessary to simulate this whole design of nine different types of chips numbering 42, in one single model. This module is so complicated that rigorous verification was necessary. The amount of work put in for verification cannot be easily explained in words. The very fact that during the simulation of this module CPU time worth $4000 was spent, shows that just an understanding of the design and the simulator does not help. The interfacing between the two is a very important factor. In case of such complicated and long simulations the following points should be noted:

1. 'Never take anything for granted!' It was experienced that if you take a very small thing for granted, it plays a monstrous role in spoiling the output. And then it is very difficult to trace back the cause of that error. If such a thing happens, it is always advisable to check each and every signal in the simulation at evry clock.

2. 'Never trust your eyes alone!' In GSP when an ouput is observed at a level 1 or above, it gives the state of 150 pins and some registers, at every time in the time queue.

If only a few pins are used the other pins are at 0 always. If checking the results is done while the screen is scrolling, the factor of human error becomes very high. If level 0 is used, the information is so much less that it becomes diffcult to debug the program. It is advisable to store the output in a file and then check it line by line.

3. There are some rules of GSP. If those rules are not satisfied, the results it gives are haphazard. It is always better to check at the beginning if all the rules of the simulator are satisfied or not.

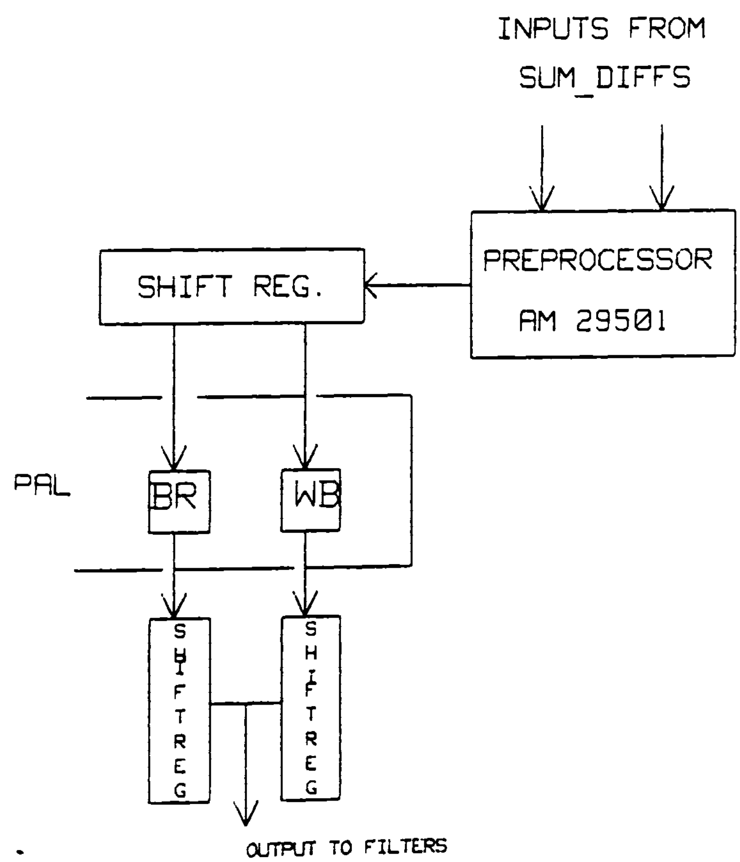
This module is the most complex of all and it needs about 7 rows of image data to get any valid output. So, simulation of it was rather a cumbersome process. And it was found that all the errors were in understanding of the module and modeling it. This complex design was the most perfect one too.

3.6 EVAL_DIFFS.

This is a relatively small module. It evaluates the B_sum and W_sum and generates the current bits of Binary_Row (BR) and White_Bits (WB). It carries out the various comparisons to thresholds and generates the required bits as functions of those comparisons. The bits are obtained as logical functions of the carry from an Am29501 (Microprogrammable ALU).

The inputs to this module are the three thresholds t1, t2, t3 stored during initialization in the Am29501 internal pipeline, and the current values of W_sum, B_sum from the output of Sum_diffs. The results of comparisons are obtained at the Cout pin and stored in a shift register. When all the comparisons have been effected the results are evaluated by logic functions associated with the two PAL functions, BR and WB.

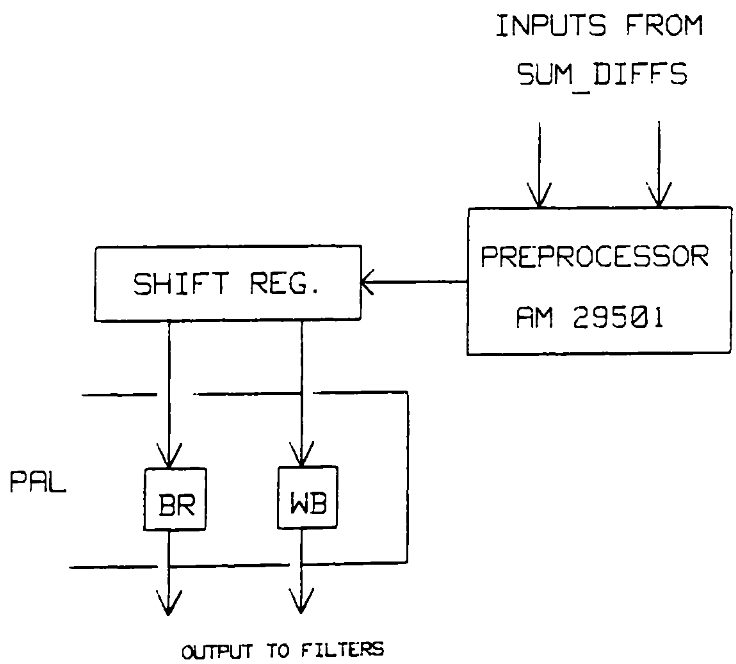
Simulation of this module was very easy. There was very little time spent on verification and validation. But it was in the simulation of this module that a major timing error in the next module, i.e., Filters, was found, which will be explained in detail in the next section. From the simulation of this module it was found that there were two chips used that were not necessary and the PAL function was too complex. So, a simple design was suggested with a smaller PAL. Figure 14 on page 49 shows the old design and the old PAL function, while Figure 15 on page 50 shows the new design with the PAL function. From the figures and the PAL function it is clear that the design is very simple now. This is one of the advantages of simulation.



$$BR = (Q'_A)(Q'_B)(Q'_C)(B/WC') + (Q_A)(Q_B) * (Q'_C)(BR')(B/WC') + (Q_A)(BR) + (Q_B)(BR) + (Q_C)(BR)$$

$$WB = (Q'_A)(Q'_B)(B/WC) + (Q'_A)(WB')(B/WC) + (Q_A)(WB) + (Q_B)(WB)$$

Figure 14. The old design and PAL function of Eval_Diffs.



$$BR = (Q_G')(T7) + (Q_F')(Q_D')(T7) + (BR)(T7')$$

$$WB = (Q_C')(T7) + (Q_B')(Q_A')(T7) + (WB)(T7')$$

Figure 15. The new design and PAL function of Eval_Diffs.

This module executes the algorithm for eliminating isolated points and 1-bit-wide streaks, by the Boolean operation:

$$e' := (a + c) * (b + d) * e$$

where the variables correspond to the bits found by Eval_diffs in the positions as shown below:

```
    a
b   e   d
    c
```

There are three filters:

Input(From)		Output(To)
Binary_Row (Eval_Diffs)	:=	Binary_In (Set_Rel_T)
White_Bit (Eval_Diffs)	:=	White_In (Set_Rel_T)
Binary_In (Set-Rel_T)	:=	Binary_Out (Output Buffer)

These three sequential machines are implemented by three pairs of flipflops in a registered PAL. The states of the sequential machines are frozen when the inputs are not available. It must be clear now that for the last filtering function, the input has to come from the output of the next module, Set_Rel_T, so that function is delayed by a pixel cycle time.

This module previously contained about 17 chips of 6 different types. The period of this module was not one pixel cycle[3], but 8 pixel cycles. This was the only module with a different period. For an efficient pipeline each stage should have approximately the same period to get significant throughput. And also that the timing could not be adjusted properly. There was one control signal from the microprogram of this module which was also used in the previous one, Eval_Diffs. Now, to get a proper input from Eval_Diffs at the right time, the control signal should be adjusted properly. But it was found from the simulation that if the timing for one module is adjusted properly, that for the other would be out of phase with it. This is an unusual case in a pipline. A lot of experiments were done to get the correct timing for both the modules. All of them drew a conclusion that the design is not right. That was the biggest achievement of this simulation. This Mega-bug was detected and the exact error was pointed out. Had there been no simulation, this error would have grown to a monstrous size and a lot more time would have been required to fix it.

The module was then redesigned, with a lot less chips than before. Now the design has only 3 chips of 3 different types and the period is now one pixel cycle. The control signals are reduced tremendously, there were about 30 control signals in the previous design whereas there are only 4 control signals now.

---

[3] One pixel cycle equals 8 clock cycles.

After the new design was ready it was very easy to simulate it. But then it was necessary to simulate the previous module again. Both the modules were simulated and tested for timing adjustment thoroughly. There was no difficulty in verifying the model and then validating the design.

3.8 SET_REL_T.

This stands for Set Relative Threshold. This is the module added to incorporate the blackfill with the pseudo-laplacian edge detector. This module does process_gray and process_black, generating a bit stream into the output filter as well as the next value of avg_black.

The principal features of this module are the selection of microprogram page by the two bits, Binary_In and White_In, and the manner of executing the conditional correction in the recursive computation of avg_black. The inputs to this module are: Binary_In and White_In from Filters, and Gray_val from Find_Diffs.

In the simulation of this module the right time of Gray_val coming from Find_Diffs was a point of contention. But then after a careful observation and computation it was resolved. When the correct offset was known it was not too difficult to get the model running. The model was verified and the design was validated after a few error corrections in the PAL functions.

After all this rigorous verification and validation process, a reference is ready for actual hardware debugging. The models are so easy to run that anyone can just see the instructions and run the models. In the documentation of these models all the details such as Inputs, Outputs, Variables etc. are given. After this exhausting exercise of simulation the next task was to modify the design to make it reconfigurable and to define all the initialization sequences for all the modules. These two topics are covered in the next chapter.

4.0 THE DESIGN MODIFICATIONS

If work done for this thesis is represented by a "black box", the input to this black box was the design of IPB for one set of parameters and GSP; the output from that black box is a perfect design of IPB for different sets of parameters ready for wiring.

The purpose of this thesis is twofold:

1. To simulate the design of IPB to get the errors in the design corrected, so as to have a reference for hardware debugging.

2. To add to the design features a capability of reconfiguration, which includes design for making reconfiguration and the definition of initializations according to the configuration.

The first purpose has already been discussed in detail in the previous chapters. The following sections describe how the design was made reconfigurable. There are some very important points one should always pay attention to, when a design has to be modified:

1. If the system is not designed by the person who is modifying it, that person should make sure he understands the original design perfectly.

2. While modifying the design, features of the original system should not be lost.

3. After designing the modification, it is essential to ensure that the modification and the original design blend well.

4. All possible effects on all the parts of the design due to the modification should be considered. It was experienced many times during this work, that a lot of time and effort may be spent unnecessarily if all the effects are not considered.

4.1 THE RECONFIGURABLE IPB.

The parameters that can be changed by the user of Telesign are as follows:

1. Spatial definition: 256 X 256 pixels or 128 X 128 pixels. This parameter has been tested for 256 X 256, 128 X 128 and 85 X 85 non-interlaced by a software engineer. It seems reasonable to use 256 X 256 or 128 X 128.

2. Temporal definition: Between 25 images/sec. to 12.5 images/sec.

3. Use of interlace: This will give an intermediate definition, 2 X 128 X 256 at double frame rate. This should give half the bit rate of a full 256 X 256, and may satisfy the temporal requirement.

4. The size of the pseudo-laplacian window (3 X 3, 5 X 5, 7 X 7) and the number of components used within that window. Studies have shown that there are only three combinations of window size and number of components, worth considering: 7 X 7 with 32 components, 7 X 7 with 20 components and 5 X 5 with 8 components.

5. Some thresholds:

   - TDIFF :- The minimum difference in thresholds between neighborhood pixels in the image that qualifies as an edge element.

   - T1 :- The minimum number of edge elements associated with a given pixel required to independently determine whether the edge elements constitute the black or white side of an edge discontinuity.

- T2 :- The minimum number of edge elements associated with a given pixel that guarantees the decision of black and/or white to be made regardless of white or black edge elements, respectively.

- T3 :- The minimum difference between the number of black and white edge elements required to assign a black and/or white label to a given pixel.

- TBF :- The minimum difference required between the running graytone average of "black" pixels and the actual graytone of a given pixel to enable the pixel to be blackfilled.

There are other parameters also, but these are fixed after a deep study. Those parameters are as follows:

1. Use of subsampling after processing or no subsampling. It is found that subsampling gives much better results.

2. Use of blackfill or not. After discussion with some future users of Telesign, it was felt that they would feel comfortable if blackfill is included. So, blackfill will be in the system, and it need not be optional.

Let us now see how the design was made capable of handling changes in all these parameters.

The first three parameters, spatial definition, temporal definition and use of interlace, depend upon the data and clock given to IPB by VPB[4]. Since IPB operates at every clock cycle, it was decided to change the clock to IPB rather than changing the data-rate. This can be achieved in two ways:

1. Keeping the clock to IPB at the same frequency all the time and incorporating some logic on IPB to change the clock frequency according to the choice of these parameters. A simple logic design to achieve this is given in the last section of this chapter.

2. Providing IPB with a clock already adjusted to accomodate these parameters. This appears to be the best way. The parameter which decides the size of pseudo-laplacian window and the number of components used within that window, has to be accommodated by IPB itself.
   Figure 16 on page 60 shows the three combinations which produce considerably good results.

After a deep study it was found that out of five modules of the design, only Sum-Diffs is affected by this parameter.

---

[4] Video Processing Board, see section 1.1.

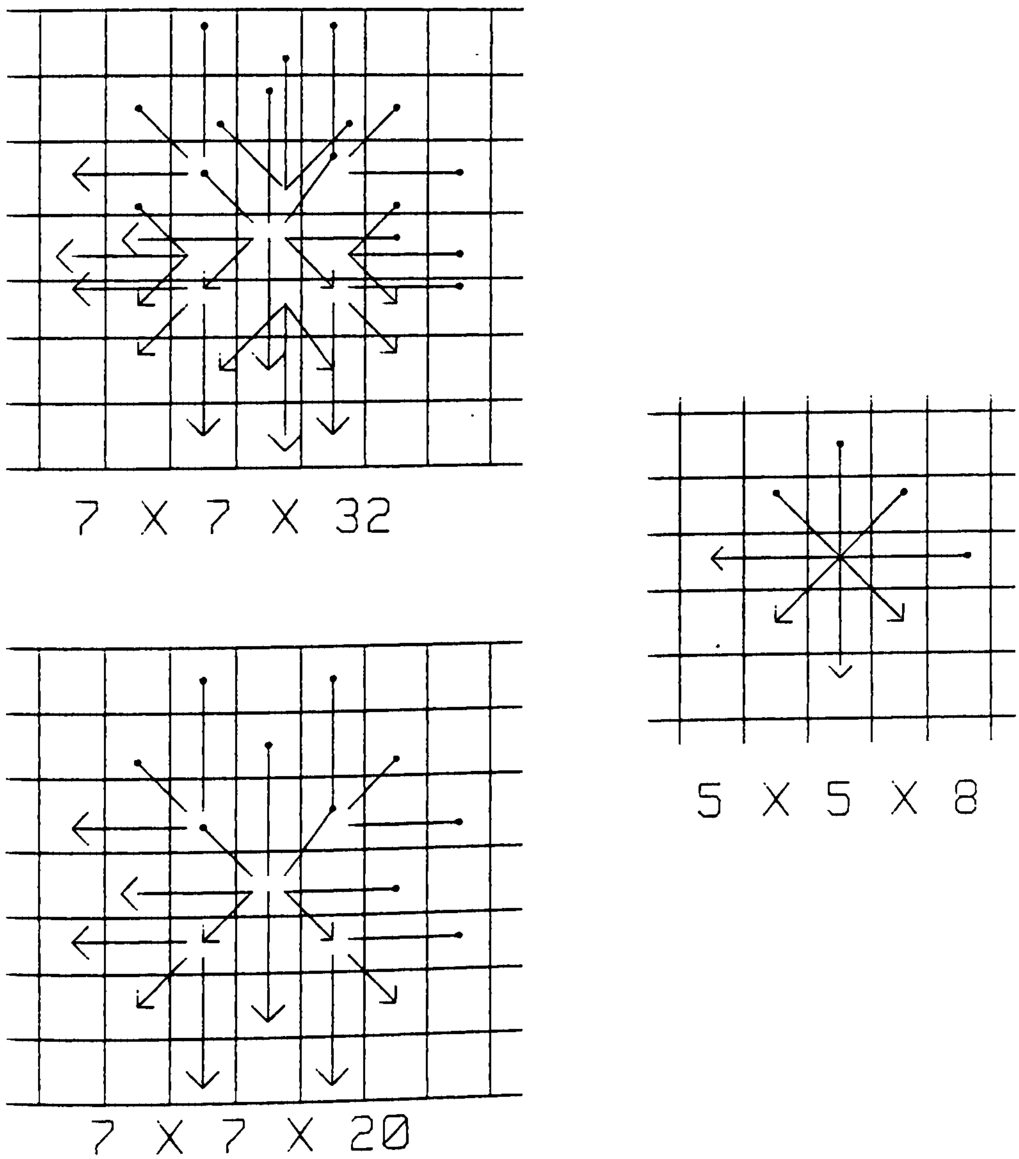


Figure 16. The three laplacian window-vector combinations

In Sum-Diffs the addressing sequence for the reads and writes of the scratch pad memories changes with the change in this parameter. The sequence of writes may remain constant, but that of reads changes, because the number of writes can be for 32 vectors, but number of reads are 32, 20 or 8. So, after some investigation the new addressing sequences were found.

Figure 17 on page 62 gives the scratch-pad addressing sequence for 7 X 7 window with 32 vectors. Figure 18 on page 63 gives the scratch-pad addressing sequence for 7 X 7 window with 20 vectors. Figure 19 on page 64 gives the scratch-pad addressing sequence for 5 X 5 window with 8 vectors. These addressing sequences were put in, and the output of Sum-Diffs, for the same input data, was checked and verified with that from the GIPSY program. After a few error corrections in data entry, the design was validated. These addressing sequences will be stored in a bigger PROM (or 3 PROMs), and two bits will select the proper addressing sequence. If a bigger PROM is used these two bits will be additional address bits to that PROM, if three smaller PROMs are used three bits would be required to select the proper PROM chip. The default will always be 7 X 7 with 20 vectors. All the thresholds are the part of initialization of the modules. The initialization sequences are defined and detailed in the next section.

