THE UNIVERSITY OF LIVERPOOL

A PARALLEL IMAGE PROCESSING SYSTEM

Thesis submitted in accordance with the
requirements of the University of Liverpool
for the degree of Doctor of Philosophy by

Kin Wing Tse

Department of Electrical Engineering and Electronics

March 1997

For my beloved wai-man

"Unless the Lord builds the house, its builders labor in vain. Unless the Lord watches over the city, the watchmen stand guard in vain." **Psalm 127:1**

Acknowledgements

I would like to thank my supervisor, Dr. J.S. Smith, for his support, guidance and encouragement throughout this research; and also for his invaluable support and regards when I was involved in a serious traffic accident in April 1995. Many thanks are also due to Professor J. Lucas, who is leading the Computer Electronics and Robotics Research group, and without whom much of this work would not have been undertaken. I also wish to thank the present and previous Heads of the department, Professor J.D. Parsons and Professor G.R. Jones, for providing laboratory facilities.

I would also like to thank Dr. S.R. Wylie and Dr. T.M. Colclough for their corrections. Finally, I would like to express my gratitude to the Overseas Research Students Awards Scheme and British Nuclear Fuels plc for funding me during my research.

Abstract

For many vision-based applications, it is necessary, or advantageous, to process image data in real-time. Custom-designed image processing systems have been used to achieve a high performance for specific applications, but, as these are non-programmable, they cannot be adapted to suit different applications without major modifications.

Advanced VLSI technology has enabled parallel processor based systems, which can be programmed to perform a wide range of algorithms, to replace these custom-designed systems.

The aim of this research was to design and build a generic parallel processor based vision system, which was able to capture, process and display image data in real-time. The hardware for these three functions was integrated onto a single board, which, by simple interprocessor links or port connections, could be connected to a multi-processor network.

Contents

List of Figures

List of Tables

Glossary of Terms Used

This thesis contains several abbreviations, some of which are in common usage and some of which are specific to this research. Most of the abbreviations can be found in this glossary of terms.

ADC ..... Analog To Digital Converter
AHDL ..... Altera Hardware Description Language
ANSI .....American National Standards Institute
CCD .....Charge Coupled Device
CMOS .....Complementary Metal Oxide Silicon
CPU ..... Central Processing Unit
DAC ..... Digital To Analog Converter
DIPB .....Dedicated Image Processing Board
DMA .....Direct Memory Access
DRAM ..... Dynamic Random Access Memory
DSP ..... Digital Signal Processor
EEPROM .....Electrically Erasable Programmable Read Only Memory
EPROM ..... Erasable Programmable Read Only Memory
FPGA .....Field Programmable Gate Array
HTRAM .....High-Performance Transputer Module
IC .....Integrated Circuit
I/O ..... Input/Output
MFLOPS .....Millions of Floating Point Operations Per Second
MIPS ..... Million Instructions Per Second
Mpps .....Million Pixels Per Second
MPU ..... Microprocessor Unit
MUX .....Multiplexer
NDL .....Network Description Language
NTSC .....National Television System Committee
PAL ..... Phase Alternate Line
PC .....Personal Computer
PCB ..... Printed Circuit Board
PGA .....Pin Grid Array
PLD ..... Programmable Logic Device
ROM .....Read Only Memory
SECAM ..... Séquential couleur à mémoire
SRAM .....Static Random Access Memory
VCP ..... Virtual Channel Processor
VLSI ..... Very Large Scale Integration
VRAM ..... Video Random Access Memory

Chapter 1

Introduction

Image processing is playing an increasingly important role in many manufacturing processes, many of which require "real-time" performance. "Real-time" performance requires an image processing system to capture, process and display image data within one frame period and without any loss of frames [1]. To achieve this performance, with cameras capturing images at full frame rate, requires considerable processing power. This can be achieved by either using a very high performance single processor system, whose cost would make it impractical for many applications, or by using many low cost interconnected processors.

The T800 transputer based TIPS [2] image processing system was developed at the University of Liverpool in 1990. This system overcame the traditional problem of using transputers [3], which is the low data bandwidth of links, in image processing applications by allowing multiple processors, up to 8, to simultaneously process the captured image. This was achieved by allowing each processor to have a full copy of the image in its own local memory through the TIPS's backplane system. The image could then be processed and the results displayed on an external monitor.

Recently, advanced VLSI technology has allowed parallel processors to perform interprocessor communication, at a rate which is sufficient to transmit images in real-time. The IMS T9000 transputer [4] and the TMS320C40 digital signal processor [5] are parallel processors with high data bandwidth interprocessor communication links, or ports, suitable for building parallel image processing systems. Consequently, Quintek Ltd has built a modular image processing

system, based around the T9000 transputer, and Sundance Multiprocessor Technology Ltd has built another with the TMS320C40 digital signal processor.

For the Quintek QT9 Multi-Capture and Multi-Display System [6], the functions of image capture and image display are separately resident in the two T9000 transputer based boards, consequently at least two processors are involved for any application, unless one of the functions is ignored. For the image processing system developed by the Sundance, it is a combination of the SMT303 [7] frame-grabber and the SMT304 [8] graphic accelerator. These two modules are TMS320C40 DSP based and hence its basic configuration involves two processors.

The aim of this research was to design and build a single image processing board, which provides both the image capture function and image display function. In addition, the frame buffers can be accessed by the capture subsystem, the display subsystem and the processor system simultaneously. As a result, the overall image processing system is fully scaleable, enabling processing performance to be enhanced by the addition of multiple processors to the basic single processor system.

The second chapter of this thesis aims to discuss the key features of a processor, and a programmable logic device, which can be used to construct a real-time parallel image processing system. The IMS T9000 transputer and the TMS320C40 digital signal processor, were investigated and one was chosen to construct the image processing system. Altera and Xilinx are the two main manufacturers of large scale programmable logic devices. The devices available from these companies were investigated, and are discussed.

Chapter three describes the organisation and operation of the image processing system. The system, in general, consists of a dedicated image processing board, a number of IMS B927 HTRAMs [9], or IMS B926 HTRAMs [10], and the IMS

B108 HTRAM motherboard [11]. The hardware design of the dedicated image processing board is detailed in this chapter.

Chapter four introduces two configurations for the image processing system. The configuration of the image processing system for Single-T9 mode is similar to that for a T9000 single transputer network [12], and the configuration for Multi-T9 mode is similar to that for a T9000 multi transputer network. Nevertheless, this chapter explains the parameters used in the configurations, with particular reference to the image processing system which has been built.

The aim of chapter five is to highlight some areas of interest concerning the operation of the image processing system. The three frame buffers in the dedicated image processing board are purposely managed for different configurations and applications. For a parallel image processing architecture, the image is divided into a number of sub-images, so that an image processing task is shared by a number of processors. An image partition technique is required, and the principle behind the technique is explained. The end of chapter five introduces a number of image processing algorithms that were used to test the performance of the image processing system in different configurations.

Chapter six presents the results of a number of tests on the image processing system in different configurations. As the system performance is related to the test conditions, such as the link connections, the program structure, the programming language, and the number of processors; these conditions are stated, and their influence on system performance is discussed.

Chapter seven contains conclusions on the work undertaken and recommendations for future research.

Chapter 2

Determination of the Processor and the PLDs Used in the System

This chapter aims to discuss the key features of a processor and a programmable logic device, which determine its suitability for the development of parallel image processing systems. The T9000 transputer and the TMS320C40 digital signal processor were considered for the processor, whilst the Altera and Xilinx devices were considered for the programmable logic device. The programmable logic devices were required to provide the glue logic between the processor, the video digitizer, the video encoder, and the memory.

2.1 Processors

This section discusses the features which a processor should have in order to construct a parallel image processing system. The T9000 transputer and the TMS320C40 digital signal processor are parallel processors which are primarily designed to facilitate system configurability and provide massive parallelism, with very little, or no, glue logic. These two parallel processors were investigated and compared, and one of them was chosen as the processor for the system.

2.1.1 The Key Features of the Processor

Image processing is typified by large data structures, requiring many operations to be performed at high speed, in order to achieve real-time processing. It encompasses a wide variety of techniques and applications, such as image enhancement, image restoration, and image segmentation. As different applications require different processing speeds, a desirable trait of a parallel

system is the ability to optimise the cost/performance ratio by changing the number of processors.

Although the processing power of a parallel processing system can be increased by employing additional processors, a processor with a high-speed integer unit and a high-speed floating point unit is still advantageous. It is beneficial to have a system which is powerful enough to deal with a number of basic image processing algorithms, such as, a simple filtering operation for a whole frame area, using a single processor.

In addition to high processing speed, a high data bandwidth for the memory interface is also important. For some applications, a processor is required to read the entire original image, and write the processed image to memory for display. In this case, the data bandwidth must be high enough to support fetching instructions from the program memory, and transporting data to and from the data memory in real-time.

The address space of the processor should be large enough to access the program memory, the frame buffer memory, the overlay buffer memory and some I/O ports directly. Although the page memory system can make the processor access physical memory outside the address space of the processor, the additional time for paging memory would cause a reduction in the real-time processing performance.

The video signal, generated by a camera, is asynchronous to the image processing system. One method which can be used to synchronise the system and the video signal, is to program the processor to repeatedly poll an input port which is connected to the frame sync signal. Alternatively, an interrupt approach could also be used. In this case, the interrupt pin of the processor is connected to the frame sync signal, and the processor is interrupted once every frame period. For

this approach, the processor should have a pin for the interrupt and respond to it in one percent of the frame period or less.

The frame buffers are designed to be a shared memory, which is accessed by three parties, namely the processor, the frame capture subsystem, and the frame display subsystem. A state machine [13] was designed to manage the time-division multiplexing, such that each subsystem accessed the memory during its allocated time. In fact, the frame capture subsystem and the frame display subsystem are indirectly controlled by the same state machine, so this results in no loss of synchronisation between them. The processor and the state machine are two individual units, and are not synchronised. With some arrangements, the wait state [14] of a processor can be used, in such a way, that the processor is synchronised with the buffer memory access time. As a result, the processor should have a wait state pin for this purpose. Chapter three details the principle of using the wait state for synchronisation.

The main purpose of the interprocessor communication ports of the processor, for a parallel image processing system, is to transmit image data. The transfer rate of the communication ports should be high enough to enable video signals to be transmitted in real-time. In addition, the processor should be designed to provide massive parallelism, with very little or no glue logic. Furthermore, the availability of a C language [15] compiler for the processor is a great advantage, as it reduces the time spent learning a new language for the processor and allows easy portability of existing code.

Data traffic between processors in a parallel image processing system is quite heavy. A separate unit which controls this data transfer is necessary as it can maximise the CPU performance by alleviating this data transfer burden from the CPU.

2.1.2 Investigation of T9000 and TMS320C40

This section is aimed at providing introductions to the T9000 transputer and the TMS320C40 digital signal processor. As these two processors have some features which can fulfil the key features of a processor for building a parallel image processing system, it is advantageous to summarise their structures and operations before selecting which of the two will be used.

2.1.2.1 T9000

The T9000 transputer is a 32-bit CMOS microprocessor, which integrates a high performance central processing unit, a 16 KByte cache [16], a communication system, and other support functions, on a single chip. Figure 2.1 shows its major operational units.

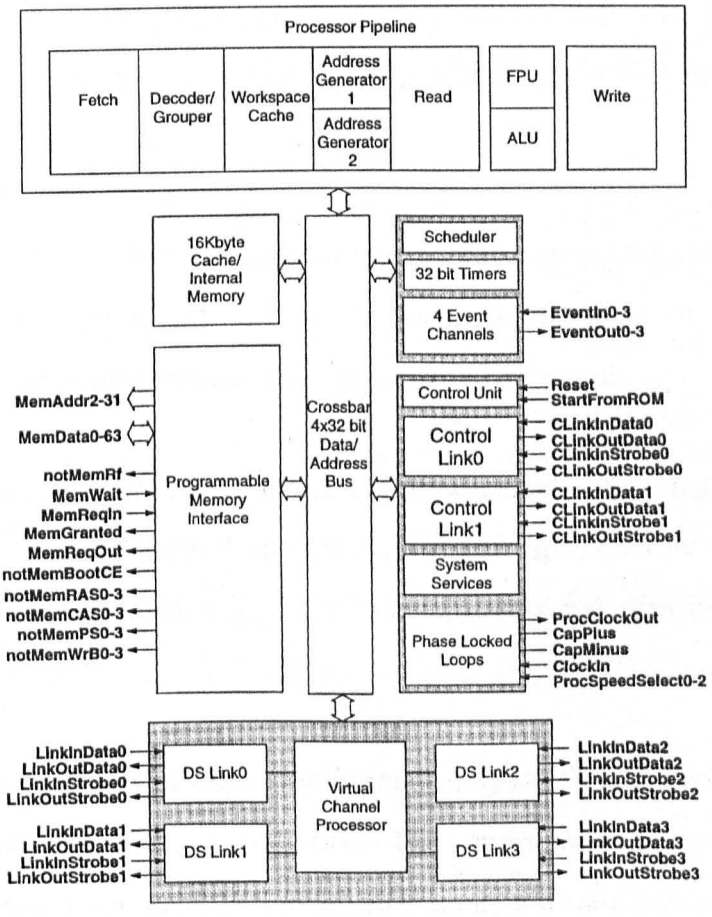


Figure 2.1 Block Diagram of T9000 Transputer
[from The T9000 Transputer Hardware Reference Manual]

The T9000 transputer has a pipelined superscalar architecture, which allows multiple instructions to be executed every processor cycle. Running at a processor speed of 20 or 25 MHz, it is capable of reaching 100 MIPS and 12.5 MFLOPS.

The 16 KByte cache provides a peak bandwidth of 120 Mwords/sec. It can also be programmed to function as 16 KByte of on-chip memory, or as 8 KByte of on-chip memory and 8 KByte of cache. This allows small applications to run with no external memory, and guarantees deterministic code behaviour for applications where this is critical.

The highly integrated programmable memory interface has a 4 GByte physical address space, and provides a peak bandwidth of 30 Mwords/sec. Four independent banks of external memory are supported to build a mixed memory system, which can contain DRAM, SRAM, EPROM and VRAM. The T9000 transputer has a 64-bit data bus, and each bank of memory can be configured to be 8, 16, 32 or 64 bits wide.

Four Event I/O pins provide asynchronous handshake interfaces between external events and internal processes, and can be used as interrupts or as controls for external peripherals. The response time is sub-microsecond.

The T9000 transputer has four serial communication links for interprocessor communication. The link speed can be configured up to a raw bit rate of 100 Mbit/sec. The four DS-Links support a total bidirectional data bandwidth of 40 Mbytes/sec.

Virtual channels are used in the communication system to control the routing of messages around a processor network. The operation of these channels is implemented solely by a virtual channel processor, without burdening the central processing unit. In addition, the VCP enables the T9000 to support up to 64k virtual channels over its four serial links.

Software support for parallel processing includes T9000 ANSI C [17] and T9000 occam 2 [18] toolsets, which were developed and are supported by INMOS.

2.1.2.2 TMS320C40

The TMS320C40 digital signal processor is a 32-bit floating-point processor, which integrates a high performance central processing unit, a program cache, six communication ports, a six-channel DMA coprocessor and other support functions, on a single chip. Figure 2.2 and Figure 2.3 show its major operational units.

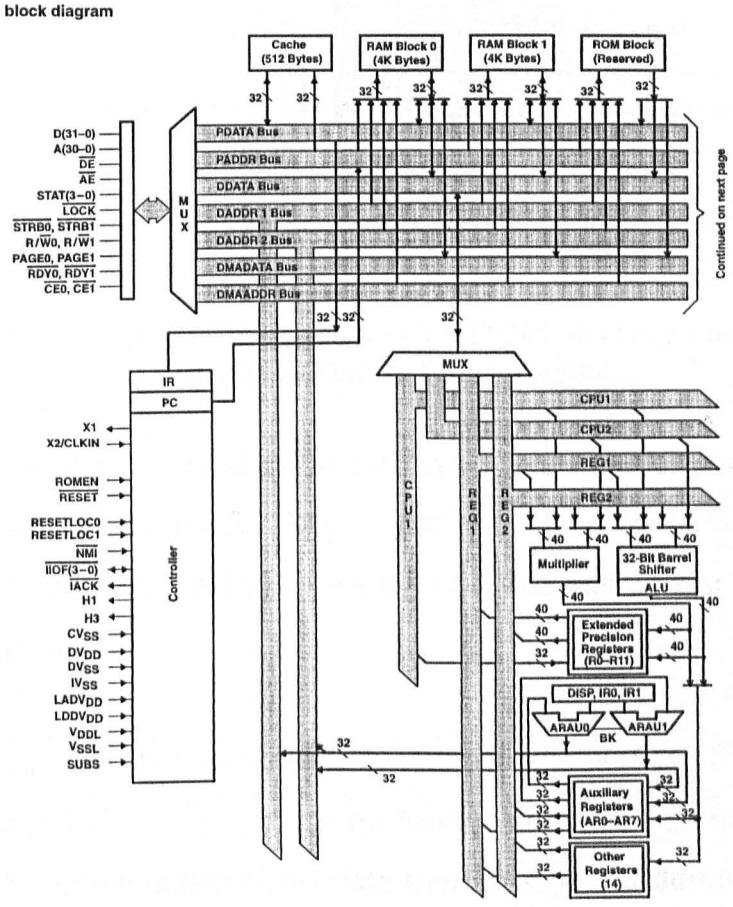


Figure 2.2 Block Diagram of TMS320C40 (Part One) [from TMS320C4x User's Guide]

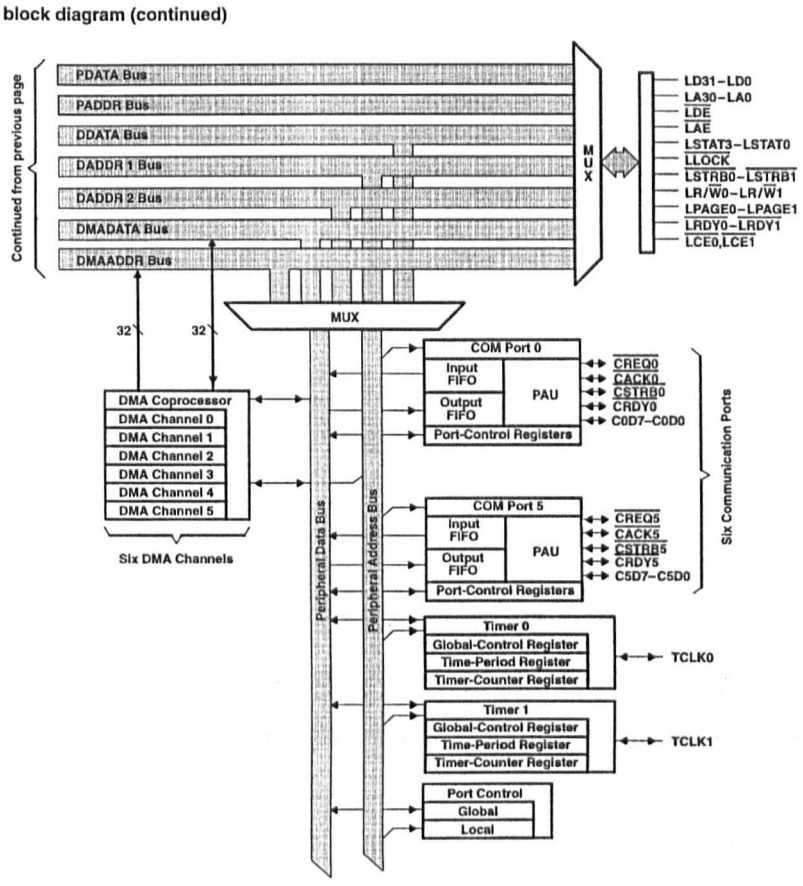


Figure 2.3 Block Diagram of TMS320C40 (Part Two)
[from TMS320C4x User's Guide]

The TMS320C40 CPU is configured for high-speed internal parallelism for the highest sustained performance. It is capable of 60 MFLOPS and 30 MIPS performance. Moreover, it provides hardware divide and square root support for improved performance.

The TMS320C40 DSP has six parallel communication ports, with eight data lines and four control lines each, for interprocessor communication. Each port can achieve 20 Mbytes/sec bidirectional data transfer rate. In addition, no glue logic is required to connect processors together through these ports, and the communication software is easy to use.

The six-channel DMA coprocessor is designed for concurrent I/O and CPU operation, thereby maximising sustained CPU performance, by alleviating the

CPU's burden in I/O data transfer. This coprocessor can be auto-initialised, which further maximises the sustained CPU performance.

Two identical external data and address buses support shared memory systems and a high data rate. Each consists of a 32-bit data bus with a peak bandwidth of 120 Mbytes/sec ('C40-60), a 31-bit address bus and two sets of control signals. Both buses can be used to address external program/data memory or I/O space. The buses also have external **RDY** signals for wait-state generation, with wait states inserted under software control.

On-chip program cache and dual-access/single-cycle RAM increase memory access performance. The 512-Byte instruction cache stores frequently used code and the 8-Kbyte single-cycle dual-access RAM stores data for easy access.

The TMS320C40 supports four external interrupts, a number of internal interrupts, a nonmaskable external interrupt, and a nonmaskable external RESET signal, which sets the processor to a known state. The DMA and communication ports have their own internal interrupts.

The TMS320C40 is supported by a host of parallel-processing development tools [19] for developing and simulating code easily, and for debugging parallel-processing systems. The code generation tools include an ANSI C, C, C++ and Ada compilers.

2.1.3 Discussions and Conclusion

Section 2.1.1 has discussed the key features of a processor, which is suitable for applying to a parallel image processing system, and sections 2.1.2.1 and 2.1.2.2 have introduced some features of two parallel processors. Before comparisons can be made, the spatial resolution and the intensity resolution of a frame must be fixed. Obviously, the higher the spatial resolution, and the intensity resolution, for each frame, the higher the required performance of the processor.

In this image processing system, a Raytheon TMC22071 video digitizer [20] was chosen, which converts standard baseband composite NTSC or PAL video [21] into 8-bit digital composite video data. It also provides five timing options, including the PAL-601 standard, which was selected for use. With this option, the size of the frame was fixed at 864 pixels by 512 lines with 25 Hz frame rate.

In the case of both input and output being full sized images, with the output being an improved version of the input, the required memory interface bandwidth is $864 \times 512 \times 25 \times 2$ byte per second, or 21.1 Mbyte/sec. This calculation also applies to the interprocessor communication port. For the T9000 transputer, the memory interface bandwidth is 4×30 Mbyte/sec with 32-bit data bus configured, and the total bidirectional data bandwidth of the four links is 17.58×4 Mbyte/sec, or 70.32 Mbyte/sec. For the TMS320C40 DSP, the peak memory interface bandwidth is 100 Mbyte/sec, and the bidirectional data bandwidth of each communication port is 20 Mbyte/sec. With consideration to the control bits used by protocol, there is no doubt that both processors are suitable.

For the other criteria such as interrupt, wait state, software development tools, and the simplicity in constructing a parallel system, both processors are also suitable. To conclude, these two processors are very competitive in their suitability for this project.

The Department of Electrical Engineering and Electronics, at the University of Liverpool, has built up considerable expertise in the construction of transputer based hardware. As a result, there was more technical support available if the parallel image processing system employed the T9000 transputer. This led to the selection of the 20 MHz T9000 transputer.

2.2 Programmable Logic Device

This section aims to discuss the features of programmable logic devices, discuss the key features which are applicable to the parallel image processing system and introduce two families of PLD for consideration. Of the PLDs currently on the market, the MAX7000 family of Altera [22] and the XC3000 series of Xilinx [23] were investigated and are discussed in this section.

2.2.1 The Attractions of Programmable Logic

Programmable logic devices are a category of Integrated Circuits whose ultimate function is determined by the designer. Programming is performed either by loading internal storage registers, or by inducing permanent or reversible physical changes in selected parts of the circuit.

There are a lot of advantages to using PLDs in digital system design. Firstly, the number of ICs in a system can be reduced, as many logic gates and registers are fabricated into a single package. Secondly, minor circuit modification can be made conveniently at the prototype stage, by simply reprogramming the PLD without having to alter the board layout. Thirdly, the time taken building a system is much shorter than if discrete logic gates and registers are used, as interconnections of gates and registers are made by programming. Fourthly, the simulation function of the development tools can help the designer to quickly identify and correct logical errors. Finally, PCB design can be simplified because of the freedom to allocate signals to pins.

2.2.2 The Key Features of The Device

It is desirable to keep the number of chips used on the board to a minimum, so a programmable logic device should consist of a large number of gates and cells, so that one or two chips are sufficient to support all the combinational and sequential logic. In addition to the number of gates and cells available, the

number of I/O pins is also a factor which will affect the total number of PLDs required.

For the prototyping board, a high density speedwire board with PGA area manufactured by VERO Electronics [24] was chosen. A PGA package was chosen to populate the PGA area. Since the width of the PGA area on the prototyping board was 17-way, the size of the PLD package must not be greater than 17 × 17.

The frame buffer memory and the overlay buffer memory of the design are triple-ported memory and dual-ported memory respectively. These memories are made up with static RAMs, tri-state buffers, and latches. The last two items are implemented by PLDs. Therefore the PLD must contain sufficient tri-state buffers.

Since the state machines and counters are driven by a clock at 27 MHz, which is twice the pixel clock, the counter speed of PLDs must be equal to, or greater than, this value. The duration of each state of the system state machine is approximately 37 ns, so the propagation delay of any logic should be much less than 37 ns, so as to produce desirable timing waveforms.

The PLD should be reprogrammable for the sake of quick and efficient modifications during design, development, and debug cycles. If the device can be electrically erasable, the time to erase the program in the PLD would be negligible.

2.2.3 Investigation of Altera Devices and Xilinx Devices

This section aims to give a brief introduction to the Altera MAX7000 family and the Xilinx XC3000 series of programmable logic devices. The devices from both the MAX7000 family and the XC3000 series are capable of meeting the key

features mentioned in the previous section, but one device, the EPM7256EGC192-12, was preferable.

2.2.3.1 MAX7000 Family

In general, the MAX7000 family of high-density, high-performance CMOS devices is based on Altera's second-generation MAX architecture. Fabricated with advanced EEPROM-based CMOS technology, the MAX7000 family provides 600 to 5,000 usable gates, pin-to-pin delays as fast as 5 ns, and counter speeds up to 178.6 MHz.

The higher-density members of the MAX7000 family, called MAX7000E devices, include the EPM7128E, EPM7160E, EPM7192E, and EPM7256E devices. These devices have several enhanced features, such as additional global clocking and additional output enable controls.

MAX7000 devices are available in a wide range of packages, including plastic J-lead chip carrier (PLCC), ceramic pin-grid array (PGA), plastic quad flat pack (PQFP), power quad flat pack (RQFP), and 1-mm thin quad flat pack (TQFP).

The MAX7000 devices use CMOS EEPROM cells to implement logic functions. The user-configurable MAX7000 architecture accommodates a variety of independent combinational and sequential logic functions. The devices can be reprogrammed and are guaranteed for 100 program and erase cycles.

The MAX7000 family is supported by Altera's MAX+PLUS II development system [25], which offers schematic, text and waveform design entry; compilation and logic synthesis; simulation and timing analysis; and a device programmer.

2.2.3.2 XC3000 Series

The XC3000 series Field Programmable Gate Array (FPGAs) provide a group of high-performance, high-density, digital integrated circuits. Using high-performance CMOS static memory technology, this series provides 1,000 to 7,500 logic gates, guaranteed toggle rates of 70 to 370 MHz, logic delays from 9 to 1.5 ns, and system clock speeds in excess of 80 MHz.

The FPGA user logic functions and interconnections are determined by the configuration program data, stored in the internal static memory cells. The configuration program can be loaded into the device an infinite number of times.

The XC3000 series devices are available in a wide range of packages, including plastic and ceramic surface-mount and pin-grid-array packages, as well as thin, and very thin, Quad Flat Pack (TQFP and VQFP) options.

The XC3000 series is supported by the XACT *step* development system [26], which provides: schematic capture; automatic place and route; logic and timing simulation; an interactive design editor for design optimization; timing calculation; and interfaces to popular design environments.

2.2.4 Discussion

Programmable logic devices can be integrated together to implement a large logic design. Any device from a desirable family can be used, since they share the same features. However, the logic design for the image processing board involves a large number of gates, registers and I/O pins, so only devices with a large number of gates, registers and I/O pins were considered. For example, the address, data and control lines of the T9000 transputer are connected to PLDs, and the address, data, and control lines of the buffer memories are also connected to them. The number of these I/O pins is considerable indeed.

In the MAX7000 family, the EPM7256E is the top of the range. It has 5,000 usable gates and 164 I/O pins. The 192-Pin PGA package is available for this device, and the size of this package is 17 × 17, which can be fitted onto the PGA area of the prototyping board. Thus, the EPM7256E was a suitable choice for use in this project.

In the XC3000 series, the XC3195A is the top of the range. It is available in two PGA packages, one with 223 pins and the other with 175 pins. The 223-Pin PGA package does not fit onto the PGA area of the prototyping board because its size is 18 × 18 way, so it was not considered in the prototyping stage. However, after the prototyping stage, it could be used to build the hardware on PCB. For the package of 175-Pin PGA, it can be fitted onto the PGA area at the cost of the number of I/O pins dropped to 144, so this package was also considered for the design.

In the University of Liverpool, the Computer Electronics and Robotics Research group of the Department of Electrical Engineering and Electronics, is currently using Altera's MAX+PLUS II development system. Preference in choosing PLDs would be given to the Altera devices unless the Xilinx devices were much better than the Altera devices. Since the devices from Xilinx were comparable to devices from Altera, programmable logic devices were chosen from Altera and finally two devices, EPM7256EGC192-12, were employed.

Chapter 3

The T9000 Transputer Based Image Processing System

This chapter describes the organisation and operation of the image processing system. This system consists of a dedicated image processing board for image capture, processing and display, a number of general purpose T9000 processor boards, such as the IMS B926 HTRAM or the IMS B927 HTRAM, which enhance the overall processing power, and an IMS B108 motherboard, for supporting HTRAMs and providing the PC bus to DS-Link interface.

3.1 The Dedicated Image Processing Board

The dedicated image processing board was designed to convert an analogue video signal from a CCD camera into a digital video signal. It provides real-time video output of captured images, processed images, or captured images overlaid with processed results, as well as transputer access for image analysis. The major components are: a video decoder, a video encoder, the IMS T9000 transputer, three frame buffers, and one overlay buffer. As the three frame buffers are accessible by the frame capture subsystem, the frame display subsystem and the T9000 transputer, and the overlay buffer is accessible by the frame display subsystem and the T9000 transputer, a multiple bus system was built, which is controlled by state machines. One of these is the system state machine, which has eight states, as shown in Figure 3.1, that define the memory access sequence. This state machine is only used during bus multiplexing time, which occurs when pixel data is being written by the video decoder or read by the video encoder. At any time outside the bus multiplexing time, the bus for accessing the frame buffers is used only by the T9000 transputer. The text design file for the system state machine is shown in Appendix B [pp147].