P	T	B/WSP (W) R1	B/WSP (W) R2	B/WSP (SE) R1	B/WSP (SE) R2	B/WSP (S) R1	B/WSP (S) R2	B/WSP (SW) R1	B/WSP (SW) R2
	0	0	5	0	8	0	8	0	5
	1	5	1	9	1	6	1	6	1
	2	2	8	2	11	2	12	2	8
0	3	6	2	12	2	9	2	9	2
	4	X	1	X	4	7	X	X	13
	5	2	X	5	X	10	X	14	X
	6	4	1	4	4	4	10	4	14
	7	4	3	5	3	7	3	14	3
	0	3	8	3	11	3	11	3	8
	1	8	4	12	4	9	4	9	4
	2	5	11	5	14	5	15	5	11
1	3	9	5	15	5	12	5	12	5
	4	X	4	X	7	10	X	X	0
	5	5	X	8	X	13	X	1	X
	6	7	4	7	7	7	13	7	0
	7	7	6	8	6	10	6	1	6

Figure 17. Addressing Sequence for 7 x 7 window with 32 vectors

P	T	B/WSP (W) R1	B/WSP (W) R2	B/WSP (SE) R1	B/WSP (SE) R2	B/WSP (S) R1	B/WSP (S) R2	B/WSP (SW) R1	B/WSP (SW) R2
	0	0	XX	0	8	0	8	0	5
	1	5	1	XX	1	6	1	6	1
	2	2	8	2	11	2	12	2	8
0	3	6	2	12	2	XX	2	XX	2
	4	X	XX	X	XX	XX	X	X	13
	5	2	X	5	X	10	X	XX	X
	6	4	1	4	XX	4	10	4	XX
	7	4	3	5	3	7	3	14	3
	0	3	XX	3	11	3	11	3	8
	1	8	4	XX	4	9	4	9	4
	2	5	11	5	14	5	15	5	11
1	3	9	5	15	5	12	5	XX	5
	4	X	XX	X	XX	XX	X	X	0
	5	5	X	8	X	XX	X	XX	X
	6	7	4	7	XX	7	13	7	XX
	7	7	6	8	6	10	6	1	6

Figure 18. Addressing Sequence for 7 x 7 window with 20 vectors

P	T	B/WSP (W) R1	B/WSP (W) R2	B/WSP (SE) R1	B/WSP (SE) R2	B/WSP (S) R1	B/WSP (S) R2	B/WSP (SW) R1	B/WSP (SW) R2
	0	0	XX	0	8	0	8	0	XX
	1	XX	1	XX	1	XX	1	XX	1
	2	2	8	2	XX	2	XX	2	8
0	3	XX	2	XX	2	XX	2	XX	2
	4	X	XX	X	XX	XX	X	X	XX
	5	XX	X	5	X	XX	X	XX	X
	6	4	1	4	XX	4	10	4	XX
	7	XX	3	XX	3	XX	3	14	3
	0	3	XX	3	11	3	11	3	XX
	1	XX	4	XX	4	XX	4	XX	4
	2	5	11	5	XX	5	XX	5	11
1	3	XX	5	XX	5	12	5	XX	5
	4	X	XX	X	XX	XX	X	X	XX
	5	XX	X	8	X	XX	X	XX	X
	6	7	4	7	XX	7	13	7	XX
	7	XX	6	XX	6	XX	6	1	6

Figure 19. Addressing Sequence for 5 x 5 window with 8 vectors

4.2 THE INITIALIZATION SEQUENCES

In the original hardware description of IPB, at the end of each module the initialization requirements are specified. Taking that as a reference and studying the control signals for all the initializations the initialization sequences are defined for each module. Some of these initializations have been simulated in the respective modules and are validated.

The latches used in the design are special chips: Am29818. In brief, they have a serial shadow register (SSR) and writable control store (WCS) pipeline register [15]. The latches are connected by their SID/SOD pins to form a "serpent" within each module. These serpents are used during hardware debugging and in some of the modules they are also used for initialization . So, the initialization data can be given to the respective latch serially, and data from each module can be read serially during debug mode.

There will be 11 signals from the Link board to IPB for initialization. Those 11 signals are:

1. The data required for initialization of different chips. Since sometimes a serpent of latches is used to load the data to the chips for initialization, the data line gives a number of zeros along with the valid data required. All the required bytes of data are defined for each module. There are three data lines.

2. The signals to control the chips that need to be initialized. These signals go into a control latch. And at the end of each sequence this latch is output enabled. There are three such lines from the Link, one for the control latch and the microprogram output latch in Find_Diffs and one each for the other two control latches.

3. Four control signals to control the four control latches. There are three extra latches for this purpose and one latch from microprogram output latches of Find_Diffs is used for output enabling the latches carrying the data needed for initialization of Find_Diffs.

4. A clock signal for the serial clock of the latches.

The initialization is achieved with a principle similar to the "Domino Principle." First the initialization data come from the link and creep through the serpent. The control data then fill the control latch. When that is finished, Link gives the data to control the Control Latche's MODE to transfer SSR to pipeline register, and to its output enable (OE). Then the control data control the chips to be initialsed with the initialization data standing at their data lines. And thus the chips are initialized. There are some chips that do not need any data for initialization; in that case more control data come to the control latches and those chips are initialized. The following sub-sections describe initialization of each module.

4.2.1 INITIALIZATION OF FIND_DIFFS.

The following steps are required for this initialization:

1. Clear Work-Zone SRAMS. CLR' of these SRAM's is 0 for clearing and 1 for normal operation. So, CLR' of WZ0,WZ1 & WZ2 = 0 1 1 1 ....

2. Store row-size + 255 in DMA Address Generator's registers, HI byte in Word-Counter (WC) and LO byte in Address-Counter (AC). There can be two values of row-size, 255 or 127. For 256 x 256 image row-size is 255, so FFFF needs to be stored in the DMA, FF in WC and FF in AC. For 128 x 128 image row-size is 127, so 7F is stored in AC and FF in WC. The control sequence for this initialization is:

I2	I1	I0	D0 - D7
1	0	1	FF
1	1	0	ROW-SIZE

3. Set DMA mode to 0. The control registers CR0, CR1, CR2 of the chip (Am2940) should have a 4 (i.e. 0, 0, 1 respectively). The control sequence is:

I2	I1	I0	D0 - D7
0	0	0	04

4. Load TDIFF into the register A3 of both the Am 29501's. The control sequence is:

I12	I11	MI0 - MI7 (MSP)
0	0	TDIFF

5. Clear Work-Zone Counters WZC0, WZC1, WZC2. And shift a 1 to WZC0. For clearing the counters CLR' of WZC0, WZC1 & WZC2 = 0 1 1 1 ..... . For shifting 1 to WZC0 PRESET' of WZC0 = 0 1 1 1 ..... . But, PRESET' of WZC1, WZC2 is always at 1.

6. Clear WCI and BCI. This is done by a giving 0 1 1 1 .... to CLR' of these flip-flops.

7. After the end of each row:

   - Clear the microprogram address counter (SN74AS163), and

   - Rotate the ring counter WZC.

This is done cleverly by the end of line (EOL) signal from the Link board.

Thus, taking into consideration all the control sequences and the latches involved, the connections are as shown in Figure 20 on page 70 and the initialization sequence is as follows:

Data-bytes: 0 0 0 0 0 0 0 04 0 0 0 0 0 0 255

Control-bytes: 0 0 0 0 0 0224[5] 120 0 0 0 0 0224 125

Data-bytes: 0 0 0 0 0 0 0 255/127[6] 0 0 0 0 0 0

Control-bytes: 0 0 0 0 0 0224 123 239 247 255 159 255

4.2.2 INITIALIZATION OF SUM_DIFFS

Although this is the most complex module of all, there are very control sequences, and except for one row-initialization which is done by EOL' signal, all the other signals are taken from the control latch used in Find_Diffs initialization. There are no data-bytes required here. There are following initializations to done:

1. Clear B/W_Set SRAM's: CLR' SRAM's = 0 1 1 1 .....

---

[5] This control byte is for OE' of latches holding data.

[6] This is to account for the row-size of 256 or 128.

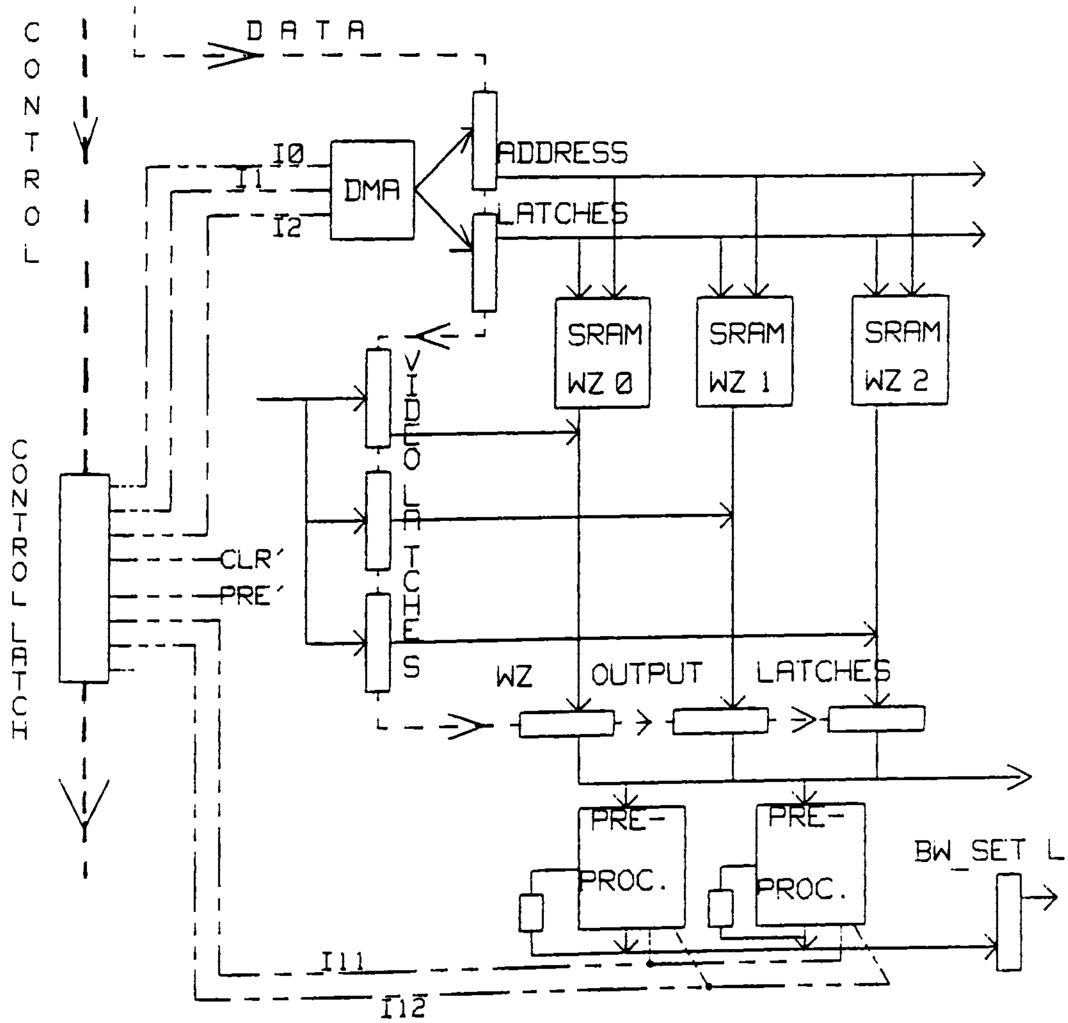


Figure 20. Initialisation of Find_diffs

2. Clear microprogram address counter: CLR' UPAC = 0 1 1 1 .....

3. Set scratch-pad address counter to 1: The controls for this are:

S0	S1	A B C D E F G H
1	0	1 0 0 0 0 0 0 0
1	1	0 0 0 0 0 0 0 0

4. Set scratch-pad address counter to 1 after each row: The same sequence as above applies. So, to incorporate both the session and row initialization a combinational logic is designed, where the functions are as follows:

   S1 = (CLR') * (EOL') and A = (S1)'

5. Clear microprogram address counter after each row: This is done by an EOL' (INV(End-of-line)) signal.

6. Clear BC, WC, SDC0 and SDC1[7]: BC and WC are cleared by a CLR' signal and SDC0 and SDC1 are cleared automatically due to the PAL function [4].

---

[7] These are flip-flops storing intermediate values.

4.2.3 INITIALIZATION OF EVAL_DIFFS

This initialization consists of storing the three thresholds into the three registers of Am 29501[8] . In order to have a concise control sequence the thresholds are stored in the following sequence:

1. t3 --> A1 of the Am 29501.

2. t2 --> B1 of the Am 29501.

3. A1 --> A2.

4. t1 --> A1 of the Am 29501.

The control sequence is as follows:

I7	I8	I9	I10	I13	I14	MODE	DI0 - DI7
1	0	1	1	1	1	1	t3
1	1	1	1	1	0	1	t2
1	1	0	1	1	1	1	XX
1	0	1	1	1	1	1	t1

The sequence expected from the link board is as follows:

Data-bytes:	t3	t2	xx	t1
Control-bytes:	125	95	123	125

---

[8] This is a microprogrammable preprocessor.

4.2.4 INITIALSIATION OF FILTERS

The only signal needed here is a clear signal for clearing SRAM and the shift register. This signal is taken from Find_Diffs control sequence.

4.2.5 INITIALIZATION OF SET_REL_T

This initialization is similar o that of Find_Diffs, which involves initialization of the DMA address generator. The steps required are:

1. Load row-size + 4 + 255 to DMA for Gray-Val FIFO.

2. Set DMA mode to 0.

3. Clear all the latches and micrprogram address counter with a CLR' signal from the first control latch in Find_Diffs.

The DMA initialization sequence is as follows:

Data-bytes:	0	0	04	0	0	03	0	0	0/128[6]
Control-bytes:	0	0	08	0	0	13	0	0	11

4.3 FUTURE DESIGN ALTERATIONS

The present design of the IPB is not the only possible way to design this board. Just as there can be many ways to program one operation in software, there can be many ways to implement hardware of a particular design. Some anticipated design alterations are mentioned here in brief:

1. The first module of the IPB design (Find_Diffs) contains three memory banks of 8k x 8 each. But for the operation of Find_Diffs only two of them are necessary; the third memory bank stores the pixels for giving it to the last module (Set_Rel_T) as Gray_Val. The Gray_Val FIFO in Set_Rel_T stores <u>one</u> row and four pixels previous to the one in reference. If the Gray_Val FIFO in Set_Rel_T is made a little bigger so as to store <u>two</u> rows and four pixels previous to the one presently in reference, there will be no need for the third memory bank in Find_Diffs.

2. There are two microprogrammable preprocessors (Am 29501's) in this module. If the number of comparisons is reduced, only one Am 29501 would carry out

the same computations at a slower rate. This needs to be investigated.

3. In the module Sum_Diffs, a set of black and white bits corresponding to each direction for a pixel (BW_Set) is stored in one of the four 8k x 8 memory banks (according to whichever is enabled for the present row). Then the set is read from the memory and is split up into individual directional bits and stored in a 2-bit scratch-pad memory. Instead of assembling these bits and then disassembling them again, if the memory banks of 8k x 1 are used to store each bit and then if the bits are read from those memories as needed, a lot of wiring and big chips would be saved. But, of course the design details and the addressing sequence should be worked out carefully.

5.0 CONCLUSION

Although this is a conclusion of the topic of this thesis, it is not a conclusion of the responsibilities of the author nor is it the completion of the Telesign project. A detail description of the work still to be done on the subsystems of the Telsign project is beyond the scope of this thesis. This work will be declared finished only when the deaf people will start communicating using this system.

One of the major things to be done in the very near future is to construct a command procedure on VAX 11/785 which will extract essential information from the output files of the simulation runs and compare it with that of the GIPSY program. This will automate the validation process and it will ensure reliability too. There are some command procedures present already on VAX 11/785, which produce input command files to the different modules from the input given by the GIPSY program. So, very soon the whole validation process will be totally managed by the computer and it will be much more reliable. This is going to be of a great help while debugging the hardware. The output from the actual hardware can be stored in a file which the computer will compare with that of the simulation cycle-by-cycle and find out where the outputs are not matching.

The accomplishments of this work include, mainly, the first successful interface between the first chip-level simulator and the first microprogrammed reconfigurable image processing board. From a survey of the literature on logic simulators and from the discussion with some of the experts in this field, it was concluded that this is the first attempt to simulate such a huge and complex hardware system using GSP. Just as in the general case of pioneers, a lot of difficult situations and deadlocks were faced. Almost all the precautions to avoid such difficulties in future and solutions to those kind of difficulties from the experience of this work are given in the previous chapters. Secondly, the design of the IPB was completed with an added feature of reconfigurability. The board is being laid out using the HP CAD/CAM system.

From the suggestions to improve GSP, given in the next section, it will be clear that had all these features already been in GSP, the work would have been easier and the results would have been more effective.

5.1 SUGGESTIONS TO IMPROVE GSP

The improvements suggested here are in the same order as the topics are covered in GSP User's Guide. [5].

1. The time queue length: This is the most significant part of simulation. The user should be able to decide the length of the time queue. The default length may be kept small. If the user wants to use it for a large number of events, either he should be introduced to the simulator's structure so that the user could go into the simulator and change the queue length or there should be a provision to fix it to any value, the user needs, from outside. The problems faced due to the fixed small queue length are mentioned earlier. Also, the variables in the simulator that need to be changed, to change the queue length, are given in Appendix F.

2. The level (L) command in the Command File determines how much information is presented while the simulator is running. As given in [5], Level 0 displays only those pins which are connected to the front panel if they change state. This information is in hexadecimal form, while in Level 1 or above 150 pins are displayed, with the binary values. (The pins

that are not at all used in the source file are always at 0.) There is no intermediate level which will display the pins connected to the front panel at the time the user wants to observe the data in decimal form. It is not practical to observe those pins in level 1 or above, because of the size of the output.

3. The input data can be given through a command file in decimal form only. The output can be in hexadecimal (level 0) or in decimal form (level 1 or above). This is rather unconventional. The user should be able to decide the form of input as well as the output at any level.

4. The output is displayed on the screen, even if the user wants to store the output in a file only. This reduces the speed of simulation, which in turn reduces the efficiency of the simulator.

5. The user decides whether an output file is to be created or not. After the simulation is finished the user is asked if the output file is to be created or not. Even if the simulator is not asked to create an output, it creates an output file. This

becomes very irritating when the user has had enough simulation!

6. While the simulator is running and the output is being displayed on screen, if the user realizes that he has made some error there is no way to halt the simualtor. On VAX 11/785 [CNTRL] Y can stop the simulation. But then three temporary files are created: DATAAUTO.TMP, GSPTMPLOG.TMP and GSPTMPREG.TMP. The GSPTMPLOG.TMP file which contains the part of the simulation is added to the output file when the simulation is run completely the next time. It is very confusing when the user opens the output file. There should be a provision for the user to decide whether the part of the previous simulation run be included in the output file or not.