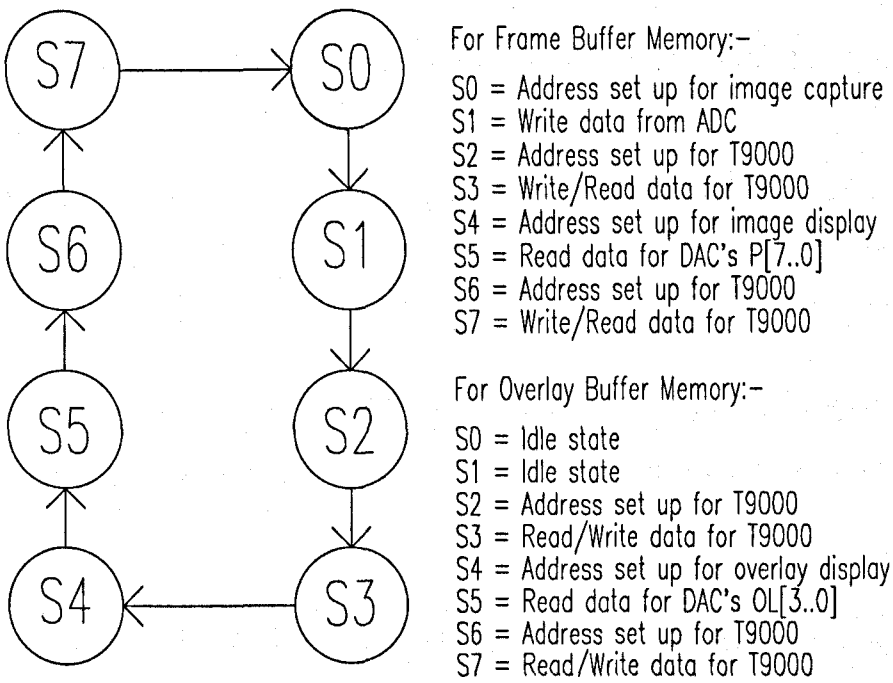


Figure 3.1 The System State Machine

3.1.1 Block Description

Although the digitised image contained 8 bits per pixel, the data width of the T9000 transputer was set to 32-bit for the frame buffer memory in order to reduce the frequencies of both the frame capture subsystem, and the frame display subsystem accessing the frame buffer memory, by a factor of four. The advantage of this arrangement was to allow the T9000 transputer to have more time to read or write to the memory. The frame buffer memory access frequencies could have been further reduced if the data width was set to 64-bit. Unfortunately, the frame buffer memory would then have been cacheable, as this is a property of the T9000 transputer's memory interface, and hence prevents it being used as a shared memory.

Figure 3.2 shows the main functional blocks of the dedicated image processing board. The CCD video camera encodes incoming images into a composite video signal, which is fed to a high-speed flash analogue to digital converter for digitisation. Four consecutive 8-bit digitised pixels are sequentially stored into a

32-bit buffer of the pixel-in box, to make it compatible with the 32-bit frame buffers, the top-left pixel being stored at the lowest address and the bottom-right pixel being stored at the highest address.

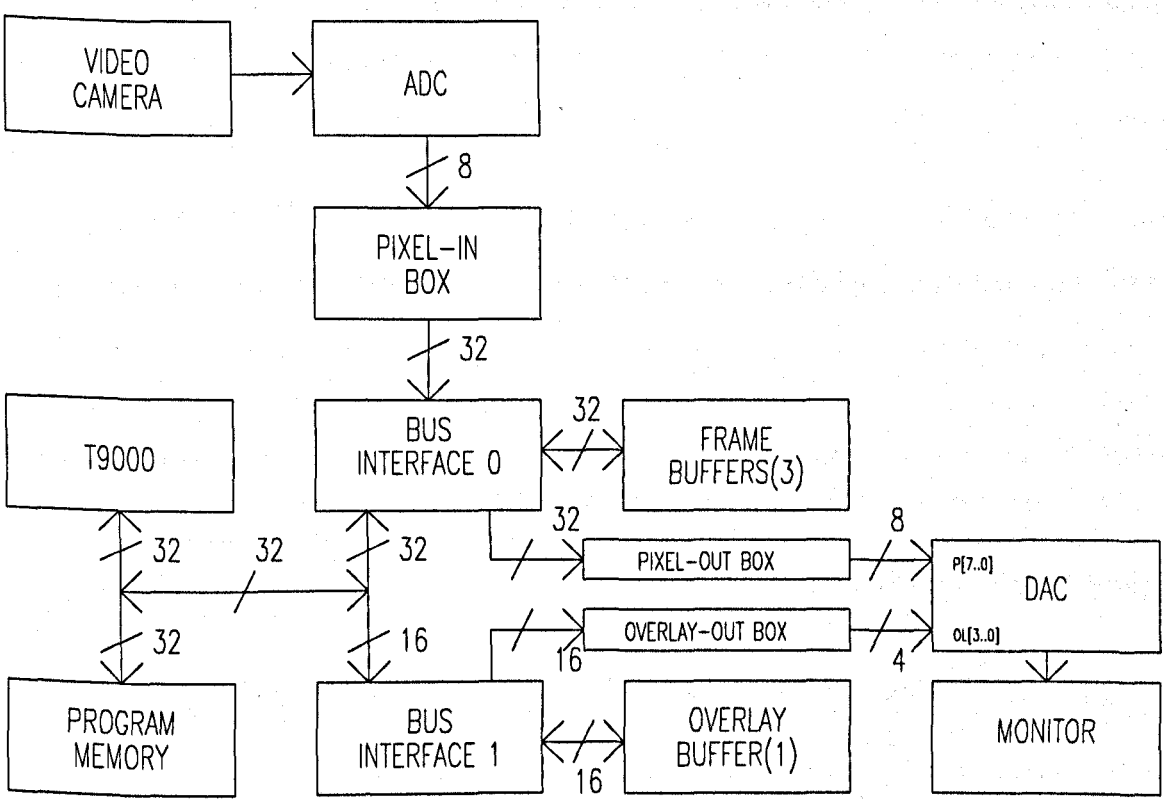


Figure 3.2 The Block Diagram of The Dedicated Image Processing Board

Bus interface 0 allows the T9000 transputer, the frame capture and the frame display subsystems to access the frame buffers, whilst bus interface 1 allows the T9000 transputer and overlay subsystems to access the overlay buffer. There are three frame buffers, each being organised as 512 × 1024 byte with an 8-bit data width and one overlay buffer, which is organised as 512 × 1024 byte with a 4-bit data width.

To display the contents of the frame buffers and the overlay buffer, 32-bit data is read from one of the frame buffers, stored into the latch of the pixel-out box, and then multiplexed as four groups of 8-bit data, which is sequentially fed to the pixel input of the digital to analogue converter. Similarly, the 16-bit data from the overlay buffer is stored into the latch of the overlay-out box, and then

multiplexed as four 4-bit data groups, which are sequentially fed to the overlay input of the digital to analogue converter. With the composite sync signal provided by the video sync separator device, the digital to analogue converter encodes the pixel data and the overlay data into the standard composite video signal for display on a monitor.

3.1.2 Memory Map

The external address space of the T9000 transputer was partitioned into four banks. The timing of each of the banks could be programmed separately, which enabled a mixed memory system to be used without the need for external hardware support.

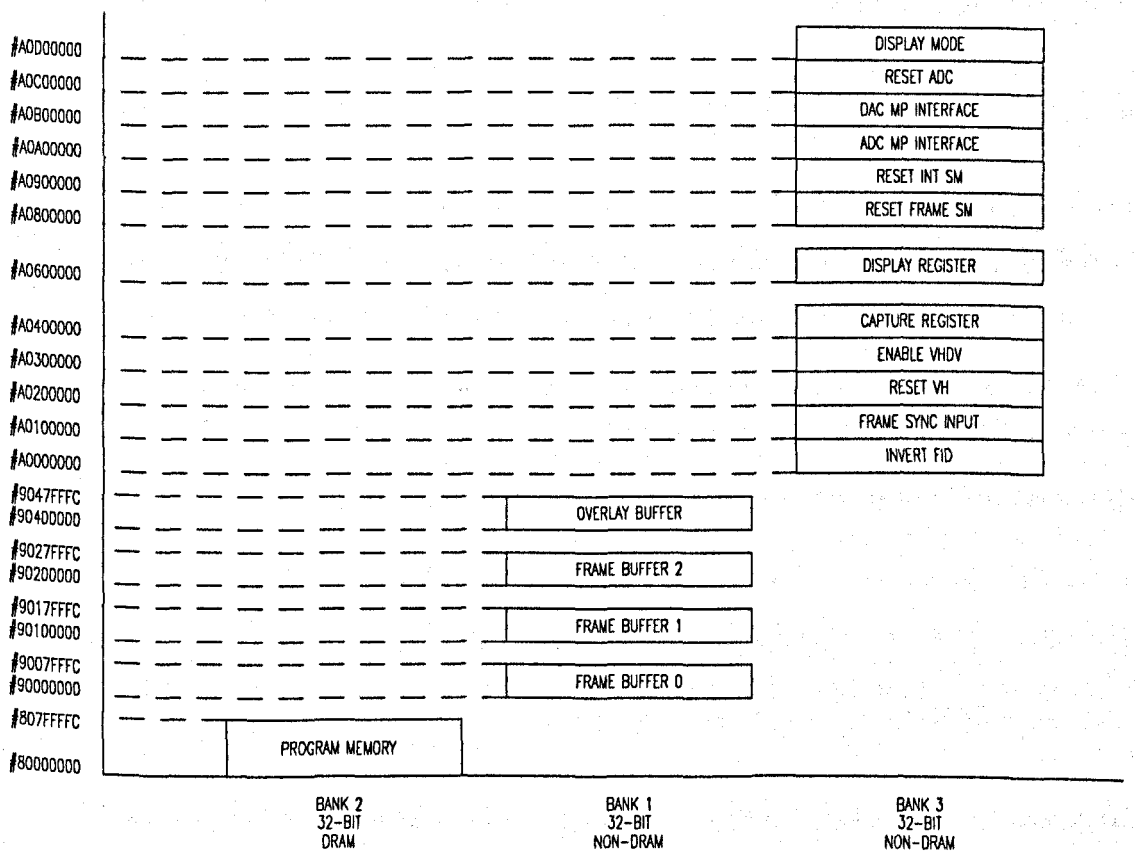


Figure 3.3 Memory Map

Figure 3.3 shows the memory map of the dedicated image processing board. As there was no pin on the T9000 transputer to indicate either memory access or I/O

access, all the I/O devices were mapped in the memory space. Bank 1 was configured to support Static RAMs, which were used for the frame buffers and the overlay buffer. Bank 2 was programmed to support Dynamic RAMs, which were used for program memory. Bank 3 was assigned for I/O devices, the timing in this bank satisfied the timings of the microprocessor interfaces of both the analogue to digital converter and the digital to analogue converter, as well as some of the data latches. Bank 0 was not used.

In order to simplify the address decoder circuitry, partial decoding [27] was employed. With partial decoding, the same memory cell can be accessed at many different memory addresses. All memory and I/O blocks were carefully placed so that they did not appear at the wraparound addresses of others.

3.1.3 Memory Operations

The performance of the microprocessor system is related to the speed of instruction code fetching. In general, the Dynamic RAM access time is longer than that of Static RAM, however, there are several reasons why Dynamic RAMs rather than Static RAMs were chosen to provide the program memory.

First of all, the density of a storage cell of Dynamic RAM is much higher than that of Static RAM, so more storage cells can be put into a single chip, hence the number of memory chips required is reduced. Secondly, the T9000 transputer has 16 Kbytes of cache memory, which stores the currently executed block of instructions for faster access by the CPU. Normally, 16 Kbytes of cache memory is large enough to significantly reduce the number of accesses to the external Dynamic RAMs for most image processing applications. Thirdly, Dynamic RAM is less expensive per bit than Static RAM. Fourthly, the T9000 programmable memory interface was designed to provide efficient support for Dynamic RAM without additional circuitry for refreshing and address multiplexing.

Video RAM [28] is normally used for frame memory. However, it was not chosen for this project as its serial port cannot be simultaneously accessed by the frame capture subsystem and frame display subsystem in real-time operation, in spite of a multiple bus interface which has been added to it. Instead, Static RAM equipped with a multiple bus interface was selected which enables the T9000 transputer, the frame capture subsystem and the frame display subsystem to access the frame memory in real-time.

3.1.3.1 Program Memory

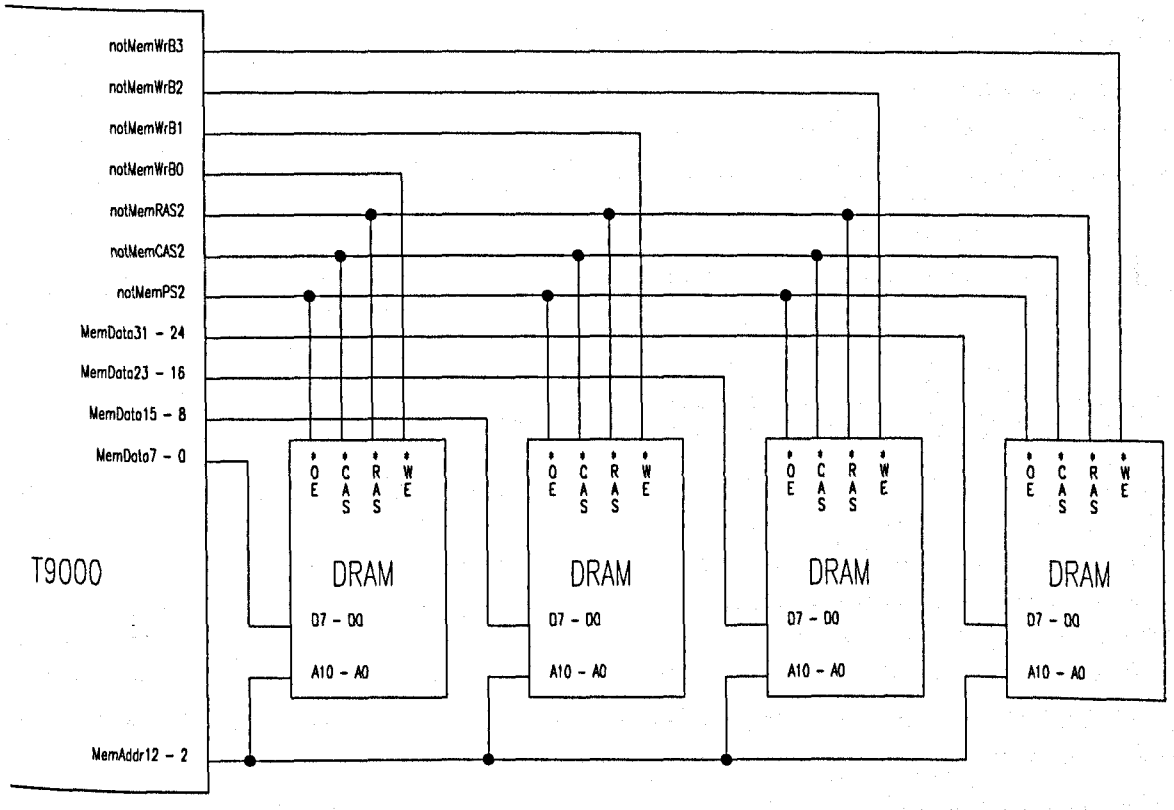


Figure 3.4 Program Memory Interface

Four HM5117800 Dynamic RAMs [29] were used for the program memory. They were altogether organised as 2097152-word × 32-bit memory, in which each byte, within a word, can be selected for writing. Figure 3.4 shows these RAMs interfacing to the T9000 transputer.

The notMemWrB0-3 lines are the write strobes used for selecting a byte within a word for writing. The notMemRAS2, notMemCAS2 and notMemPS2 lines are programmable strobes which are allocated to bank 2, they are active only when the T9000 transputer accesses the addresses assigned for this bank. The timings of these strobes are controlled by the corresponding registers, which are detailed in chapter four. It should be noted that no external glue logic was required.

3.1.3.2 Frame Buffer Memory

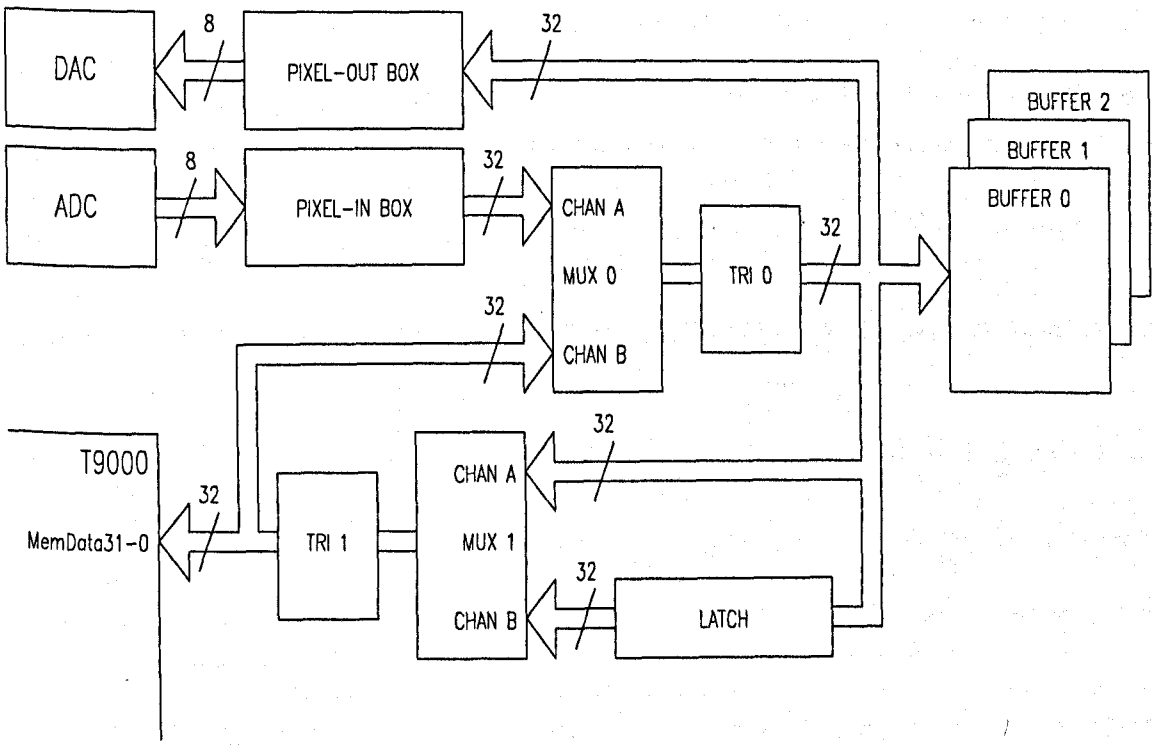


Figure 3.5 Interface for The Frame Memory's Data Bus

Twelve W241024AK-20 Static RAMs [30] were used for the frame buffer memory. They were grouped to constitute three frame buffers, each buffer has a memory size of 512 × 1024 bytes.

As the W241024AK-20 Static RAMs are not multi-port memory devices, a multiple bus system was applied to these RAMs in such a way that the T9000 transputer, the frame capture subsystem and the frame display subsystem could

access any one of these frame buffers in real-time. Figure 3.5 shows the main components in the multiple bus system for the data bus of the frame buffers.

In Figure 3.1, the eight states for the frame buffer memory were defined. However, for simplicity, state 0 and state 1 are collectively called the ADC state, state 2 and state 3 the T9000 state, state 4 and state 5 the DAC state and, state 6 and state 7 the T9000 state again. In the ADC state, channel A of multiplexer 0 was chosen, the tri-state buffer 0 was enabled and the tri-state buffer 1 was disabled. In the DAC state, data was stored to the latch of the pixel-out box, tri-state buffer 0 and tri-state buffer 1 were disabled. In the T9000 state for writing to the frame memory, channel B of multiplexer 0 was selected, tri-state buffer 0 was enabled and tri-state buffer 1 was disabled. In the T9000 state for reading from frame memory, tri-state buffer 0 was disabled, data was stored to the latch, channel B of multiplexer 1 was chosen, and tri-state buffer 1 was enabled. As mentioned before, the system state machine was only used during the bus multiplexing time. At any time outside the bus multiplexing time, if the T9000 transputer reads from the frame memory, tri-state buffer 0 was disabled, tri-state buffer 1 was enabled, and channel A of multiplexer 1 was selected. Figures 3.6-3.10 show the data paths for each of the situations.

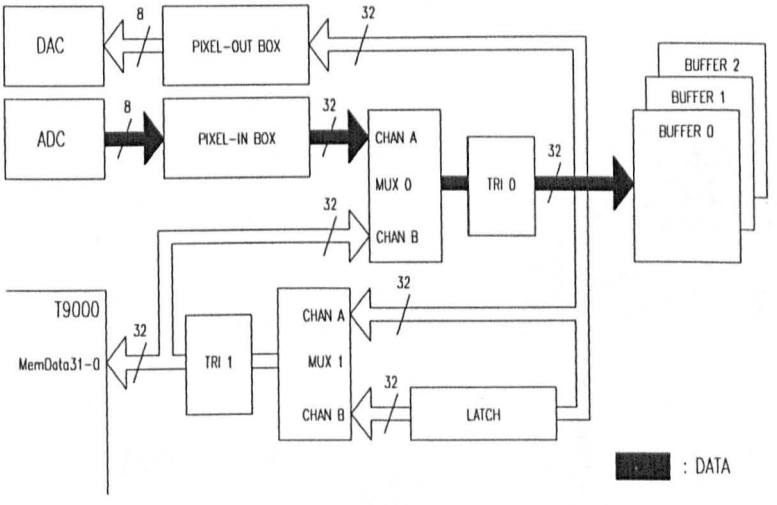


Figure 3.6 Data Path of the Frame Memory in the ADC State

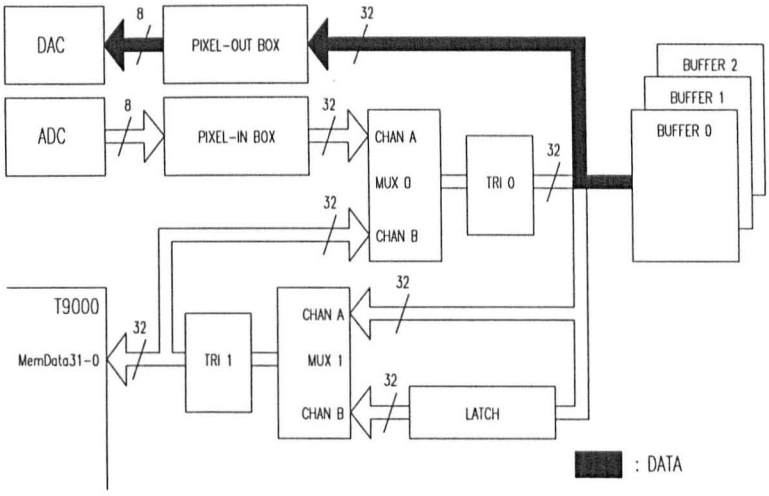


Figure 3.7 Data Path of the Frame Memory in the DAC State

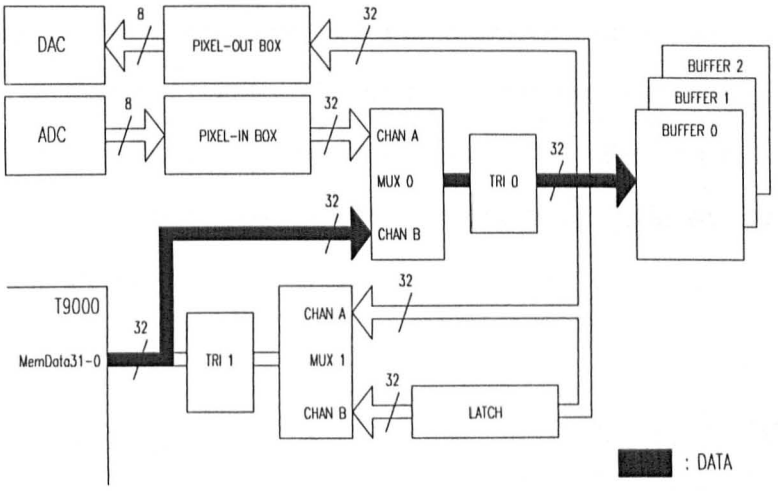


Figure 3.8 Data Path of the Frame Memory in the T9000 State for Write Cycle

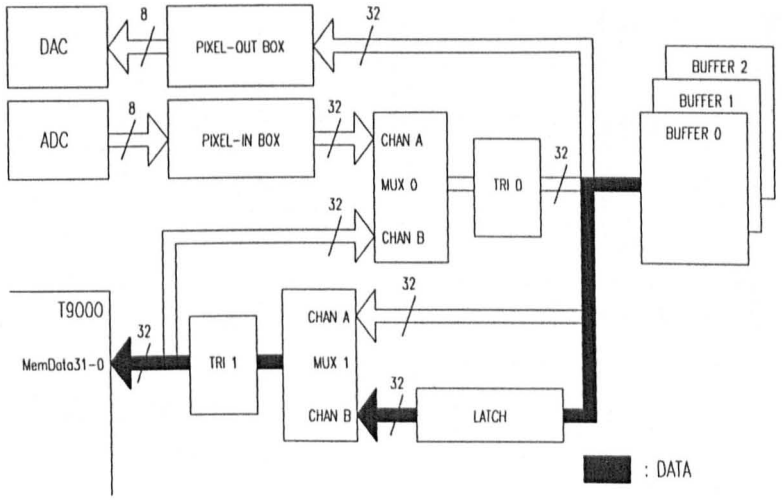


Figure 3.9 Data Path of the Frame Memory in the T9000 State for Read Cycle

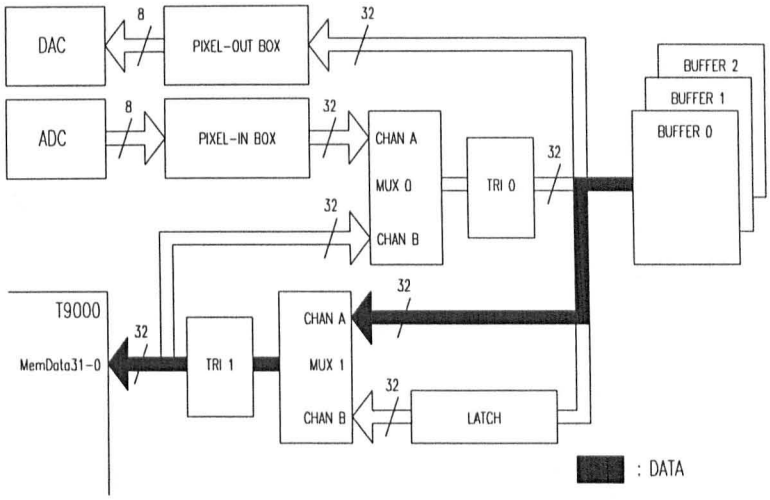


Figure 3.10 Data Path of the Frame Memory When the T9000 Reads from the Frame Memory outside the Bus Multiplexing Time

Figure 3.11 shows the organisation of the multiple bus system relating to the local address bus and the control bus of the frame memory. The frame capture subsystem and the frame display subsystem share the same address counters, and the contents of the counter remains constant during a cycle of the system state machine. A cycle starts from state 0 and ends in state 7.

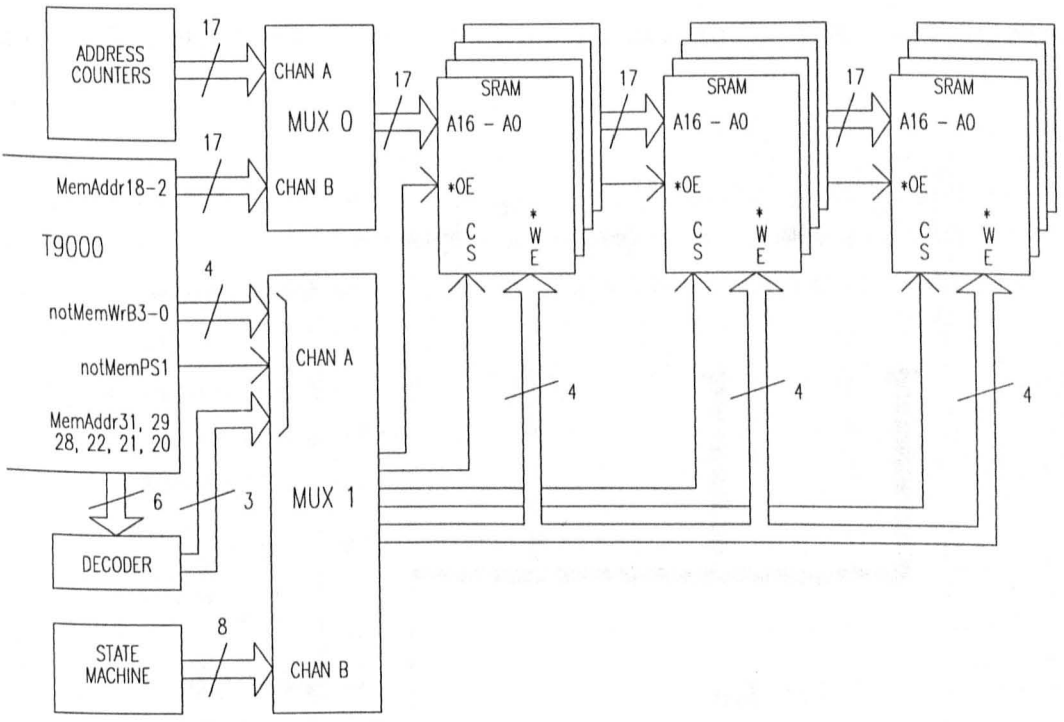


Figure 3.11 Interface for The Frame Memory's Address and Control Buses

In both the ADC and DAC states, channel A of multiplexer 0 and channel B of multiplexer 1 were selected; whereas in the T9000 state, channel B of multiplexer 0 and channel B of multiplexer 1 were selected. At any time outside the bus multiplexing time, when the T9000 accesses the frame memory, channel B of multiplexer 0 and channel A of multiplexer 1 were selected. Figures 3.12-3.14 show the signal paths for each of the situations.