   (There is a solution to this problem when the simulation is done on VAX 11/785. The following series of commands will erase the unwanted part of the previous simulation run:

   $[9]CLOSE LOGFILE

   $CLOSE REGFILE

   $DELETE *.TMP;*

[9] This is the prompt on VAX 11/785.

This might be a problem only with VAX computers, where different versions of one file are created.)

7. The maximum length of a register is 8 bits. So, the maximum number which can be handled without any manipulation is 255. The registers should be of variable length from 8 bits to 32 bits.

8. Since the maximum length of an index register is also 8, only 256 memory locations can be addressed through one index register. The maximum size of the memory should be at least 8 kilobytes.

9. There are instructions such as increment (INC), rotate right (ROR), shift right (SHR). While simulating big hardware systems the counterparts of these instructions, i.e. decrement, rotate left and shift left, are also necessary. Similarly, the branch instructions, branch if equal (BEQ) and branch if not equal (BNE), should also have other forms - branch if greater than and branch if less than.

10. Branch to a program label is a similar operation to jump to a subroutine (JSR). This is the only instruction to call a subroutine. There should also be instructions to call a subroutine on some condi-

tions, jump to a subroutine if equal or not equal and greater than or less than.

11. In the command file, when a bunch of pins from one module is connected to a bunch of pins from another module, each pin connection has to be mentioned. There should be a provision to connect a bunch of pins to another in one command.

12. This simulator does not recognize special control characters such as TAB. It only recognizes spaces. It should be easy to incorporate the control characters in its list of characters.

13. It recognizes only upper case letters. It should also recognize lower case letters.

BIBLIOGRAPHY

1. Pearsen D. E. : "Visual Communication at Very Low Data Rates." IEEE Special Issue on Visual Communication Systems, April 1985.

2. Nadler M., et al : "The Telesign Project." IEEE Special Issue on Visual Communication Systems, April 1985.

3. Crosbie et al : "Toward Real-Time Simulation." Simulation Councils Proceedings Series, vol. 6, Number 2, 1976.

4. Nadler M. : "Logical Design for 4 MHz Pseudo-Laplacian Preprocessor with Blackfill." Dept. of Electrical Engineering, Virginia Tech. 1985.

5. GSP User's Guide. Virginia Tech, EE Dept., December 1982.

6. Armstrong J.R. : "Chip-level Modeling of VLSI Devices." IEEE Transactions on Computer-Aided Design, Vol. CAD-3, No. 4, October 1984.

7. Nadler M.: "The Telesign Project Report Submitted to NSF." Interim report of the Telesign project for the year 1984-1985.

8. Parker A.C. et al : "ISPS: Retrospective View", 4th Intl. Sympsium on Computer Hardware Description Languages, (pp. 21-28). October 1979.

9. Nadler M. : Research proposal on three-phase clocking system for VLSI chips. Virginia Tech, EE Dept., May 1985.

10. Gupta A.K. : "Functional Fault Modeling and Test Vector Development for VLSI Systems." Master's thesis submitted to the EE Dept. of Virginia Tech. March 1985.

11. Kogge P.M.:The Architecture of Pipelined Computers. McGraw Hill Book Co., 1981.

12. Daly E.B. : "Management of Software Development", IEEE Transactions on Software Engineering, SE-3, (pp. 229-242). 1977.

13. Myers G.J.:Digital Sytem Design With LSI Bit-Slice Logic. Wiley-Interscience Publication. 1980.

14. Nadler M.: " PASCAL program of the pseudo-laplacian operator."

15. Bipolar Microprocessor Logic and Interface Data Book. Advanced Micro Devices, 1983.

16. Toshiba Data Book.

17. Texas Instruments Data Book.

18. Hitachi Data Book.

APPENDIX A. THE SOURCE CODE OF THE MODULE FIND_DIFFS

As explained earlier there are three parts of this source code, DMA, MEM and NAD. Source codes for these parts are included in this section. There is also an example of the output file of the DMA Address Generator model.

A.1 DMA.SOR

```
;--DMA           ADDRESS GENERATION FOR MEMORY BANKS.
;
;
;
;PURPOSE
;
; THIS MODULE SIMULATES A BUNCH OF CHIPS FROM THE FIRST
; MODULE OF THE IPB DEISIGN: FIND_DIFFS.
; THE CHIPS INCLUDED IN THIS MODULE ARE :
; AM2940  :    DMA ADDRESS GENERATOR.
; AM29818 :    TWO ADDRESS LATCHES.
;
;ENTRY POINT
;
; GSPASM DMA -- TO ASSEMBLE THE SOURCE FILE.
```

```
;
; GSPSIM DMA -- TO START LINKING AND THEN SIMULATING.
;
;                  THIS IS AN INTERACTIVE PROCEDURE.
;
;
;
;
;DATA FORMAT
;
; INPUT  : AS GIVEN IN COMMAND FILE - DECIMAL.
; OUTPUT : HEXADECIMAL.
;
;LIMITATIONS
;
; RUNS FOR THE TIME SPECIFIED BY THE CLOCK INPUT.
;REMARKS
;
; THIS IS JUST AN EXAMPLE OF HOW THE ADDRESSES ARE
; GENERATED, THEY MAY NOT BE THE ACTUAL ADDRESSES NEEDED.
;
;PRECAUTIONS
;
; EVERY TIME THE SOURCE FILE IS CHANGED AND ASSEMBLED, IT IS
;
; NECESSARY TO DELETE THE PREVIOUS LINK AND LOG FILES.
```

```
;
;**********************************************************
;
;
;REGISTERS 2940.
;
REG(8)   COLRG,ROWRG
;
REG(1)   OLCL1,OLCL2,BEG,LOERG
;
;REGISTERS  29818.
;
REG(8)   SHAD1,PPLN1,SHAD2,PPLN2
;
; 2940 PINS.
PIN      ROCLK(1),ROWCO(2),COCLK(3),COLAD(4,11),ROWAD(12,19)
;
PIN      AGO(20),CLK2(21),COLCO(22),DONE(23)
;
;29818 PINS.
;
PIN      LOE1(24),CLK1(25),I8181(26,33),SDI1(34)
;
PIN      SDO1(35),D8181(36,43),MODE1(44)
;
PIN      LOE2(45),CLK11(46),I8182(47,54),SDI2(55)
```

```
;
PIN     SDO2(56),D8182(57,64),MODE2(65)
;
PIN     DUMMY(151)
;
EVW     W30(30),W80(80),W64(64),W224(224)
;
;
        BNE     CLK2,OLCL2,ADGN1
START : EXR
;
ADGN1 : MOV     CLK2,OLCL2          ; STORE THE CLOCK VALUE
        BNE     #1,OLCL2,START      ; RESTART IF LOW GOING CLOCK
        BEQ     #1,BEG,ADGN2        ; IS THIS THE FIRST CYCLE ?
        MOV     #29,ROWRG           ; COLUMN-SIZE
        MOV     #29,COLRG           ; ROW-SIZE
        MOV     #1,BEG              ; FLAG THE FIRST CYCLE
;
;
ADGN2 : BEQ     COLRG,RODEC         ; IS A ROW FINISHED ?
        SUB     COLRG,BEG,COLRG     ; IF NOT DECRMENT COLUMN
        BRU     OUTP                ;
RODEC : SUB     ROWRG,BEG,ROWRG     ; IF YES DECREMENT ROW
        MOV     #29,COLRG           ; RESET COLUMN REG
;
OUTP :  MOV(W80) ROWRG,ROWAD         ; OUTPUT AFTER 80 NS
```

```
          MOV(W80) COLRG,COLAD
;
          BRU         START
;
END
```

A.2 EXAMPLE OF AN OUTPUT FILE: DMA.LOG

```
;--LOG     TO SHOW A COMMAND FILE AND AN OUTPUT FILE.
;
;
;PURPOSE
;
; THIS IS A LOG FILE FOR THE MODEL OF THE DMA AND
; TWO LATCHES. IT SHOWS THE COMMAND FILE WITH
; INITIALIZATIONS, INTERCONNECTIONS AND INPUTS.
; IT ALSO SHOWS THE OUTPUT AS IT APPEARS ON THE SCREEN.
;
;ENTRY POINT
; GSPASM DMA
; GSPSIM DMA  ... AND THEN THE INTERACTIVE PROCEDURE
;                 STARTS THE SIMULATION.
;
;**********************************************************
;
; NOW STARTS THE INTERACTIVE PROCEDURE OF STARTING THE
```

```
;
; SIMULATION, AS IT IS SEEN ON THE SCREEN.
;
GSPSIM: GSPLNK file is DMA.
     Enter object file names, one per line. Terminate loop
     with a carriage return.
Enter name of OBJfile: DMA
Are you satisfied with loading?(Y/N) default Yes:
Enter command input file (default DMA):
Enter output device for console log(Print,Term,Disk):
Enter name for log file (default DMA):
GSPSIM running
;
;* * * * * * * * * * * * * * * * * * * * * * *
;

MONI:           ; THIS IS THE "MONITOR" PROMPT; "F" ENTERED
;COLAD  ROWAD   ; HERE STARTS READING THE COMMAND FILE
Y
#MOD=  1 L1= 275 L2=  55 LL= 238 LE= 152 LC=   0
T 2100               ;MAXIMUM TIME FOR THIS SIMULATION
                     ;AS GIVEN IN COMMAND FILE.
N 1 21               ;AN INPUT PIN DECLARED ASYNC.
LINE IS NOW ASYNC.
N 1 24 25            ;
LINE IS NOW ASYNC.
```

```
N 1 34                ;
LINE IS NOW ASYNC.
N 1 44                ;
LINE IS NOW ASYNC.
N 1 55                ;
LINE IS NOW ASYNC.
N 1 65                ;
LINE IS NOW ASYNC.
;
; A LIST OF INTERCONNECTIONS
;
C 1,22  1,1           ; PIN 22 FROM MODULE #1 IS CONNECTED
                      ; TO PIN #1 FROM MODULE #1
C 1,4  0,1            ; THESE ARE THE OUTPUT PINS
C 1,5  0,2            ; CONNECTED TO THE FRONT PANEL
C 1,6  0,3            ; IN LEVEL 0 THESE PINS WILL BE
C 1,7  0,4            ; DISPLAYED IF THEY CHANGE STATES
C 1,8  0,5
C 1,9  0,6
C 1,10  0,7
C 1,11  0,8
C 1,12  0,17
C 1,13  0,18
C 1,14  0,19
C 1,15  0,20
C 1,16  0,21
```

```
C 1,17  0,22
C 1,18  0,23
C 1,19  0,24
;
; PIN 21 IS THE CLOCK INPUT PIN
;
A 1 21 40 1           ; ADD TO MODULE #1 PIN 21
;                     ; AT 40 NS A "1"
A 1 21 168 0          ; ADD TO MODULE #1 PIN 21
A 1 21 296 1          ; AT 168 NS A "0"
A 1 21 424 0
A 1 21 552 1
A 1 21 680 0
A 1 21 808 1
A 1 21 936 0
A 1 21 1064 1
A 1 21 1192 0
A 1 21 1320 1
A 1 21 1448 0
A 1 21 1576 1
A 1 21 1704 0
A 1 21 1832 1
A 1 21 1960 0
A 1 21 2088 1
;
;
```

```
;
L 1 0                ; OUTPUT LEVEL OF MODULE #1 IS "0"
X
X= 1


MONI:


MONI:



MONI: SIMULATION STARTED





SIMULATION TIME =        0
--------------------------
;COLAD  ROWAD
00         00




SIMULATION TIME =      120
--------------------------
;COLAD  ROWAD
1D         1D
```

```
SIMULATION TIME =      376
--------------------------
;COLAD  ROWAD
1C          1D



SIMULATION TIME =      632
--------------------------
;COLAD  ROWAD
1B          1D



SIMULATION TIME =      888
--------------------------
;COLAD  ROWAD
1A          1D
```

```
SIMULATION TIME =     1144
--------------------------
;COLAD  ROWAD
19         1D



SIMULATION TIME =     1400
--------------------------
;COLAD  ROWAD
18         1D



SIMULATION TIME =     1656
--------------------------
;COLAD  ROWAD
17         1D



SIMULATION TIME =     1912
--------------------------
;COLAD  ROWAD
16         1D
```

```
MONI: SIMULATION ENDED


MONI:
DO YOU WANT TO NOTE THE RESULT OF
THIS SIMULATION SESSION ?(Y¬N)
DEFAULT YES).
ZZZZZZ
```

A.3 MEM.SOR

```
;--MEMORY  MEMORY BANKS WITH THE VIDEO LATCHES
;
;PURPOSE
;
; THIS MODULE SIMULATES THE THREE MEMORY CHIPS AND THE
; THREE VIDEO LATCHES FROM THE FIRST MODULE OF THE IPB
; DESIGN: FIND-DIFFS. THE CHIPS INCLUDED IN THIS MODULE:
; TC5564P(L)-10 : 8K X 8 STATIC RAM.
; AM29818       : SSR DIAGNOSTIC/WCS
;                 PIPELINE REGISTER, VIDEO INPUT LATCHES.
;
;ENTRY POINT
;
```

```
; GSPASM MEM
; GSPSIM MEM
;
;DATA FORMAT
;
; INPUT  : DATA FROM THE COMMAND
;          FILE IN DECIMAL.
; OUTPUT : SIGNALS ON THE OUTPUT
;          PINS IN BINARY.
;
;LIMITATIONS
;
; SIMULATES UPTO THE TIME GIVEN
; IN COMMAND FILE BY " T  ##### ".
;
;REMARKS
;
; THIS IS DEVELOPED TO ILLUSTRATE THE READING FROM
; AND WRITING INTO THE MEMORY FROM THE VIDEO
; LATCHES. THE DATA MAY NOT BE NECESSARILY THE ACTUAL
; OF THE SIMULATION. ANY DESIRED DATA CAN BE OBSERVED
; ON THREE SETS OF 8 PINS: OUT1, OUT2 AND OUT3.
; IF THE DESIRED REGISTER IS REG1 ADD THE FOLLOWING
; INSTRUCTION IN THIS FILE WHEREVER DESIRED.
;     MOV(DELAY)  REG1,OUT*
; DELAY IS THE DESIRED DELAY, IF OMITTED DELAY IS O NS.
```

```
        ; OUT* MEANS OUT1, OUT2 OR OUT3.
        ;
***************************************************************
        ;
        ;
        REG(8)       VIDR1,VIDR2,VIDR3,VIDR
        REG(8)       MACT
        REG(1)       RWRG,VLE1R,VLE2R,VLE3R,BEG
        PIN          VIDAT(1,8),OUT1(9,16),OUT2(17,24),OUT3(25,32)
        PIN          CLVD(33),VLE1(34),VLE2(35),VLE3(36),RW(37)
        PIN          DUMMY(151)
        ;
        EVW          ROTIME(7680)
        ;
        START: BNE  VLE1,VLE1R,BEGIN  ; BEGIN IF ANY OF THE LATCHES
               BNE  VLE2,VLE2R,BEGIN  ; IS ENABLED
               BNE  VLE3,VLE3R,BEGIN
               EXC
        ;
        BEGIN: MOV   VLE1,VLE1R     ; STORE THE NEW VALUES
               MOV   VLE2,VLE2R
               MOV   VLE3,VLE3R
               MOV   #1,BEG
               MOV   RW,RWRG
```

```
        BEQ     #1,RWRG,READ    ; START READING IF RW = 1
;                               ; START WRITING IF RW = 0
WRITE:  BEQ     #1,RWRG,READ    ; CONFIRM THE RW SIGNAL
        MOV     VIDAT,VIDR      ; LATCH THE VIDEO DATA
        IDX     MACT(0),8,1
        BEQ     #1,VLE1R,WX0    ; WRITE INTO THE RESPECTIVE
        BEQ     #1,VLE2R,WXSE   ; MEMORY BANK
        BEQ     #1,VLE3R,WXS
        BRU     START
;                           ; WRITE INTO THE FIRST MEMORY BANK
WX0 :   MOV     #0,VLE1         ; RESET THE LATCH ENABLE
        MOV     #0,VLE1R
        MOV(ROTIME) #1,VLE2
        MOV     VIDR,MEM1@1
        BRU     START
;                           ; WRITE INTO THE SECOND MEMORY BANK
WXSE :  MOV     #0,VLE2
        MOV(ROTIME) #1,VLE3
        MOV     VIDR,MEM2@1
        BRU     START
;                           ; WRITE INTO THE THIRD MEMORY BANK
WXS   : MOV      VIDR,MEM3@1
        MOV      #0,VLE3
        MOV(ROTIME) #1,VLE1
        BRU      START
;
```

```
 READ: BEQ    #0,RWRG,WRITE ; RECHECK FOR RW SIGNAL
;
       INC    MACT,MACT
;
FIVE : IDX    MACT(0),8,2
       BEQ    #1,VLE1R,RXO  ; READ FROM THE RESPECTIVE
       BEQ    #1,VLE2R,RXSE ; MEMORY BANKS
       BEQ    #1,VLE3R,RXS
       BRU    START
;                           ; READ FROM THE FIRST MEMORY BANK
RXO :  MOV    MEM1@2,VIDR1
       MOV    #0,VLE1
       MOV    #0,VLE1R
       MOV(ROTIME) #1,VLE2
       BRU    START
;                           ; READ FROM THE SECOND MEMORY BANK
RXSE : MOV    MEM2@2,VIDR2
       MOV    #0,VLE2
       MOV    #0,VLE2R
       MOV(ROTIME) #1,VLE3
       BRU    START
;                           ; READ FROM THE THIRD MEMORY BANK
RXS  : MOV    MEM3@2,VIDR3
       MOV    #0,VLE3
       MOV    #0,VLE3R
       MOV(ROTIME) #1,VLE1
```

```
        BRU     START
;
;THREE MEMORY TABLES OF 256 BYTES EACH.
;
MEM1   : BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
```

```
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
;
;
MEM2    : BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
```

```
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
;
;
MEM3    : BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
```

```
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
```

```
          BYT  #0, #0, #0, #0, #0, #0, #0, #0
;
;
END
```

A.4 NAD.SOR

```
;--NAD SIMULATION OF PREPROCESSORS GENRATING B/W_SET.
;
;
;PURPOSE
;
; THIS IS A SOURCE CODE OF A MOULE THAT SIMULATES THE
; TWO PREPROCESSORS AND A FEW FLIP-FLOPS.
; THIS IS BASICALLY THE COMPUTING UNIT OF FIND_DIFFS.
;
;ENTRY POINT
;
; GSPASM NAD   AND   THEN   GSPSIM NAD
;
;DATA FORMAT
;
; INPUT  : DECIMAL DATA OF THE PIXELS
:          FROM MEMORY OUTPUT
;          LATCHES THROUGH THE COMMAND FILE.
; OUTPUT : BINARY DATA ON OUTPUT PINS:
```

```
;           B/W_SET ON PIN NUMBERS 25 THROUGH 32.
;
;
;**************************************************
;
;
; THIS IS FOR THE TWO 29501 ' S.
; REGISTERS :
;
REG(1) CAR1,CAR2,ALCLR,BEG,CARW,CARB
REG(4) CNTR
REG(8) MSPR,MSPR1,AREG,BREG,A21C,B21C,A32C,B32C,MSPRC
;
;REGISTERS FOR AM29501
; FOR WHITE_DIFFS COMPUTATIONS
;
REG(8) A11,A21,A31,B11,B21,B31,AREG,A21C,B21C,B31C
;
;REGISTERS FOR AM 29501
; FOR BLACK_DIFFS COMPUTATIONS
;
REG(8) A21,A22,A32,B21,B22,B32,BREG,A22C,B22C,A32C,B32C
;
;
PIN     MSP(1,8),TDIFF(9,16),A(17),WHITE(18),BLACK(19)
PIN     OUT(20,23),JUST(24),WB(25),WW(26),SWB(27),SWW(28)
```

```
PIN     SB(29),SW(30),SEB(31),SEW(32),IN(33,40)
PIN     OUT1(41,48),OUT2(49,56),OUT3(57,64)
PIN     ALCL(151)                ; THIS IS A SELF-CALL PIN
;
EVW        W5(5),W32(32),W64(64),W96(96)
;
START: BNE    ALCL,ALCLR,BEGIN   ; BEGIN ON CLOCK CHANGE
       EXR
;
BEGIN: MOV    ALCL,ALCLR
       BNE    #1,ALCLR,START ; RESTART IF CLOCK LO GOING
       MOV    #0,ALCL
       MOV    #0,ALCLR
       MOV(W32) #1,ALCL ; SCHDULE NEXT CALL AFTER 32NS
       MOV    #1,BEG
       MOV    TDIFF,A31   ; STORE THRESHOLD TDIFF IN A3
       MOV    TDIFF,A32   ; OF BOTH THE AM 29501'S
       MOV    MSP,MSPR1
;
       IDX    CNTR(0),4,1
       BRU    ALUBR@1     ; FOR THE 8 CLOCK CYCLES
;
ALUBR: BYT    99, 100, 200, 300, 400, 500, 600, 700
;
99:    NOP                ; THE FIRST CLOCK CYCLE TO
       INC      CNTR,CNTR
```

```
        EXR                     ;  EXIT AND RESTART
;
100 : MOV   MSPR1,MSPR          ; STORE THE DATA ON MSP
      COM   A32,A32C            ; THIS IS FOR
      ADD   MSPR,A32C,B32     ; MSP - A3 = B3 (1'S COMPL.)
      MOV   C,CARB            ; STORE THE CARRY
      COM   CARB,CARB         ; COMPLEMENT THE CARRY
      ADD   MSPR1,A31,B31     ; MSP + A3 = B3
      MOV   C,CARW            ; STORE THE CARRY
      COM   B31,B31C          ; FOR FUTURE USE
      MOV   CARW,WHITE        ; OBSERVE WHITE CARRY
      MOV   CARB,BLACK        ; OBSERVE BLACK CARRY
      INC   CNTR,CNTR
      EXR
;
200:  NOP
      INC   CNTR,CNTR
      EXR
;
300:  COM   A22,A22C
      ADD   B32,A22C,BREG   ; B3 - A22 = DUMMY REG.
      BEQ   #1,CARB,FORC1   ; IS PREVIOUS CARRY 0?
      MOV   C,CAR2          ; IF NOT STORE THIS CARRY
      BRU   BOUT1
FORC1: MOV     #0,CAR2        ; IF YES INHIBIT CARRY
BOUT1: NOP
```

```
        ADD     B31C,A21,AREG  ; - B3 + A2
        BEQ     #1,CARW,INHB1  ; IS PREVIOUS CARRY 1?
        MOV     C,CAR1         ; IF NOT STORE THIS CARRY
        BRU     COUT1
INHB1:  MOV     #0,CAR1        ; IF YES INHIBIT THIS CARRY
COUT1:  MOV     CAR1,WHITE
        MOV     CAR2,BLACK
        MOV(W96) CAR1,WW       ; WHITE BIT OF WEST SIDE
        MOV(W96) CAR2,WB       ; AND BLCK BIT OF WEST SIDE
        INC     CNTR,CNTR      ; PUT THEM TO THE BW_SET LATCH
        EXR
;
400:    MOV     A11,A21        ; SHIFT A1 TO A2
        MOV     A12,A22
        MOV     MSPR,A11       ; MOV MSP TO A1
        MOV     MSPR,A12
        MOV     B21,OUT1
        MOV     B22,OUT2
        COM     B22,B22C       ; B3 - B2 = DUMMY REG.
        ADD     B32,B22C,BREG
        BEQ     #1,CARB,FORC2  ; INHIBIT CARRY SIMILAR
        MOV     C,CAR2         ; TO THE LAST ONE
        BRU     BOUT2
FORC2:  MOV     #0,CAR2
BOUT2:  NOP
        ADD     B31C,B21,AREG  ; - B3 + B2 = DUMMY REG.
```

```
        BEQ      #1,CARW,INHB2
        MOV      C,CAR1
        BRU      COUT2
INHB2:  MOV      #0,CAR1
COUT2:  MOV      CAR1,WHITE
        MOV      CAR2,BLACK
        MOV(W64) CAR1,SWW       ; WHITE BIT OF SOUTH-WEST
        MOV(W64) CAR2,SWB       ; BLACK BIT OF SOUTH-WEST
        INC      CNTR,CNTR
        EXR
;
500 :   MOV      MSPR1,MSPR     ; STORE MSP
        COM      MSPR,MSPRC
        ADD      B32,MSPRC,BREG ; B3 - MSP = DUMMY REG.
        BEQ      #1,CARB,FORC3
        MOV      C,CAR2
        BRU      BOUT3
FORC3:  MOV      #0,CAR2
BOUT3:  NOP
        ADD      B31C,MSPR,AREG ; - B3 + MSP = DUMMY REG.
        BEQ      #1,CARW,INHB3
        MOV      C,CAR1
        BRU      COUT3
INHB3:  MOV      #0,CAR1
COUT3:  MOV      CAR1,WHITE
        MOV      CAR2,BLACK
```

```
        MOV(W32) CAR1,SW
        MOV(W32) CAR2,SB
        INC      CNTR,CNTR
        EXR
;
600 :   MOV      B11,B21           ; SHIFT B1 TO B2
        MOV      B12,B22
        MOV      MSPR1,B11         ; MOV MSP TO B1
        MOV      MSPR1,B12
        MOV      MSPR1,MSPR
        COM      MSPR,MSPRC
        ADD      MSPRC,B32,BREG ; - MSP + B3 = DUMMY REG.
        BEQ      #1,CARB,FORC4
        MOV      C,CAR2
        BRU      BOUT4
FORC4:  MOV      #0,CAR2
BOUT4:  NOP
        ADD      B31C,MSPR,AREG ; - B3 + MSP = DUMMY REG.
        BEQ      #1,CARW,INHB4
        MOV      C,CAR1
        BRU      COUT4
INHB4:  MOV      #0,CAR1
COUT4:  MOV      CAR1,WHITE
        MOV      CAR2,BLACK
        MOV      CAR1,SEW          ; WHITE BIT OF SOUTH-EAST
        MOV      CAR2,SEB          ; BLACK BIT OF SOUTH-EAST
```

```
        INC     CNTR,CNTR
        EXR
;
700 :   MOV     #0,CNTR
        MOV     #0,WW           ; CLEAR ALL FLIP-FLOPS
        MOV     #0,WB
        MOV     #0,SWW
        MOV     #0,SWB
        MOV     #0,SW
        MOV     #0,SB
        MOV     #0,SEW
        MOV     #0,SEB
        MOV     #0,WHITE
        MOV     #0,BLACK
        MOV     #0,CARW
        MOV     #0,CARB
        EXR
;
END
```