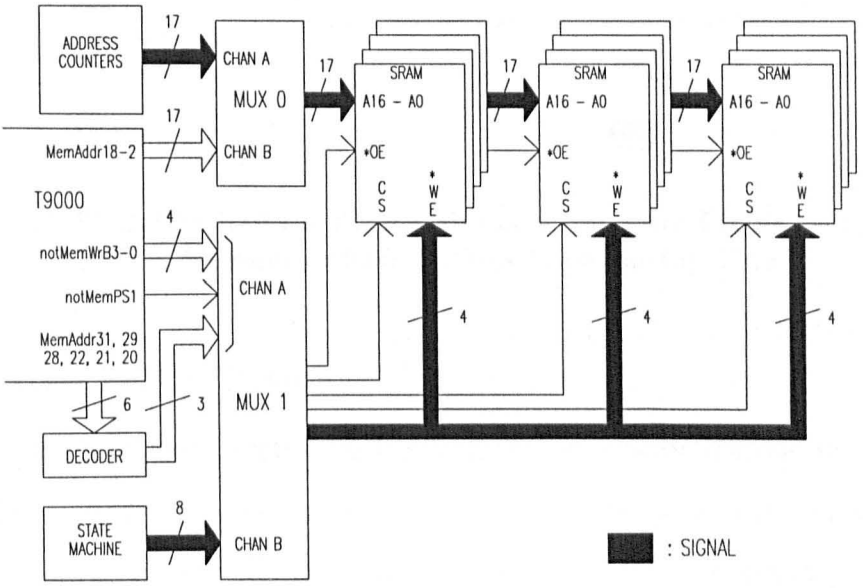


Figure 3.12 Signal Path of the Frame Memory in Both the ADC and DAC States

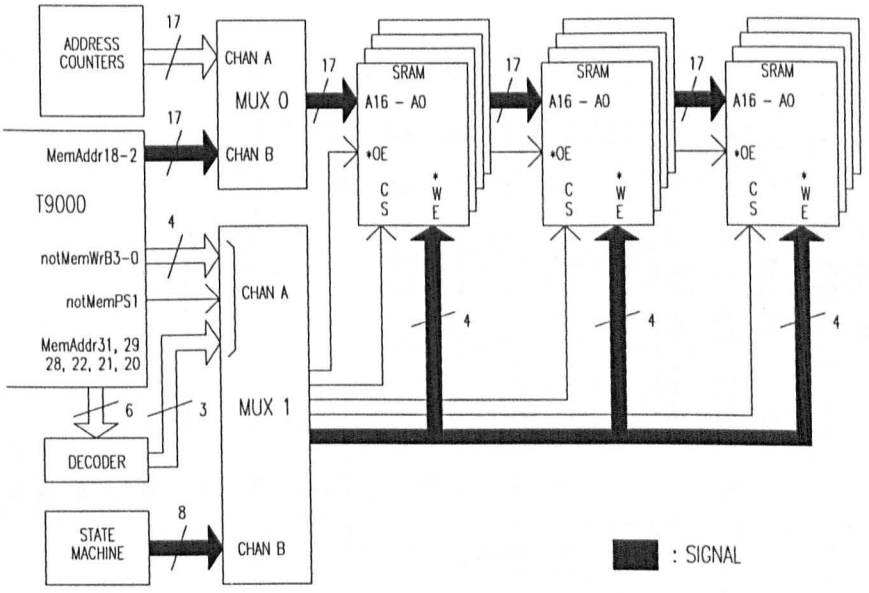


Figure 3.13 Signal Path of the Frame Memory in the T9000 State

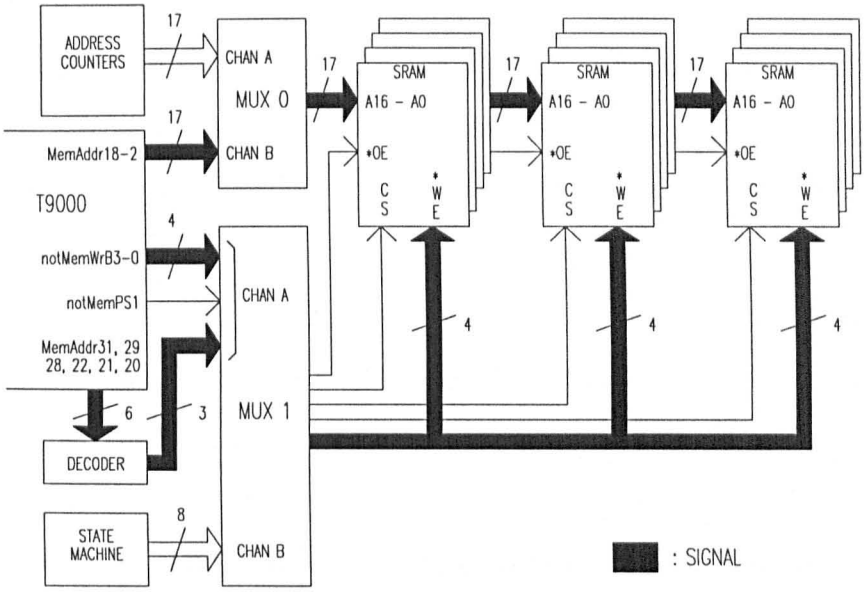


Figure 3.14 Signal Path of the Frame Memory when the T9000 Accesses the Frame Memory outside the Bus Multiplexing Time

3.1.3.3 Overlay Buffer Memory

The organisation of the overlay buffer memory was very similar to that of the frame buffer memory. Instead of a triple-ported memory, a dual-ported memory was used to allow access for the T9000 transputer, and the overlay subsystem. Two W241024AK-20 Static RAMs were used to give a 524288 word × 16-bit memory. Figure 3.15 shows the multiple bus interface for the data bus.

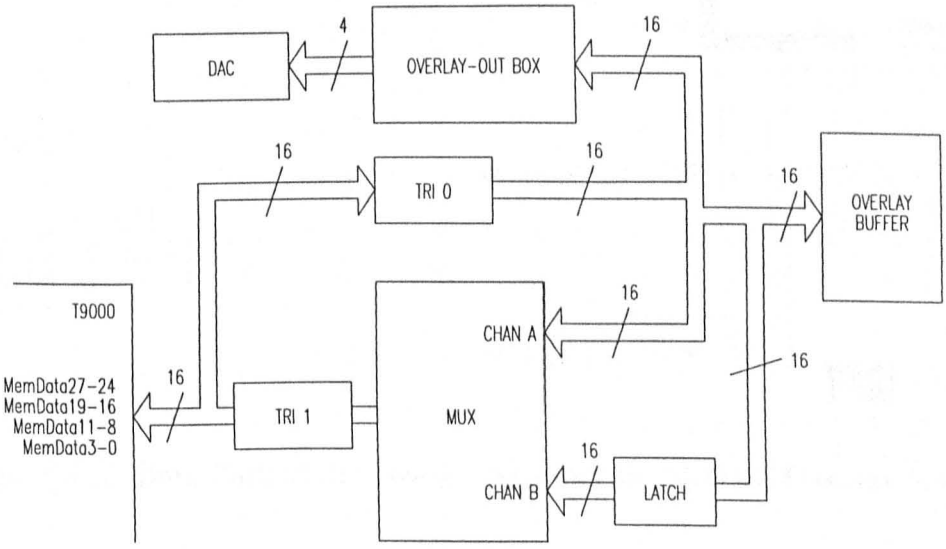


Figure 3.15 Interface for The Overlay Memory's Data Bus

There are eight defined states in the system state machine for the overlay buffer memory, as shown in Figure 3.1. For the sake of simplicity, the first two states are called the idle state, state 2 and state 3 are called the T9000 state, state 4 and state 5 the DAC overlay state and state 6 and state 7 are called the T9000 state. In the idle state, tri-state buffers 0 and 1 were disabled. In the DAC overlay state, data was stored in the latch of the overlay-out box, tri-state buffer 0 and tri-state buffer 1 were disabled. In the T9000 state for reading from the overlay buffer memory, tri-state buffer 0 was disabled, data was stored to the latch, channel B of the multiplexer was chosen and tri-state buffer 1 was enabled. As mentioned before, the system state machine is only applied when the local buses are multiplexed. At any time outside the bus multiplexing time, when the T9000 reads from the overlay buffer memory, tri-state buffer 0 was disabled, tri-state buffer 1 was enabled, and channel A of the multiplexer was selected. In the T9000 state for writing to the overlay buffer memory, tri-state buffer 0 was enabled and tri-state buffer 1 was disabled. Figures 3.16-3.19 show the data paths for each of the situations.

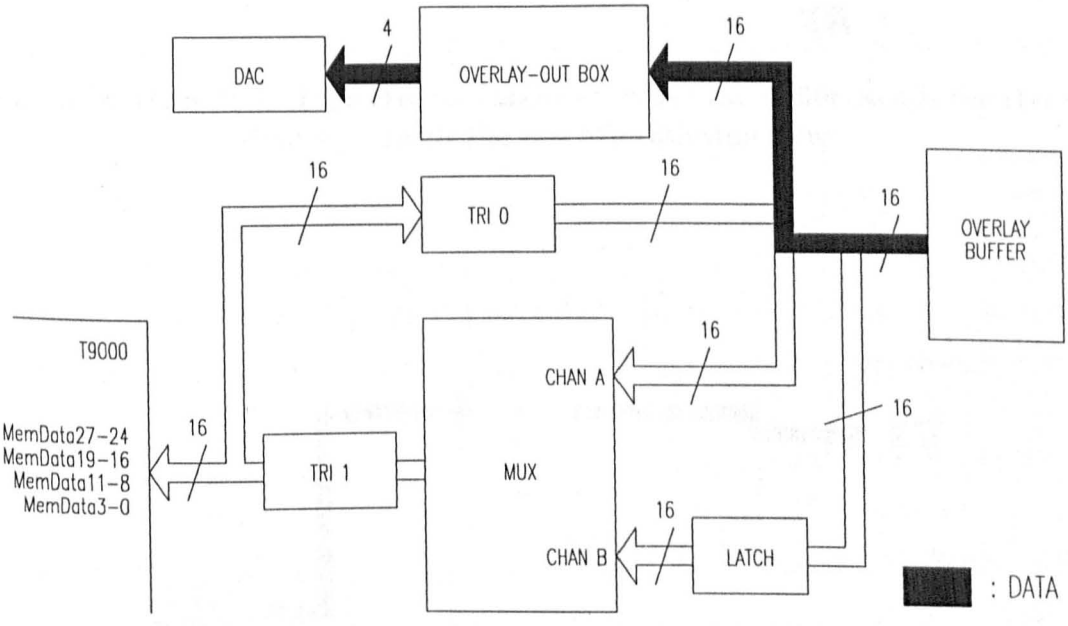


Figure 3.16 Data Path of the Overlay Memory in the DAC Overlay State

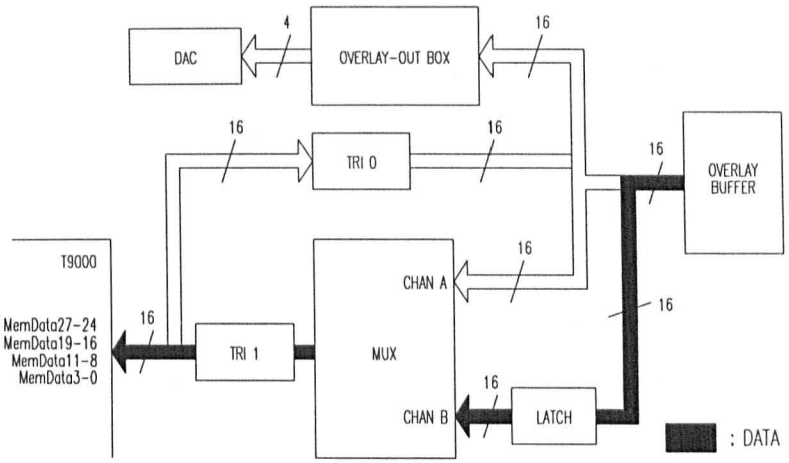


Figure 3.17 Data Path of the Overlay Memory in the T9000 State for Read Cycle

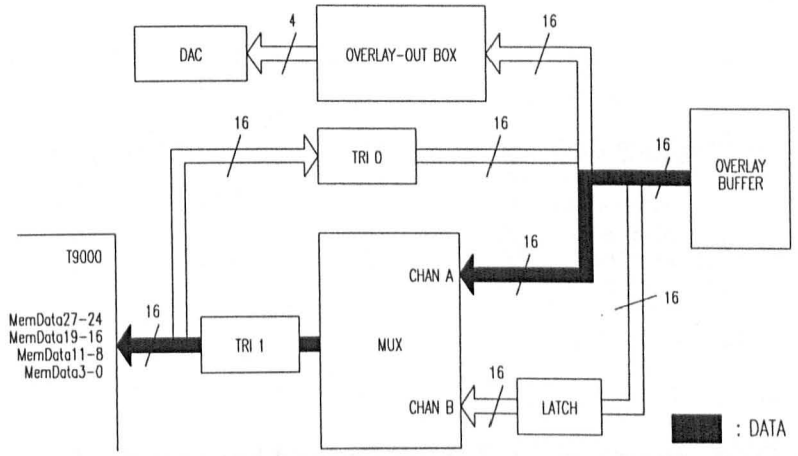


Figure 3.18 Data Path of the Overlay Memory When the T9000 Reads the Overlay Memory outside the Bus Multiplexing Time

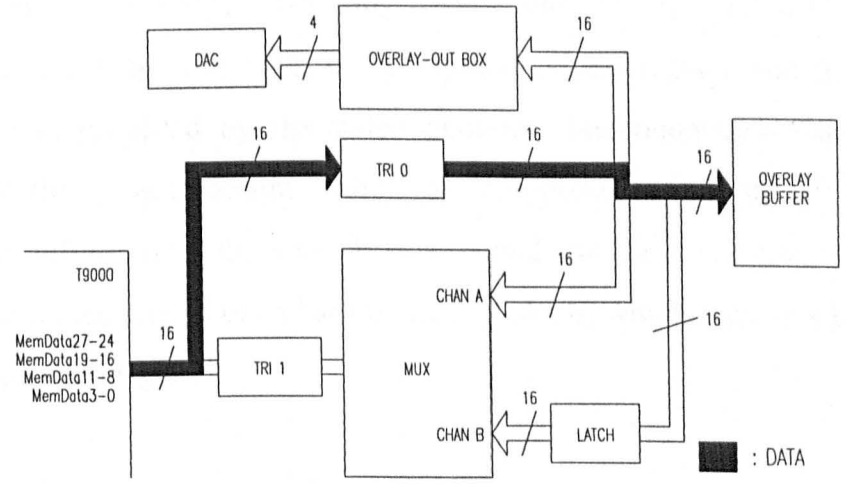


Figure 3.19 Data Path of the Overlay Memory in the T9000 State for Write Cycle

The local address bus, the output enable signal and the write enable signal for the overlay buffer memory are those used for the frame buffer memory. Figure 3.20 shows how these buses provided for these two memories.

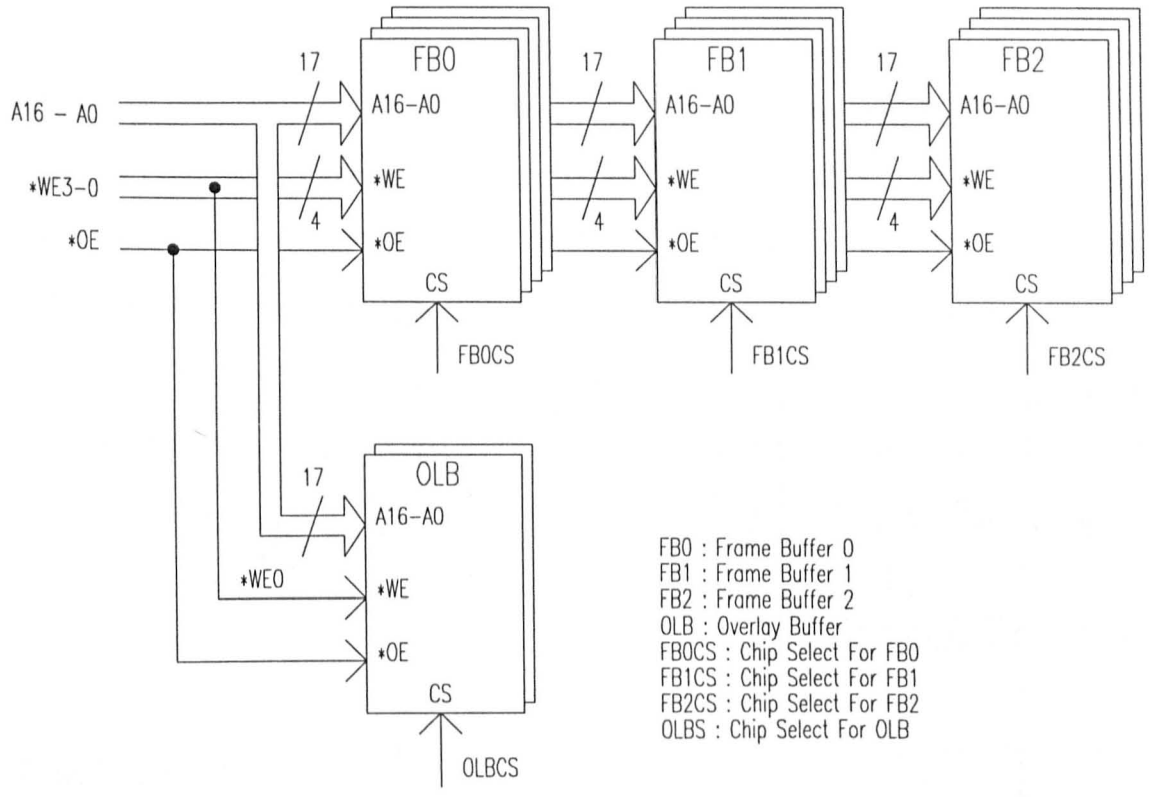


Figure 3.20 Local Address and Control Buses For The Frame and Overlay Buffers

3.1.4 Video Timings

The operation of the image processing board, was closely related to the video timings, which are derived from the composite video signal, and a 2 × pixel clock, which is provided by the video decoder. The necessary video timings included in the board design were the composite sync signal, the field identification information, the vertical sync signal and the horizontal sync signal. The 2 × pixel clock was chosen for the system clock, which was synchronised to the incoming video signal.

3.1.4.1 Composite Sync and Field Identification Extraction

The LM1881 video sync separator [31] extracts timing information including composite and vertical sync, burst/back porch timing, and odd/even field information from the standard negative going sync NTSC, PAL and SECAM video signals. This chip was only used to extract composite sync and odd/even field information. The composite sync was fed to the RAMDAC Bt481A [32] to generate a composite video signal for display, and the odd/even field signal was used as one of the address lines for storing a frame in a non-interlace format. Figure 3.21 shows the circuitry of this sync separator.

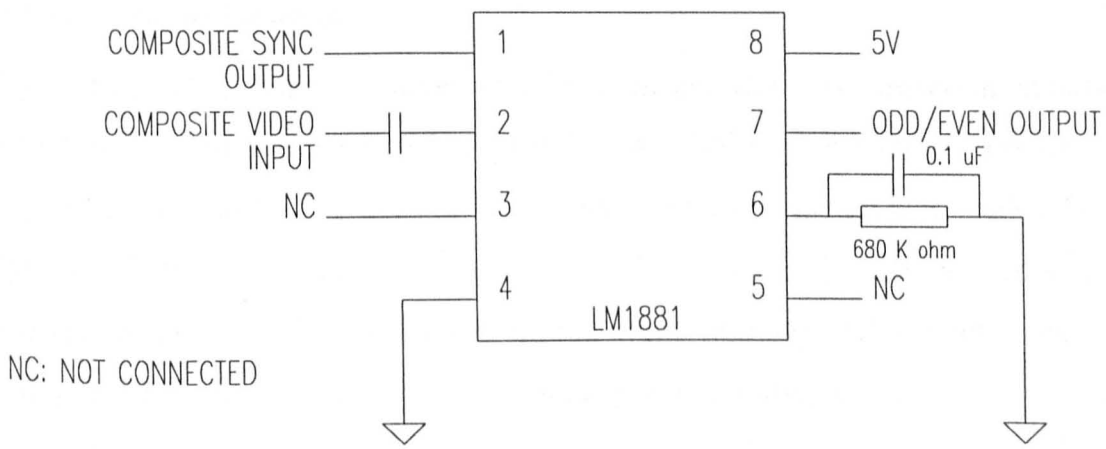


Figure 3.21 LM1881 Video Sync Separator Circuit

3.1.4.2 Frame Sync Generation

The TMC22071 video digitizer and the LM1881 video sync separator do not give a frame sync signal. However, a simple circuit with vertical sync and odd/even field signal can generate it. Figure 3.22 shows how this was achieved. The frame sync was used to signal to the T9000 transputer when the start of a frame occurred, so that the T9000 transputer could be synchronised to the incoming video signal.

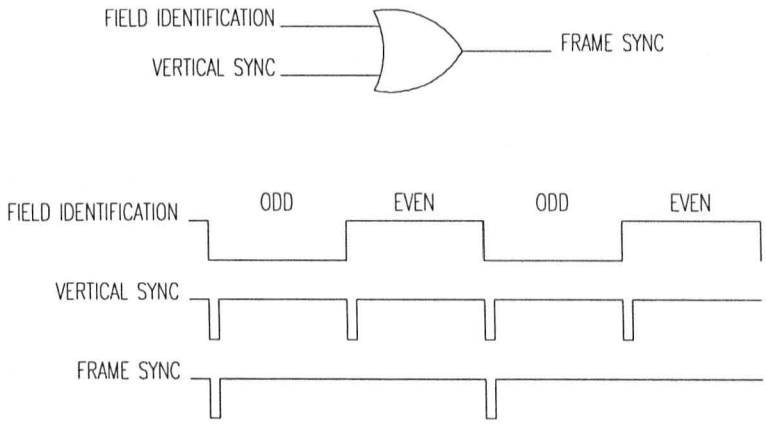


Figure 3.22 Frame Sync Generator

3.1.4.3 Frame Timings

The TMC22071 video digitizer can be configured to the different standards which are defined in Table 3.1. PAL-601 was chosen to set the dimension of a captured frame to be 512 lines × 640 pixels. With the line rate at 15.625 kHz, the time taken to store a picture consisting of 512 lines is 32.768 ms. This frame storage is managed by horizontal timing logic and vertical timing logic. The following two sections detail the functioning of this timing logic.

Standard	Field Rate (Hz)	Line Rate (kHz)	Pixel Rate (Mpps)	PXCK Frequency (MHz)	Pixels Per Line
NTSC	59.94	15.734264	12.2727+	24.54+	780
NTSC-601	59.94	15.734264	13.50	27.0	858
PAL_A	50.00	15.625	14.75	29.5	944
PAL_B	50.00	15.625	15.00	30.0	960
PAL-601	50.00	15.625	13.50	27.0	864

Table 3.1 TMC22071 Timing Options

3.1.4.3.1 Horizontal Timing Logic

The main components in the horizontal timing logic, were the horizontal state machine and a pixel counter. The horizontal state machine was used to find the start of a line, decide where to start capturing pixels in a line, where to stop capturing, and indicate that pixels were being captured. The pixel counter and

some logic gates assisted the operation of the horizontal state machine. The most significant eight bits of the counter were taken as part of the address lines for the frame buffer memory and the overlay buffer memory. Figure 3.23 shows the flow-chart of the horizontal state machine and Figure 3.24 is an overview of the horizontal timing logic. The first pixel was defined as the 42nd pixel of a line and the last pixel was defined as the (42 + 639)th pixel of the same line. The text design files for the horizontal state machine and pixel counter are shown in Appendix B [pp148-151].

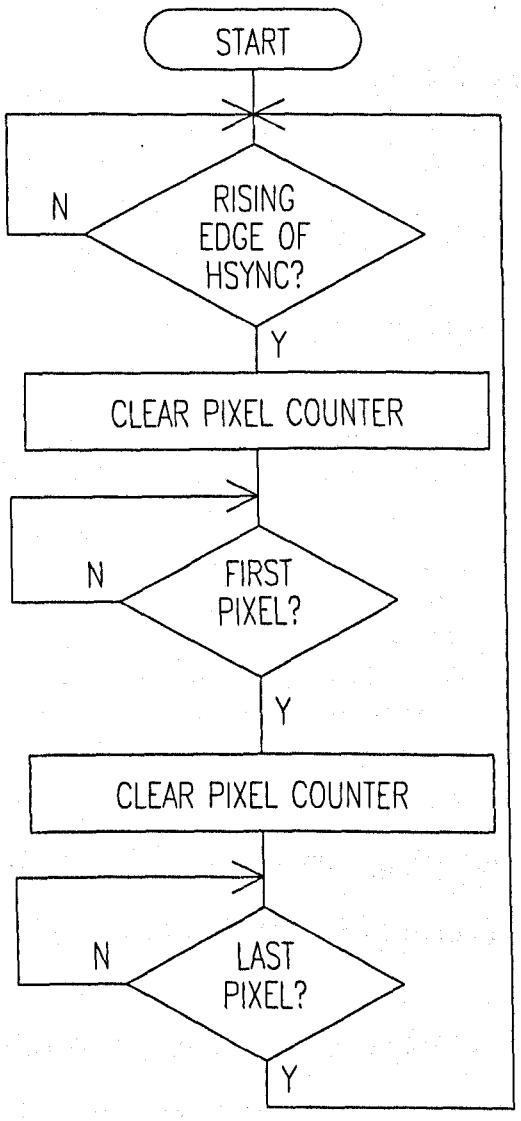


Figure 3.23 Flow-Chart of The Horizontal State Machine

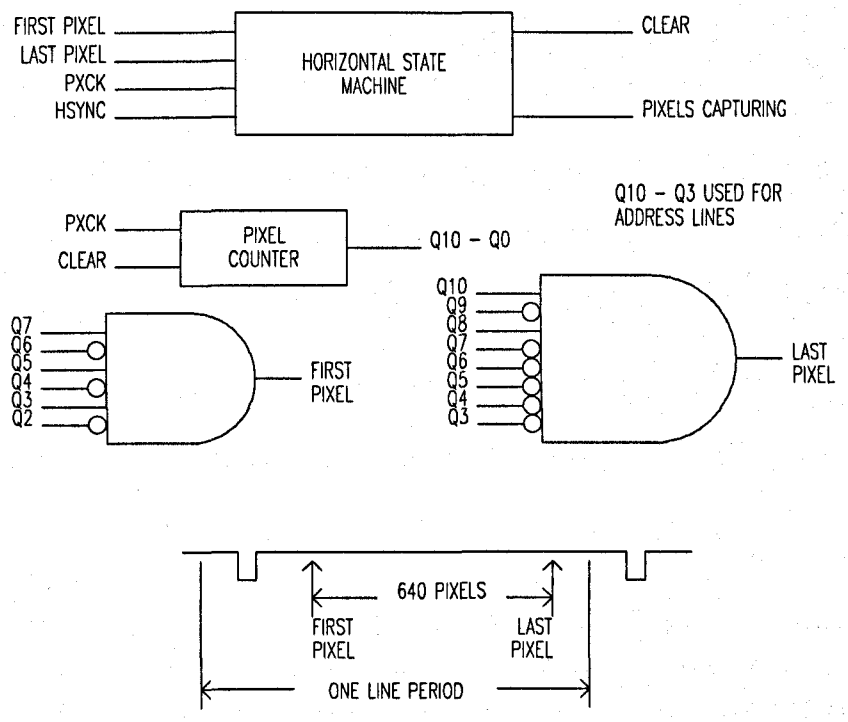


Figure 3.24 Horizontal Timing Logic

3.1.4.3.2 Vertical Timing Logic

The design of the vertical timing logic is similar to that of the horizontal timing logic. Instead of handling individual pixels, the vertical timing logic controlled the lines of pixels. The main components in the vertical timing logic were the vertical state machine and a line counter. The vertical state machine was used to find the start of a line of a field, decide where to start capturing lines in a field, where to stop capturing and indicate that lines were being captured. The line counter was driven by a system clock with an enable signal, so that one vertical sync increments the counter value by one. The eight bits of the counter were all taken as part of the address lines for frame buffer memory and the overlay buffer memory. Figure 3.25 shows the flow-chart of the vertical state machine, and Figure 3.26 shows the vertical timing logic. The first line was defined as the 35th line of a field and the last line was defined as the (35 + 255)th line of the same field. The text design files for the vertical state machine and line counter are shown in Appendix B [pp151-153].

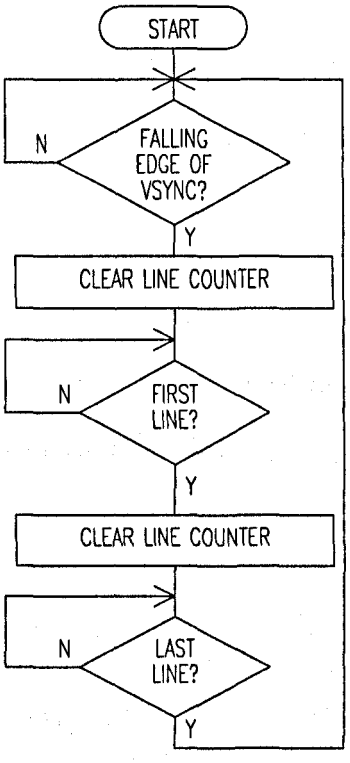


Figure 3.25 Flow-Chart of The Vertical State Machine

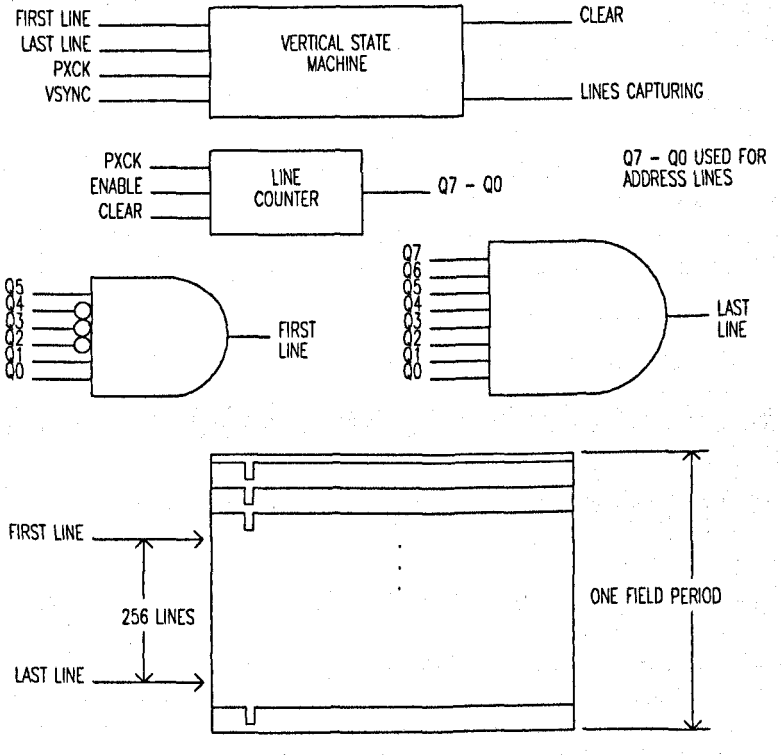


Figure 3.26 Vertical Timing Logic

3.1.4.4 Blank Signal Generation

The RAMDAC Bt481A has a pin called BLANK. When this pin is a logical zero, the pixel and overlay inputs are ignored. This signal can be obtained from the Logical AND of the pixels capturing signal and the lines capturing signal. When both capturing signals are active or logically high, the output of the AND gate is high and hence the pixel and overlay inputs are displayed; on the other hand, when either of the capturing signals is inactive, or logically low, the output of the AND gate is logically low, and hence black is displayed.

3.1.5 Frame Capture Subsystem

The TMC22071 video digitizer was a main component in the frame capture subsystem. It converts standard baseband composite NTSC, or PAL video, into 8-bit digital composite video data. It has a control register for configuration. This register is accessed through a microprocessor interface. The timing diagrams of the data read sequence and the data write sequence on the interface, are shown in Figure 3.27 and Figure 3.28 respectively. R/*W and A0 are latched by the TMC22071 video digitizer on the falling edge of *CS, and data input D0 is latched on the rising edge of *CS. Data read from D0 is enabled by the falling edge of *CS and disabled by the rising edge of *CS.

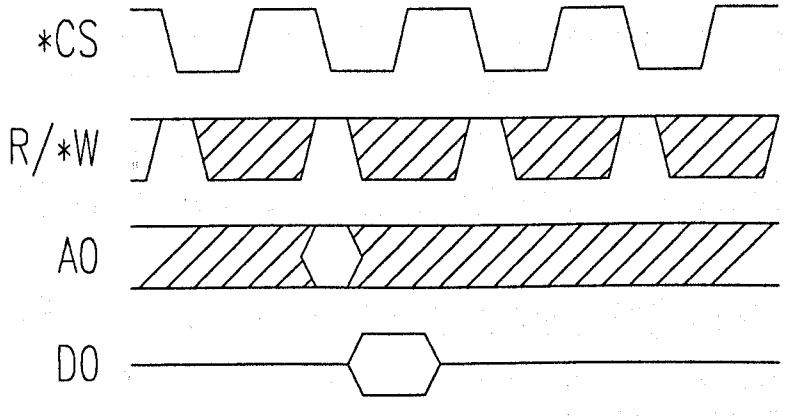


Figure 3.27 Data Read Sequence of TMC22071

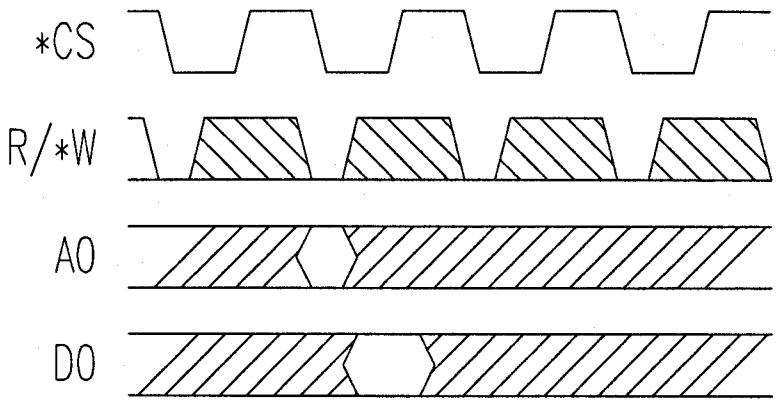


Figure 3.28 Data Write Sequence of TMC22071

Further, Figure 3.29 and Figure 3.30 respectively show the timing diagrams of the read cycle and the write cycle of Bank 3 of the T9000 transputer, they were purposely set to match the timings required for the microprocessor interface of the TMC22071.

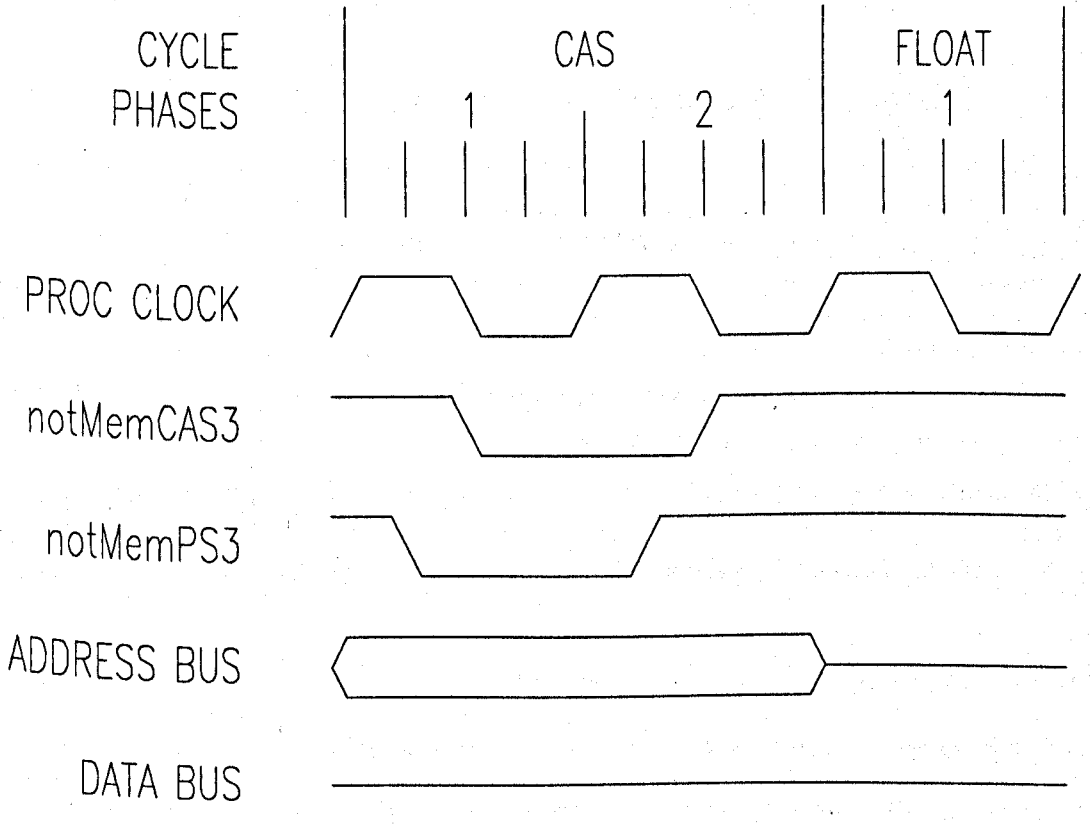


Figure 3.29 Timing Diagram of Bank 3 Read Cycle

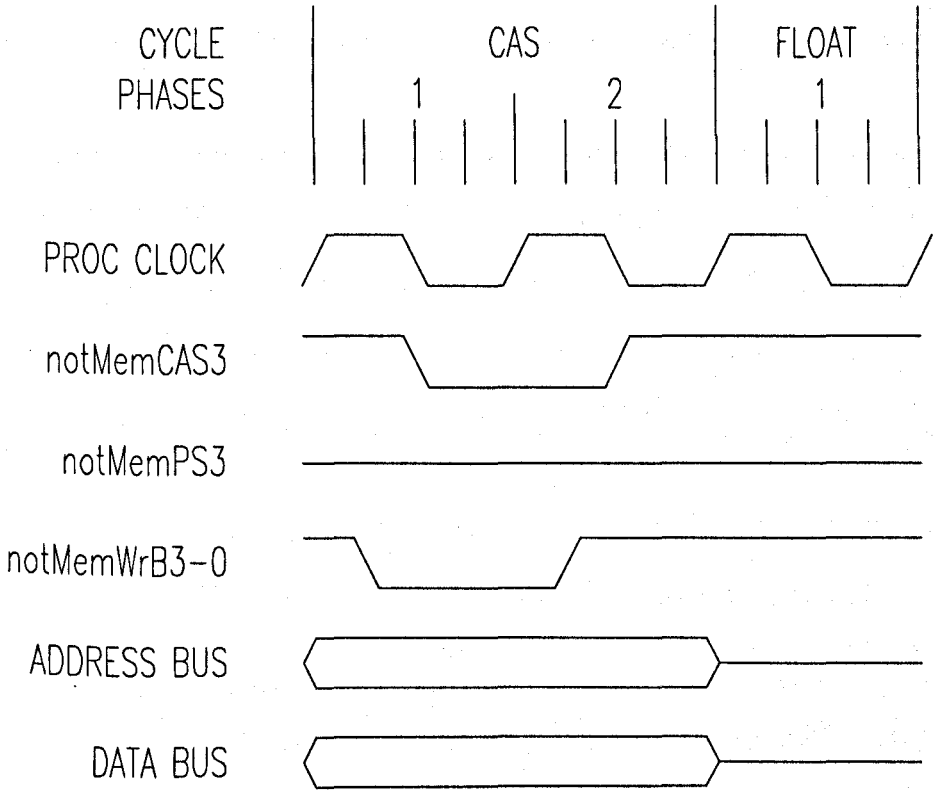


Figure 3.30 Timing Diagram of Bank 3 Write Cycle

Signals generated for the interface are derived by the following logic equations.

!*RESET = MemAddr31 & MemAddr29 & !MemAddr28 & MemAddr23
& MemAddr22 & !MemAddr21 & !MemAddr20 & !notMemCAS3

D0 = MemData0

A0 = MemAddr31 & MemAddr29 & !MemAddr28 & MemAddr23
& !MemAddr22 & MemAddr21 & !MemAddr20 & MemAddr16

!*CS = MemAddr31 & MemAddr29 & !MemAddr28 & MemAddr23
& !MemAddr22 & MemAddr21 & !MemAddr20 & !notMemCAS3

R/*W = notMemWrB0

There were three frame buffers in the system. A 3-bit register, called the capture register, determined which buffer would store the digitised image data. The outputs of the register are logically connected to the chip selects of the frame buffers. Bit 0, 1 and 2 are connected to the chip selects of frame buffer 0, 1 and 2

respectively. When the capture register stored a value of zero, none of the frame buffers were selected, and hence the data in the three buffers remained the same.

The video digitizer outputs 8-bit digital image data; however, the frame buffer memory was 32-bit. Five 8-bit latches were grouped together in such a way that four consecutive digital image pixels were accumulated before going into memory. Figure 3.31 shows the whole picture of the design. Image pixels were continuously clocked into the latches sequentially by the outputs of the 2-line to 4-line decoder, and hence a 32-bit data was obtained. Latch 4 was purposely added to buffer the least significant byte of the 32-bit data, so that this byte would not be overwritten before it was copied into memory.

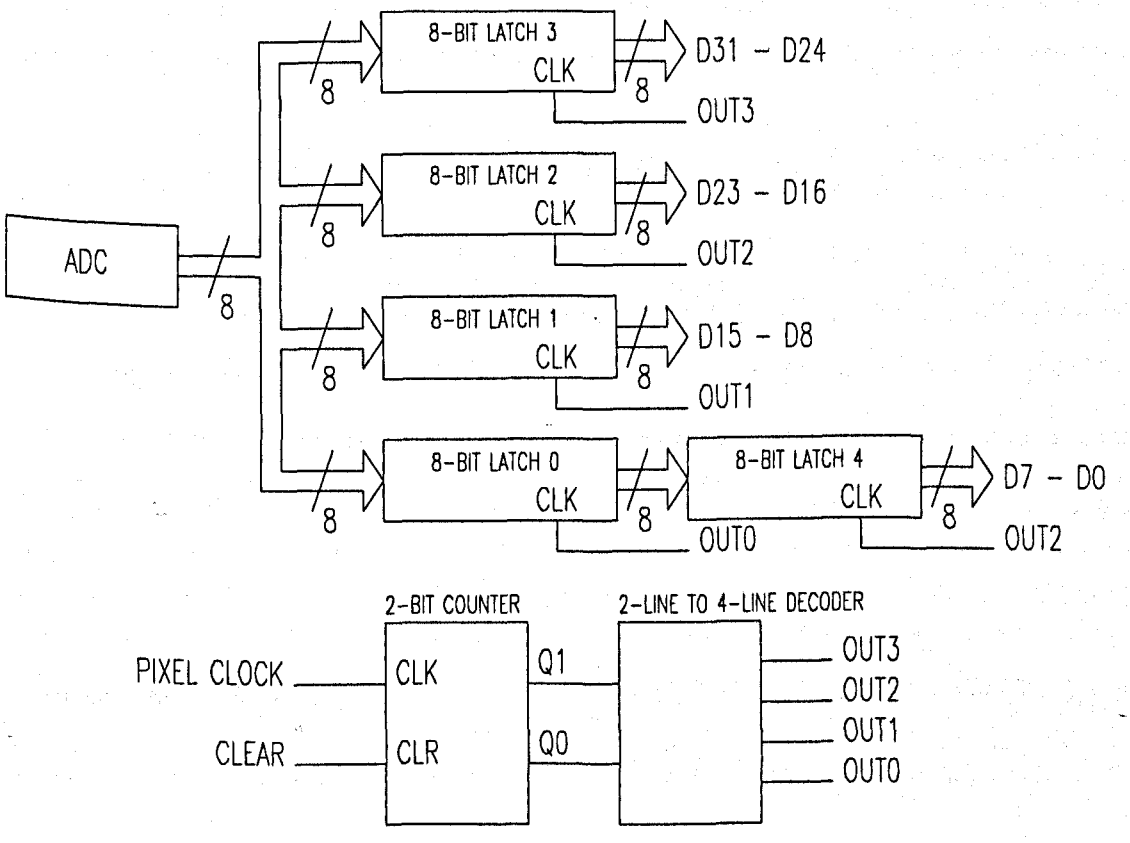


Figure 3.31 Architecture of Pixel-In Box

Single-frame capture mode and multiple-frame capture mode can be achieved by programming. For the former, when a frame sync signal is detected a frame buffer value is written by the T9000 transputer to the capture register for buffer

selection. Afterwards, the next frame sync signal is checked. When it is detected, a value of zero is written by the T9000 transputer to the capture register, so as to stop further image data being stored into buffers. Finally, a complete frame is stored.

For multiple-frame capture mode, the procedure is just a repeated procedure prepared for the single-frame capture mode but with a little modification. When a frame sync signal is detected, a value is written to the capture register for buffer selection. Afterwards, another value is written to the capture register when the next frame sync signal has just arrived. If the above steps are repeated continuously, multiple-frame capturing is achieved.

3.1.6 Frame Display Subsystem

The RAMDAC Bt481A is the main component in the frame display subsystem. It supports 8-bit pseudo-colour format, generating RS-343-compatible video signals into a doubly-terminated load. It also supports a standard MPU bus interface, allowing the T9000 transputer direct access to the colour palette RAM, the overlay colour registers and command register A, and indirect access to the command register B. Figure 3.32 shows the timing diagram of the read and write cycles of the MPU interface. This interface was accessed by the T9000 transputer through Bank 3.

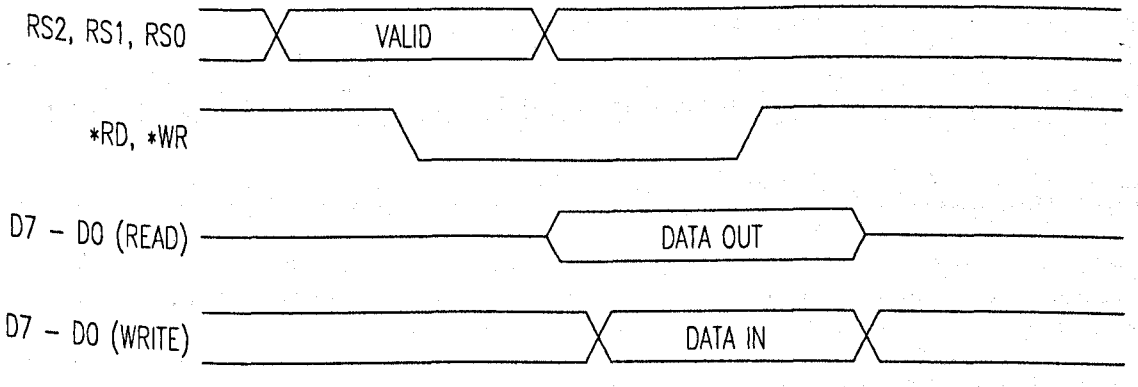


Figure 3.32 MPU Read/Write Timing of Bt481A

Signals generated for the interface were derived from the following logic equations.

RS0 = MemAddr31 & MemAddr29 & !MemAddr28 & MemAddr23
& !MemAddr22 & MemAddr21 & MemAddr20 & MemAddr16

RS1 = MemAddr31 & MemAddr29 & !MemAddr28 & MemAddr23
& !MemAddr22 & MemAddr21 & MemAddr20 & MemAddr17

RS2 = MemAddr31 & MemAddr29 & !MemAddr28 & MemAddr23
& !MemAddr22 & MemAddr21 & MemAddr20 & MemAddr18

*RD = MemAddr31 & MemAddr29 & !MemAddr28 & MemAddr23
& !MemAddr22 & MemAddr21 & MemAddr20 & notMemPS3

*WR = MemAddr31 & MemAddr29 & !MemAddr28 & MemAddr23
& !MemAddr22 & MemAddr21 & MemAddr20 & notMemWrB0

D0 - D7 = MemData0 - MemData7

A 3-bit register, called the display register, determined which one of the three buffers was going to be displayed. Bit 0, 1 and 2 were logically connected to the chip select of buffer 0, 1 and 2 respectively. If none of the bits are set high, nothing is displayed on the monitor. On the other hand, if more than one bit was loaded high, local data bus contention occurs in the frame buffer memory. In normal operation, only one bit was set high so that only one buffer was selected.

The RAMDAC Bt481A was not compatible with the frame buffer memory with regard to the data width for pixel data. Five 8-bit latches incorporated with a multiplexer and a 2-bit counter, were used to interface them together. As shown in Figure 3.33, 32-bit data from the frame buffer memory was clocked into the latch 0, latch 1, latch 2 and latch 3. The 2-bit counter driven by the pixel clock operates the multiplexer so that 32-bit data is multiplexed into 8-bit data for the RAMDAC Bt481A. Latch 4 was added to buffer the most significant byte of data, so that it was not overwritten before it was copied to the RAMDAC.

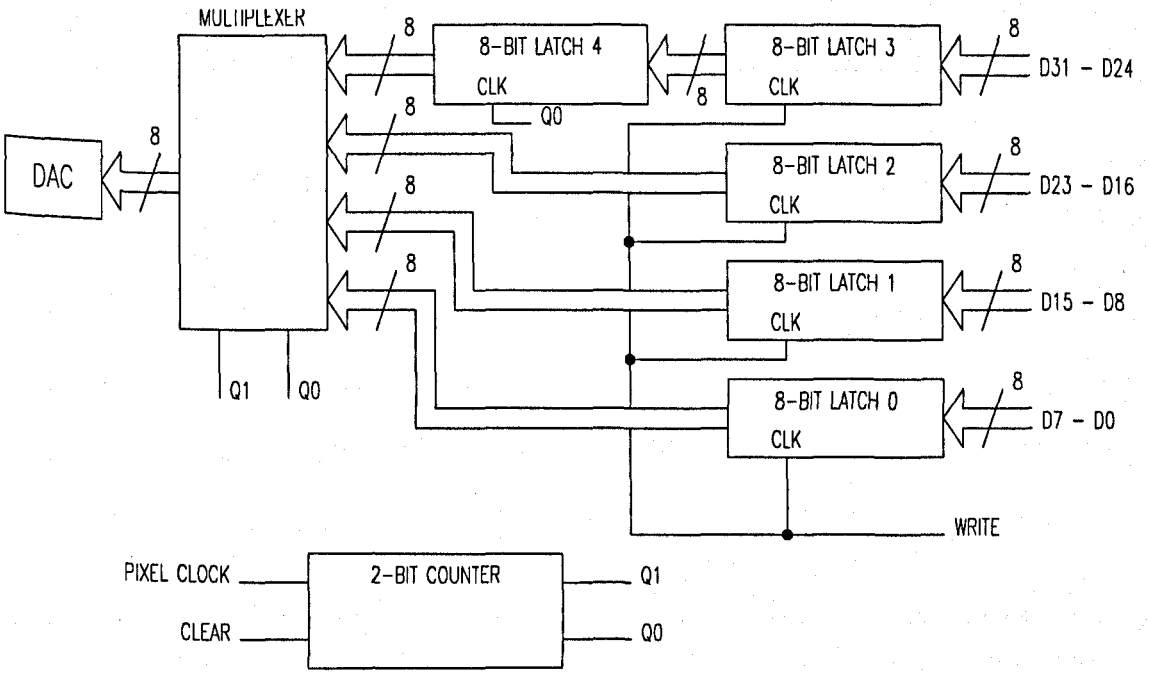


Figure 3.33 Architecture of The Pixel-Out Box

The overlay part of the RAMDAC also required a circuit to multiplex 16-bit data into 4-bit data, as the data width of the overlay input was not the same as that of the overlay buffer memory. The design of the circuit is the same as that applied to the pixel input, except for the width of data.

3.1.7 DS Link Interface

The first generation of transputers, such as the T2, T4, and T8 transputers [33], made use of OS-Links [34] for point-to-point communication between devices. However, the T9000 transputer adopts DS-Link [35] technology for its communication links to support higher bandwidth serial communication between devices, up to 100 Mbits/s, and also supports virtual channels. Each physical link consists of four wires, two in each direction, one carrying data and one carrying a strobe. The links are therefore referred to as data-strobe links or DS-Links.

The dedicated image processing board was not designed to interface to the PC bus directly. Instead, it was designed to provide a standard control and data DS-Link interface, and to make connections to the host computer through the IMS

B108 motherboard. With regard to the IMS B108, it provides standard control and data link interface to the T9000 network, either internal or external. For convenience in making the prototype, the image processing board was designed to interface to the IMS B108 externally. Since the IMS B108 made use of the fast 41-series transceivers [36] by AT&T for external connections, these transceivers were also selected to buffer the DS-Link interface in the image processing board.

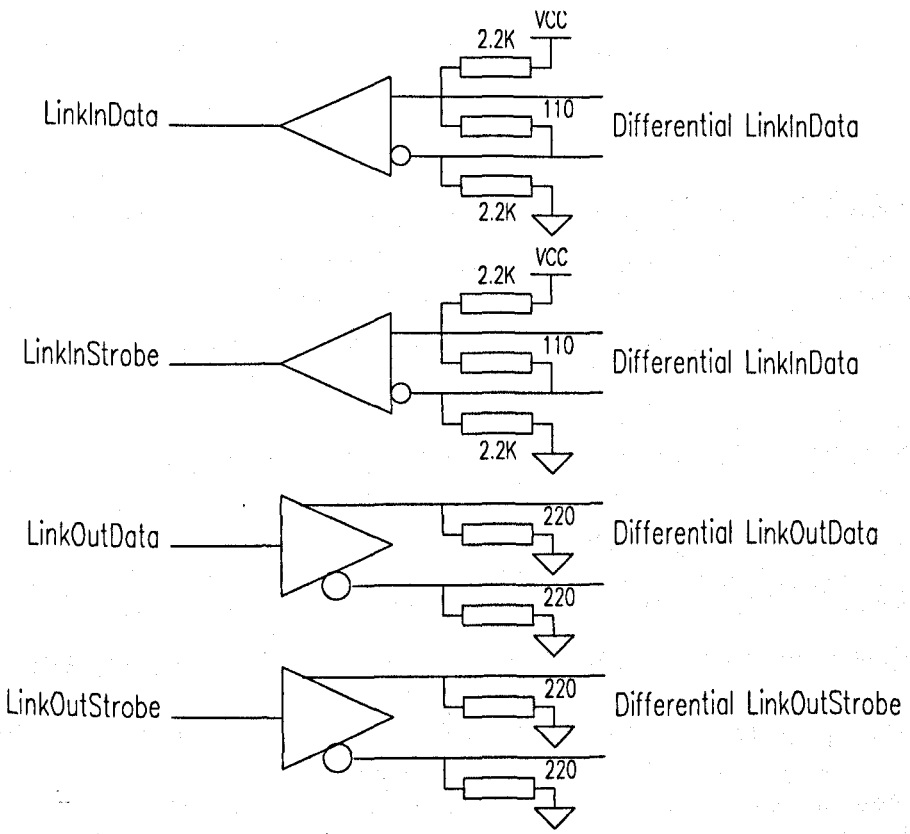


Figure 3.34 A Pair of Buffered DS-Links

Only the 1141MK dual differential transceivers were available on the market during the image processing board development, so they were chosen. There are no built-in termination resistors in this package, so external resistors of 220 Ω were needed to connect between each driver pin and ground, and resistors of 110 Ω were used between the differential pair in each receiver. Figure 3.34 is an example of a pair of buffered DS-Links. Pull-up and pull-down 2.2k Ω resistors are used to avoid inputs floating when they are not connected. The differential

signals go to the 10-way modular I/O connectors designed by Harting [37] for external connections. Figure 3.35 shows the DS-Link connector pinout.

Signal	Pin	Pin	Signal
DoutPlus	1 a	2 a	DoutMinus
SoutPlus	1 b	2 b	SoutMinus
Reserved	1 c	2 c	Reserved
SinMinus	1 d	2 d	SinPlus
DinMinus	1 e	2 e	DinPlus

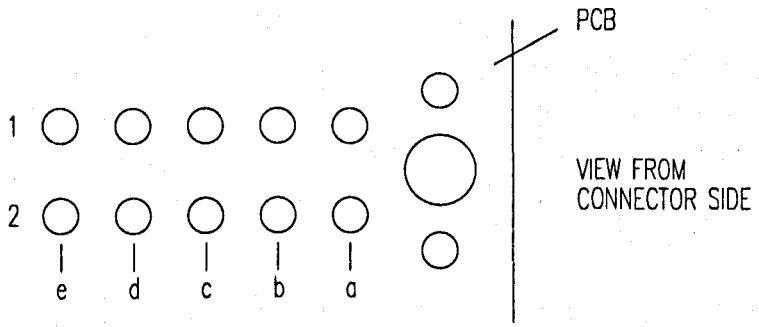


Figure 3.35 DS-Link Connector Pinout

3.1.8 Control System

The first generation transputer's system of reset, analyse and error, booting from link, peek and poke, and memory interface programming [38] is replaced by a system of control links on the T9000 transputer. The control links should be connected into a control network which is completely separate from the normal data link network. The T9000 transputer has two control links, Clink0 and Clink1, which are effectively the up and down control links, allowing them to be daisy-chained.

The Up/Down ports for reset signals are employed in the IMS B108 and the dedicated image processing board. The use of notResetIn and notResetOut allows the host PC to reset the external hardware under software control. In order to make sure of a complete reset chain, the reset acknowledgement signal must be

fed back to the master IMS B108, which is the first element in a daisy-chained network. Figure 3.36 shows the Up/Down ports for the reset signals in the image processing board. The jumper J1 is set at position 2 if the board is at the end of a daisy chain; otherwise it is set at position 1.

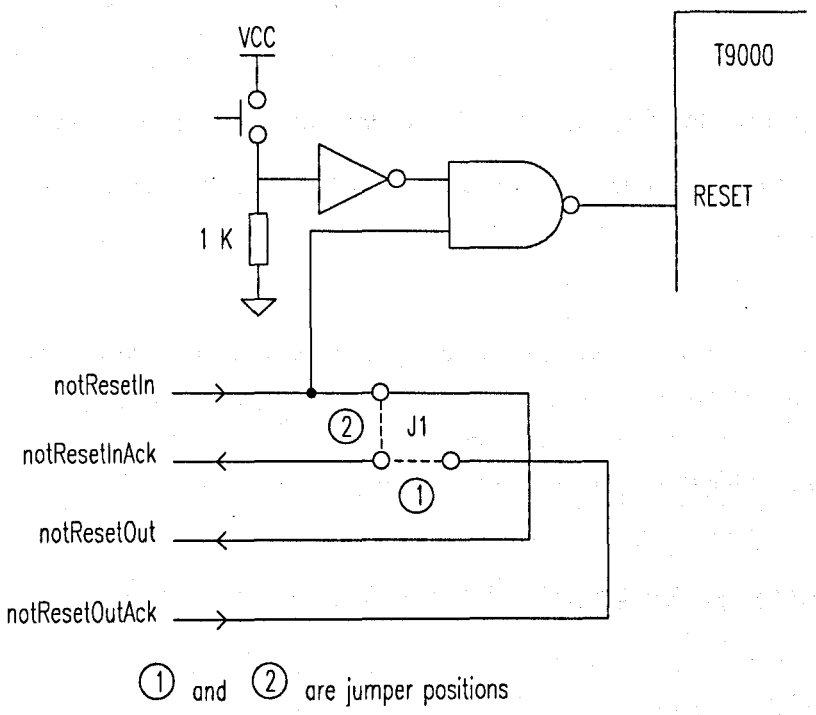


Figure 3.36 Up/Down Ports for Reset Signals

3.1.9 The Topologies of The System Synchronisation

In the Dedicated Image Processing Board, there are a number of subsystems requiring synchronisation to other signals. For example, the pixel demultiplexer was synchronised to the timing of the ADC state; the bus multiplexing mode switching was synchronised to the read or write cycle of the T9000 transputer when the read or write cycle for the buffer memory was operating, the frame buffer memory or the overlay buffer memory accessed by the T9000 transputer was synchronised to the system state machine and image capture to the buffer memory was synchronised to the incoming video signal. The following sections detail how the synchronisation was achieved.

3.1.9.1 Pixel Multiplexer and Demultiplexer

The pixel demultiplexer in the frame capture subsystem was indirectly driven by the pixel clock. The order of pixels at the output of the demultiplexer was dependent on the time to reset the 2-bit counter, which is shown in Figure 3.31. A state machine was dedicated to reset the counter such that four consecutive pixels, in the correct order, appear at the output of the demultiplexer during the ADC state and the next four consecutive pixels, in the correct order, appear during the next ADC state. Thus, 32-bit pixel data was available for memory storage in the allocated time.

For the pixel multiplexer in the image display subsystem, a slightly different approach was used. The sequence of pixels clocked out from the pixel multiplexer depends on the time taken to reset the 2-bit counter, which is shown in Figure 3.33. Another state machine was assigned to reset the counter such that four bytes of data are multiplexed in an order with the least significant byte going first, and most significant byte going last. Thus, a sequence of 8-bit pixel data was available for the RAMDAC pixel input port.

3.1.9.2 Bus Multiplexing Mode Switching

There are several internal buses for the frame buffer memory and the overlay buffer memory. These internal buses are supported by the multiple bus interface so the frame capture subsystem, the frame display subsystem, and the T9000 transputer could access the buffer memory through time-division-multiplexing mode. In this mode, wait states were inserted to the T9000 transputer to maintain synchronisation. Actually, time-division-multiplexing mode was not needed when the frame capture subsystem and the frame display subsystem did not access the buffer memory, as only the T9000 transputer used the internal buses. Although the time-division-multiplexing mode could be applied all the time, in order to reduce the buffer memory accessing time of the T9000 transputer, this mode was only applied when the frame capture subsystem, and the frame display

subsystem were active as wait states were not required outside the time of bus multiplexing.

If bus multiplexing mode was switched whilst the T9000 transputer was accessing the buffer memory, it was likely that the read or write cycle would be corrupted. In order to avoid this corruption, a state machine was added to check whether the T9000 transputer was accessing the buffer memory when it was the time to switch the bus multiplexing mode. If it was the case, the switching time was delayed until the completion of the current read or write cycle. The following shows the text design file of the state machine written in AHDL [39].

```
SUBDESIGN vhdvsta
(
 reset, pxck, t9nras1, vhdv : INPUT;  % t9nras1 showing chip enable %
 sync_vhdv                  : OUTPUT;
)
VARIABLE
 ss: MACHINE OF BITS (sync_vhdv)
    WITH STATES (s0 = b"0", s1 = b"1");
BEGIN
 ss.clk  = pxck;
 ss.reset = reset;
 CASE ss IS
   WHEN s0 =>
    IF vhdv & t9nras1 THEN
     ss = s1;
    ELSE
     ss = s0;
    END IF;
   WHEN s1 =>
    IF !vhdv & t9nras1 THEN
     ss = s0;
    ELSE
     ss = s1;
    END IF;
 END CASE;
END;
```

In this file, the **vhdv** was a signal to switch the bus multiplexing mode on or off, and it was not synchronous to the T9000 transputer's read or write cycle. The **t9nras1** was a signal to indicate the presence of the T9000 transputer's read or write cycle and it was checked by the state machine when it was time to switch the bus multiplexing mode. The **sync_vhdv** was the output of the state machine,

it was synchronous to the T9000 transputer's read or write cycle and used to switch the bus multiplexing mode.

3.1.9.3 T9000 Accessing Buffer Memory

With reference to the system state machine, shown in Figure 3.1, there are four defined states for the on-board T9000 transputer. When the T9000 transputer accesses buffer memory, it is not synchronised to the system state machine so these accesses may not occur within the defined states. Obviously, the T9000 transputer cannot access buffer memory directly by normal read or write cycles. Additional hardware was required to complete the memory access. The role of this hardware, was to keep tracking the system state machine and provide appropriate control signals to the T9000 transputer and the buffer memory.

For a write cycle, the hardware provides the memory wait signal to the T9000 transputer, to suspend the current state of the cycle and, at the same time, finds the coming T9000 state to provide the 'write' control signal and chip select signal to the buffer memory at the appropriate time. After the 'write' control signal has been asserted, the memory wait signal is deasserted to let the T9000 transputer finish the rest of the cycle. Effectively, the write operation occurs during the T9000 state, relative to the buffer memory; however, it happened over a few system states relative to the T9000 transputer. During the above-mentioned T9000 state, the address and data were provided by the T9000 transputer, and the chip select signal and 'write' signal were supplied by the additional hardware. Figure 3.37 shows the relevant signals in the T9000 state for the write operation. The additional hardware is called **assistant**.