APPENDIX B. SDF32.SOR

[Source Code for the model of Sum_Diffs]

```
;--SDF SIMULATION OF THE WHOLE SUM_DIIFS MODULE.
;
;
;PURPOSE
;
;  THIS IS A SOURCE CODE OF THE MODULE THAT SIMULATES
;  THE WHOLE SUM_DIFFS. THIS VERSION IS FOR WINDOW SIZE
;  7 X 7 WITH 32 VECTORS. THE OTHERS VERSIONS FOR
;  7 X 7 X 20 AND 5 X 5 X 8 ARE IN  THE DIRECTORY
;  US1:[DEO.GSP] ON VAX 11/785 AS SDF20.SOR AND
;  SDF8.SOR THE COMMAND FILE FOR ALL THE THREE VERSIONS
;  IS SDF.GCM.
;
;  ENTRY POINT
;
;  GSPASM SDF32  AND  THEN  GSPSIM  SDF32.
;
;  DATA FORMAT
;
;  INPUT  :  DECIMAL DATA FROM FIND_DIFFS
;            THROUGH SDF.GCM
;  OUTPUT :  BINARY DATA ON THE OUTPUT PINS:
```

```
;             B_SUM : PIN # 9 TO 16
;             W_SUM : PIN # 17 TO 25
;
;
;*******************************************
;
;
; THIS MODULE HAS FOUR MAIN MEMORY CHIPS,
; 8 SCRATCH PAD MEMORY CHIPS, ABOUT 15
; LATCHES, TWO PREPROCESSORS AND SO ON.
;
; IF MORE OUTPUTS ARE TO BE OBSERVED,
; OUT1 AND OUT2 ARE KEPT FOR THAT USE.
; THE ROW-SIZE CAN BE VARIED BY GIVING
; THE REQUIRED DATA ON
; PIN # 43, THROUGH THE COMMAND FILE.
;
;****************************************************
;
REG(8)  BWSR1,BWSR2,BWSR3,BWSR4,BWSR,MACT1
REG(8)  MACT,ROSIZ,BWDUM,ROW,SPLNU,WSLO,BSLO
REG(8)  CON1,CON2,CON3,CON4,CON5,CON6,BWST1,BWST2,SPL
REG(8)  ADW1,ADW2,ADSE1,ADSE2,ADS1,ADS2,ADSW1,ADSW2
REG(8)  REG1,REG2,REG3,REG4,CNTRG,CON7
REG(8)  SAVE1,SAVE2,SAVE3,SAVE4,DUPL1,DUPL2,DUPL3,DUPL4
REG(8)  BSUMR,WSUMR,BSUML,WSUML
```

```
REG(7)  SPADR
REG(5)  UPAD1,UPAD2,UPAD3
REG(3)  EK3,DO3,TIN3,CHAR3,PACH3,CHHE3,SAT3
REG(3)  PER,UPAD,ROWNO
REG(1)  EK1,DO1,TIN1,CHAR1,PACH1,CHHE1,SAT1
REG(1)  RWRG,SLE1R,SLE2R,SLE3R,SLE4R,BEG,DUMRG,INRG
REG(1)  BC,WC,SDB1,SDW1,DI,FDUP
;
;
PIN     BWSET(1,8),BSUMO(9,16),WSUMO(17,24),OUT3(25,32)
PIN     OUT4(33,40),RW(41),CLBW(42),ROWSZ(43,50)
PIN     CNTR(51,54)
PIN     DUMMY(151),SLE1(152),SLE2(153)
PIN     SLE3(154),SLE4(155)
;
EVW     W2(2),W16(16),W32(32),W64(64),W96(96)
EVW     W240(240),W254(254),W255(255),W256(256)
;
START:  BNE   DUMMY,DUMRG,BEGIN  ; BEGIN IF SELF-CALLED
        EXC
;
BEGIN:  MOV   DUMMY,DUMRG
        BEQ   DUMRG,START     ; GO BACK IF LO GOING EDGE
        MOV   #1,EK1          ; INITIALIZE SOME REGISTERS
        MOV   #1,EK3          ; FOR COMPARISONS
        MOV   #2,DO3
```

```
        MOV    #3,TIN3
        MOV    #4,CHAR3
        MOV    #5,PACH3
        MOV    #6,CHHE3
        MOV    #7,SAT3
        MOV    #0,DUMMY
        MOV    #0,DUMRG
        BEQ     INRG,INIT
        MOV(W32) #1,DUMMY  ; CLOCK PERIOD IS 32 NS
        BRU     REGOP       ; FOR REGULAR OPERATION
INIT :  MOV(W256) #1,DUMMY   ; IT IS 256 NS FOR
REGOP:  NOP                 ; INITIALIZING MEMORY BANKS
        MOV    BWSET,BWDUM
        MOV    ROWSZ,ROSIZ  ; INPUT ROW SIZE
;
WRITE: NOP
       MOV     SLE1,SLE1R  ; THIS IS DONE FOR ONE FORM
       MOV     SLE2,SLE2R  ; OF IMPLEMENTING A MICROPROG.
       MOV     SLE3,SLE3R
       MOV     SLE4,SLE4R
       BNE     INRG,NOTIN
       BEQ     EK1,SLE1R,IN0 ; THIS IS FOR ANOTHER FORM
       BEQ     EK1,SLE2R,IN1 ; OF IMPLEMENTING A MICROPROG.
       BEQ     EK1,SLE3R,IN2
       BEQ     EK1,SLE4R,IN3
```

```
        BRU     START
;
INO :   MOV      BWDUM,BWSR
        IDX      MACT(0),5,1
        MOV      #1,CNTR        ; OUTPUT A REFERENCE NUMBER
        BNE      INRG,NOTIN
        MOV      BWSR,MEM1@1 ; STORE FIRST ROW OF BW_SET
        MOV      #0,SLE1        ; IN FIRST MEMORY BANK
        MOV(W254) #1,SLE1       ; SCHEDULE NEXT ENABLE
        INC      MACT,MACT      ; NEXT ADDRESS
        BNE      ROSIZ,MACT,START ; START IF ROW NOT OVER
OTPTO : MOV      #0,MACT        ; IF ROW OVER: RESET ADDRESS
        MOV(W255) #0,SLE1 ; DISABLE FIRST LATCH AND MEM.
        MOV(W254) #1,SLE2 ; ENABLE SECOND LATCH AND MEM.
        BRU      START
;
IN1:    MOV      BWDUM,BWSR
        IDX      MACT(0),5,1
        MOV      #2,CNTR
        BNE      INRG,NOTIN
        MOV      BWSR,MEM2@1 ; WRITE BW_SET IN SECOND MEM.
        MOV     #0,SLE2         ; THE ABOVE PROCESS CONTINUES
        MOV(W254) #1,SLE2       ;
        INC      MACT,MACT      ;
        BNE      ROSIZ,MACT,START
OTPT1:  MOV      #0,MACT
```

```
        MOV(W255) #0,SLE2
        MOV(W254) #1,SLE3
        BRU       START
;
IN2:    MOV       BWDUM,BWSR
        IDX       MACT(0),5,1
        MOV       #3,CNTR
        BNE       INRG,NOTIN
        MOV       BWSR,MEM3@1
        MOV       #0,SLE3
        MOV(W254) #1,SLE3
        INC       MACT,MACT
        BNE       ROSIZ,MACT,START
OTPT2:  MOV       #0,MACT
        MOV(W255) #0,SLE3
        MOV(W254) #1,SLE4
        BRU       START
;
IN3:    MOV       BWDUM,BWSR
        BNE       #1,SLE4R,START
        IDX       MACT(0),5,1
        MOV       #4,CNTR
        BNE       INRG,NOTIN
        MOV       BWSR,MEM4@1
        MOV       #0,SLE1
        MOV       #0,SLE2
```

```
        MOV     #0,SLE3
        MOV     #0,SLE4
        MOV     #0,SLE1R
        MOV     #0,SLE2R
        MOV     #0,SLE3R
        MOV     #0,SLE4R
        MOV(W254) #1,SLE4
        INC      MACT,MACT
        BNE      ROSIZ,MACT,START ; NOW IF ROW OVER
;
OVER:   MOV(W255) #0,SLE4        ; INITIALIZE FOR NORMAL
        MOV(W254)      #1,SLE1   ; OPERATION
        MOV      #1,INRG
        MOV      #0,SPADR
        MOV      #0,PER
        MOV      #0,ROWNO
        MOV      #0,MACT
        MOV      #192,CON1  ; MASKS FOR MASKING OUT
        MOV      #3,CON2    ; UNWANTED BITS FROM A BYTE
        MOV      #12,CON3
        MOV      #48,CON4
        MOV      #15,CON5
        MOV      #240,CON6
        BRU      START
;
;
```

```
;
NOTIN : IDX     MACT(0),8,1
 ;IDX      UPAD(0),5,7   ; THIS IS ONE WAY OF
 ;MOV      SDUP@7,UPAD1  ; IMPLEMENTING A MICROPROG.
 ;IDX      UPAD1(0),1,7
 ;MOV(W32)      @7,SLE1
 ;IDX      UPAD1(1),1,8
 ;MOV(W32)      @8,SLE2
 ;IDX      UPAD1(2),1,7
 ;MOV(W32)      @7,SLE3
 ;IDX      UPAD1(3),1,8
 ;MOV(W32)      @8,SLE4
        IDX     PER(0),3,2
        IDX     SPADR(0),7,3
        BRU     TBL@2
;      THE 8 CLOCK CYCLES
 TBL   : BYT      0, 1, 2, 3, 4, 5, 6, 7
;
1      : NOP
;
        BEQ    SDB1,BLACK      ; ADJUST 5TH BIT OF
        BIS    #4,BSUML        ; B_SUM AND W_SUM
BLACK : BEQ    SDW1,WHITE
        BIS    #4,WSUML
WHITE : NOP
```

```
;
;
        MOV     #1,DI             ; A SIGNAL FROM FIND_DIFFS
        BEQ     ROWNO,RST0        ; THIS IS ANOTHER WAY OF
        BEQ     EK3,ROWNO,RST1    ; IMPLEMENTING MICROPROG.
        BEQ     DO3,ROWNO,RST2
        BEQ     TIN3,ROWNO,RST3
RST0  : MOV     BWSR,DUPL1       ; SAVE THE BW_SET BYTE
        MOV     BWSR,MEM1@1      ; WRITE INTO THE FRIST MEMORY
        MOV     BWSR1,SAVE1
        BRU     WSET0
;
RST1  : MOV     BWSR,DUPL2
        MOV     BWSR,MEM2@1      ; WRITE INTO SECOND MEMORY
        MOV     BWSR2,SAVE2
        BRU     WSET1
;
RST2  : MOV     BWSR,DUPL3
        MOV     BWSR,MEM3@1      ;
        MOV     BWSR3,SAVE3
        BRU     WSET2
;
RST3  : MOV     BWSR,DUPL4
        MOV     BWSR,MEM4@1      ;
        MOV     BWSR4,SAVE4
        BRU     WSET3
```

```
 ;BEQ      #1,SLE1R,WSET0
 ;BEQ      #1,SLE2R,WSET1
 ;BEQ      #1,SLE3R,WSET2
 ;BEQ      #1,SLE4R,WSET3
;
ZERO: NOP
      MOV      #1,FDUP
      IDX      SPLNU(0),8,6  ; ACCORDING TO CONTENTS OF
      MOV        CNTR1@6,CON7; SCRATCH PAD OUTPUT LATCH
      MOV(W32) CON7,CNTRG    ; GET THE COUNTER DATA
      MOV        BWSR3,OUT3
      MOV        BWSR4,OUT4
      MOV     BSUML,BSUMO
      MOV     WSUML,WSUMO
;
      MOV     #0,BSUML      ; INITIALIZE LATCHES
      MOV     #0,WSUML
      MOV     #0,BSLO
      MOV     #0,WSLO
      MOV     #0,BC
      MOV     #0,WC
      MOV     #0,SDB1
      MOV     #0,SDW1
      JSR     SUMO         ; FOR SUMMING UP W AND B BITS
;
      MOV     #2,PER
```

```
        BRU     START
;
WSET0: NOP
        MOV     SAVE1,BWST1 ; PASS BW_SET FROM FIRST LATCH
        JSR     WRRD        ; TO WRITE IN R1 AND READ FROM R2
;
        BEQ     EK3,PER,ZERO ; GO BACK TO RESPECTIVE ORIGIN
        BEQ     TIN3,PER,TWO
        BEQ     SAT3,PER,SIX
;
        BRU     START
;
WSET1: NOP
        MOV     SAVE2,BWST1  ; OUTPUT ENABLE SECOND LATCH
        JSR     WRRD
        BEQ     EK3,PER,ZERO
        BEQ     TIN3,PER,TWO
        BEQ     SAT3,PER,SIX
;
        BRU     START
;
WSET2: NOP
        MOV     SAVE3,BWST1 ; OUTPUT ENABLE THIRD LATCH
        JSR     WRRD
        BEQ     EK3,PER,ZERO
        BEQ     TIN3,PER,TWO
```

```
        BEQ     SAT3,PER,SIX
;
         BRU     START
;
WSET3:  NOP
        MOV     SAVE4,BWST1 ; OUTPUT ENABLE FORTH LATCH
        JSR     WRRD
        BEQ     EK3,PER,ZERO
        BEQ     TIN3,PER,TWO
        BEQ     SAT3,PER,SIX
;
        BRU     START
;
2     : MOV     BWSR1,SAVE1 ; SAVE CONTENTS OF ALL LATCHES
        MOV     BWSR2,SAVE2
        MOV     BWSR3,SAVE3
        MOV     BWSR4,SAVE4
;
        MOV     #0,DI
;
         BEQ     ROWNO,RSET3
         BEQ     EK3,ROWNO,RSET0
         BEQ     DO3,ROWNO,RSET1
         BEQ     TIN3,ROWNO,RSET2
;
 ;BEQ     #1,SLE1R,RSET0
```

```
;BEQ     #1,SLE2R,RSET1
;BEQ     #1,SLE3R,RSET2
;BEQ     #1,SLE4R,RSET3
;
ONE  :  NOP
;
        MOV      #0,DI
        MOV      #1,FDUP
        IDX      SPLNU(0),8,6
        MOV       CNTRO@6,CON7
        MOV(W32) CON7,CNTRG
        JSR      SUMO
;
        INC      UPAD,UPAD
        MOV      #3,PER
        BRU      START
;
RSETO: NOP
       MOV      SAVE1,BWST2; OUTPUT ENABLE FIRST LATCH
       JSR      RDWR     ; READ FROM R1 AND WRITE IN R2
       BEQ      DO3,PER,ONE; GO BACK TO RESPECTIVE ORIGINS
       BEQ      CHAR3,PER,THRI
       BEQ      PACH3,PER,FOUR
       BEQ      CHHE3,PER,FIVE
       BEQ      PER,SEVEN
;
```

```
        BRU     START
;
RSET1 : NOP
        MOV     SAVE2,BWST2; OUTPUT ENABLE SECOND LATCH
        JSR     RDWR
        BEQ     DO3,PER,ONE
        BEQ     CHAR3,PER,THRI
        BEQ     PACH3,PER,FOUR
        BEQ     CHHE3,PER,FIVE
        BEQ     PER,SEVEN
;
        BRU     START
;
RSET2 : NOP
        MOV     SAVE3,BWST2 ; OUTPUT ENABLE THIRD LATCH
        JSR     RDWR
        BEQ     DO3,PER,ONE
        BEQ     CHAR3,PER,THRI
        BEQ     PACH3,PER,FOUR
        BEQ     CHHE3,PER,FIVE
        BEQ     PER,SEVEN
;
        BRU     START
;
RSET3 : NOP
        MOV     SAVE4,BWST2 ; OUTPUT ENABLE FORTH LATCH
```

```
        JSR      RDWR
        BEQ      DO3,PER,ONE
        BEQ      CHAR3,PER,THRI
        BEQ      PACH3,PER,FOUR
        BEQ      CHHE3,PER,FIVE
        BEQ      PER,SEVEN
;
        BRU      START
;
;
3     : NOP
        MOV       #0,DI
        INC       MACT,MACT
        BNE       ROSIZ,MACT,ENDRO
        MOV       #0,MACT
        NOP
ENDRO:  NOP
;
;
        BEQ       ROWNO,WSTO
        BEQ       EK3,ROWNO,WST1
        BEQ       DO3,ROWNO,WST2
        BEQ       TIN3,ROWNO,WST3
;
WSTO  : MOV       BWSR,BWSR1   ; OUTPUT ENABLE FIRST LATCH
        MOV       BWSR1,SAVE1 ; SAVE ITS CONTENTS
```

```
        BRU         WSET0      ; PASS IT FOR WRITE IN R1
;
WST1  : MOV         BWSR,BWSR2  ; SAME WITH SECOND LATCH
        MOV         BWSR2,SAVE2
        BRU         WSET1
;
WST2  : MOV         BWSR,BWSR3  ; SAME WITH THIRD LATCH
        MOV         BWSR3,SAVE3
        BRU         WSET2
;
WST3  : MOV         BWSR,BWSR4  ; SAME WITH FORTH LATCH
        MOV         BWSR4,SAVE4
        BRU         WSET3
;
;BEQ            #1,SLE1R,WSET0
;BEQ            #1,SLE2R,WSET1
;BEQ            #1,SLE3R,WSET2
;BEQ            #1,SLE4R,WSET3
;
TWO   : NOP
        MOV     #0,DI
        MOV     #1,FDUP
        IDX     SPLNU(0),8,6; GET COUNTER DATA ACCORDING TO
        MOV     CNTR0@6,CON7    ; S-P LATCH CONTENTS
        MOV(W32)    CON7,CNTRG
        JSR     SUM0        ; SUM UP W AND B BITS
```

```
        NOP
        MOV     BWSR3,OUT3
        MOV     BWSR4,OUT4
;
        INC         UPAD,UPAD
        MOV         #4,PER
        BRU         START
;
;
4     :  NOP
        MOV     #0,DI
        NOP
        MOV     DUPL1,SAVE1   ; RECALL THE SAVED DATA
        MOV     DUPL2,SAVE2
        MOV     DUPL3,SAVE3
        MOV     DUPL4,SAVE4
;
        BEQ     ROWNO,RSET0
        BEQ     EK3,ROWNO,RSET1
        BEQ     DO3,ROWNO,RSET2
        BEQ     TIN3,ROWNO,RSET3
;
;BEQ    #1,SLE1R,RSET0
;BEQ    #1,SLE2R,RSET1
;BEQ    #1,SLE3R,RSET2
```

```
 ;BEQ     #1,SLE4R,RSET3
;
THRI :  NOP
        MOV      #0,DI
        MOV      #1,FDUP
        IDX      SPLNU(0),8,6   ; GET THE COUNTER DATA
        MOV      CNTRO@6,CON7
        MOV(W32)   CON7,CNTRG
        JSR      SUMO         ; SUM UP W AND B BITS
;
        INC      UPAD,UPAD
        MOV      #5,PER
        BRU      START
;
;
5     :  NOP ;
        MOV      #0,DI
        MOV      BWSR3,OUT3
        MOV      BWSR4,OUT4
 FOUR : MOV      #1,RW
        MOV      #1,FDUP
        JSR      WRRD      ; WRITE IN R1 AND READ FROM R2
        IDX      SPLNU(0),8,6   ; PROPER COUNTER DATA
        MOV      CNTRO@6,CON7
        MOV(W32)    CON7,CNTRG
        JSR      SUMO         ; SUM UP W AND B BITS
```

```
        INC     UPAD,UPAD
        MOV     #6,PER
        BRU     START
;
6     : NOP
        MOV     #1,DI
FIVE  : NOP
        MOV     #1,FDUP
        JSR     RDWR            ; READ FROM R1, WRITE IN R2
        IDX     SPLNU(0),8,6
        MOV     CNTR1@6,CON7    ; GET PROPER COUNTER DATA
        MOV(W32)      CON7,CNTRG
        JSR     SUMO
        INC     UPAD,UPAD
        MOV     #7,PER
        BRU     START
;
7     : NOP
        MOV     MEM1@1,BWSR1    ; OUTPUT ENABLE MEMORY BANKS
        MOV     MEM2@1,BWSR2    ; LATCH THE NEXT BW_SETS
        MOV     MEM3@1,BWSR3
        MOV     MEM4@1,BWSR4
        MOV     #1,BEG
        MOV     #1,DI
        MOV     BWSR1,SAVE1     ; SAVE CONCTENTS OF ALL
        MOV     BWSR2,SAVE2     ; THE FOUR LATCHES
```

```
        MOV     BWSR3,SAVE3
        MOV     BWSR4,SAVE4
        BEQ     ROWNO,WSET2
        BEQ     EK3,ROWNO,WSET3
        BEQ     DO3,ROWNO,WSET0
        BEQ     TIN3,ROWNO,WSET1
;
 ;BEQ    #1,SLE1R,WSET0
 ;BEQ    #1,SLE2R,WSET1
 ;BEQ    #1,SLE3R,WSET2
 ;BEQ    #1,SLE4R,WSET3
;
SIX :   NOP
        MOV     #0,FDUP
        IDX     SPLNU(0),8,6    ; GET THE PROPER DATA FROM
        MOV     CNTR1@6,CON7    ; THE COUNTER
        MOV(W32)     CON7,CNTRG
        JSR     SUMO         ; SEND IT FOR SUMMING UP W AND B
        MOV     #0,PER
        INC     UPAD,UPAD
        INC     ROW,ROW         ; ONE PIXEL FINISHED
        BNE     ROSIZ,ROW,START; IS THE ROW OVER?
        MOV     #0,ROW          ; IF NOT START AGAIN
FINAL : INC     ROWNO,ROWNO    ; IF YES, START THE NEXT ROW
        MOV     ROWNO,UPAD
        MOV     #0,SPADR
```

```
          BRU     START
;
O      :  MOV      BWDUM,BWSR
          MOV     #1,DI
          MOV     BWSR1,SAVE1
          MOV     BWSR2,SAVE2
          MOV     BWSR3,SAVE3
          MOV     BWSR4,SAVE4
;
          BEQ     ROWNO,RSET1
          BEQ     EK3,ROWNO,RSET2
          BEQ     DO3,ROWNO,RSET3
          BEQ     TIN3,ROWNO,RSET0
;
 ;BEQ    #1,SLE1R,RSET0
 ;BEQ    #1,SLE2R,RSET1
 ;BEQ    #1,SLE3R,RSET2
 ;BEQ    #1,SLE4R,RSET3
SEVEN : MOV    #1,FDUP
        IDX    SPLNU(0),8,6
        MOV    CNTR1@6,CON7
        MOV(W32)    CON7,CNTRG
        JSR    SUMO
;
;
```

```
        NOP
;
        MOV     #1,PER
        INC     UPAD,UPAD
        BRU     START
;
;
;  SUBROUTINE 'WRRD' FOR WRITE IN R1 AND READ FROM R2.
;
WRRD: AND   CON1,BWST1,REG1  ; MASK OFF 6 BITS KEEPING
      AND   CON2,BWST1,REG2  ; 2 BITS THAT ARE REQUIRED
      AND   CON3,BWST1,REG3  ; FOR EACH DIRECTION OUT OF
      AND   CON4,BWST1,REG4  ; S, SW, SE AND W
;
      MOV      #0,RW
      MOV      #0,SPL          ; CLEAR S-P LATCH
;
;
;
      MOV(W32) WR1@3,ADW1   ; LATCH ADDRESS FOR WR1
      IDX      ADW1(0),8,4
      BEQ      PACH3,PER,RDWR2  ; SKIP WRITE IN WR1 IF T5
      MOV      REG1,SPW1@4     ; OTHERWISE WRITE IN WR1
RDWR2: NOP
      MOV(W32)   WR2@3,ADW2   ; LATCH ADDRESS FOR WR2
      IDX      ADW2(0),8,5
```

```
        MOV       SPW2@5,SPL        ; READ FROM WR2 INTO S-P LATCH