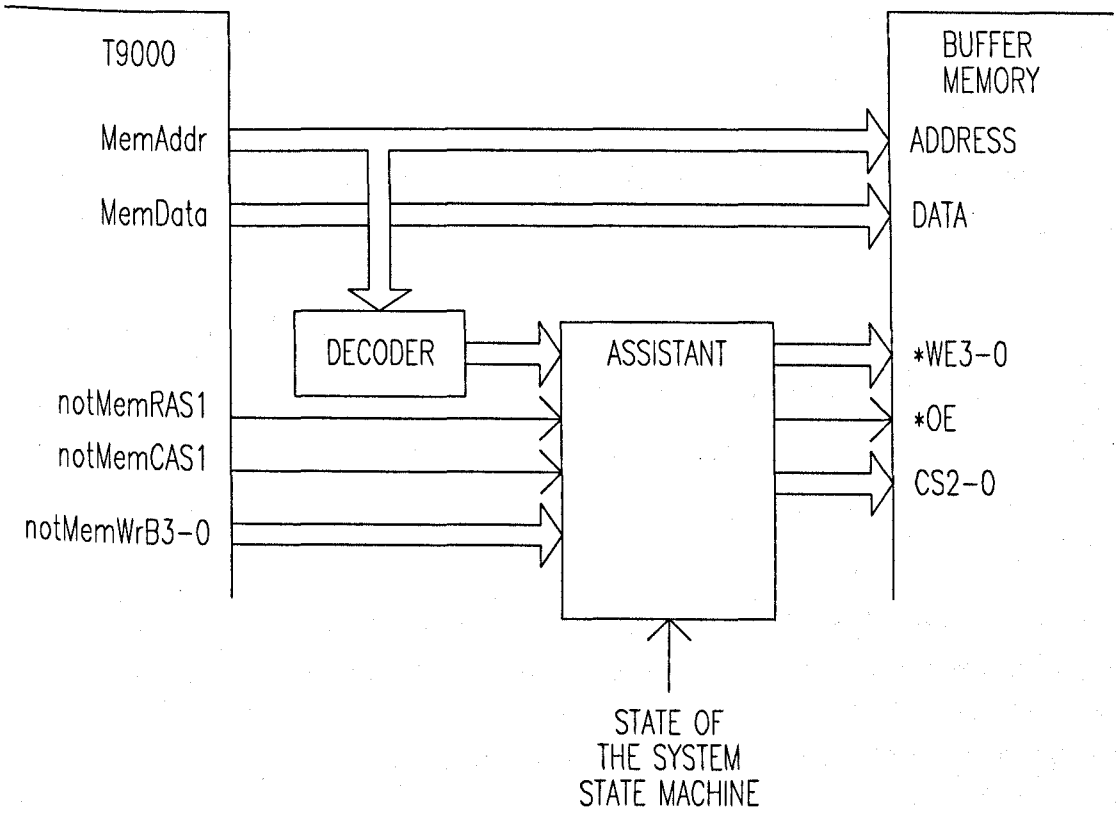


Figure 3.37 Relevant Signals in The T9000 State for Write Cycle

For a read cycle, this hardware also provided the memory wait signal to the T9000 transputer to extend the cycle. Once the T9000 state is found, the located data was latched from the buffer memory to a register, so that the data is still valid on the T9000 transputer's data bus before the conclusion of the read cycle. The memory wait signal was set low after the data had been latched. Effectively, the read cycle occurred during the T9000 state, relative to the buffer memory; however it happened over a few system states relative to the T9000 transputer. During the above-mentioned T9000 state, the address was provided by the T9000 transputer, and the rest of the signals were provided by the hardware. Figure 3.38 shows the relevant signals in the T9000 state for the read operation.

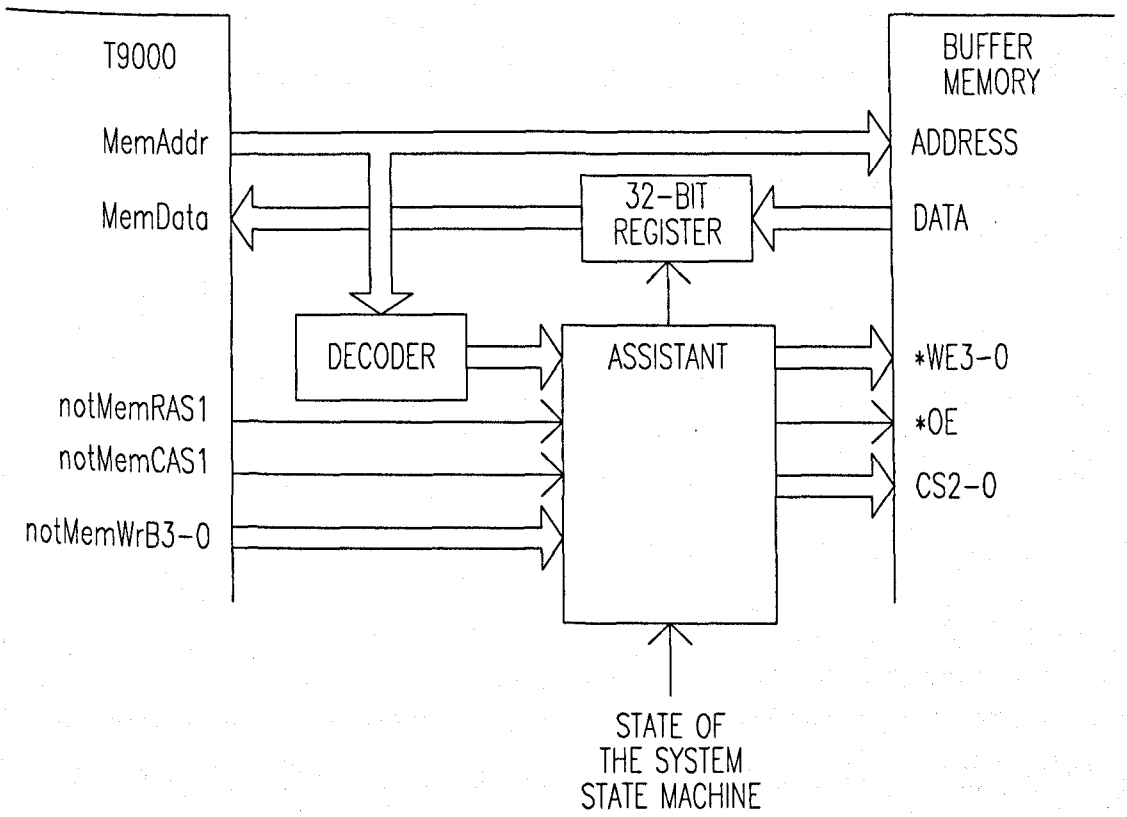


Figure 3.38 Relevant Signals in The T9000 State for Read Cycle

3.1.9.4 Frame Synchronisation

The frame capture subsystem was partially controlled by the T9000 transputer, which can determine when to start capturing an image into the frame buffer memory, and when to stop it. In order to capture a complete image into the buffer, the frame sync signal was fed to an input port, which was polled by the T9000 transputer, to check for the presence of the frame sync signal. For capturing a single frame, a frame buffer is enabled when the frame sync signal is detected, and is disabled by the time the next frame sync signal occurs.

3.2 PLD Design Verification

The development of the PLD was accompanied with a large number of simulation files for verification. Four of them were chosen to show the timings of the ADC writing to the frame buffer memory, the DAC reading from the frame buffer memory as well as the overlay buffer memory, the T9000 transputer writing to

the frame buffer memory and the T9000 transputer reading from the frame buffer memory. T9000 transputer access to the frame buffer memory occurred only during the bus multiplexing time.

3.2.1 Frame Capture

In Figure 3.39, **adcd[7..0]** represents the image data from the video digitizer, **buffa[16..0]** the address for the buffer memory, **buffnwr[3..0]** the write control signals for the buffer memory, and **buffd[31..0]** the data for the frame buffer memory.

The four pixels with values **E2**, **E3**, **E4** and **E5** from the video digitizer were continuously clocked into the latches and they appeared as a 32-bit word **E5E4E3E2** at the **buffd[31..0].** When this word appeared, the address **00306** generated by the address counter and the write control signals generated by the state machine were given at the appropriate time, which was around 1.85μs.

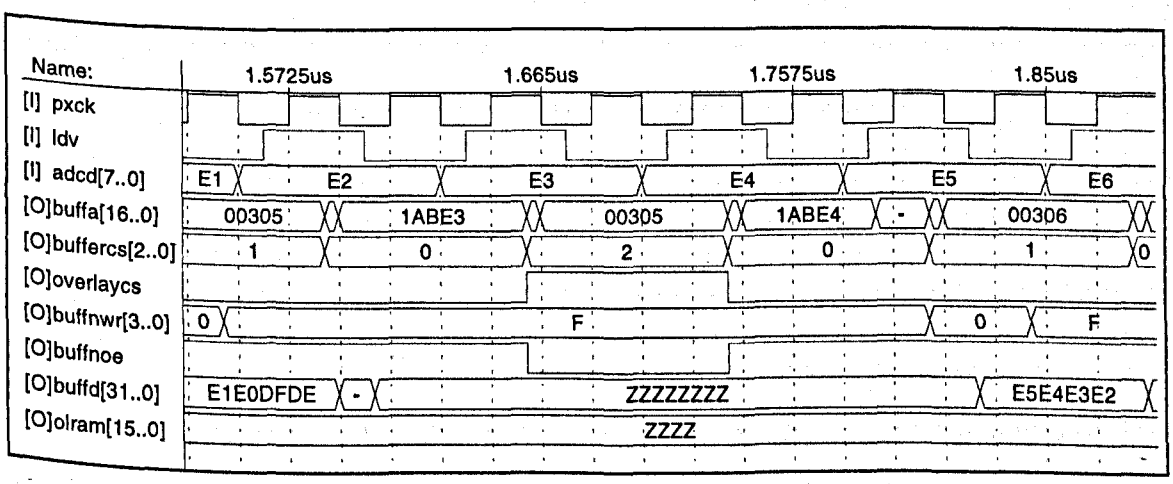


Figure 3.39 Frame Capture Verification

3.2.2 Frame Display

In Figure 3.40, **olram[15..0]** represents the data from the overlay buffer memory, **buffd[31..0]** the data from the frame buffer memory, **buffa[16..0]** the address for the buffer memory, **buffnoe** the output enable signal for the buffer memory,

bt481a_p[7..0] the data for the pixel input of the RAMDAC and **dacol[3..0]** the data for the overlay input of the RAMDAC.

The 16-bit data **ABD7** from the overlay buffer memory was read by providing the buffer address **00302** and setting **buffnoe** low. It was then multiplexed into 4-bit data in a sequence of **7**, **D**, **B**, and **A** for the overlay input. The 32-bit data **23456794** from the frame buffer memory was read by the same way, and it was multiplexed into 8-bit data in a sequence of **94**, **67**, **45**, and **23** for the pixel input. There was no discontinuity in providing data for the RAMDAC.

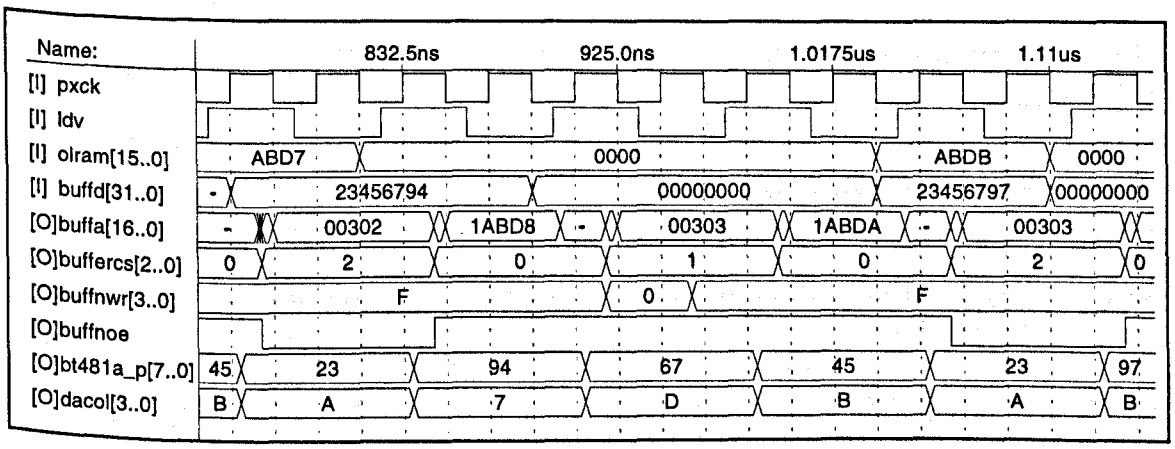


Figure 3.40 Frame Display Verification

3.2.3 T9000's Write Cycle

In Figure 3.41, **t9a31**, **t9a29**, **t9a28**, **t9a[23..20]** and **t9a[18..2]** represent the address lines of the T9000 transputer, **t9d[31..0]** the data from the T9000 transputer, **buffa[16..0]** the address for the buffer memory, **buffnwr[3..0]** the write control signals for the buffer memory, **buffd[31..0]** the data for the frame buffer memory, and **wait_state** the signal connected to the MemWait pin of the T9000 transputer.

The T9000 transputer started writing to the frame buffer memory when the logic level of **t9a31** changed from low to high. At that moment, the system state machine was in the ADC state, so **wait_state** was set high to suspend the write

cycle. The 32-bit word **12345678** from **t9d[31..0]** appeared at **buffd[31..0]** during the following T9000 state. In this state, the address **1ABF0** from **t9a[18..2]** was selected to occur at **buffa[16..0]** and the write signals **buffnwr[3..0]** were given by the state machine to complete a write operation.

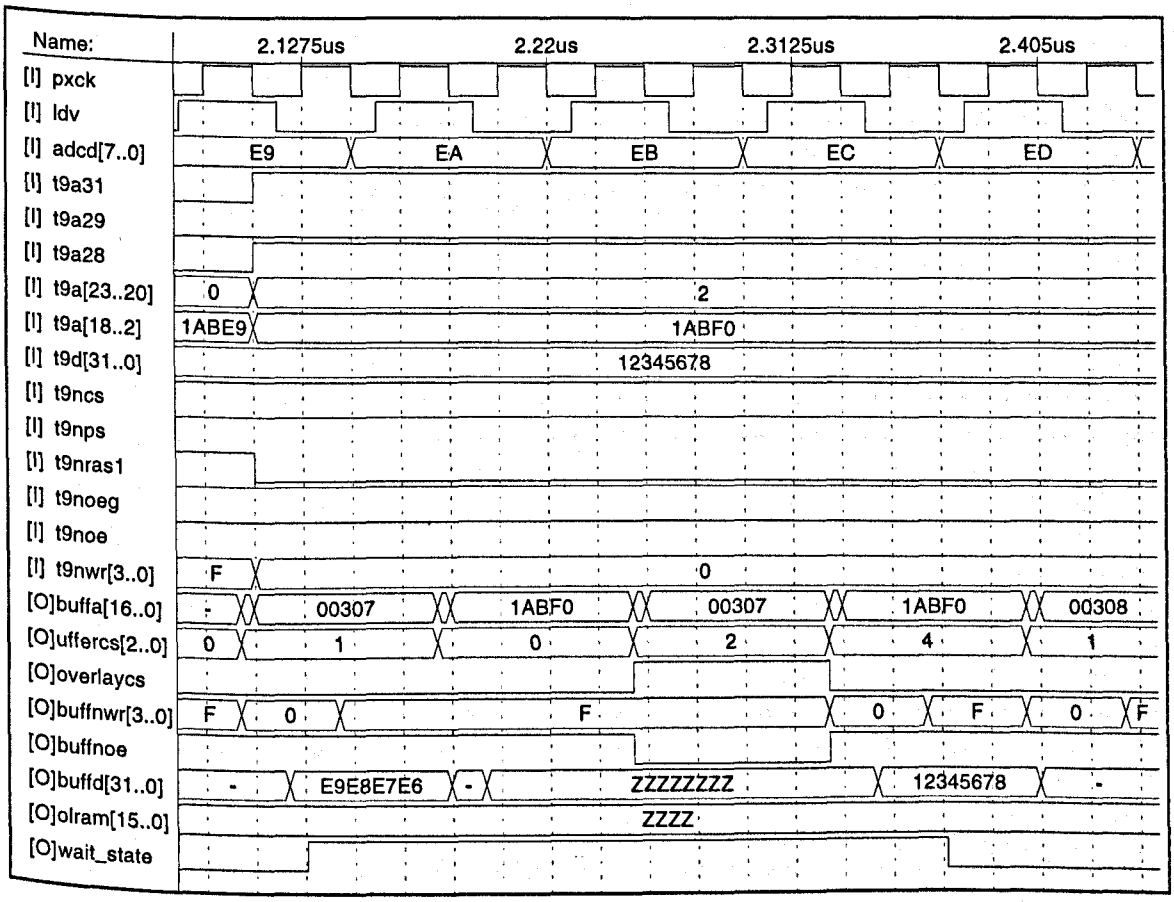


Figure 3.41 T9000's Write Cycle Verification

3.2.4 T9000's Read Cycle

In Figure 3.42, **buffd[31..0]** represents the data from the frame buffer memory, **t9a31**, **t9a29**, **t9a28**, **t9a[23..20]** and **t9a[18..2]** the address lines of the T9000 transputer, **buffnoe** the output enable signal for the buffer memory, **wait_state** the signal connected to the MemWait pin of the T9000 transputer, and **t9d[31..0]** the data for the T9000 transputer.

The T9000 transputer started reading from the frame buffer memory when the logic level of **t9a31** changed from low to high. At that time, the system state

machine was in the ADC state, so **wait_state** was set high to suspend the T9000 transputer read cycle. The addressed data **2345679F** from the frame buffer memory was latched a short time after the second falling edge of **buffnoe** during the T9000 state. Finally, the latched data **2345679F** appeared at **t9d[31..0]** until the end of the read cycle.

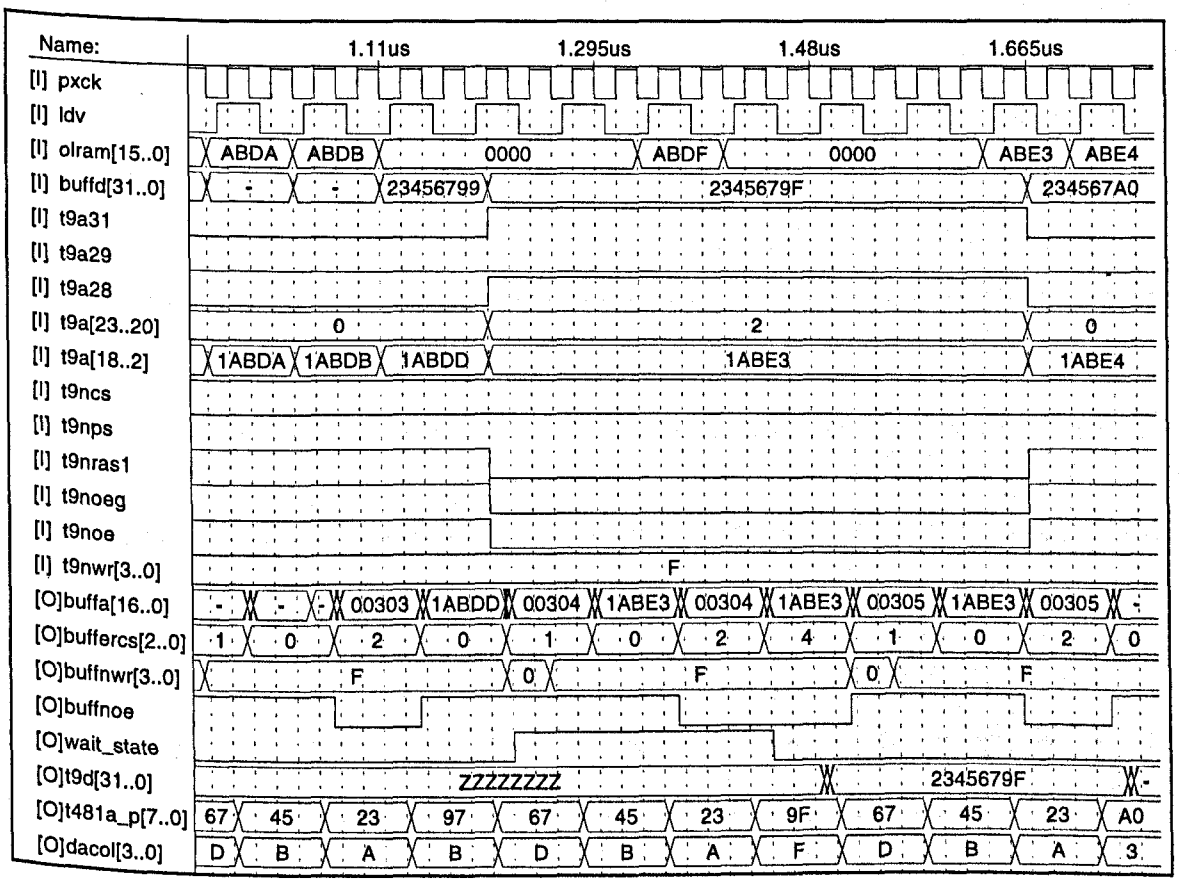


Figure 3.42 T9000's Read Cycle Verification

3.3 Prototype Construction

The prototype of the dedicated image processing board was constructed in two stages. The first stage was to make a small PCB for the T9000 transputer and the DS-Link connectors. The second stage was to populate the speedwire board with all the required components and connect them together.

3.3.1 T9000 Transputer Adaptor Board

The T9000 transputer was available in a 208 pin ceramic leaded chip carrier (CLCC) package. This package does not allow the T9000 transputer to be directly fixed on the speedwire board. A small PCB was made to extend the necessary signal lines of the T9000 transputer to the two connectors with pitch size compatible to the speedwire board. With this PCB fixed on to the speedwire board, the T9000 transputer could be connected to the on-board components.

The DS-Link connectors also do not fit to the speedwire board. It is necessary to make an adaptor for them. As DS-Links are intended to be used in electrically quiet environments, these connectors were mounted on the same PCB. This PCB was called T9000 transputer adaptor board and it is shown in Figure 3.43.

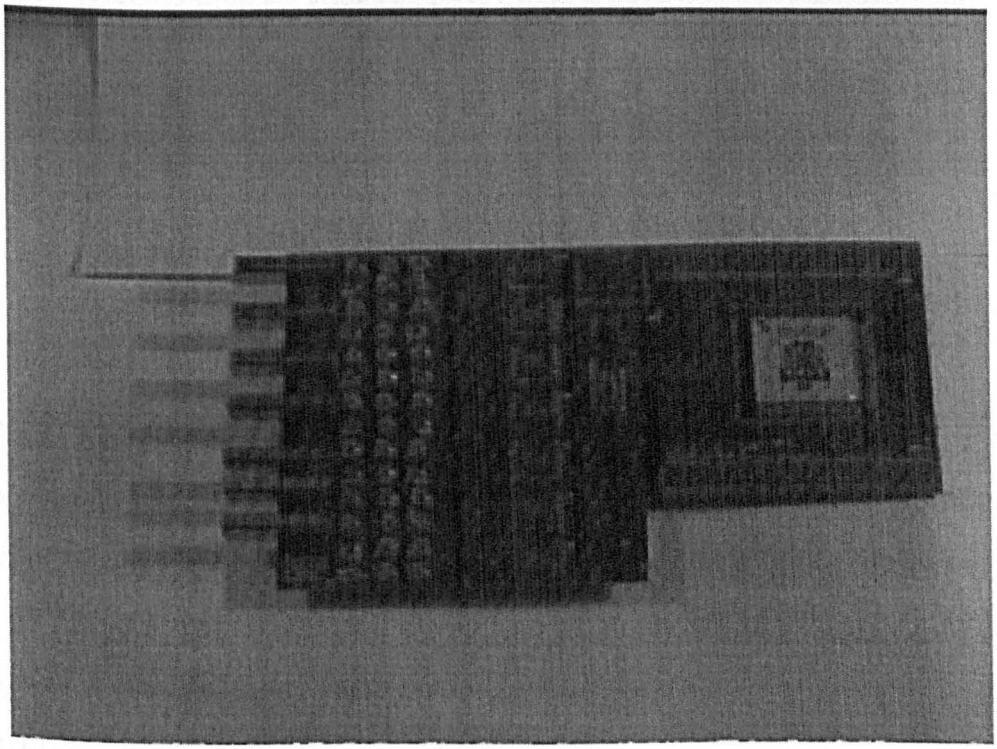


Figure 3.43 T9000 Transputer Adaptor Board

3.3.2 Full System Prototype Using Speedwire Board

The prototype of the dedicated image processing board was built by using a speedwire board. This prototype is shown in Figure 3.44. It contains the

populated T9000 transputer adaptor board, four DRAM chips, fourteen SRAM chips, two programmable logic devices, one video digitizer, one RAMDAC, and some support components.

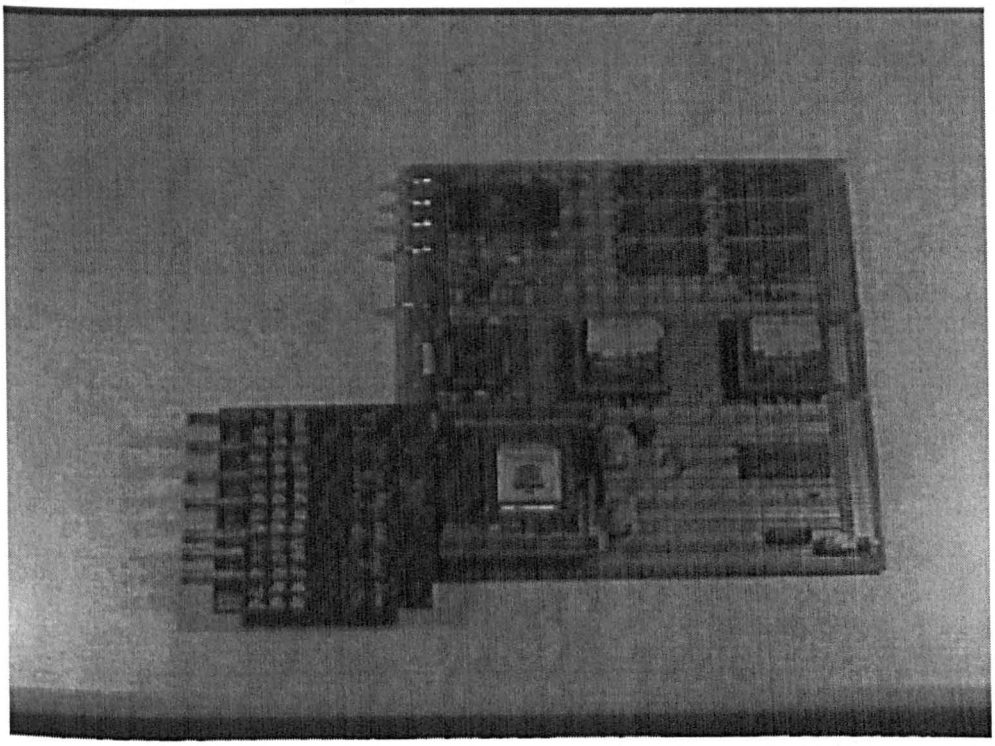


Figure 3.44 The prototype of the DIPB

3.4 IMS B108 and IMS B927

The IMS B108 is a full length PC-AT format HTRAM motherboard which supports up to two size 2 or size 4 HTRAMs. It also provides a standard control and data DS-Link interface between the PC-AT bus and a T9000 network. In the other words, the IMS B108 can support small networks of HTRAMs which are wholly inside the PC host, T9000 networks which are wholly external to the PC, or a network that is both internal and external.

The IMS B927 is a second generation transputer module (HTRAM), which integrates the high performance IMS T9000 transputer with 8 Mbytes of fast DRAM. The memory is organised as four 64-bit wide banks, all of which are cacheable. The memory system performs automatic page mode accesses

whenever possible to provide maximum memory bandwidth. The IMS B927 is fully compatible with the IMS B108 HTRAM motherboard.

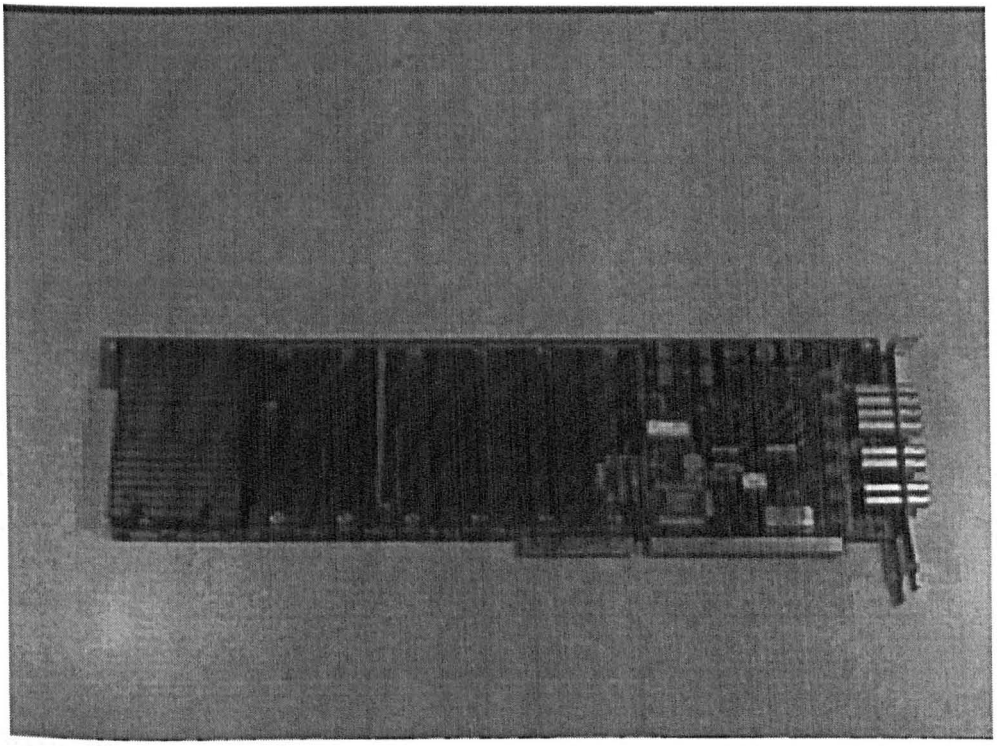


Figure 3.45 The IMS B108 with the IMS B927 HTRAM

Chapter 4

The Configurations of The Image Processing System

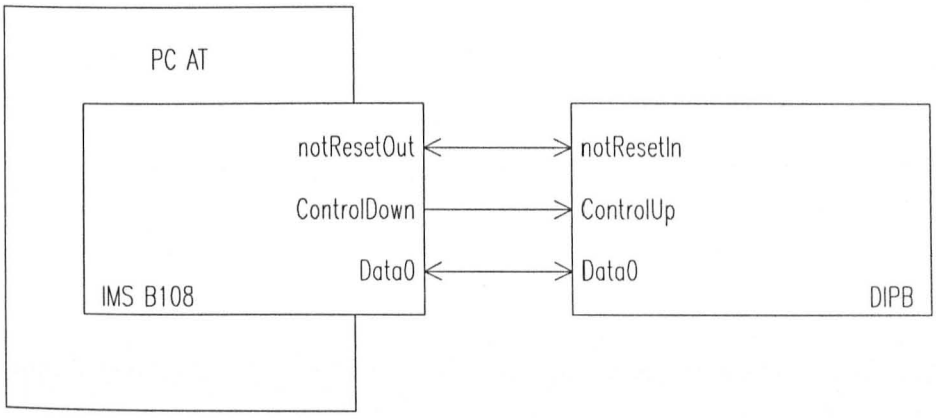
This chapter illustrates the configurations of the image processing system for two modes, Single-T9 mode and Multi-T9 mode. In Single-T9 mode, only the dedicated image processing board and the IMS B108 are used. There is only one T9000 transputer involved in this mode. In Multi-T9 mode, the overall system consists of one dedicated image processing board, a number of IMS B108's, and a number of IMS B927's. There is more than one T9000 transputer involved in this mode.

4.1 Single-T9 Mode

In Single-T9 mode, there is only one transputer performing the computation, so low processing power for this mode is expected. However, some applications, such as non real-time processing, do not require high processing power, so this mode may be sufficient to complete these tasks.

4.1.1 System Interconnection

The image processing system is not a stand-alone system, it is connected to a computer for downloading and running programs. The global elements of the system in this mode are the computer, the IMS B108, and the dedicated image processing board. Since data links are not used to transmit image data, only one data link is connected, which is required in the control system of the T9000 transputer. Figure 4.1 shows the system interconnection in Single-T9 mode.



DIPB : Dedicated Image Processing Board

Figure 4.1 System Interconnection in Single-T9 Mode

In the IMS B108, the data link jumpers enable reconfiguration of a small number of the data links from the two HTRAM slots and the data link ST C101. The jumpers must be used in pairs, JP1 with JP2, and JP3 with JP4. Each pair of jumpers can be in two positions. The inside position is defined as all jumpers towards the inside of a line drawn between the centres of the pins. The outside position is defined as all jumpers towards the outside of the pins. Table 4.1 presents the jumper positions and the selected connections. During Single-T9 mode operation, JP1 and JP2 are set to the inside position, as shown in the option at the top of the table. Jumpers JP3 and JP4 are set to either position since slot 0 and slot 1 are not used.

Jumper	Positions	Connection(s)
JP1 and JP2	Inside	C101 **Data** to **Data0** SLOT 0 Links 0 and 1 N/C
JP1 and JP2	Outside	C101 **Data** to SLOT 0 Link 0 SLOT 0 Link 1 to **Data0**
JP3 and JP4	Inside	SLOT 0 Link 2 to **Data2** SLOT 1 Links 1 and 3 N/C
JP3 and JP4	Outside	SLOT 0 Link 2 to SLOT 1 Link 1 SLOT 1 Link 3 to **Data2**

Table 4.1 IMS B108 JP 1-4 Connection

The control links on the IMS B108 are connected into a pipeline with options to reconfigure the connection of the head of the pipeline and to bypass slots. The actual control link connections on the board are selected by switches. Table 4.2 is the IMS B108 control link switch connections. Switches SW-1, SW1-2 and SW1-3 are all set to OFF, which is the option at the bottom of the table.

SW1-1	SW1-2	SW1-3	Connection(s)
ON	ON	ON	External **ControlUp** to SLOT 0 **ControlUp** SLOT 0 **ControlDown** to SLOT 1 **ControlUp** SLOT 1 **ControlDown** to external **ControlDown**
ON	ON	OFF	External **ControlUp** to SLOT 0 **ControlUp** SLOT 0 **ControlDown** to external **ControlDown** SLOT 1 **ControlUp** and **Down** N/C (SLOT 1 jumped)
ON	OFF	ON	External **ControlUp** to SLOT 1 **ControlUp** SLOT 1 **ControlDown** to external **ControlDown** SLOT 0 **ControlUp** and **Down** N/C (SLOT 0 jumped)
ON	OFF	OFF	External **ControlUp** to external **ControlDown** SLOT 0 **ControlUp** and **Down** N/C (SLOT 0 jumped) SLOT 1 N/C **ControlUp** and **Down** (SLOT 1 jumped)
OFF	ON	ON	C101 **Control** to SLOT 0 **ControlUp** SLOT 0 **ControlDown** to SLOT 1 **ControlUp** SLOT 1 **ControlDown** to external **ControlDown**
OFF	ON	OFF	C101 **Control** to SLOT 0 **ControlUp** SLOT 0 **ControlDown** to external **ControlDown** SLOT 1 **ControlUp** and **Down** N/C (SLOT 1 jumped)
OFF	OFF	ON	C101 **Control** to SLOT 1 **ControlUp** SLOT 1 **ControlDown** to external **ControlDown** SLOT 0 **ControlUp** and **Down** N/C (SLOT 0 jumped)
OFF	OFF	OFF	C101 **Control** to external **ControlDown** SLOT 0 **ControlUp** and **Down** N/C (SLOT 0 jumped) SLOT 1 **ControlUp** and **Down** N/C (SLOT 1 jumped)

Table 4.2 IMS B108 Control Link Switch Connections

4.1.2 Hardware Configuration

This section describes how to configure the hardware components of the system by file or program. The frame capture subsystem, the frame display subsystem and the T9000 transputer require configuration before they are ready for use. Hardware configuration for the T9000 transputer determines the transputer types, the root transputer, link speeds, how the transputers are to be booted, the amount

of on-chip memory to be used as cache, how the memory is to be configured and how the transputer is connected. Basically, the hardware configuration for the frame capture subsystem requires the programming of the control register of the video digitizer TMC22071, whereas the hardware configuration for the frame display subsystem involves setting the look-up tables for pixel colour palette and overlay colour palette, and the programming of the command registers.

4.1.2.1 Memory Interface Configuration

The external memory interface is highly programmable, allowing large memory systems, containing different types of devices, to be built with little or no external logic. It has a 64-bit data bus, and each bank of memory can be configured to be 8, 16, 32 or 64 bits wide. All banks programmed to be 64-bit wide memory are defined as cacheable, and always transfer a full 64-bit operand to and from external memory to the internal cache, providing fast cache refill.

4.1.2.1.1 Program Memory

Four HM5117800BTT6 dynamic RAMs were chosen to provide the program memory. Figure 4.2 shows the section of the memory configuration file that handles this RAM which occupies bank 2. The dynamic RAMs were placed at the bottom of the address space, so the base address was #80000000. Since the total memory size is 8 Mbytes, the address mask is calculated as:

$$\begin{aligned} \text{Bank.Address.Mask} &= \text{BITNOT (8 Mbtyes - 1)} \\ &= \text{BITNOT (\#007FFFFF)} \\ &= \text{\#FF800000} \end{aligned}$$

The memory is 32 bits (or 4 bytes) wide, so the two least significant bits of the transputer address select a byte within the width of the memory. The column address is the ten bits, 2 to 11, and the row address is the eleven bits, 12 to 22, as shown in Figure 4.3.

```
                    Page 5 - Input Data, Bank 2
                    ===========================

 Cacheable                                  Bank.Base.Address  := 80000000
 Bank.Address.Mask  := FF800000             Port.Size          := 32 bits
 Page.Address.Bits  := 001FFFFC             Page.Address.Shift := 10 bits

Ras.Strobe                                  Cas.Strobe
 RAS2                                        CAS2
 Active.During := Read & Write              Active.During := Read & Write
 Time.To.Falling.Edge := 2 Phases           Time.To.Falling.Edge := 2 Phases
 Time.To.Rising.Edge  := 8 Phases           Time.To.Rising.Edge  := 7 Phases

Programmable.Strobe                         Write.Strobe
 OE2                                         WRITE2
 Active.During := Read                      Active.During := Write
 Time.To.Falling.Edge := 0 Phases           Time.To.Falling.Edge := 0 Phases
 Time.To.Rising.Edge  := 7 Phases           Time.To.Rising.Edge  := 7 Phases

 Ras.Precharge.Time :=  1 cycle             Ras.Edge.Time      :=  2 phases
 Ras.Cycle.Time     :=  1 cycle             Cas.Cycle.Time     :=  2 cycles
 Bus.Release.Time   :=  1 cycle             Wait.pin           := Disabled
```

Figure 4.2 Sample imem Display Page for the Program Memory

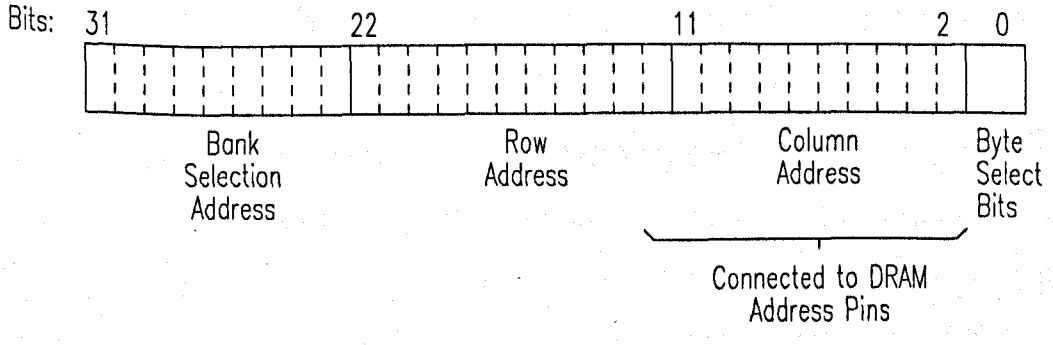


Figure 4.3 The HM5117800BTT6 Dynamic RAMs Addressing

Three parameters Ras.Cycle.Time, Page.Address.Shift and Ras.Edge.Time are provided to support multiplexed addresses. The parameter, Page.Address.Bits, is provided to support page mode. The Page.Address.Shift is the number of bits the row address must be shifted to place it on the DRAM address pins, which is 10 in this case. Two addresses in this bank are in the same page (i.e. the same row) if bits 12 to 22 are the same. The Page.Address.Bits parameter is the mask for the page address bits, i.e. the mask with bits 12 to 22 set, which is #007FF000 in this case. However, when this value was checked, it was found not to work. With reference to the memory configuration file for the IMS B927, the parameter for

Page.Address.Bits is #001FFFFC. It is not a reasonable value for DRAM, but it did work. This phenomenon might be caused by a T9000's bug at the silicon level. With trial and error, it was found that the value used by the IMS B927 also worked for the HM5117800BTT6 dynamic RAMs.

The parameters for Ras.Strobe, Cas.Strobe, Programmable.Strobe, Write.Strobe, Ras.Precharge.Time, Ras.Edge.Time, Ras.Cycle.Time, Cas.Cycle.Time and Bus.Release.Time were set according to the timings of the HM5117800BTT6 shown in the Hitachi data book. For the refresh section, it contains three timing parameters, which are called Dram.Refresh.Interval, Dram.Refresh.Time and Dram.Refresh.Ras.High.

Dram.Refresh.Interval is the time, in cycles, between successive refresh cycles. The refresh period of the DRAM is 32 ms for 2048 cycles and the processor speed is 20 MHz, hence:

$$\begin{aligned} \text{Dram.Refresh.Interval} &= 32 \times 20 \times 1000 \;/\; 2048 \\ &= 312.5 \text{ Cycles} \end{aligned}$$

Dram.Refresh.Interval can be set to 300 cycles which is less than the calculated value.

4.1.2.1.2 Buffer Memory

W241024A static RAMs were chosen for the frame buffer memory and the overlay buffer memory. Actually, the buffer memory is not directly connected to bank 1 of the T9000 transputer, but the multiple bus interface is. The timing parameters of this bank are set so that the T9000 transputer can read or write data to the buffer memory via the bus interface. Figure 4.4 shows the section of the memory configuration file for the buffer memory in bank 1. The buffer memory was placed at the address #90000000 and 16 Mbytes address space was reserved for it. The address mask is calculated as:

$$\begin{aligned} \text{Bank.Address.Mask} &= \text{BITNOT (16 Mbtyes - 1)} \\ &= \text{BITNOT (\#00FFFFFF)} \\ &= \text{\#FF000000} \end{aligned}$$

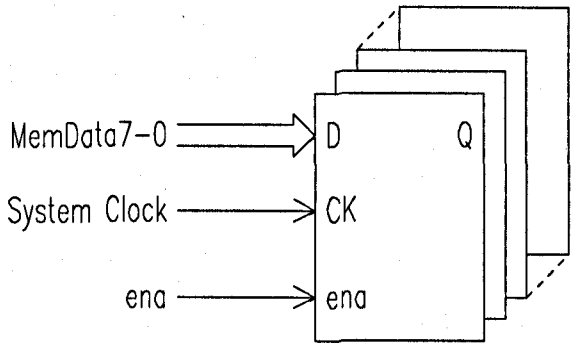
Wait.Pin is enabled for incorporating the bus multiplexing mode. Bank 1 is marked as non-cacheable since the buffer memory is a shared memory. Cache coherency problems would occur if a shared memory was cached.

```
                    Page 4 - Input Data, Bank 1
                    ===========================

 Non-Cacheable                          Bank.Base.Address  := 90000000
 Bank.Address.Mask  := FF000000         Port.Size          := 32 bits

Strobe.1                               Strobe.2
 RAS1                                   CAS1
 Active.During := Read & Write          Active.During := Read
 Time.To.Falling.Edge := 2 Phases       Time.To.Falling.Edge := 2 Phases
 Time.To.Rising.Edge  := 14 Phases      Time.To.Rising.Edge  := 14 Phases

Programmable.Strobe                    Write.Strobe
 PS1                                    WRITE1
 Active.During := Read                  Active.During := Write
 Time.To.Falling.Edge := 2 Phases       Time.To.Falling.Edge := 2 Phases
 Time.To.Rising.Edge  := 13 Phases      Time.To.Rising.Edge  := 13
Phases

 Cycle.Time      := 4 cycles            Bus.Release.Time   := 2 cycles
 Wait.pin        := Enabled
```

Figure 4.4 Sample imem Display Page for the Buffer Memory

4.1.2.1.3 I/O Devices

Bank 3 is allocated for I/O devices. Although the timings for these devices are not unique, one set of timings is defined so that it is compatible with all the timings for the different devices. The devices to be considered are registers, the video digitizer and the video encoder. Registers were made using D type flip-flops with an enable input. The interface for the register is shown in Figure 4.5. The **ena** signal must be set low before the end of valid data on the input of register. The rising edge of notMemWrB0 in bank 3 must appear before the end of each cycle time.



ena = Decoded Address & ! notMemWrB0

Figure 4.5 Interface of Registers

The MPU interfaces of the video digitizer and the video encoder were discussed in the frame capture subsystem and the frame display subsystem sections of chapter three. Time.to.falling.edge of Cas.Strobe is purposely set for the video digitizer, whereas the parameters: Programmable.Strobe and Write.Strobe, are set for the video encoder. Figure 4.6 shows the section of the memory configuration file in bank 3.

```
                Page 6 - Input Data, Bank 3
                ===========================

 Device.only                          Bank.Base.Address  :=
A0000000
 Bank.Address.Mask  := FF000000       Port.Size          := 32 bits

Strobe.1                             Strobe.2
 Inactive                             CAS3
                                      Active.During := Read & Write
                                      Time.To.Falling.Edge := 2
Phases
                                      Time.To.Rising.Edge  := 6
Phases

Programmable.Strobe                  Write.Strobe
 PS3                                  WRITE3
 Active.During := Read                Active.During := Write
 Time.To.Falling.Edge := 1 Phase      Time.To.Falling.Edge := 1
Phase
 Time.To.Rising.Edge  := 5 Phases     Time.To.Rising.Edge  := 5
Phases

 Cycle.Time       := 2 cycles         Bus.Release.Time   := 1 cycle
 Wait.pin         := Disabled
```

Figure 4.6 Sample imem Display Page for the I/O Ports

4.1.2.2 Network Configuration

The Network Description Language [40] is used to describe the available hardware such as the types of processors, their attributes, and how they are connected. Processor attributes include a description of its memory map and its link speeds. The network description will not normally be changed unless the hardware is changed. Figure 4.7 shows a network description for the image processing system configured in Single-T9 mode.

```
#INCLUDE "stdndl.inc"

VAL mem.base IS #80000000:
VAL sys.size IS 2*K:
VAL sram.size IS 2*M:
VAL dram.size IS 8*M:
VAL sram.base IS #90000000:
VAL dram.base IS #80000000:
VAL system.base IS dram.base + (dram.size - sys.size):

CONTROLPORT host :
NODE wingt9:
ARC HostLink :

NETWORK
 DO
  SET DEFAULT (link.speed.multiply  := 10)
  SET DEFAULT (link.speed.divide    := [1])
  SET DEFAULT (control.speed.divide := [8])

  -- set wingt9 processor type
  SET wingt9(type                 :="T9000")
  SET wingt9(root                 := TRUE )
  SET wingt9(local.rom            := FALSE)
  SET wingt9(pmi.config.inrom     := FALSE)
  SET wingt9(cachesize            := 16 )
  SET wingt9(memconfig            := "myramio.mem")
  SET wingt9(memory               := [[dram.base, dram.size - sys.size, USER+RAM],
                                      [sram.base, sram.size, RESERVED+RAM],
                                      [system.base, sys.size, SYSTEM+RAM]])

  -- Connect control tree
  CONNECT host[control]    TO wingt9[control.up]

  -- Connect data tree
  CONNECT host[data]       TO wingt9[link][0] WITH HostLink
:
```

Figure 4.7 A Network Description for Single-T9 Mode

4.1.2.2.1 Terminology

Controlport is an interface to the host or controlling processor. It has a control link connection and a data link connection which will be used to initialize and load the system respectively.

A **node** in a T9000-series network is a device, or part of a device, which is initialised by means of a control link or a ROM. For example, for the IMS T9000 transputers, asynchronous packet switches, and system protocol converters, are all nodes.

Connections to nodes, and to devices external to the network, are referred to as **edges**. An edge for an external device is also known as a network edge, or an external edge.

Connections between nodes and the outside world are referred to as **arcs**. Arcs always connect edges, and arcs between nodes need not be named. If they are named, they can be referred to in the section of channel mapping or channel placement.

4.1.2.2.2 The Declarations of Node, Arc and Control Port

The host is the computer which incorporates the IMS B108. Wingt9 represents the dedicated image processing board and HostLink is the connection of the host's data link to wingt9's data link.

4.1.2.2.3 The Link Speed Attributes

There are three attributes to set the link transmission speeds. Every link must have its transmission speed set. The link speeds may be set for individual devices and links, but normally the links of the entire data network will all run at a single speed, as will the links of the control network.

The transmission speed of a link is derived from a 10 MHz base clock by multiplying and dividing by the available factors. For a single device, there is one multiplication factor for all the links. This is defined by the attribute link.speed.multiply and can take any of the values 8, 10, 12, 14, 16, 18 or 20. From this multiplied speed, each link can have its own speed division so that the links can run at different speeds. The speed divisions are set by the arrays link.speed.divide for the data links and control.speed.divide for the control links, and the division factor can be set to 1, 2, 4, or 8.

With reference to Figure 4.7, link.speed.multiply is set to 10, link.speed.divide is set to 1, and control.speed.divide is set to 8. This resulted in setting all the data links' speeds to 100 Mbits/sec and all control links' speeds to 12.5 Mbits/sec. The lowest speed is purposely set for the control links so as to increase their reliability, for system control and monitoring.

4.1.2.2.4 The Type and Root Attributes

Since there is only one processor, called wingt9, it must be the root processor for the whole system. The wingt9 node is actually an IMS T9000, so the node type is set to "T9000".

4.1.2.2.5 The Memory and Memconfig Attributes

The **memconfig** attribute is used to specify the name of the memory configuration file. The memory configuration file for the dedicated image processing board is called myramio.mem, and it defines all the timings of bank 1, bank 2 and bank 3.

The memory is divided into blocks. Each block is declared with one possible usage, either RAM or ROM, and RAM can be either **USER**, **SYSTEM** or **RESERVED**. Five constants, **SYSTEM**, **USER**, **RESERVED**, **RAM** and **ROM**, are defined to assign the usage. These constants are intended to be added

together in various combinations, and Table 4.3 describes some of the possible cases.

Usage	Explanation
ROM	used for any ROM
SYSTEM + RAM	used for RAM to be used during the loading and bootstrapping
USER + RAM	used for general purpose RAM, to be allocated by the configurer
RESERVED	used for special memory (such as screen RAM) and non-memory devices

Table 4.3 The Memory Usage of the Dedicated Image Processing Board

Three blocks of memory are required for the program memory, the buffer memory and the system memory. The buffer memory includes the frame buffer memory and the overlay buffer memory. The memory space for the program memory is declared as **USER + RAM** since this memory is used for general purpose. The memory range for the buffer memory is declared as **RESERVED + RAM** for preventing the configurer from downloading the program code into this memory space.

The memory locations for the I/O devices are not declared because they are not used as memory. If the I/O devices are accessed in a manner of memory access, they are declared as **RESERVED + RAM** in the memory attribute.

4.1.2.2.6 The Control and Data Tree

The T9000 transputer has a control.up link (CLink0), a control.down link (CLink1) and an array of four data links. The control port has one control link and one data link only. The control link in the host control port is connected to the control.up link of the wingt9 and its data link is connected to data link 0 of the T9000 transputer.

4.1.2.3 Setting Up The Frame Capture Subsystem

For the frame capture subsystem, configuration is only required for the TMC22071 video digitizer. In addition to this configuration, the state machine for the pixel demultiplexer should be reset before running application processes.

The TMC22071 is controlled by a single 47-bit long Control Register [41]. Bit 0 of the register is set to binary 1 to start and run the internal state machines. Bits 3-1 are set to binary 100 for a PAL input signal and to set the pixel rate at 13.5 Mpps. Since the video connector for the incoming video signal is connected to Vin1, bits 8-7 are set to binary 00. Video gain is set to unity by setting bit 9 to binary 0. In order to keep the alignment of HSYNC and the incoming video, bits 24-17 are set at binary value 01111011. Bit 25 is set to low, so that the automatic gain control operation is continuous for two fields, and then held. Bit 26 is set to low to lock PXCK into incoming video. Bit 31 is set to high to enable GHSYNC\ and GVSYNC\. Bits 43-40 are set to binary 011 to fix the output sync level to the hexadecimal value 07.

4.1.2.4 Setting Up The Frame Display Subsystem

The only element requiring configuration in the frame display subsystem is the RAMDAC Bt481A. It is also necessary to reset the state machine for the pixel multiplexer before running application processes.

Before the RAMDAC Bt481A is available for use, the command register A and the command register B need to be initialised and the look-up tables for the colour palette RAM data and the overlay/cursor colour data are filled with appropriate values.

The RAMDAC Bt481A was used to support an 8-bit pseudo-colour format. The command register A is initialised to obtain this format. In the OCCAM [42] program, the procedure to initialise this register is:

```
PROC write.command.reg.a ()
  SEQ
    rs2.rs1.nrs0 ! #00
:
```

where rs2.rs1.nrs0 is the port used to access the command register A, and its location is #A0B60000 in the machine map.

It is not necessary to initialise the command register B, since its default value at power-up is already appropriate for the system.

RS2	RS1	RS0	Addressed by MPU
0	0	0	address register (palette write mode)
0	1	1	address register (palette read mode)
0	0	1	colour palette RAM
0	1	0	pixel read mask register
1	0	0	address register (overlay write mode)
1	1	1	address register (overlay read mode)
1	0	1	overlay registers
1	1	0	command register A

Table 4.4 Control Input Truth Table of Bt481A

Table 4.4 is the control input truth table of the RAMDAC Bt481A. To write colour data, the T9000 transputer writes the address register (RAM write mode) with the address of the colour palette RAM location which is to be modified. Then the T9000 transputer performs three successive write cycles (8 bits each for red, green, and blue), using RS0-RS2 to select the colour palette RAM. The following OCCAM program shows how a block of colour values, in consecutive locations, is written to the colour palette RAM, by writing the start address and performing continuous RGB write cycles until the whole block has been written.

```
PROC write.color.palette ()
  SEQ
    nrs2.nrs1.nrs0 ! #00  -- address register (palette write mode)
    SEQ i = 0 FOR 256
      SEQ
        nrs2.nrs1.rs0 ! i
        nrs2.nrs1.rs0 ! i
        nrs2.nrs1.rs0 ! i
:
```

The nrs2.nrs1.nrs0 is the port used to access the address register (palette write mode), which is located at #A0B00000 in the machine map, and the nrs2.nrs1.rs0 is the port used to access the colour palette RAM, which is located at #A0B10000 in the machine map.

To write the overlay data, the T9000 transputer writes the address register (overlay write mode) with the address of the overlay location to be modified. Then the T9000 transputer performs three successive write cycles (8 bits each for red, green, and blue), using RS0-RS2 to select the overlay registers. The following OCCAM program shows how a block of colour values in consecutive locations is written to the overlay registers by writing the start address and performing continuous RGB write cycles until the entire block has been filled.

```
PROC write.overlay ()
  SEQ
    rs2.nrs1.nrs0 ! #00  -- address register (overlay write mode)
    SEQ i = 1 FOR 15
      SEQ
        rs2.nrs1.rs0 ! i
        rs2.nrs1.rs0 ! i
        rs2.nrs1.rs0 ! i
:
```

The rs2.nrs1.nrs0 is the port used to access the address register (overlay write mode) which is located at #A0B40000 in the machine map and the rs2.nrs1.rs0 is the port used to access the overlay registers, which is located at #A0B50000 in the machine map.

4.1.3 Software Configuration

Software configuration is primarily concerned with mapping. Processes are mapped onto logical processors, and logical processors are mapped onto physical processors. In addition, channels between processes are mapped or placed onto physical links.

In Single-T9 mode, all processes are mapped to the wingt9 processor, since it is the only processor in the system. As all processes are resident in one processor, no channel mapping is needed.

4.2 Multi-T9 Mode

In Multi-T9 mode, more than one T9000 transputer are integrated together for the purpose of increasing the computational power of the whole system. To illustrate this, a two transputer network is discussed in this section.

4.2.1 System Interconnection

The global elements of the image processing system are the computer, the IMS B927, which is fixed to the slot 0 position of the IMS B108, the IMS B108, and the dedicated image processing board. Figure 4.8 shows the system interconnection in Multi-T9 mode.

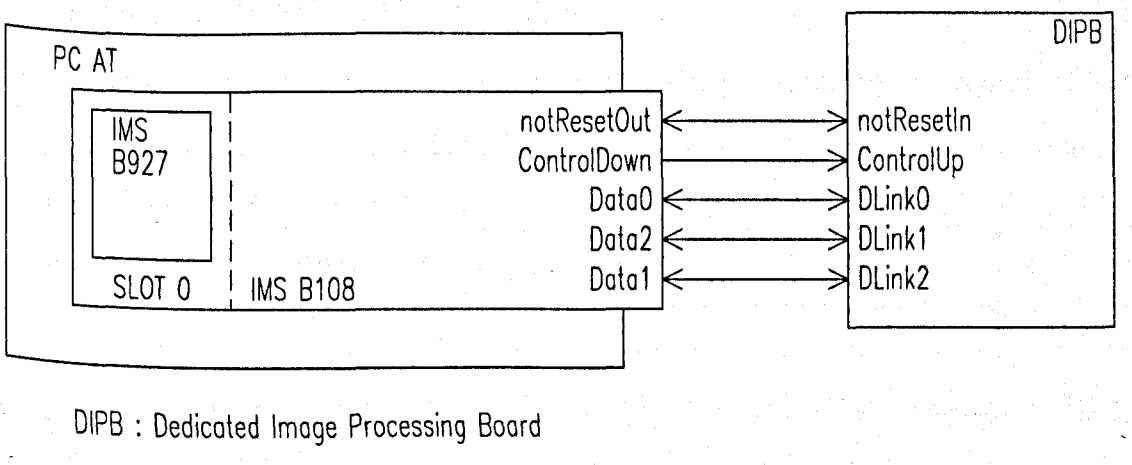


Figure 4.8 System Interconnection for Multi-T9 Mode

For the data link configuration in the IMS B108, JP1 and JP2 are set to the outside position, and JP3 and JP4 are set to the inside position [Table 4.1]. For the control link configuration, switches SW-1, SW1-2 and SW1-3 are set to OFF, ON, and OFF respectively [Table 4.2].

As a result, the IMS B927 is connected to the host and so becomes the root transputer in the network. The data link 1, 2 and 3 of IMS B927 are connected externally to DLink0, DLink1 and DLink2 of the dedicated image processing board respectively. Meanwhile, the CLink1 of the IMS B927 is externally connected to the CLink0 of the board.

4.2.2 Hardware Configuration

The hardware configuration for a two processor network involves two memory configuration files and one network configuration file. The contents of the two memory configuration files remain unchanged from those used in Single-T9 mode, but the contents of network configuration file would be changed. For example, the contents of the data tree and the control tree sections of a network configuration file must be changed if the physical link connections, between two transputers, have been changed.

4.2.2.1 Memory Interface Configuration

The memory configuration file for the IMS B927 is provided by its manufacturer, and is called b927-20b.mem. This file is shown in Appendix D. The memory configuration file for the dedicated image processing board is the same as that described in the Single-T9 mode section.

4.2.2.2 Network Configuration

Figure 4.9 shows a network description for the image processing system configured in Multi-T9 mode. In order to set the links to consistent speeds, the link speeds are set by defaults with the desired parameters. For the attributes of the b927, the attribute, root, is set to TRUE because the b927 is connected to the host. For the rest of the attributes, they are set according to information provided by its manufacturer. All connected data links are declared so that they can be mapped to the logical channels if necessary.

```
#INCLUDE "stdndl.inc"

VAL mem.base IS #80000000:
VAL sys.size IS 2*K:
VAL sram.size IS 2*M:
VAL dram.size IS 8*M:
VAL sram.base IS #90000000:
VAL dram.base IS #80000000:
VAL system.base IS dram.base + (dram.size - sys.size):

CONTROLPORT host:
NODE b927:
NODE wingt9:
ARC HostLink:
ARC Link0:
ARC Link1:
ARC Link2:

NETWORK
 DO
  SET DEFAULT (link.speed.multiply  := 10)
  SET DEFAULT (link.speed.divide    := [1])
  SET DEFAULT (control.speed.divide := [8])

  -- set b927 processor type
  SET b927(type                 := "T9000")
  SET b927(root                 := TRUE )
  SET b927(local.rom            := FALSE)
  SET b927(pmi.config.inrom     := FALSE)
  SET b927(cachesize            := 16 )
  SET b927(memconfig            := "b927-20b.mem")
  SET b927(memory               := [[#80000000, #3FF000, RAM],
                                    [#803FF000, #1000, RAM + SYSTEM] ])

  -- set wingt9 type
  SET wingt9(type               :="T9000")
  SET wingt9(root               := FALSE )
  SET wingt9(local.rom          := FALSE)
  SET wingt9(pmi.config.inrom   := FALSE)
  SET wingt9(cachesize          := 16 )
  SET wingt9(memconfig          := "myramio.mem")
  SET wingt9(memory             := [[dram.base, dram.size - sys.size, USER+RAM],
                                    [sram.base, sram.size, RESERVED+RAM],
                                    [system.base, sys.size, SYSTEM+RAM]])

  -- Connect control tree
  CONNECT host[control]         TO b927[control.up]
  CONNECT b927[control.down] TO wingt9[control.up]

  -- Connect data tree
  CONNECT host[data]          TO b927[link][0] WITH HostLink
  CONNECT b927[link][1]       TO wingt9[link][0] WITH Link0
  CONNECT b927[link][2]       TO wingt9[link][1] WITH Link1
  CONNECT b927[link][3]       TO wingt9[link][2] WITH Link2
```

Figure 4.9 Network Description for Multi-T9 Mode

4.2.3 Software Configuration

Since the number of physical processors is two, software configuration is essential. Process mapping and logical processor mapping must be stated in the software description file. However, channel mapping is optional. If channel mapping is not mentioned in the software description file, the configurer will assign physical links to the channels which connect the two physical processors.

Chapter 5

The Operations of The Image Processing System

This chapter can essentially be split into four sections. The first section introduces the frame buffer management, which shows how the three frame buffers are organised to serve different applications. The second section talks about the technique to divide an image into a number of sub-images so that the work of image processing can be shared by a number of processors. The third section introduces four methods to transfer sub-images to a T9000 transputer network. The fourth section contains details of image processing algorithms which were used to test the performance of the system.

5.1 Frame Buffer Management

In Single-T9 mode, the three frame buffers on the dedicated image processing board are usually organised for the four functions, image capturing, captured image reading, processed image writing, and processed image displaying. In Multi-T9 mode, they are usually arranged for image capturing, captured image sending, processed image receiving, and received image displaying. Obviously, the three frame buffers need to be mapped onto the four functions. This frame buffer mapping actually depends on the applications being carried out by the system.

5.1.1 Single-T9 Mode

Figure 5.1 shows an example of the organisation of the three frame buffers in Single-T9 mode. In this case, one of the three frame buffers is simultaneously,

implementing two functions, captured image reading and processed image writing. The rest of the frame buffers only perform a single function.

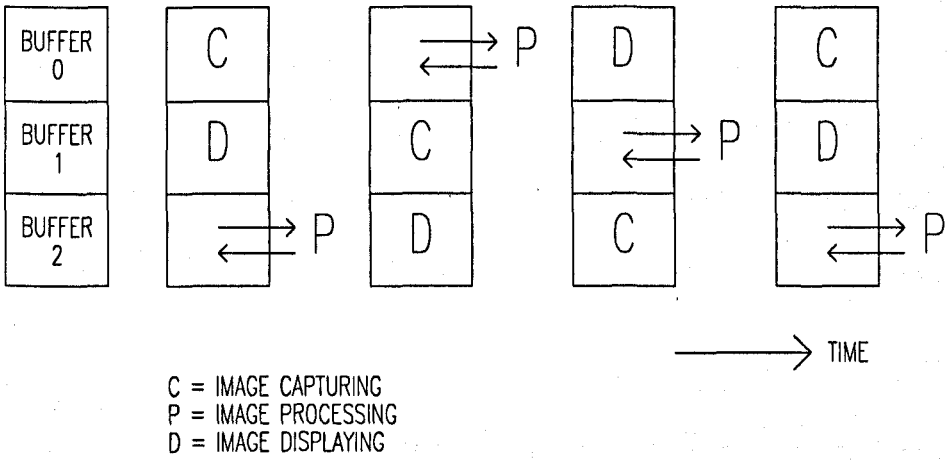


Figure 5.1 Frame Buffer Operation In Single-T9 Mode

The operation principle behind this organisation is to process the image which has been previously captured into a buffer, and to display the image that has been previously processed. Firstly, buffer 0 is used for the capture operation, buffer 1 for the display operation and buffer 2 for both the captured image reading and the processed image writing operations. After the completion of one operation, the role of the buffers rotates downward such that buffer 1 is used for the capture operation, buffer 2 performs the display operation and buffer 0 is used for both the captured image reading and the processed image writing operations. In general, the role of the buffers rotates downward by one step after each operation.