;
;
        MOV(W32) SER1@3,ADSE1; LATCH ADDRESS FOR SER1
        IDX       ADSE1(0),8,4
        BEQ       PACH3,PER,RDSE2 ; SKIP WRITE IN SER1 IF T5
        MOV       REG2,SPSE1@4    ; OTHERWISE WRITE IN SER1
RDSE2 :   NOP
        MOV(W32) SER2@3,ADSE2; LATCH ADDRESS FOR SER2
        IDX       ADSE2(0),8,7
        MOV       SPSE2@7,REG2    ; READ FROM SER2 INTO REG2
        OR        REG2,SPL,SPL    ; PUT W AND SE BITS AT
;                                 ; PROPER PLACES IN S- LATCH
;
;
        MOV(W32) SR1@3,ADS1   ; LATCH ADDRESS FOR SR1
        IDX       ADS1(0),8,4
        BEQ       PACH3,PER,NOSR2  ; SKIP BOTH OPER. IF T5
        MOV       REG3,SPS1@4     ; OTHERWISE WRITE IN SR1
        MOV(W32) SR2@3,ADS2   ; ADDRESS FOR SR2
        IDX       ADS2(0),8,5
        MOV       SPS2@5,REG3    ; READ FROM SR2
        OR        REG3,SPL,SPL   ; PUT S BITS WITH W AND SE BITS
        BRU       NORM
;
NOSR2 :   NOP                    ; IF T5, READ FROM SR1 ONLY
```

```
        MOV     SPS1@4,REG3
        OR      REG3,SPL,SPL
;
;
;
NORM :  NOP
        MOV(W32)      SWR1@3,ADSW1; SAME OPERATIONS
        IDX     ADSW1(0),8,4    ; FOR SW
        BEQ     PACH3,PER,RDSW2
        MOV     REG4,SPSW1@4
RDSW2 : NOP
        MOV(W32)  SWR2@3,ADSW2
        IDX     ADSW2(0),8,5
        MOV     SPSW2@5,REG4
        OR      REG4,SPL,SPL
        MOV(W32)  SPL,SPLNU
        INC     SPADR,SPADR
RET1  : RTS                     ; RETURN FROM SUBROUTINE
;
;
;  SUBROUTINE 'RDWR' FOR READ IN R1 AND WRITE FROM R2.
;  THIS SUBROUTINE IS EXACTLY SIMILAR i
;  TO THE PREVIOUS ONE, EXCEPT THAT
;  IT IS FOR WRITING IN R2 AND READING FROM R1
;
RDWR    : AND     CON1,BWST2,REG1
```

```
        AND         CON2,BWST2,REG2
        AND         CON3,BWST2,REG3
        AND         CON4,BWST2,REG4
;
        MOV         #0,SPL
        MOV         #1,RW
        MOV(W32)         WR2@3,ADW2
        IDX         ADW2(0),8,4
        BEQ         CHHE3,PER,NOW2
        MOV         REG1,SPW2@4
NOW2 :  NOP
        MOV(W32)         WR1@3,ADW1
        IDX         ADW1(0),8,5
        MOV         SPW1@5,SPL
;
;
        MOV(W32)         SER2@3,ADSE2
        IDX         ADSE2(0),8,8
NOW   : BEQ         CHHE3,PER,NOSE2
        MOV         REG2,SPSE2@8
NOSE2 : NOP
        MOV(W32)         SER1@3,ADSE1
        IDX         ADSE1(0),8,5
        MOV        SPSE1@5,REG2
```

```
            OR          REG2,SPL,SPL
;
;
            MOV(W32)        SR2@3,ADS2
            IDX         ADS2(0),8,4
NOSE :      BEQ         CHHE3,PER,NOR2
            MOV         REG3,SPS2@4
NOR2  :     NOP
            MOV(W32)        SR1@3,ADS1
            IDX         ADS1(0),8,5
            MOV         SPS1@5,REG3
            OR          REG3,SPL,SPL
;
;
            MOV(W32)        SWR2@3,ADSW2
            IDX         ADSW2(0),8,4
            BEQ         CHHE3,PER,NOSW
            MOV         REG4,SPSW2@4
NOSW :      NOP
            MOV(W32)        SWR1@3,ADSW1
            IDX         ADSW1(0),8,5
            MOV         SPSW1@5,REG4
            OR          REG4,SPL,SPL
            MOV(W32) SPL,SPLNU
            INC         SPADR,SPADR
RET2  :     RTS
```

```
;
;   SUBROUTINE 'SUM' FOR GETIING B/W SUMS .
;
SUMO  :  NOP ;
        AND    CNTRG,CON5,BSUMR  ; MASK OFF UNWANTED 4 BITS
        ADD    BSUMR,BSUML,BSLO  ; ADD B BITS WITH PREVIOUS
        MOV    C,BC             ; LATCH THE CARRY
        MOV(W32) BSLO,BSUML       ; OUTPUT ENABLE LATCH
                                  ; AT THE NEXT CLOCK
        AND    CNTRG,CON6,WSUMR  ; SAME OPERATIONS
        MOV    #0,@7       ; INDEX REG #7 HAS TO BE CLEARED
        SHR    WSUMR       ; FOR SHR OPERATION
        SHR    WSUMR       ; THESE FOUR SHIFTS GET THE
        SHR    WSUMR       ; W BITS TO LSB POSITIONS
        SHR    WSUMR
        ADD    WSUML,WSUMR,WSLO
        MOV    C,WC             ; LATCH CARRY
        MOV(W32) WSLO,WSUML
        AND    SDB1,FDUP,SDB1      ; THE 5TH BIT OF B_SUM:
        OR     BC,SDB1,SDB1        ; (SDB1)(FDUP) + (BC)
        AND    SDW1,FDUP,SDW1      ; SIMILARLY SDW1 IS:
        OR     WC,SDW1,SDW1        ; (SDW1)(FDUP) + (WC)
;
;WHERE:
;: SDB1/SDW1 = SINGLE BIT OUTPUTS OF THIS FUNCTION
;  BC/WC     = CARRY'S FROM B AND W ADD OPERATIONS
```

```
;   FDUP        = SIGNAL FROM FIND_DIFFS
;                                  ;RETURN FROM SUBROUTINE
        RTS
;
;   FOUR MEMORY TABLES OF 256 BYTES EACH.
;
MEM1 :  BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
```

```
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
;
MEM2 :  BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
```

```
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0

;
MEM3 :  BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
```

```
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
BYT  #0, #0, #0, #0, #0, #0, #0, #0
```

```
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
;
MEM4 :  BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
```

```
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
        BYT  #0, #0, #0, #0, #0, #0, #0, #0
; A PART OF SUM_DIFFS MICROPROGRAM
SDUP :  BYT  #1, #8, #1, #1, #0, #0, #4, #2;
 ;BYT  #0, #0, #0, #0, #0, #0, #0, #1
 ;BYT  #0, #0, #0, #0, #0, #0, #1, #0
 ;BYT  #0, #1, #0, #0, #0, #0, #0, #0
;
;
SR1  :  BYT  #0,  #06, #02, #09, #07, #10, #04, #07
        BYT  #03, #09, #05, #12, #10, #13, #07, #10
        BYT  #06, #12, #08, #15, #13, #0,  #10, #13
        BYT  #09, #15, #11, #02, #0,  #03, #13, #0
        BYT  #12, #02, #14, #05, #03, #06, #0,  #03
        BYT  #15, #05, #01, #08, #06, #09, #03, #06
        BYT  #02, #08, #04, #11, #09, #12, #06, #09
        BYT  #05, #11, #07, #14, #12, #15, #09, #12
        BYT  #08, #14, #10, #01, #15, #02, #12, #15
        BYT  #11, #01, #13, #04, #02, #05, #15, #02
        BYT  #14, #04, #0,  #07, #05, #08, #02, #05
```

```
        BYT  #01, #07, #03, #10, #08, #11, #05, #08
        BYT  #04, #10, #06, #13, #11, #14, #08, #11
        BYT  #07, #13, #09, #0,  #14,  #01, #11, #14
        BYT  #10, #0,  #12, #03, #01, #04, #14, #01
        BYT  #13, #03, #15, #06, #04, #07, #01, #04
;
;
SR2 :   BYT  #08, #01, #12, #02, #0, #0, #10, #03
        BYT  #11, #04, #15, #05, #0, #0, #13, #06
        BYT  #14, #07, #02, #08, #0, #0, #0,  #09
        BYT  #01, #10, #05, #11, #0, #0, #03, #12
        BYT  #04, #13, #08, #14, #0, #0, #06, #15
        BYT  #07, #0,  #11, #01, #0, #0, #09, #02
        BYT  #10, #03, #14, #04, #0, #0, #12, #05
        BYT  #13, #06, #01, #07, #0, #0, #15, #08
        BYT  #0,  #09, #04, #10, #0, #0, #02, #11
        BYT  #03, #12, #07, #13, #0, #0, #05, #14
        BYT  #06, #15, #10, #0,  #0, #0, #08, #01
        BYT  #09, #02, #13, #03, #0, #0, #11, #04
        BYT  #12, #05, #0,  #06, #0, #0, #14, #07
        BYT  #15, #08, #03, #09, #0, #0, #01, #10
        BYT  #02, #11, #06, #12, #0, #0, #04, #13
        BYT  #05, #14, #09, #15, #0, #0, #07, #0
;
WR1  :  BYT  #0,  #05, #02, #06, #0, #02, #04, #04
        BYT  #03, #08, #05, #09, #0, #05, #07, #07
```

```
        BYT  #06, #11, #08, #12, #0, #08, #10, #10
        BYT  #09, #14, #11, #15, #0, #11, #13, #13
        BYT  #12, #01, #14, #02, #0, #14, #0, #0
        BYT  #15, #04, #01, #05, #0, #01, #03, #03
        BYT  #02, #07, #04, #08, #0, #04, #06, #06
        BYT  #05, #10, #07, #11, #0, #07, #09, #09
        BYT  #08, #13, #10, #14, #0, #10, #12, #12
        BYT  #11, #0,  #13, #01, #0, #13, #15, #15
        BYT  #14, #03, #0,  #04, #0, #0,  #02, #02
        BYT  #01, #06, #03, #07, #0, #03, #05, #05
        BYT  #04, #09, #06, #10, #0, #06, #08, #08
        BYT  #07, #12, #09, #13, #0, #09, #11, #11
        BYT  #10, #15, #12, #0,  #0, #12, #14, #14
        BYT  #13, #02, #15, #03, #0, #15, #01, #01
;
WR2  :  BYT  #05, #01, #08, #02, #01, #0, #01, #03
        BYT  #08, #04, #11, #05, #04, #0, #04, #06
        BYT  #11, #07, #14, #08, #07, #0, #07, #09
        BYT  #14, #10, #01, #11, #10, #0, #10, #12
        BYT  #01, #13, #04, #14, #13, #0, #13, #15
        BYT  #04, #0,  #07, #01, #0,  #0, #0,  #02
        BYT  #07, #03, #10, #04, #03, #0, #03, #05
        BYT  #10, #06, #13, #07, #06, #0, #06, #08
        BYT  #13, #09, #0,  #10, #09, #0, #09, #11
        BYT  #0,  #12, #3,  #13, #12, #0, #12, #14
        BYT  #03, #15, #06, #0,  #15, #0, #15, #01
```

```
        BYT  #06, #02, #09, #03, #02, #0, #02, #04
        BYT  #09, #05, #12, #06, #05, #0, #05, #07
        BYT  #12, #08, #15, #09, #08, #0, #08, #10
        BYT  #15, #11, #02, #12, #11, #0, #11, #13
        BYT  #02, #14, #05, #15, #14, #0, #14, #00
;
SER1 :  BYT  #0,  #09, #02, #12, #0, #05, #04, #05
        BYT  #03, #12, #05, #15, #0, #08, #07, #08
        BYT  #06, #15, #08, #02, #0, #11, #10, #11
        BYT  #09, #02, #11, #05, #0, #14, #13, #14
        BYT  #12, #05, #14, #08, #0, #01, #0, #01
        BYT  #15, #08, #01, #11, #0, #04, #03, #04
        BYT  #02, #11, #04, #14, #0, #07, #06, #07
        BYT  #05, #14, #07, #01, #0, #10, #09, #10
        BYT  #08, #01, #10, #04, #0, #13, #12, #13
        BYT  #11, #04, #13, #07, #0, #0, #15, #0
        BYT  #14, #07, #0, #10, #0, #03, #02, #03
        BYT  #01, #10, #03, #13, #0, #06, #05, #06
        BYT  #04, #13, #06, #0, #0, #09, #08, #09
        BYT  #07, #0,  #09, #03, #0, #12, #11, #12
        BYT  #10, #03, #12, #06, #0, #15, #14, #15
        BYT  #13, #06, #15, #09, #0, #02, #01, #02
;
SER2 :  BYT  #08, #01, #11, #02, #04, #0, #04, #03
        BYT  #11, #04, #14, #05, #07, #0, #07, #06
        BYT  #14, #07, #01, #08, #10, #0, #10, #09
```

```
         BYT  #01, #10, #04, #11, #13, #0, #13, #12
         BYT  #04, #13, #07, #14, #0, #0, #0, #15
         BYT  #07, #0, #10, #01, #03, #0, #03, #02
         BYT  #10, #03, #13, #04, #06, #0, #06, #05
         BYT  #13, #06, #0, #07, #09, #0, #09, #08
         BYT  #0, #09, #03, #10, #12, #0, #12, #11
         BYT  #03, #12, #06, #13, #15, #0, #15, #14
         BYT  #06, #15, #09, #0, #02, #0, #02, #01
         BYT  #09, #02, #12, #03, #05, #0, #05, #04
         BYT  #12, #05, #15, #06, #08, #0, #08, #07
         BYT  #15, #08, #02, #09, #11, #0, #11, #10
         BYT  #02, #11, #05, #12, #14, #0, #14, #13
         BYT  #05, #14, #08, #15, #01, #0, #01, #0
;
SWR1   : BYT  #0, #06, #02, #09, #0, #14, #04, #14
         BYT  #03, #09, #05, #12, #0, #01, #07, #01
         BYT  #06, #12, #08, #15, #0, #04, #10, #04
         BYT  #09, #15, #11, #02, #0, #07, #13, #07
         BYT  #12, #02, #14, #05, #0, #10, #0, #10
         BYT  #15, #05, #01, #08, #0, #13, #03, #13
         BYT  #02, #08, #04, #11, #0, #0, #06, #0
         BYT  #05, #11, #07, #14, #0, #03, #09, #03
         BYT  #08, #14, #10, #01, #0, #06, #12, #06
         BYT  #11, #01, #13, #04, #0, #09, #15, #09
         BYT  #14, #04, #0, #07, #0, #12, #02, #12
         BYT  #01, #07, #03, #10, #0, #15, #05, #15
```

```
        BYT  #04, #10, #06, #13, #0, #02, #08, #02
        BYT  #07, #13, #09, #0, #0, #05, #11, #05
        BYT  #10, #0, #12, #03, #0, #08, #14, #08
        BYT  #13, #03, #15, #06, #0, #11, #01, #11
;
SWR2  : BYT  #05, #01, #08, #02, #13, #0, #13, #03
        BYT  #08, #04, #11, #05, #0, #0, #0, #06
        BYT  #11, #07, #14, #08, #03, #0, #03, #09
        BYT  #14, #10, #01, #11, #06, #0, #06, #12
        BYT  #01, #13, #04, #14, #09, #0, #09, #15
        BYT  #04, #0, #07, #01, #12, #0, #12, #02
        BYT  #07, #03, #10, #04, #15, #0, #15, #05
        BYT  #10, #06, #13, #07, #02, #0, #02, #08
        BYT  #13, #09, #0, #10, #05, #0, #05, #11
        BYT  #0, #12, #03, #13, #08, #0, #08, #14
        BYT  #03, #15, #06, #0, #11, #0, #11, #01
        BYT  #06, #02, #09, #03, #14, #0, #14, #04
        BYT  #09, #05, #12, #06, #01, #0, #01, #07
        BYT  #12, #08, #15, #09, #04, #0, #04, #10
        BYT  #15, #11, #02, #12, #07, #0, #07, #13
        BYT  #02, #14, #05, #15, #10, #0, #10, #0
;
;
SPSE1 : BYT  #0,  #0,  #0,  #0,  #0,  #0,  #0,  #0
```

```
        BYT  #0,  #0,  #0,  #0,  #0,  #0,  #0,  #0
;
SPSE2 : BYT  #0,  #0,  #0,  #0,  #0,  #0,  #0,  #0
        BYT  #0,  #0,  #0,  #0,  #0,  #0,  #0,  #0
;
SPS1  : BYT  #0,  #0,  #0,  #0,  #0,  #0,  #0,  #0
        BYT  #0,  #0,  #0,  #0,  #0,  #0,  #0,  #0
;
SPS2  : BYT  #0,  #0,  #0,  #0,  #0,  #0,  #0,  #0
        BYT  #0,  #0,  #0,  #0,  #0,  #0,  #0,  #0
;
SPSW1 : BYT  #0,  #0,  #0,  #0,  #0,  #0,  #0,  #0
        BYT  #0,  #0,  #0,  #0,  #0,  #0,  #0,  #0
;
SPSW2 : BYT  #0,  #0,  #0,  #0,  #0,  #0,  #0,  #0
        BYT  #0,  #0,  #0,  #0,  #0,  #0,  #0,  #0
;
SPW1  : BYT  #0,  #0,  #0,  #0,  #0,  #0,  #0,  #0
        BYT  #0,  #0,  #0,  #0,  #0,  #0,  #0,  #0
;
SPW2  : BYT  #0,  #0,  #0,  #0,  #0,  #0,  #0,  #0
        BYT  #0,  #0,  #0,  #0,  #0,  #0,  #0,  #0
;
;
CNTR0   :  BYT  #00, #01, #16, #17, #01, #02, #17, #18
           BYT  #16, #17, #32, #33, #17, #18, #33, #34
```

```
BYT  #01, #02, #17, #18, #02, #03, #18, #19
BYT  #17, #18, #33, #34, #18, #19, #34, #35
BYT  #16, #17, #32, #33, #17, #18, #33, #34
BYT  #32, #33, #48, #49, #33, #34, #49, #50
BYT  #17, #18, #33, #34, #18, #19, #34, #35
BYT  #33, #34, #49, #50, #34, #35, #50, #51
BYT  #01, #02, #17, #18, #02, #03, #18, #19
BYT  #17, #18, #33, #34, #18, #19, #34, #35
BYT  #02, #03, #18, #19, #03, #04, #19, #20
BYT  #18, #19, #34, #35, #19, #20, #35, #36
BYT  #17, #18, #33, #34, #18, #19, #34, #35
BYT  #33, #34, #49, #50, #34, #35, #50, #51
BYT  #18, #19, #34, #35, #19, #20, #35, #36
BYT  #34, #35, #50, #51, #35, #36, #51, #52
BYT  #16, #17, #32, #33, #17, #18, #33, #34
BYT  #32, #33, #48, #49, #33, #34, #49, #50
BYT  #17, #18, #33, #34, #18, #19, #34, #35
BYT  #33, #34, #49, #50, #34, #35, #50, #51
BYT  #32, #33, #48, #49, #33, #34, #49, #50
BYT  #48, #49, #64, #65, #49, #50, #65, #66
BYT  #33, #34, #49, #50, #34, #35, #50, #51
BYT  #49, #50, #65, #66, #50, #51, #66, #67
BYT  #17, #18, #33, #34, #18, #19, #34, #35
BYT  #33, #34, #49, #50, #34, #35, #50, #51
BYT  #18, #19, #34, #35, #19, #20, #35, #36
BYT  #34, #35, #50, #51, #35, #36, #51, #52
```

```
         BYT  #33, #34, #49, #50, #34, #35, #50, #51
         BYT  #49, #50, #65, #66, #50, #51, #66, #67
         BYT  #34, #35, #50, #51, #35, #36, #51, #52
         BYT  #50, #51, #66, #67, #51, #52, #67, #68
;
CNTR1  : BYT  #00, #16, #01, #17, #16, #32, #17, #33
         BYT  #01, #17, #02, #18, #17, #33, #18, #34
         BYT  #16, #32, #17, #33, #32, #48, #33, #49
         BYT  #17, #33, #18, #34, #33, #49, #34, #50
         BYT  #01, #17, #02, #18, #17, #33, #18, #34
         BYT  #02, #18, #03, #19, #18, #34, #19, #35
         BYT  #17, #33, #18, #34, #33, #49, #34, #50
         BYT  #18, #34, #19, #35, #34, #50, #35, #51
         BYT  #16, #32, #17, #33, #32, #48, #33, #49
         BYT  #17, #33, #18, #34, #33, #49, #34, #50
         BYT  #32, #48, #33, #49, #48, #64, #49, #65
         BYT  #33, #49, #34, #50, #49, #65, #50, #66
         BYT  #17, #33, #18, #34, #33, #49, #34, #50
         BYT  #18, #34, #19, #35, #34, #50, #35, #51
         BYT  #33, #49, #34, #50, #49, #65, #50, #66
         BYT  #34, #50, #35, #51, #50, #66, #51, #67
         BYT  #01, #17, #02, #18, #17, #33, #18, #34
         BYT  #02, #18, #03, #19, #18, #34, #19, #35
         BYT  #17, #33, #18, #34, #33, #49, #34, #50
         BYT  #18, #34, #19, #35, #34, #50, #35, #51
         BYT  #02, #18, #03, #19, #18, #34, #19, #35
```

```
        BYT  #03, #19, #04, #20, #19, #35, #20, #36
        BYT  #18, #34, #19, #35, #34, #50, #35, #51
        BYT  #19, #35, #20, #36, #35, #51, #36, #52
        BYT  #17, #33, #18, #34, #33, #49, #34, #50
        BYT  #18, #34, #19, #35, #34, #50, #35, #51
        BYT  #33, #49, #34, #50, #49, #65, #50, #66
        BYT  #34, #50, #35, #51, #50, #66, #51, #67
        BYT  #18, #34, #19, #35, #34, #50, #35, #51
        BYT  #19, #35, #20, #36, #35, #51, #36, #52
        BYT  #34, #50, #35, #51, #50, #66, #51, #67
        BYT  #35, #51, #36, #52, #51, #67, #52, #68
;
END
```