In some image processing algorithms, the result of processing a pixel depends on its nearest neighbours. The processed pixels should not be written back to their original positions immediately until their pre-processed values have all been referenced by their neighbours. If a 3 × 3 operator is used, for example, the nth processed pixel line should not be written back until the (n+2)th pixel line is being processed, where the 3 × 3 operator is moving from the top of the image to the bottom of the image and the top line of the image is called the 1st pixel line.

If the processed image is allowed to shift upward, the first pixel line of the original image can be overwritten by the first processed pixel line, no buffering is required.

Alternatively, the pixels going to be referenced by the other pixels are stored in arrays. The processed values can be written back to the original positions immediately, since the pre-processed values are already resident in arrays which can be referenced during subsequent processing.

5.1.2 Multi-T9 Mode

Figure 5.2 shows an example of the arrangement of the three frame buffers in Multi-T9 mode for real-time processing. In this example, the T9000 transputer on the dedicated image processing board does not process the image. Its single role is to handle the image transmission between the dedicated board and the external T9000 transputer network. One of the three frame buffers needs to perform two functions, the image display function and the processed image receiving function. The rest of the frame buffers implement one function only. It should be noted that the rate of storing a processed image into a buffer must be higher than the rate of displaying the stored image in a real-time situation.

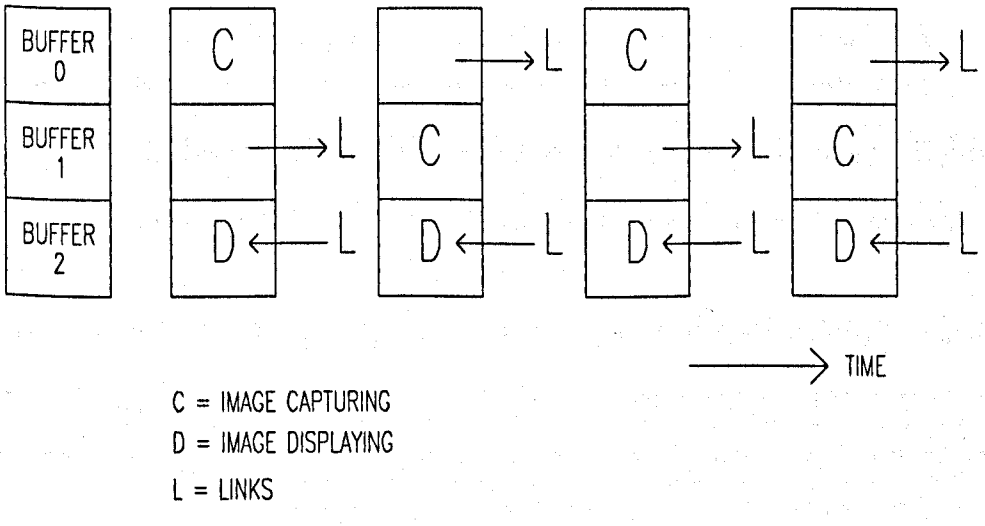


Figure 5.2 Frame Buffer Operation in Multi-T9 Mode

The operation principle behind this arrangement is to send out the raw image, which has been previously captured into a buffer, to the external T9000 transputer network for processing and to receive the image which has been previously processed by the transputer network for display. Firstly, buffer 0 is used for the capture operation, buffer 1 is used for the captured image sending operation, and buffer 2 performs both the display and the processed image receiving operations. Afterwards, the roles of buffer 0 and buffer 1 are swapped, whilst the role of buffer 2 remains the same. The operation continues in this manner until the end of the program.

5.2 Image Division Techniques

For the operation of the image processing system running in Multi-T9 mode, a captured image should be divided into a number of sub-images, which are then distributed to the elements of the T9000 transputer network for processing. Usually, the number of sub-images is equal to the number of the elements in the network.

For the image processing algorithms which do not require a pixel to interact in some way with its immediate neighbours, the image division method is simple, as the image is just divided evenly into a number of sub-images. If the result of processing a pixel depends on its immediate neighbours, the image is not simply divided evenly into a number of sub-images, but additional pixels from adjacent sub-images must be attached to the edges of each sub-image. Figure 5.3 illustrates the second case, in which a 3×3 operator is employed for image processing, and the image is divided into two sub-images. The purpose of edge attachment is to allow the pixels on the boundaries of the sub-image to interact with their immediate neighbours during image processing. In this case, one additional pixel line is attached to the boundary. Similarly, two additional pixel lines are attached to the boundary if a 5×5 operator is used.

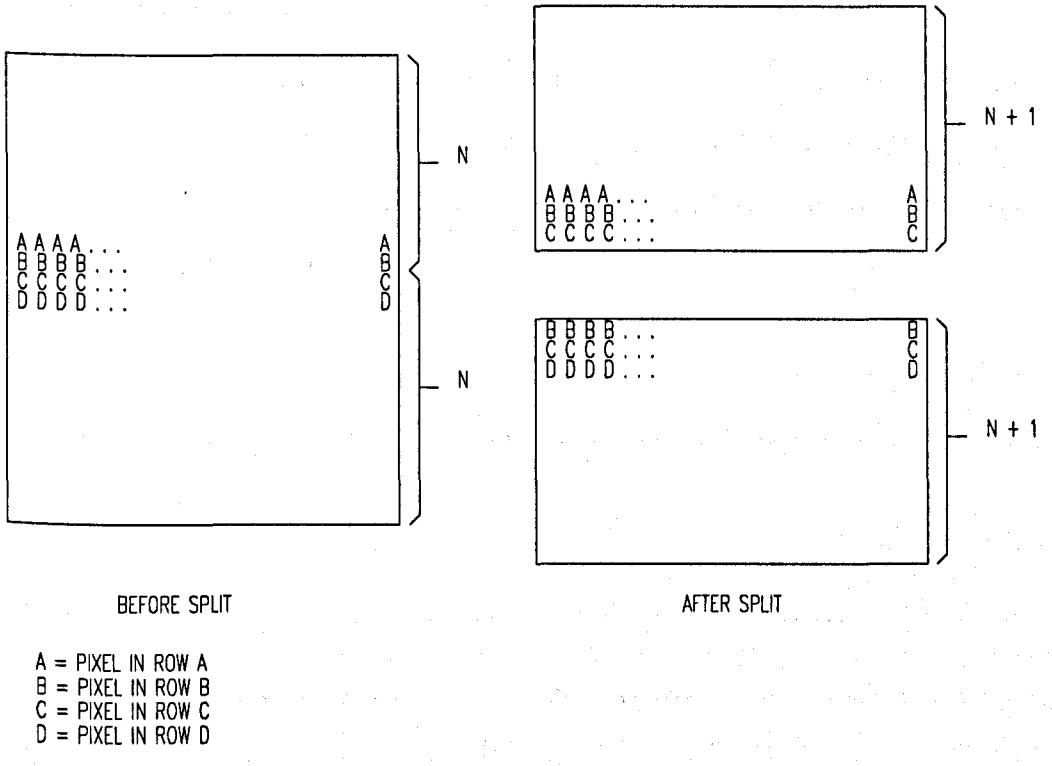


Figure 5.3 Illustration of Edge Attachment

5.3 Image Distribution Techniques

The image distribution techniques used by the image processing system are closely related to the architecture of the T9000 transputer system. The T9000 transputer was designed to use the high bandwidth DS-Links for data transmission. T9000 transputers may be connected directly, or via a network of IMS C104 dynamic routing devices [43]. Communication channels can be established between any two processes, regardless of where they are physically located, or whether the channels are routed through a network, provided there is a connecting path via a communication network. Sub-images are distributed from the dedicated image processing board to the external T9000 transputer boards, in a way that it is appropriate to the types of the T9000 transputer network. The following sections discuss the image distribution techniques in four different types of networks.

5.3.1 The Non-routed Network

Figure 5.4 shows a small non-routed, but fully connected, network of a parallel image processing system. The dedicated image processing board can directly communicate to the adjacent three T9000 transputer boards, therefore no software virtual-routing processes are required.

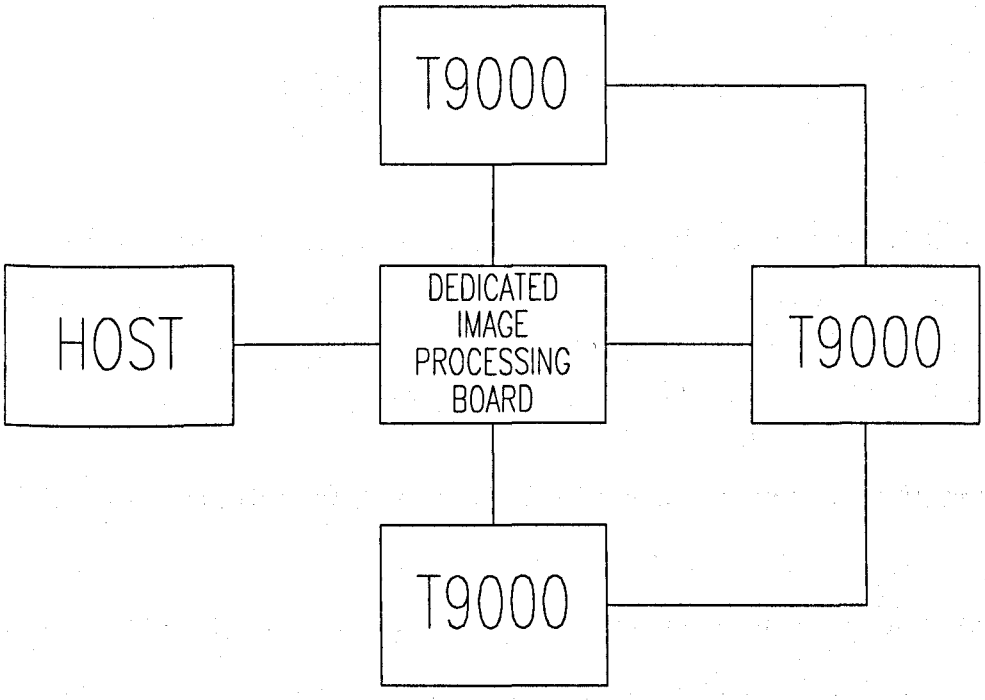


Figure 5.4 The Non-routed Image Processing System Network

5.3.2 The Routed Network without IMS C104

Figure 5.5 shows a routed network of the parallel image processing system without the IMS C104 device. The dedicated image processing board cannot directly communicate to the non-adjacent T9000 transputer boards, which are located at the corners of the network, therefore software virtual-routing processes are needed to implement the non-adjacent interprocessor communication. With regard to the comments given by INMOS, the performance of this kind of network is low.

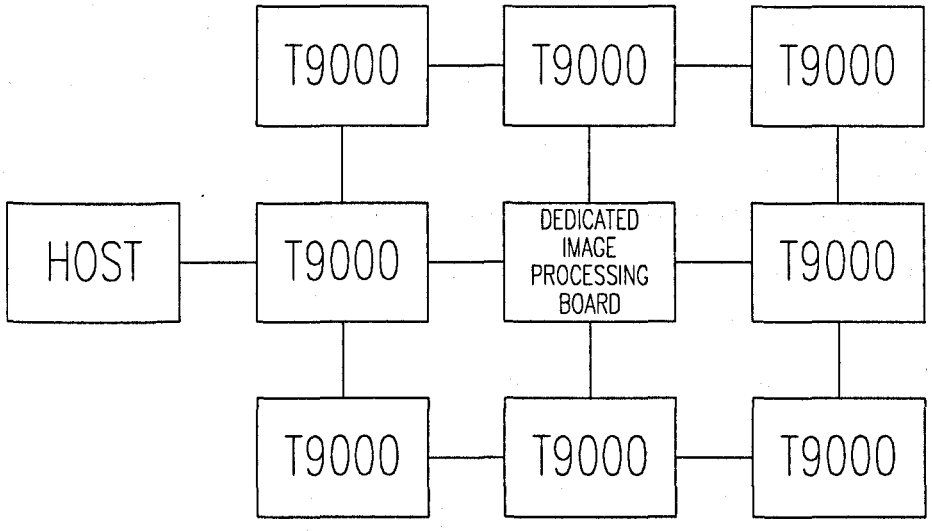


Figure 5.5 The Routed Image Processing System Network without IMS C104

5.3.3 The Routed Network with IMS C104

Figure 5.6 shows a routed network of the parallel image processing system with the IMS C104 device. Basically, the IMS C104 devices were used to improve the performance of interprocessor communications in multi-transputer networks. The use of this network is recommended when the dedicated image processing board is integrated to a large multi-transputer network for parallel processing.

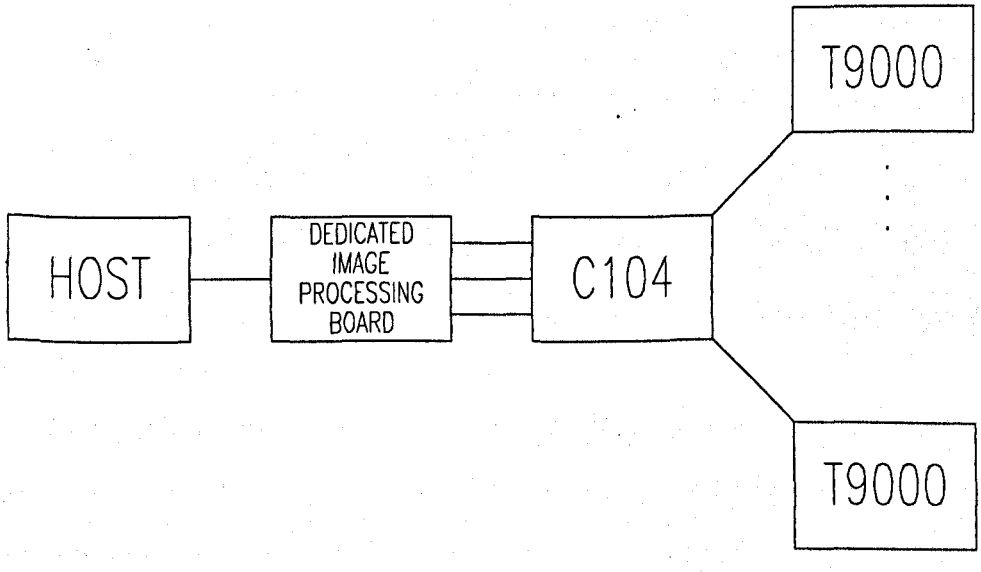


Figure 5.6 The Routed Image Processing System Network with IMS C104

5.3.4 The Two-DIPB Network

Figure 5.7 shows a network which makes use of two dedicated image processing boards. One of the boards is dedicated to image capture, the other one to image display. The use of this network is to reduce the data traffic in the frame buffer memory. This network is recommended when it is required to display a whole processed image.

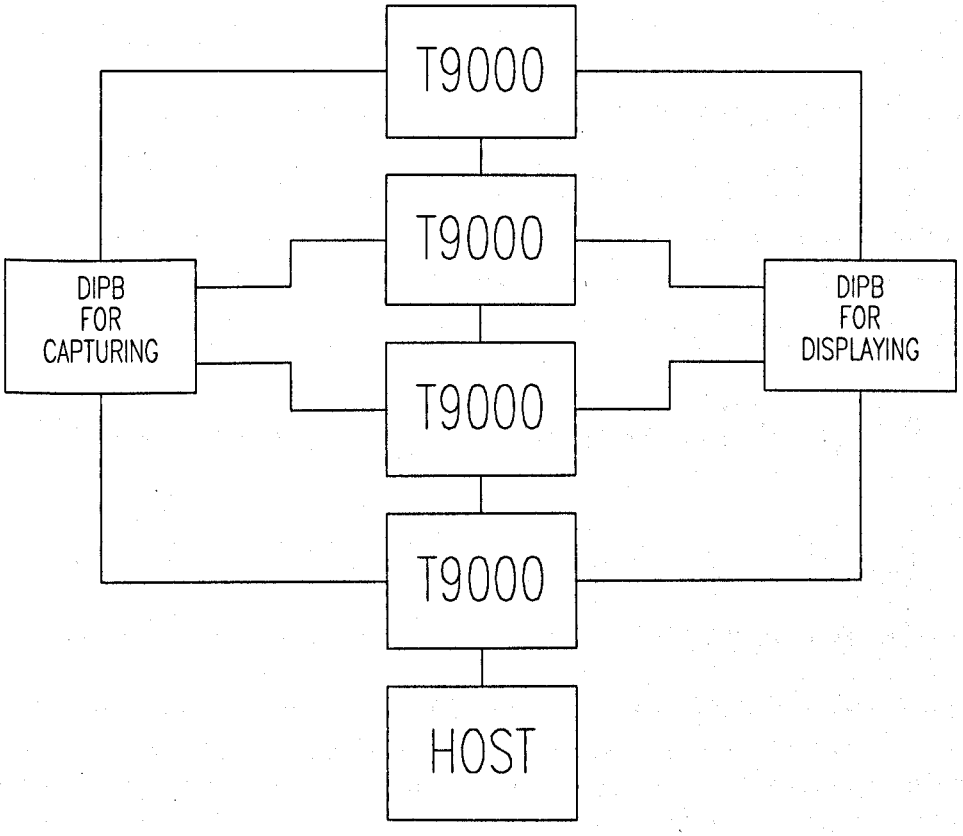


Figure 5.7 The Two-DIPB Network

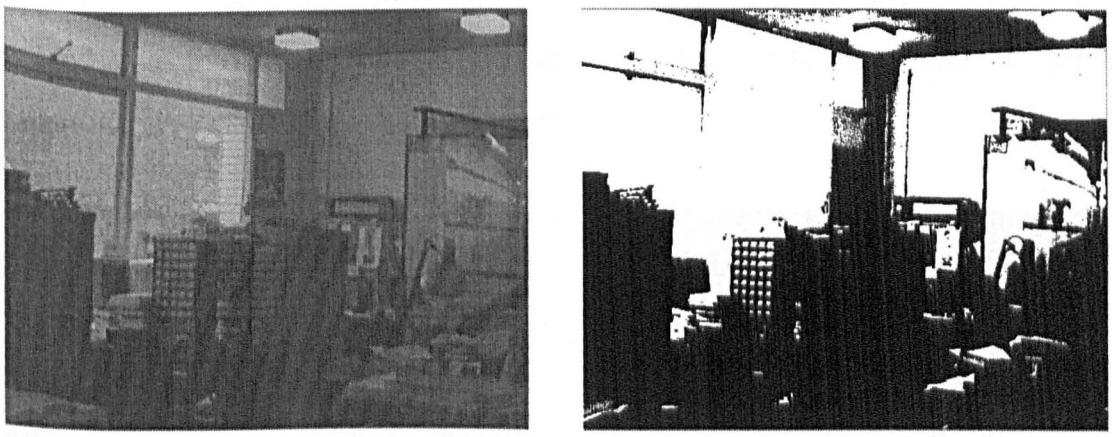
5.4 Image Processing Algorithms

The application of this project involved implementing a wide range of image processing algorithms for use in different circumstances. This section introduces the image processing algorithms selected to test the performance of the system. It is an advantage to give an overview of the functions and the complexities of these algorithms before going to the next chapter, which presents the results of the system performance tests. The selected image processing algorithms are

thresholding [44], local averaging [45], Sobel edge detection [46] and seam tracking [47].

5.4.1 Thresholding

Thresholding is one of the most important approaches to image segmentation [48]. It is designed to extract objects that have characteristic grey level ranges. A simple algorithm for thresholding produces a binary image. The grey level of each pixel is compared with a threshold value. Those pixels with grey level equal to, or greater than, the threshold value are given a value equivalent to white and those with grey level less than the threshold value are given a value equivalent to black. Figure 5.8 shows the result of applying thresholding to produce a binary image.

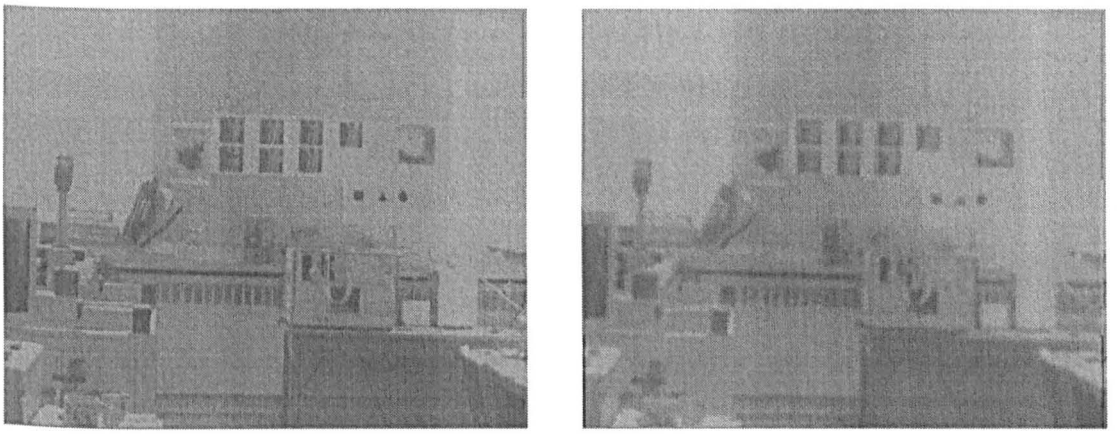


(a) Original Image (b) Binary Image

Figure 5.8 Result of Thresholding

5.4.2 Local Averaging

Local averaging is one of the methods used to reduce noise. This algorithm replaces the each pixel value with the average value of itself and its nearest neighbours. It should be noted that the degree of blurring in the resulting image is strongly proportional to the size of the window used. Figure 5.9 shows the result of applying this algorithm using a 7×7 window.

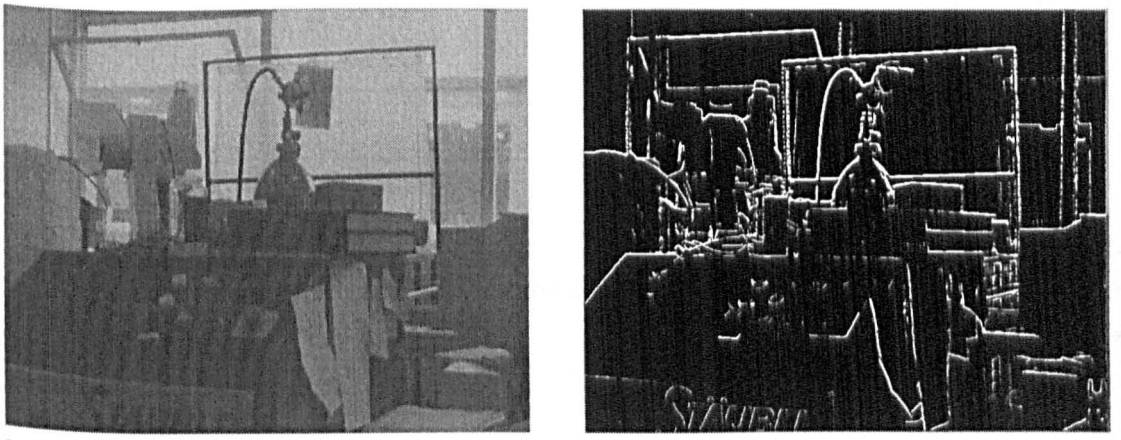


(a) Original Image (b) Result of Local Averaging

Figure 5.9 Result of Local Averaging with 7 × 7 Window

5.4.3 Sobel Operator

The Sobel operator is one of the edge detection algorithms which are primarily designed for image segmentation. It detects abrupt changes in grey level, and the detected areas indicate the end of one region and the beginning of another. Figure 5.10 shows the result of applying the Sobel operator.



(a) Original Image (b) Processed Image

Figure 5.10 Result of the Sobel Operator

In operation, two masks, X and Y, are convolved with the image to give an edge magnitude for each pixel, as shown in Figure 5.11. The horizontal edge mask gives:

$$\begin{aligned} GradH = & \{p(x-1,y-1) + 2(p(x,y-1)) + p(x+1,y-1)\} \\ & - \{p(x-1,y+1) + 2(p(x,y+1) + p(x+1,y+1)\} \end{aligned}$$

and the vertical edge mask gives:

$$\begin{aligned} GradV = & \{p(x+1,y-1) + 2(p(x+1,y)) + p(x+1,y+1)\} \\ & - \{p(x-1,y-1) + 2(p(x-1,y) + p(x-1,y+1)\} \end{aligned}$$

The gradient magnitude is approximated by | GradH | + | GradV | and the gradient direction is calculated by $\tan^{-1}$(GradV/GradH).

1 (X-1,Y-1)	2 (X,Y-1)	1 (X+1,Y-1)
0 (X-1,Y)	0 (X,Y)	0 (X+1,Y)
-1 (X-1,Y+1)	-2 (X,Y+1)	-1 (X+1,Y+1)

(a) Horizontal Edge Mask

-1 (X-1,Y-1)	0 (X,Y-1)	1 (X+1,Y-1)
-2 (X-1,Y)	0 (X,Y)	2 (X+1,Y)
-1 (X-1,Y+1)	0 (X,Y+1)	1 (X+1,Y+1)

(b) Vertical Edge Mask

Figure 5.11 Sobel Operator Masks

5.4.4 Seam Tracking

Software was also implemented to track raised seam edges using a structured lighting technique. This algorithm could be applied to such tasks as robotic cutting and welding. A low intensity laser stripe is used to detect the contours of a seam. The seam follower uses a laser stripe to produce a fine line which follows the contour of the seam. The laser line is seen by a CCD camera and an image is captured by the dedicated image processing board. In order to eliminate other sources of illumination, which could obscure the seam position, the laser

line is viewed through a band pass filter. After the filter, the image appears as a bright distorted white line on a black background. Figure 5.12 shows the setting for seam tracking. The picture at the top right corner of this figure is an image viewed through a band pass filter.

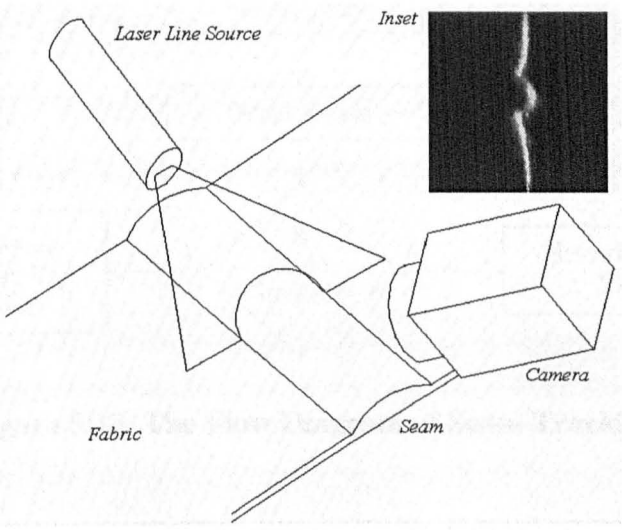


Figure 5.12 The Setting of Seam Tracking

High speed computation using pixel thinning is used to observe the contour of the laser line. When the laser line illuminates a seam, there are two points at which critical changes occur in the shape of the laser line. These two points are determined by a differential operator which produces an array of gradient data. The positions of these two points correspond to the edges of the seam, and hence the position of the seam can be determined. Prior to the seam tracking, the whole image is scanned, but during tracking, a smaller window covering the current seam location is scanned so that the overall processing time is reduced. If the software looses tracking, the whole image is scanned again until the position of the seam is determined.

Figure 5.13 shows the flow chart of the seam tracking algorithm. Figure 5.14 shows a sample image with the results overlaid on it. The result of finding the laser line profile, which is in black colour, is overlaid on the original laser line and the result of differential operations, which is in white colour, is shown on the right of the line. The two horizontal black lines denote the position of the seam.

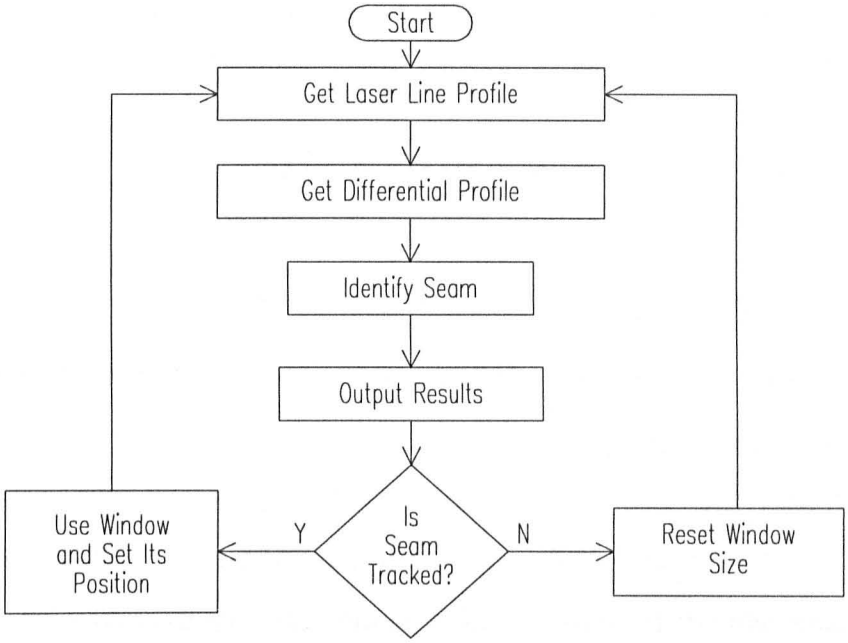


Figure 5.13 The Flow Diagram of Seam Tracking

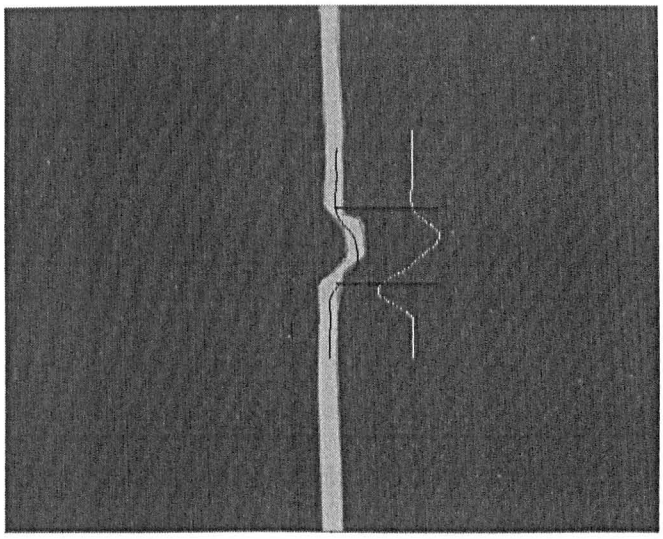


Figure 5.14 Image from the Seam Tracking

Chapter 6

System Performance

This chapter presents the results of a number of tests which were designed to investigate the performance of the image processing system for different configurations. The system performance varies with the test conditions which include the system configuration, the processor speeds of the connected transputers, the software environment and the structure of the test procedures; the relevant test conditions are stated in each test section.

6.1 Image Transmission Rates

This section examines the speed with which the dedicated image processing board can send and receive a full image and also send part of the full image to the external T9000 transputer network, in different configurations. The size of the full image was 640 pixels by 512 lines, and there were four system configurations used for these tests.

The connected transputers were 25 MHz T9000 transputers of revision gamma E03. All connected data links were set to 100 Mbits/sec. The test programs were written in the T9000 ANSI C language [17] and the compiled test programs were run under the **irun** environment [49].