APPENDIX C. EDF1.SOR

[Source Code for the model of Eval_Diffs]

```
;--EDF        SIMULAION OF THE MODULE EVAL_DIFFS
;
;
;PURPOSE
;
; THIS IS A SOURCE CODE FOR THE
; MODEL OF THE WHOLE EVAL_DIFFS
; MODULE. IT HAS A PREPROCESSOR,
; A LATCH AND A PAL.
;
;ENTRY POINT
;
; GSPASM EDF1  AND  THEN  GSPSIM EDF1.
;
;DATA FORMAT
;
; INPUT  : 2 BYTES FROM SUM_DIFFS,
;          B_SUM AND W_SUM, IN DECIMAL
;          FORM THROUGH A COMMAND
;          FILE EDF1.GCM.
; OUTPUT : BIN_ROW AND WHITE_BIT,
;          ONE BIT EACH. IT IS IN BINARY
```

```
;           FORM ON PIN #49 AND PIN #50.
;
;REMARKS
;
; IF ANY OTHER DATA IS TO BE OBSERVED,
; THREE GROUPS OF PINS ARE
; KEPT FOR THAT USE, OUT1, OUT2 AND OUT3.
;
****************************************************************
;
;
REG(8)  A1,A2,A3,B1,B2,B3,B2C,B3C,MSPR,MSPRC,LSPR
REG(8)  LSPRC,TOR,ROSIZ,PIX
REG(3)  PER
REG(1)  QG,QF,QE,QD,QC,QB,QA,BEG,T7,T7C
REG(1)  QGC,QFC,QEC,QDC,QCC,QBC,QAC
REG(1)  INT1,INT2,INT3,INT4,INT5,DUMRG,BR,WB
;
PIN     MSP(1,8),LSP(9,16),ROWSZ(17,24),T1(25,32)
PIN     T2(33,40),T3(41,48),BRO(49),WBO(50)
PIN     OUT1(51,58),OUT2(59),OUT3(60)
PIN     DUMMY(151)
;
EVW     W2(2),W32(32)
;
```

```
        BNE          DUMMY,DUMRG,BEGIN   ; BEGIN WHEN CLOCK CHANGES
START: EXR
;
BEGIN: MOV           DUMMY,DUMRG
       BEQ          DUMRG,START; GO BACK IF CLOCK LO GOING
       MOV          #0,DUMMY
       MOV          #0,DUMRG
       MOV(W32)     #1,DUMMY    ; SCHEDULE THE NEXT CLOCK CHANGE
       MOV          T1,A1       ; INITIALIZE THE PREPROCESSOR BY
       MOV          T2,B1       ; STORING THE THRESHOLDS    IN
       MOV          T3,A2       ; PROPER REGISTERS
       MOV          ROWSZ,ROSIZ; STORE THE ROW-SIZE IN A REG.
       MOV          #1,BEG
;
NOTIN: IDX           PER(0),3,1
       IDX          PIX(0),8,2 ; TO OBSERVE THE PIXEL NUMBER
       IDX          PIX(0),8,4 ; TO OBSRVE THE ROW NUMBER
       BRU          TBL@1
TBL: BYT  100,101,102,103,104,105,106,107; THE CLOCK CYCLES
;
100 : MOV           MSP,MSPR    ; B_SUM FROM SUM_DIFFS
      MOV           LSP,LSPR    ; W_SUM FROM SUM_DIFFS
      MOV           MSPR,A3     ; STORE THESE VALUES
      MOV           LSPR,B2
      COM           MSPR,MSPRC
      MOV           #0,C
```

```
        ADD         B1,MSPRC,TOR   ; B1 - MSP = DUMMY REG.
        MOV         C,QG         ; STORE THE CARRY
        COM         QG,QGC
        MOV         #1,PER
        BRU         START
;
101 :   ADD         A1,MSPRC,TOR   ; A1 - MSP = DUMMY REG.
        MOV         C,QF         ; STORE THE CARRY
        COM         QF,QFC
        MOV         #2,PER
        BRU         START
;
102 :   COM         B2,B2C
        MOV         #0,C         ; CLEAR CARRY FOR PRECAUTION
        ADD         A3,B2C,B3    ; A3 - B2 = B3
        MOV         C,QE         ; STORE THE CARRY
        COM         QE,QEC
        MOV         #3,PER
        BRU         START
;
103 :   COM         B3,B3C
        MOV         #0,C
        ADD         A2,B3C,TOR ; A2 - B3 = DUMMY REG.
        MOV         C,QD         ; STORE BTHE CARRY
        COM         QD,QDC
        MOV         #4,PER
```

```
        BRU          START
;
104 :   ADD          B1,B2C,TOR ; B1 - B2 = DUMMY REG.
        MOV          C,QC       ; STORE THE CARRY
        COM          QC,QCC
        MOV          #5,PER
        BRU          START
;
105 :   ADD          A1,B2C,TOR ; A1 - B2 = DUMMY REG.
        MOV          C,QB       ; STORE THE CARRY
        COM          QB,QBC
        MOV          #6,PER
        BRU          START
;
106 :   ADD          A2,B3,TOR  ; A2 + B3 = DUMMY REG.
        MOV          C,QA       ; STORE THE CARRY
        COM          QA,QAC
        MOV          #7,PER
        BRU          START
;
107 :   MOV          #1,T7      ; A SIGNAL FROM FIND_DIFFS
        COM          T7,T7C
;THE PAL FUNCTION IMPLEMENTATION:
        AND      QGC,T7,INT1    ; -
        AND      QFC,QDC,INT2   ; |
        AND      INT2,T7,INT3   ; |> BR = (QG')(T7) +
```

```
        AND     BR,T7C,INT4      ; |      (QF')(QD')(T7)+
        OR      INT1,INT3,INT5   ; |      (BR)(T7')
        OR      INT4,INT5,BR     ; /
;
        AND     QCC,T7,INT1      ; -
        AND     QBC,QAC,INT2     ; |
        AND     INT2,T7,INT3     ; |> WB = (QC')(T7) +
        AND     WB,T7C,INT4      ; |     (QB')(QA')(T7) +
        OR      INT1,INT3,INT5   ; |     (WB)(T7')
        OR      INT4,INT5,WB     ; /
;
        MOV         BR,BRO
        MOV         WB,WBO
        MOV(W2)     #0,BRO
        MOV(W2)     #0,WBO
        MOV         #0,PER
        INC         PIX,PIX     ; ONE PIXEL OVER
        BNE         ROSIZ,PIX,START; IS THE ROW OVER?
        MOV         #0,PIX      ; IF YES START NEW ROW
        BRU         START
;
END
```

APPENDIX D. FILT1.SOR

[Source Code for the model of Filter Module.]

```
;--FILTER SIMULAION OF THE THREE FILTERS AS ONE MODEL.
;
;
;PURPOSE
;
;THIS IS A SOURCE CODE FOR THE
;MODULE FILTERS. THERE ARE THREE
;FILTERS IN THIS MODULE. THE INPUT
;DATA IS MANIPULATED BY A
;SHIFT REGISTER AND GIVEN TO THE
;PAL WITH THE INTERMEDIATE
;VALUES STORED IN ITS FLIP-FLOPS.
;
;ENTRY POINT
;
;GSPASM  FILT1  AND  THEN  GSPSIM FILT1
;
;DATA FORMAT
;
;INPUT  : INPUT TO THE FIRST FILTER
;         IS BIN-ROW FROM EVAL_DIFFS.
;         THAT TO THE SECOND FILTER
```

```
;            IS WHITE_BITS FROM
;            EVAL_DIFFS AND THE THIRD
;            FILTER GETS ITS INPUT,
;            (BIN_ROW)' FROM SET_REL_T.
;            ALL THESE INPUTS ARE IN
;            DECIMAL FORM GIVEN THROUGH
;            THE COMMAND FILE.
;OUTPUT :    THE OUTPUTS FROM THE FIRST
;            TWO FILTERS, BIN_IN AND
;            WHITE_IN, ARE GIVEN TO
;            SET_REL_T. AND THE THIRD
;            FILTER GIVES THE OUTPUT
;            OF THE WHOLE IPB. THESE
;            OUTPUTS CAN BE OBSERVED
;            IN BINARY FORM ON PIN #'S
;            5, 6 AND 7, RESPECTIVELY.
;
;
;****************************************************
;
REG(8)      FSR,VEC1,MASK1,ROSIZ,MACT,PIX
REG(3)      PER
REG(1)      BRR,WBR,BRN,BRN1,BRN2,WBN,WBN1,WBN2
REG(1)      BWF,BWFC,BRNC,BRN1C,BRN2C,WBNC,WBN1C,WBN2C
REG(1)      BOF,BOFC,BO1,BOO,WI,WO,W1,BI,BO,B1
REG(1)      WODUM,W1DUM,BODUM,B1DUM
```

```
REG(1)  BICN,BICN1,BICN2,BICNC,BICN1C,BICN2C,BICR
REG(1)  INT1,INT2,INT3,INT4,INT5,INT6,INT7,INT8,INT9
REG(1)  INT10,INT11,INT12,INT13,INT14,INT15,INT16,INT17
REG(1)  INT18,INT19,INT20,INT21,INT22,INT23,INT24,INT25
REG(1)  DUMRG
;
PIN     BR(1),WB(2),ROWSZ(3,10),BIC(11)
;
PIN     BIN(12),WIN(13),BOUT(14),OUT1(15),OUT2(16)
PIN     OUT3(16,23)
PIN     DUMMY(151)
;
EVW     W32(32),W2(2)
;
        BNE        DUMMY,DUMRG,BEGIN; BEGIN ON CLOCK CHANGE
START:  EXR
;
BEGIN:  MOV        DUMMY,DUMRG
        BEQ        DUMRG,START    ; RESATRT IF CLOCK LO GOING
        MOV        #0,DUMMY
        MOV        #0,DUMRG
        MOV(W32)   #1,DUMMY       ; SCHEDULE THE NEXT CLOCK CHANGE
        MOV        ROWSZ,ROSIZ    ; STORE THE ROWSIZE
        MOV        #219,MASK1     ; TO MASK OFF UNWANTED BITS
        IDX        PIX(0),8,8     ; TO KEEP TRACK OF PIXEL NUMBER
        IDX        MACT(0),8,2
```

```
        IDX          PER(0),3,1
        BRU          TBL@1
;
TBL: BYT  100,101,102,103,104,105,106,107 ; CLOCK CYCLES
;
100 :   MOV          FSR,SRAM@2      ; FSR IS THE FILT. SHIFT REG.
        MOV          BI,BOUT         ; OUTPUT OF THE WHOLE SYSTEM
        MOV(W2)      #0,BOUT
101 :   INC          PER,PER
        BRU          START
;
102 :   INC          MACT,MACT
        INC          PER,PER
        BNE          ROSIZ,MACT,START
        MOV          #0,MACT
        BRU          START
;
103 :   INC          PER,PER
        BRU          START
;
104 :   INC          PER,PER
        MOV          BIC,BICR         ; ACCEPT INPUT FROM SET_REL_T
        BRU          START
;
105 : MOV          SRAM@2,VEC1      ; GET OLD DATA IN DUMMY REG.
      AND          MASK1,VEC1,VEC1    ; MASK OFF UNWANTED BITS
```

```
        MOV     BR,BRR          ; STORE INPUT FROM EVAL_DIFFS
        MOV     WB,WBR
        BEQ     BRR,NOBR        ; BR TAKES 6TH BIT OF FSR
        BIS     #5,VEC1
        BRU     YESBR
NOBR:   BIR     #5,VEC1
YESBR:   NOP
        BEQ     WBR,NOWB        ; WB TAKES 3RD BIT OF FSR
        BIS     #2,VEC1
        BRU     YESWB
NOWB:   BIR     #2,VEC1
YESWB:   NOP
        MOV     VEC1,FSR
        INC     PER,PER
        BRU     START
;
106 :   IDX     FSR(7),1,3      ; GET THE INDIVIDUAL BITS
        MOV     @3,BICN1
        IDX     FSR(6),1,4
        MOV     @4,BICN2
        IDX     FSR(5),1,5
        MOV     @5,BRN
        IDX     FSR(4),1,6
        MOV     @6,BRN1
        IDX     FSR(3),1,3
        MOV     @3,BRN2
```

```
        IDX         FSR(2),1,3
        MOV         @3,WBN
        IDX         FSR(1),1,3
        MOV         @3,WBN1
        IDX         FSR(0),1,3
        MOV         @3,WBN2
;
;
        MOV         #0,BWF    ; A SIGNAL FROM FIND_DIFFS
        COM         BWF,BWFC
        MOV         #1,BOF    ; ANOTHER SIGNAL FROM FIND_DIFFS
        COM         BOF,BOFC
        COM         BRN,BRNC
        COM         BRN1,BRN1C
        COM         BRN2,BRN2C
        COM         WBN,WBNC
        COM         WBN1,WBN1C
        COM         WBN2,WBN2C
;
; THE INTERMEDIATE VALUES STORED IN FLIP-FLOPS OF
; THE PAL CHIP ARE IMPLEMENTED.
; THOSE INTERMEDIATE FUNCTIONS ARE:
;
; BO = [BR(N-1)][BR(N)'][BR(N-2)'][BWF'] +
; [BO][BR(N-1)][BWF'] + [B1][BR(N-1)][BWF'] +
; [BO][BWF]
```

```
; B1 = [BR(N-1)][BR(N-2)][BWF'] + [BR(N-1)][BR(N)][BWF'] +
; [B1][BWF]
; WO = [WB(N-1)][WB(N)'][WB(N-2)'][BWF'] +
; [WO][WB(N-1)][BWF'] + [W1][WB(N-1)][BWF'] +
; [WO][BWF]
; W1 = [WB(N-1)][WB(N-2)][BWF'] + [WB(N-1)][WB(N)][BWF'] +
; [W1][BWF]
;
        AND         BRN1,BWFC,INT1
        AND         INT1,BRN2,INT2
        AND         INT1,BRN,INT3
        AND         B1,BWF,INT4
        OR          INT2,INT3,INT5
        OR          INT4,INT5,B1
        MOV(W32)    B1,B1DUM

;
        AND         BRNC,BRN2C,INT6
        AND         INT6,INT1,INT7
        AND         INT1,BO,INT8
        AND         BRN1,BWFC,INT9
        AND         B1,INT9,INT10
        AND         BO,BWF,INT11
        OR          INT7,INT8,INT12
        OR          INT11,INT10,INT13
        OR          INT12,INT13,BO
```

```
        MOV(W32)   BO,BODUM
;
        AND        WBN1,BWFC,INT1
        AND        INT1,WBN2,INT2
        AND        INT1,WBN,INT3
        AND        W1,BWF,INT4
        OR         INT2,INT3,INT5
        OR         INT4,INT5,W1
        MOV(W32)   W1,W1DUM
;
;
        AND        WBNC,WBN2C,INT6
        AND        INT6,INT1,INT7
        AND        INT1,WO,INT8
        AND        WBN1,BWFC,INT9
        AND        W1,INT9,INT10
        AND        WO,BWF,INT11
        OR         INT7,INT8,INT12
        OR         INT11,INT10,INT13
        OR         INT12,INT13,WO
        MOV(W32)   WO,WODUM
;
        JSR        PALBI  ; THE PAL FUNCTION FOR BI AND WI
;
        MOV       BI,BIN  ; OUTPUT BIN TO SET_REL_T
        MOV       WI,WIN  ; OUTPUT WIN TO SET_REL_T
```

```
          MOV(W2)    #0,BIN
          MOV(W2)    #0,WIN
;
          INC        PER,PER
          BRU        START
;
107 :     MOV        #0,PER
          SHR        FSR      ; SHIFT FSR FOR GETTING NEW BIC
          BEQ        BICR,NOBIC; NEW BIC TAKES 7TH BIT OF FSR
          BIS        #7,FSR
          BRU        BOFLT
NOBIC:    BIR        #7,FSR
BOFLT:    NOP
          IDX         FSR(7),1,3 ; NOW GET THE INDIVIDUAL BITS
          MOV         @3,BICN
          IDX         FSR(6),1,4
          MOV         @4,BICN1
          IDX         FSR(5),1,5
          MOV         @5,BICN2
          IDX         FSR(4),1,6
          MOV         @6,BRN
          IDX         FSR(3),1,3
          MOV         @3,BRN1
          IDX         FSR(2),1,3
          MOV         @3,BRN2
          IDX         FSR(1),1,3
```

```
        MOV     @3,WBN
        IDX     FSR(0),1,3
        MOV     @3,WBN1
        MOV     #1,BWF      ; A SIGNAL FROM FIND_DIFFS
        COM     BWF,BWFC
        MOV     #0,BOF      ; ANOTHER SIGNAL
        COM     BOF,BOFC
;
        COM     BICN,BICNC
        COM     BICN1,BICN1C
        COM     BICN2,BICN2C
        COM     BRN,BRNC
        COM     BRN1,BRN1C
        COM     BRN2,BRN2C
;
; THERE ARE TWO MORE INTERMEDIATE FUNCTIONS.
; THOSE ARE BOO AND BO1.
; BOO = [BIC(N-1)][BIC(N)'][BIC(N-2)'][BOF'] +
;[BOO][BIC(N-1)][BOF'] + [BO1][BIC(N-1)][BOF'] +
;[BOO][BOF]
; BO1 = [BIC(N-1)][BIC(N-2)][BOF'] +
;[BIC(N-1)][BIC(N)][BOF'] + [BO1][BOF]
;
        AND     BICN1,BICNC,INT1
        AND     BICN2C,BOFC,INT2
        AND     INT2,INT1,INT3
```

```
        AND     BICN1,BOFC,INT4
        AND     BOO,INT4,INT5
        AND     BO1,INT4,INT6
        AND     BOO,BOF,INT7
        OR      INT3,INT5,INT8
        OR      INT6,INT7,INT9
        OR      INT8,INT9,BOO
;
        AND     BICN1,BOFC,INT1
        AND     BICN2,INT1,INT2
        AND     BICN,INT1,INT3
        AND     BO1,BOF,INT4
        OR      INT2,INT3,INT5
        OR      INT4,INT5,BO1
;
        JSR     PALBI
;
        INC     PIX,PIX   ; ONE PIXEL OVER
        BNE     ROSIZ,PIX,START ; IS ROW OVER?
        MOV     #0,PIX     ; IF ROW OVER START NEW ROW
        BRU     START
;
;THE PAL FUNCION IS IMPLEMENTED IN THIS SUBROUTINE.
;
;THE FUNCTION IS:
;          BI = [B1][BR(N-1)][BWF'] +
```

```
;                   [B1][BO][BWF'] + [BO1][BIC(N-1)][BOF'] +
;                   [BO1][BOO][BOF']
;AND
;             WI = [W1][WB(N-1)][BWF'] + [W1][WO][BWF']
;
;
;
PALBI:   NOP
;
         AND          B1,INT1,INT14
         AND          B1,BO,INT15
         AND          INT5,BWFC,INT16
         AND          BO1,BICN1,INT17
         AND          INT17,BOFC,INT18
         AND          BO1,BOO,INT19
         AND          INT19,BOFC,INT20
         OR           INT14,INT16,INT21
         OR           INT18,INT20,INT22
         OR           INT21,INT22,BI
;
         AND          W1,WBN1,INT14
         AND          BWFC,INT14,INT15
         AND          W1,WO,INT16
         AND          INT16,BWFC,INT17
         OR           INT15,INT17,WI
;
```

```
        RTS
;
; THIS IS THE SRAM OF FILTERS.
; IT HAS 256 BYTES, ONLY FOR CONVENIENCE.
SRAM :  BYT     #0,#0,#0,#0,#0,#0,#0,#0
        BYT     #0,#0,#0,#0,#0,#0,#0,#0
        BYT     #0,#0,#0,#0,#0,#0,#0,#0
        BYT     #0,#0,#0,#0,#0,#0,#0,#0
        BYT     #0,#0,#0,#0,#0,#0,#0,#0
        BYT     #0,#0,#0,#0,#0,#0,#0,#0
        BYT     #0,#0,#0,#0,#0,#0,#0,#0
        BYT     #0,#0,#0,#0,#0,#0,#0,#0
        BYT     #0,#0,#0,#0,#0,#0,#0,#0
        BYT     #0,#0,#0,#0,#0,#0,#0,#0
        BYT     #0,#0,#0,#0,#0,#0,#0,#0
        BYT     #0,#0,#0,#0,#0,#0,#0,#0
        BYT     #0,#0,#0,#0,#0,#0,#0,#0
        BYT     #0,#0,#0,#0,#0,#0,#0,#0
        BYT     #0,#0,#0,#0,#0,#0,#0,#0
        BYT     #0,#0,#0,#0,#0,#0,#0,#0
        BYT     #0,#0,#0,#0,#0,#0,#0,#0
        BYT     #0,#0,#0,#0,#0,#0,#0,#0
        BYT     #0,#0,#0,#0,#0,#0,#0,#0
        BYT     #0,#0,#0,#0,#0,#0,#0,#0
        BYT     #0,#0,#0,#0,#0,#0,#0,#0
        BYT     #0,#0,#0,#0,#0,#0,#0,#0
```

```
        BYT     #0,#0,#0,#0,#0,#0,#0,#0
        BYT     #0,#0,#0,#0,#0,#0,#0,#0
        BYT     #0,#0,#0,#0,#0,#0,#0,#0
        BYT     #0,#0,#0,#0,#0,#0,#0,#0
        BYT     #0,#0,#0,#0,#0,#0,#0,#0
        BYT     #0,#0,#0,#0,#0,#0,#0,#0
        BYT     #0,#0,#0,#0,#0,#0,#0,#0
        BYT     #0,#0,#0,#0,#0,#0,#0,#0
        BYT     #0,#0,#0,#0,#0,#0,#0,#0
        BYT     #0,#0,#0,#0,#0,#0,#0,#0
;
END
```

APPENDIX E. SET1.SOR

[Source Code for the model of Set_Rel_T]

```
;--SET    SIMULATION OF THE WHOLE SET_REL_T MODULE
;
;
;PURPOSE
;
; THIS IS A SOURCE CODE FOR THE MODEL OF THE WHOLE
; SET_REL_T MODULE. IT HAS A PREPROCESSOR,
; TWO MEMORY BANKS, FOUR LATCHES, A PAL AND SO ON.
; THE IMPORTANT ASPECT OF IT IS SELECTING A SET
; OF MICROOPERATIONS ACCORDING TO THE BIN AND WIN
; FUNCTIONS. THIS COMPARISON IS DONE IN
; EVERY CYCLE TO AVOID REPITITION OF
; THE CODE AND TO SPEED UP THE
; SIMULATION. THE VALUES OF BIN AND
; WIN ARE COMPUTED FOR THE 8 CLOCK CYCLES.
;
;ENTRY POINT
;
; GSPASM SET1  AND  THEN  GSPSIM SET1.
;
;DATA FORMAT
; INPUT:
```

```
;  THERE ARE THREE TYPES OF INPUT DATA:
;  GRAY_VAL = GRAY VALUE OF CORRESPONDING PIXEL FROM
;            FIND_DIFFS.
;  BIN_IN  AND WHITE_IN FROM FILTERS.
;  TBF = THRESHOLD VALUE, GIVEN DURING INTIALIZATION.
;  INIT= #127, GIVEN DURING ROW INTIALIZATION TO RC.
;OUTPUT :
;  THE ONLY BIT GIVEN BY THIS MODULE APPEARS AS BIC ON
;  PIN # 27 IN DECIMAL FORM.
;
;*****************************************************
;
;
;
REG(8)      A1,A2,A3,B1,B2,B3,MIOR,MIR,DIOR,RA,RB,RC
REG(8)      A2B2,PIX
REG(8)      GVIL,KOR,K12R,DUM1,DMA,GVDMA,ROSIZ
REG(3)      UPAC,PER
REG(1)      BININ,WININ,WINR,BINR,BIC,CSR,QA,QB,QC
REG(1)      CINR1,CINR2,DUMRG,BEG,WINC,QBC,QAC,QCC
REG(1)      INT1,INT2,INT3,INT4,INT5,INT6,INT7,INT8
REG(1)      INT9,INT10,INT11,INT12,INT13,INT14,INT15
;
PIN         GRAY(1,8),TBF(9,16),ROWSZ(17,24)
PIN         BIN(25),WIN(26),BINC(27),BINRO(28)
PIN         WINRO(29),OUT1(30),OUT2(31),OUT3(32)
```

```
PIN         OUT4(33,40),OUT5(41,45),INIT(46,53)
;
PIN         DUMMY(151)
;
EVW         W192(192),W64(64),W32(32),W2(2)
;
;
        BNE     DUMMY,DUMRG,BEGIN   ; BEGIN ON CLOCK CHANGE
START:  EXR
;
BEGIN:  MOV     DUMMY,DUMRG   ; START AGAIN IF
        BEQ     DUMRG,START   ; CLOCK LO GOING
        MOV     #0,DUMMY
        MOV     #0,DUMRG      ; SCHEDULE THE NEXT
        MOV(W32) #1,DUMMY     ; CLOCK CHANGE
;
        BNE     BEG,NOTIN     ; INTIALIZE BEFORE STARTING
        MOV     TBF,A3        ; STORE THRESHOLD IN A3
        MOV     INIT,RC       ; STORE #127 IN RC
        MOV     ROWSZ,ROSIZ   ; STORE THE ROW-SIZE
        MOV     #1,BEG
NOTIN:  NOP
        IDX     PER(0),3,1
        IDX     DMA(0),8,2
        IDX     GVDMA(0),8,3
        IDX     UPAC(0),1,4   ; THE SET_REL_T MICROPROG.
```

```
        MOV     @4,QA           ; ADDRESS COUNTER OUTPUT IS
        COM     QA,QAC          ; USED IN THE PAL FUNCTION
        IDX     UPAC(1),1,4     ; TO RESTORE THE VALUES
        MOV     @4,QB
        COM     QB,QBC
        IDX     UPAC(2),1,4
        MOV     @4,QC
        COM     QC,QCC
        IDX     UPAC(0),3,5
;
        BRU     TBL@1     ; THE CLOCK CYLCES
TBL :   BYT       100,101,102,103,104,105,106,107
;
100: MOV    GRAY,GVIL   ; ACCEPT INPUT FROM FIND_DIFFS
     MOV    GVIL,GVFIF@3 ; AND STORE IN THE FIFO
     BEQ    #1,BINR,BI1TO ; ALL THE COMPARISONS WITH BIN
;                         ; AND WIN ARE DONE TO
;                       ; SELECT PROPER MICROOP. SET
     MOV    A2,MIOR     ; (A2) --> MIO
     MOV    MIOR,RA     ;MIO --> RA.. LATCH A.
     MOV    RA,SRAM@2   ;RA  --> SRAM
     BRU    FINTO
;
BI1TO: MOV   B1,MIOR    ; (B1)  --> MIO
       MOV   MIOR,RC    ;MIO  --> RC
       BIR   #7,RC     ; THE LAST BIT OF RC IS "0"
```

```
        MOV   MIOR,RB      ;MIO  --> RB
        BEQ   CSR,NOCO     ; LAST BIT OF RB IS CARRY
        BIS   #7,RB
        BRU   CO
NOCO :  BIR   #7,RB
CO :    MOV   RB,SRAM@2    ; RB  --> SRAM
FINTO:  INC   PER,PER
        INC   UPAC,UPAC    ; INCR. MICROPROG. ADD. CNTR.
        BRU   START
;
101 :   BEQ   BINR,NOBI    ; BRANCH ACCORDING TO BIN
        MOV   RC,DIOR      ; RC  --> DIO
        MOV   DIOR,A1      ; DIO --> A1
NOBI :  INC   PER,PER
        INC   UPAC,UPAC
        BRU   START
;
102 :   INC   DMA,DMA
        BNE   ROSIZ,DMA,GVINC
        MOV   0,DMA
        NOP
GVINC:  NOP
        INC     GVDMA,GVDMA
        BNE     ROSIZ,GVDMA,PERIN
        MOV     #0,GVDMA
        MOV     #127,RC
```

```
        NOP
PERIN:  NOP
        INC     PER,PER
        INC     UPAC,UPAC
        BRU     START
;
103 :   COM      WINR,WINC
        AND     CSR,WINC,INT1     ; \
        AND     INT1,QCC,INT2     ; |
        AND     BIC,QC,INT3       ; |> BIC = (CS)(WI')(QC')
        AND     BINR,QCC,INT4     ; |         + (BI)(QC').
        OR      INT2,INT3,INT5    ; |         + (BIC)(QC)
        OR      INT4,INT5,BIC     ; /
        MOV     BIC,BINC        ; OUTPUT OF THE MODULE
        MOV     BINR,BINRO      ; CHECK THE VALUE OF BINR
        MOV     WINR,WINRO      ;
;
104 :   INC     PER,PER
        INC     UPAC,UPAC
        BRU     START
;
105 :   MOV     SRAM@2,KOR      ; READ SRAM INTO A DUMMY REG.
        ROR     KOR             ; DIVIDE BY TWO
        MOV     C,CINR1         ; STROE CARRY FOR ADDING
        ADD     CINR1,KOR,MIR     ; SO, KO/2 + CIN --> MI
        ROR     A2              ; SIMILARLY FOR A1 I.E. [K1]
```

```
        MOV     C,CINR2
        ADD     CINR2,A2,K12R   ; [K1]/2 + CIN
        ADD     K12R,MIR,A2   ; MI + [K1]/2 + CIN --> A2
        MOV     GVFIF@3,DIOR ; READ GRAY_VAL FIFO INTO DIO
        MOV     DIOR,B1       ; STORE THAT GRAY_VAL IN B1
        INC     PER,PER
        INC     UPAC,UPAC
        BRU     START
;
106 :   SHR     A2,A2B2       ; DIVIDE A2 BY 2
        MOV     A2B2,MIOR     ; [A2]/2 --> MIO
        MOV     MIOR,RC       ; MIO  --> RC
        BIR     #7,RC         ; THE LAST BIT OF RC IS "0"
        ADD     B1,A3,B2      ; B1 + A3 --> B2
        MOV     C,CSR         ; STORE THE CARRY
        INC     PER,PER
        INC     UPAC,UPAC
        JSR     PAL           ; PAL FUNCTION IS A SUBROUTINE
        BRU     START
;
107 :   MOV     BIN,BININ     ; STORE BIN AND WIN GIVEN
        MOV     WIN,WININ     ; BY THE PAL FUNCTION IN
;                             ; PREVIOUS PIXEL CYCLE
        MOV     RC,DIOR       ; RC  --> DIO
        MOV     DIOR,A1       ; DIO --> A1
        BEQ     BINR,BIO      ; BRANCH IF BIN IS ZERO
```

```
        MOV    B1,MIOR           ; B1 --> MIO
        BRU    OVER
BIO :   BEQ    WINR,WIO          ; BRANCH IF WIN IS ZERO
        MOV    A2,MIOR           ; MOVE A2 TO RA VIA MIO
        MOV    MIOR,RA
        BRU    OVER
WIO :   MOV    A2,MIOR           ; MOVE A2 TO RA VIA MIO
        MOV    MIOR,RA
        SUB    A2,B2,DUM1        ; A2 - B2
        MOV    C,CSR             ; LATCH THE CARRY
OVER :  INC    PER,PER
        INC    UPAC,UPAC
        INC    PIX,PIX           ; ONE PIXEL OVER
        BNE    ROSIZ,PIX,START ; IS THE ROW OVER?
        MOV    #0,UPAC           ; ROW INIT. IF ROW IS
        MOV    #127,RC           ; OVER
        BRU    START
;
;
; THE PAL FUNCTION TO COMPUTE BIN AND WIN IS
; IMPLEMENTED AS A SUBROUTINE.
; THE PAL FUNCTION IS:
; BINR = (BININ)(QA)(QB)(QC) + (BI)(QA') +
; (BI)(QB') + (BI)(QC')
; WINR = (WININ)(QA)(QB)(QC) + (WI)(QA') +
; (WI)(QB') + (WI)(QC')
```

```
; ACCORDING TO THESE VALUES THE MICROOPERATIONS ARE
; CARRIED OUT. THERE ARE THREE TYPES OF DIFFERENT
; MICROOPERATIONS: FOR BIN = 1 WITH WIN 0 OR 1,
; FOR BIN = 0 AND WIN = 0; AND FOR BIN = 0 AND
; WIN = 1. THESE ARE DECIDED FROM THE COMPARISONS
; IN EVER CYCLE, BUT THE VALUES ARE SAME FOR ALL THE
; 8 CLOCK CYCLES IN A PIXEL CYCLE.
;
;
PAL :  NOP
       MOV        BIN,BININ
       MOV        WIN,WININ
       MOV        BININ,IN1
       MOV        WININ,IN2
       MOV        QAC,IN3
       MOV        QB,IN4
       MOV        QC,IN5
       AND        WININ,QAC,INT1
       AND        QB,QC,INT2
       AND        INT1,INT2,INT3
       AND        WINR,QAC,INT4
       AND        WINR,QBC,INT5
       AND        WINR,QCC,INT6
       OR         INT3,INT4,INT7
       OR         INT5,INT6,INT8
       OR         INT7,INT8,INT9
```

```
        MOV(W64)    INT9,WINR
;
        AND         BININ,QAC,INT1
        AND         QB,QC,INT2
        AND         INT1,INT2,INT3
        AND         BINR,QAC,INT4
        AND         BINR,QBC,INT5
        AND         BINR,QCC,INT6
        OR          INT3,INT4,INT7
        OR          INT5,INT6,INT8
        OR          INT7,INT8,INT9
        MOV(W64)    INT9,BINR
;
        RTS
;
;
; THIS IS THE SRAM TO STORE KO.
; IT IS INTIALIZED TO MAX-BLACK (255) .
;
SRAM : BYT      #255,#255,#255,#255,#255,#255,#255,#255
       BYT      #255,#255,#255,#255,#255,#255,#255,#255
       BYT      #255,#255,#255,#255,#255,#255,#255,#255
       BYT      #255,#255,#255,#255,#255,#255,#255,#255
       BYT      #255,#255,#255,#255,#255,#255,#255,#255
       BYT      #255,#255,#255,#255,#255,#255,#255,#255
       BYT      #255,#255,#255,#255,#255,#255,#255,#255
```

```
        BYT     #255,#255,#255,#255,#255,#255,#255,#255
        BYT     #255,#255,#255,#255,#255,#255,#255,#255
        BYT     #255,#255,#255,#255,#255,#255,#255,#255
        BYT     #255,#255,#255,#255,#255,#255,#255,#255
        BYT     #255,#255,#255,#255,#255,#255,#255,#255
        BYT     #255,#255,#255,#255,#255,#255,#255,#255
        BYT     #255,#255,#255,#255,#255,#255,#255,#255
        BYT     #255,#255,#255,#255,#255,#255,#255,#255
        BYT     #255,#255,#255,#255,#255,#255,#255,#255
        BYT     #255,#255,#255,#255,#255,#255,#255,#255
        BYT     #255,#255,#255,#255,#255,#255,#255,#255
        BYT     #255,#255,#255,#255,#255,#255,#255,#255
        BYT     #255,#255,#255,#255,#255,#255,#255,#255
        BYT     #255,#255,#255,#255,#255,#255,#255,#255
        BYT     #255,#255,#255,#255,#255,#255,#255,#255
        BYT     #255,#255,#255,#255,#255,#255,#255,#255
        BYT     #255,#255,#255,#255,#255,#255,#255,#255
        BYT     #255,#255,#255,#255,#255,#255,#255,#255
        BYT     #255,#255,#255,#255,#255,#255,#255,#255
        BYT     #255,#255,#255,#255,#255,#255,#255,#255
        BYT     #255,#255,#255,#255,#255,#255,#255,#255
        BYT     #255,#255,#255,#255,#255,#255,#255,#255
        BYT     #255,#255,#255,#255,#255,#255,#255,#255
        BYT     #255,#255,#255,#255,#255,#255,#255,#255
        BYT     #255,#255,#255,#255,#255,#255,#255,#255
;
```

```
;
;
; THIS IS THE RAM TO STORE GRAY-VAL.
;
GVFIF: BYT        #0;#0,#0,#0,#0,#0,#0,#0
       BYT        #0,#0,#0,#0,#0,#0,#0,#0
       BYT        #0,#0,#0,#0,#0,#0,#0,#0
       BYT        #0,#0,#0,#0,#0,#0,#0,#0
       BYT        #0,#0,#0,#0,#0,#0,#0,#0
       BYT        #0,#0,#0,#0,#0,#0,#0,#0
       BYT        #0,#0,#0,#0,#0,#0,#0,#0
       BYT        #0,#0,#0,#0,#0,#0,#0,#0
       BYT        #0,#0,#0,#0,#0,#0,#0,#0
       BYT        #0,#0,#0,#0,#0,#0,#0,#0
       BYT        #0,#0,#0,#0,#0,#0,#0,#0
       BYT        #0,#0,#0,#0,#0,#0,#0,#0
       BYT        #0,#0,#0,#0,#0,#0,#0,#0
       BYT        #0,#0,#0,#0,#0,#0,#0,#0
       BYT        #0,#0,#0,#0,#0,#0,#0,#0
       BYT        #0,#0,#0,#0,#0,#0,#0,#0
       BYT        #0,#0,#0,#0,#0,#0,#0,#0
       BYT        #0,#0,#0,#0,#0,#0,#0,#0
       BYT        #0,#0,#0,#0,#0,#0,#0,#0
       BYT        #0,#0,#0,#0,#0,#0,#0,#0
       BYT        #0,#0,#0,#0,#0,#0,#0,#0
       BYT        #0,#0,#0,#0,#0,#0,#0,#0
```

```
        BYT         #0,#0,#0,#0,#0,#0,#0,#0
        BYT         #0,#0,#0,#0,#0,#0,#0,#0
        BYT         #0,#0,#0,#0,#0,#0,#0,#0
        BYT         #0,#0,#0,#0,#0,#0,#0,#0
        BYT         #0,#0,#0,#0,#0,#0,#0,#0
        BYT         #0,#0,#0,#0,#0,#0,#0,#0
        BYT         #0,#0,#0,#0,#0,#0,#0,#0
        BYT         #0,#0,#0,#0,#0,#0,#0,#0
        BYT         #0,#0,#0,#0,#0,#0,#0,#0
        BYT         #0,#0,#0,#0,#0,#0,#0,#0
;
;
END
```

APPENDIX F. LIST OF PARAMETERS TO CHANGE QUEUE LENGTH

This chapter lists the parameters to be changed in GSPSIM.FOR, the FORTRAN program of the simulator. The present values of these parameters are also given at the end. The square brackets ([ ]) give the line numbers of the occurences of those parameters.

1. COMMON/QUEUE/QPINS(*),QTIME(*),QVALU(*),QLINK(*),QMOD(*)
   .... [86,973,1153,1226,1277,2776]

2. LABEL(*) .... IN COMMON STATEMENTS .
   DATA LABEL/ / .... [1030]

3. CODE1(*),CODE2(*) ...... [1028,1029 AND IN COMMON]

4. QLINK (**) ....
   [86,973,1153,1226,1259,1277,1571,2776]

5. QMOD (**) .... [87,974,1154,1227,1278,2777]

6. DATA BOTTOM / / ..... [988]
   DATA QPINS/ / ..... [990]
   DATA QTIME/ / ..... [991]
   DATA QVALU/ / ..... [992]

```
DATA     QLINK/      /        ..... [993]
DATA     QMOD/       /        ..... [994]
```

7. A global variable "MXQSIZ" has been declared to specify the maximum Queue size. Initialize it in BLOCK DATA statements.

   .... [1253, 1565]

8. BOTTOM = ... [1262, 1574]

9. MXQSIZ ..... [86].

PRESENT VALUES :

QPINS(*),QTIME(*).....	200.
LABEL(*)	3000.
CODE1(*)	5000.
CODE2(*)	10000.

The vita has been removed from the scanned document