6.1.1 System Configurations

For convenience, the four configurations were named so that it was much easier to refer to them in the following two sections. T(3,4) was given to the configuration which involved three T9000 transputers and four connected data links for image transmission. T(3,2) was given to the configuration which

involved three T9000 transputers and two connected data links for image transmission. T(2,2) was given to the configuration which involved two T9000 transputers and two connected data links for image transmission. Finally, T(2,1) was given to the configuration which involved two T9000 transputers and one connected data link for image transmission. These four configurations are shown in Figure 6.1, Figure 6.2, Figure 6.3 and Figure 6.4 respectively.

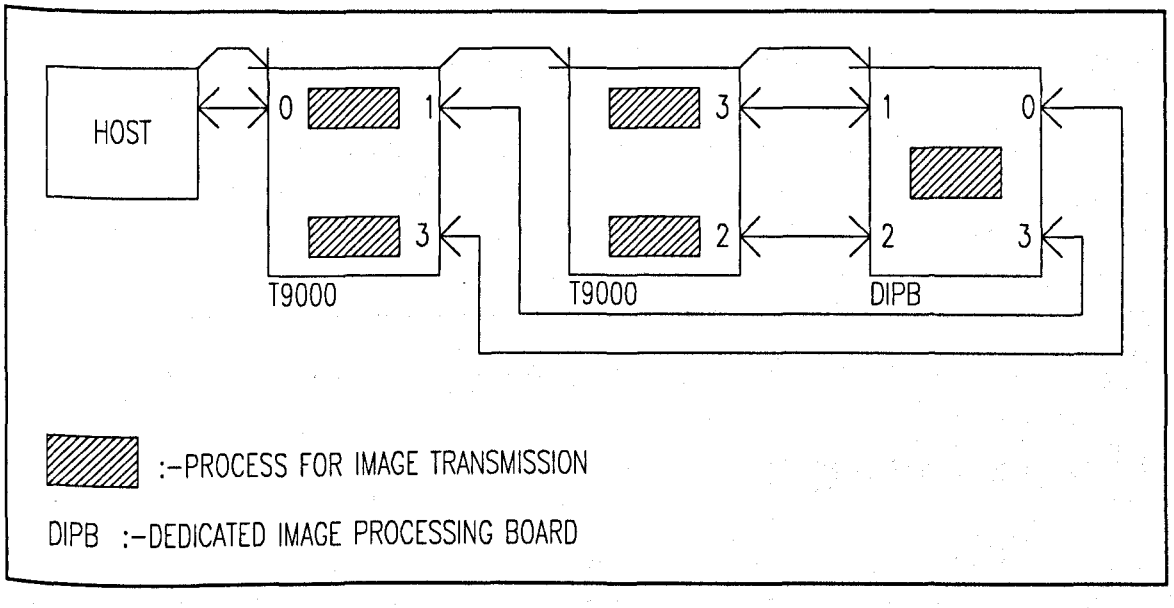


Figure 6.1 Configuration T(3,4)

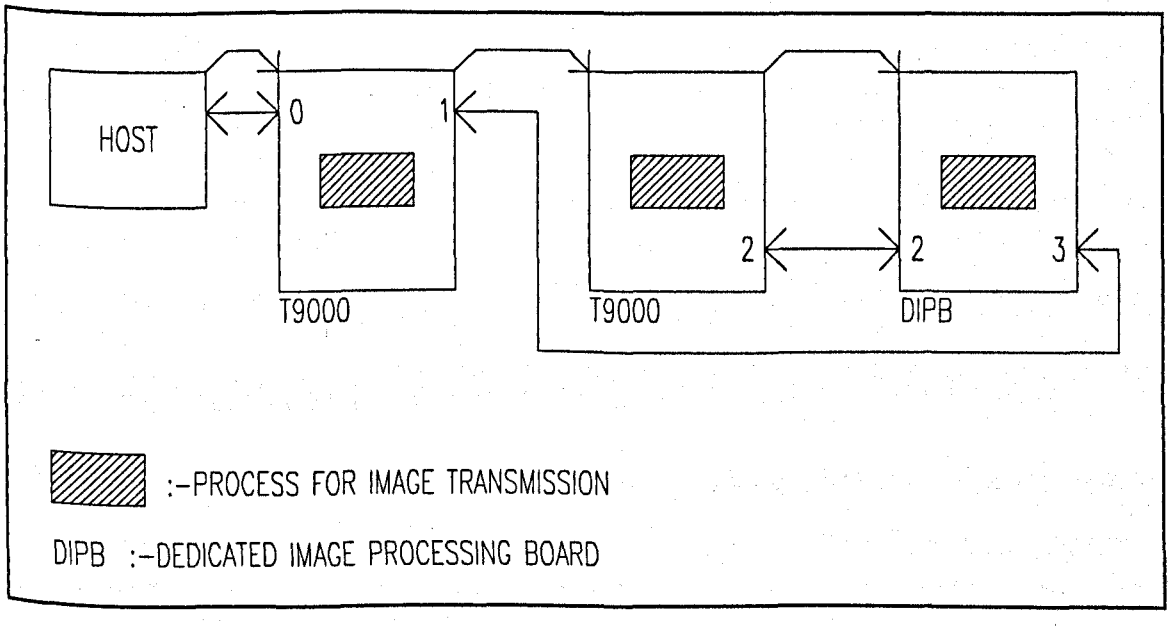


Figure 6.2 Configuration T(3,2)

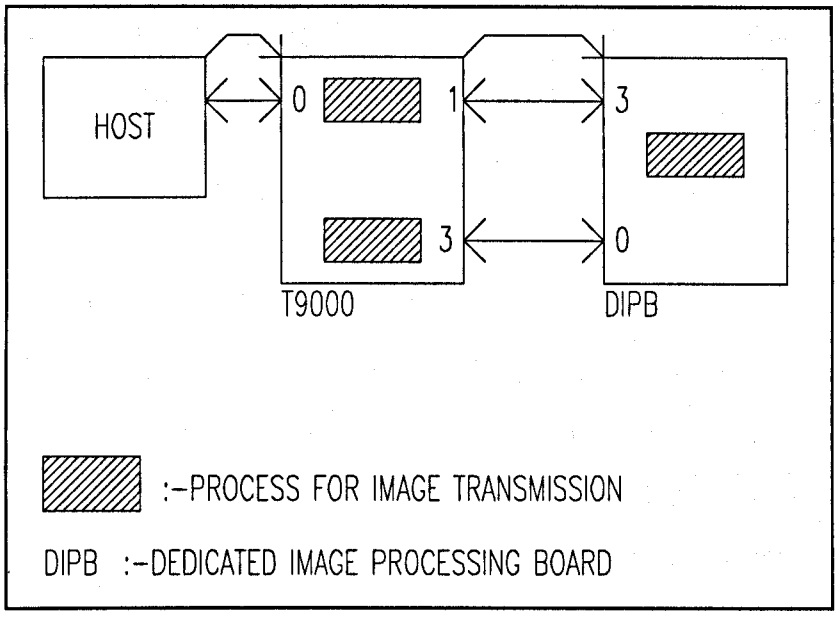


Figure 6.3 Configuration T(2,2)

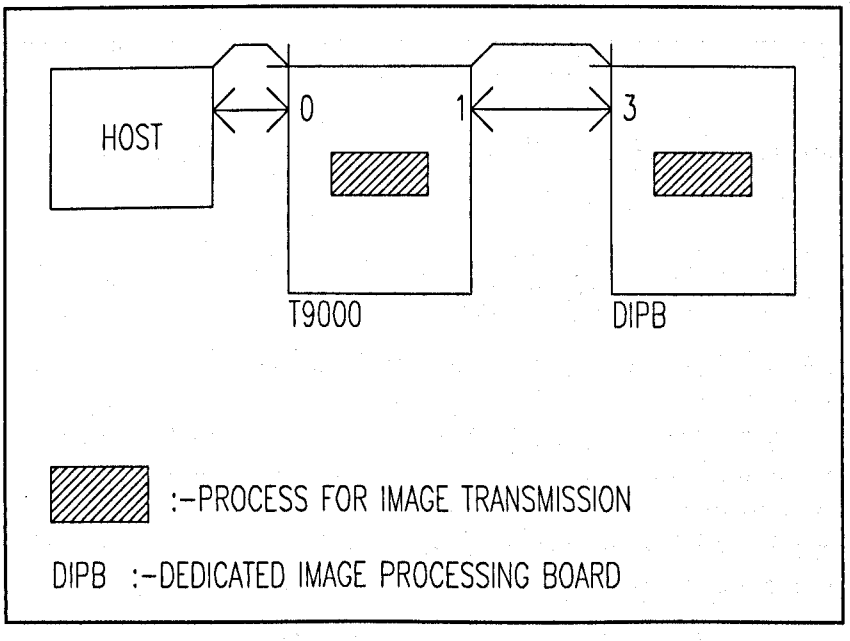


Figure 6.4 Configuration T(2,1)

6.1.2 Unidirectional Transmission

For the unidirectional transmission tests in Section 6.1.2.1, one image was sent out from the dedicated image processing board to the connected T9000 transputer network. For the tests in Section 6.1.2.2, one image was sent out from the dedicated image processing board to the connected T9000 transputer network and then received back.

The data links in between transputers were only used for this data transmission during the testing, and they were not shared with other channels for other purposes.

6.1.2.1 Sending One Image

The width of the image was varied from 512 pixels to 640 pixels for the tests. Real-time image transmission was achieved for a range of image sizes. Using the PAL video standard at 25 frames per second requires image transmission to take place in under 40 ms for real-time performance. Figure 6.5 shows the results of these tests and Table 6.1 lists the maximum width of an image, with which the image can be transmitted in real-time.

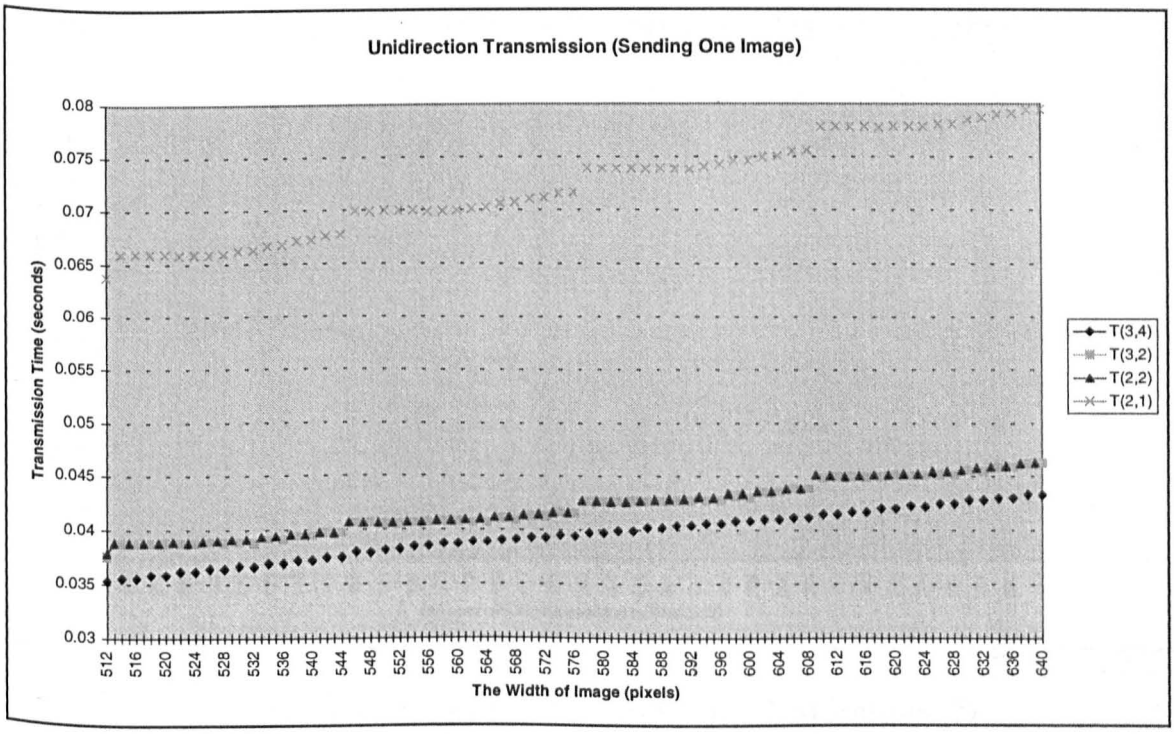


Figure 6.5 Unidirectional Transmission Test Results (1)

Configurations	T(3,4)	T(3,2)	T(2,2)	T(2,1)
Maximum Width (pixels)	580	544	544	< 512

Table 6.1 Unidirectional Transmission Test Results for Real-time Performance

6.1.2.2 Sending One Image and Receiving One Image

For some image processing techniques, the full image is divided into an upper and lower part. The upper part of the image is processed by the DIPB and the lower part of the image is transmitted to the connected T9000 transputer network for processing. The image processed by the network is then sent back to the DIPB for display. The tests in this section were to find out the overall transmission time for sending and receiving different sizes of images using different configurations. The size of the upper part of the image was described in terms of the number of lines. Figure 6.6 shows the results of these tests and Table 6.2 lists the results of the tests in which a full image was transmitted.

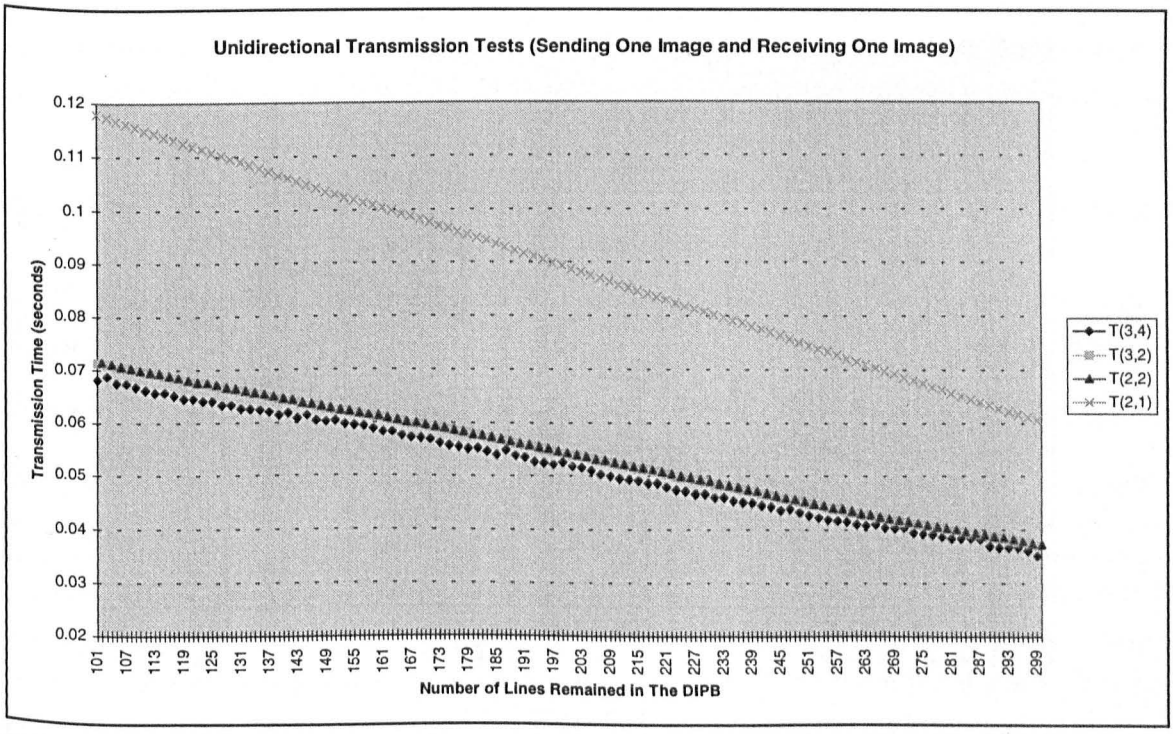


Figure 6.6 Unidirectional Transmission Test Results (2)

Configurations	T(3,4)	T(3,2)	T(2,2)	T(2,1)
Transmission Time (s)	0.0842	0.0887	0.0892	0.1466

Table 6.2 Unidirectional Transmission Test Results for a Full Image

6.1.3 Bidirectional Transmission

For the bidirectional transmission tests, two images were sent simultaneously, one from the dedicated image processing board to the connected transputer network, and the other from the connected T9000 transputer network to the dedicated image processing board. The data links in between the transputers were also reserved for this transmission only, and the division of the full image was identical to that used in the unidirectional transmission tests. Figure 6.7 shows the results of the tests in which only part of the full image was transmitted and Table 6.3 lists the full image transmission test results.

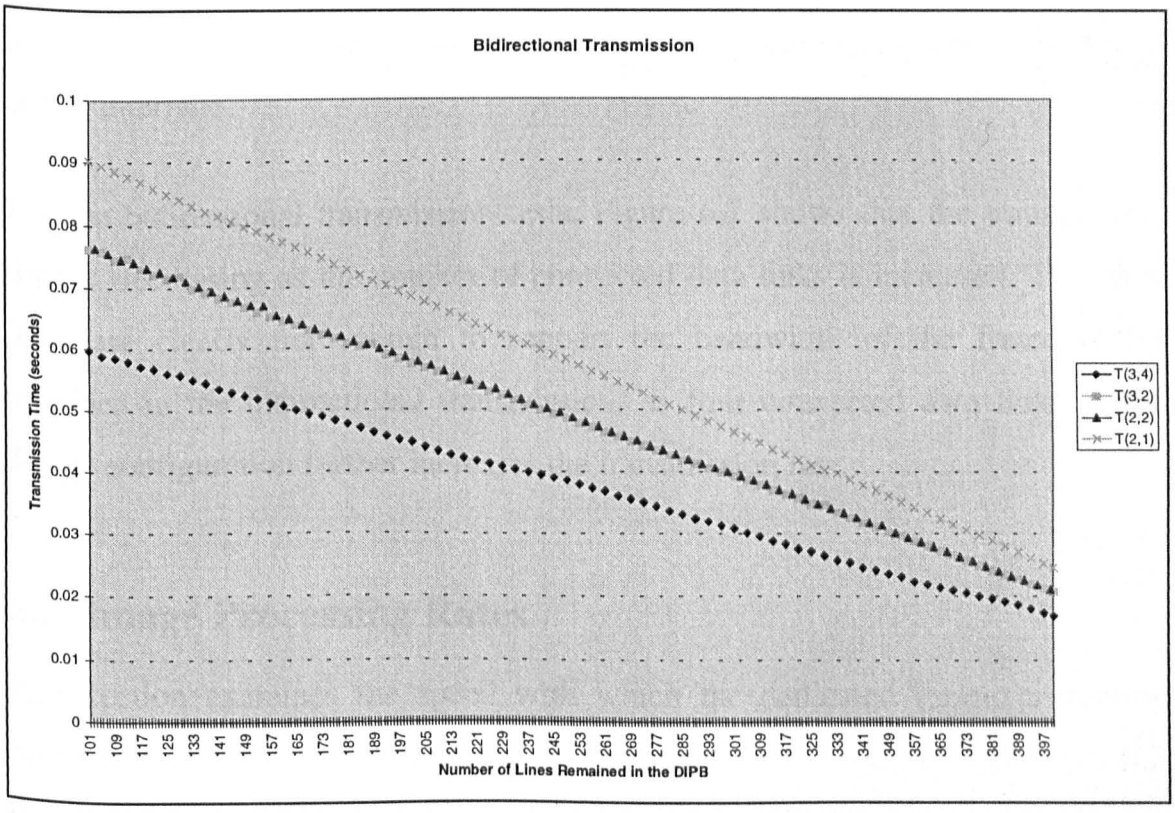


Figure 6.7 Bidirectional Transmission Test Results

Configurations	T(3,4)	T(3,2)	T(2,2)	T(2,1)
Transmission Time (s)	0.0736	0.0944	0.0954	0.1123

Table 6.3 Bidirectional Transmission Test Results for a Full Image

6.1.4 Conclusions

For the unidirectional transmission tests, Figure 6.5 and Figure 6.6 show that the transmission time for the configurations T(3,4), T(3,2), and T(2,2) are very close to each other, but the transmission time for the configuration T(2,1) is exceptionally high. The transmission rate through each link is limited by its bandwidth. Using two data links, however, reduced the transmission rate per link sufficiently to overcome this problem whilst still supporting the bandwidth of the frame buffer interface, in the unidirectional transmission. As a result, no significant reduction in transmission time was achieved by using the four connected data links in the T(3,4). For the configurations with two or more data links connected, the transmission rate is limited by the bandwidth of the frame buffer interface.

For the bidirectional transmission tests, Figure 6.7 shows that the transmission time is decreasing as the number of connected data links is increased. Two data links are clearly not enough to support the bandwidth of the frame buffer interface in the bidirectional transmission, as four connected data links in the T(3,4) configuration further increases the transmission rate.

6.2 Image Processing Rates

This section examines the speed with which the dedicated image processing board with or without the connected T9000 transputer network, can process a full image, using different image processing algorithms and different configurations.

The size of the full image was 640 pixels by 512 lines, and there were five system configurations set for the tests. The connected transputers were 25 MHz T9000 transputers of revision gamma E03. All the connected data links were set to 100 Mbits/sec. The test programs were written in T9000 ANSI C language [17] and the compiled test programs were run under the **irun** environment [49].

When performing the algorithms for local averaging and for the Sobel operator, each pixel in the image is read more than once. Since the frame buffer memory of the dedicated image processing board is not cacheable, but program memory is cacheable, image lines were purposely buffered in the program memory in the form of arrays, so that they would be stored into the cache memory and hence the processing time would be reduced.

6.2.1 System Configurations

For referencing purposes, the five configurations under test were given names. They were named $P(n_t, n_d, n_p)$, where n_t is the number of T9000 transputers, n_d is the number of connected data links, and n_p is the number of image processing processes. These five configurations are shown in Figures 6.8-6.12.

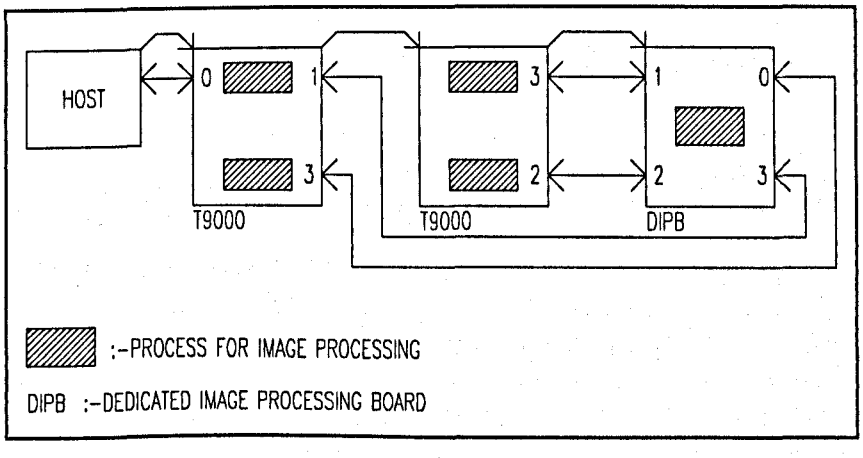


Figure 6.8 Configuration P(3,4,5)

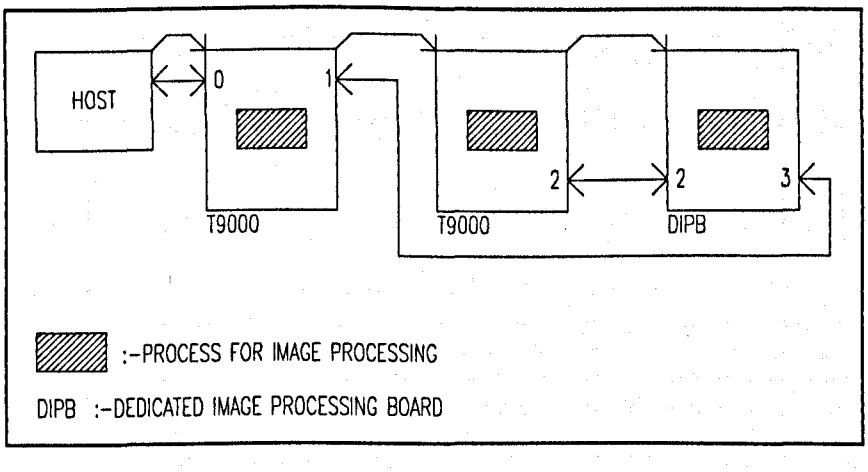


Figure 6.9 Configuration P(3,2,3)

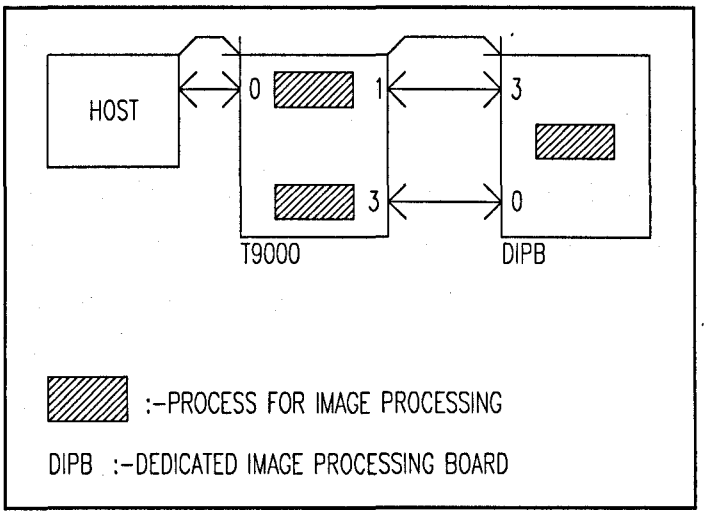


Figure 6.10 Configuration P(2,2,3)

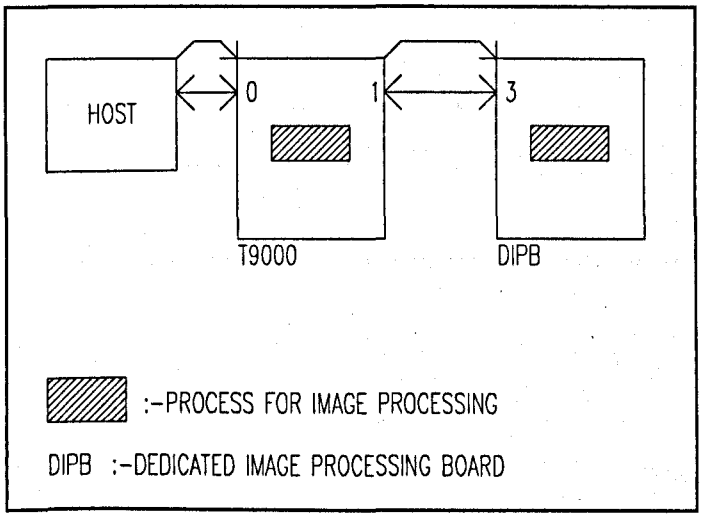


Figure 6.11 Configuration P(2,1,2)

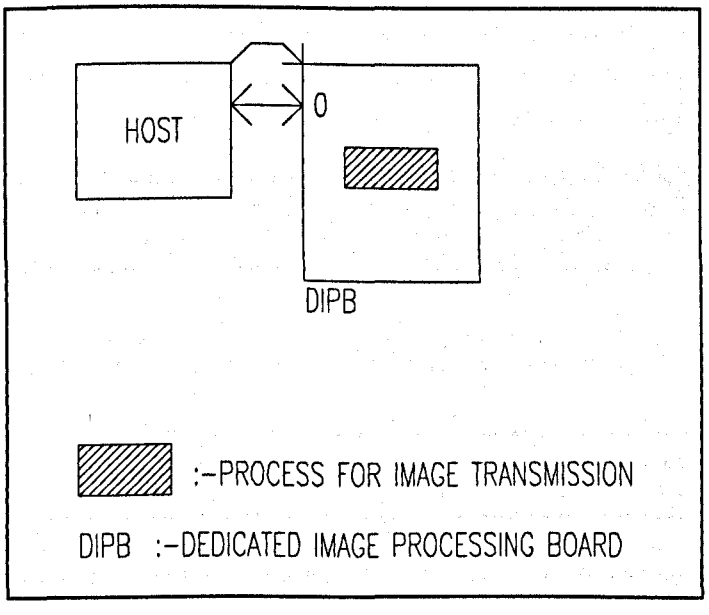


Figure 6.12 Configuration P(1,0,1)

6.2.2 Thresholding

The system performance when implementing the thresholding algorithm, as described in Section 5.4.1, is shown in Figure 6.13. Table 6.4 shows the fastest time that could be achieved in the different configurations. The routine for the thresholding algorithm is shown below.

```
for (y = 0; y < y_size; y++)
 {
  for (x = 0; x < x_size; x++)
    {
     if (image[y][x] > edge)
        result = 255;
     else
        result = 0;
     image[y][x] = result;
    }
 }
```

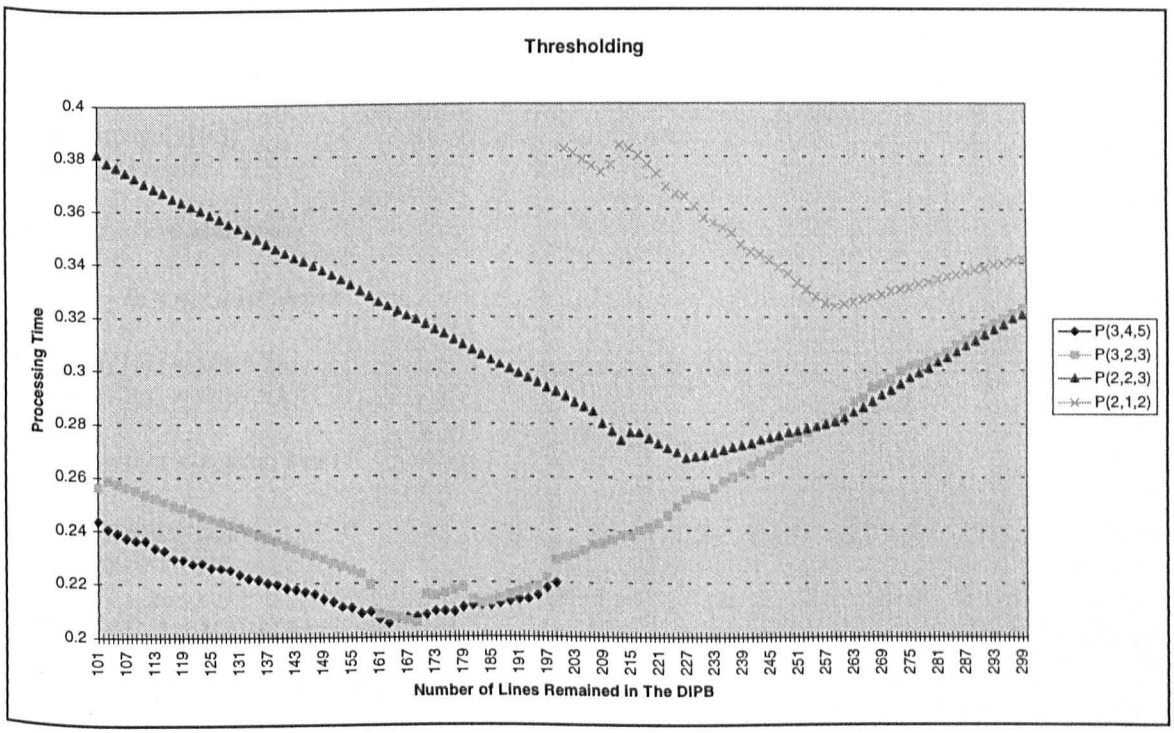


Figure 6.13 Thresholding Test Results

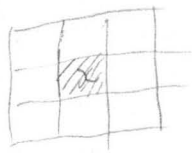
Configurations	P(3,4,5)	P(3,2,3)	P(2,2,3)	P(2,1,2)	P(1,0,1)
Processing Time (s)	0.2053	0.2056	0.2671	0.3237	0.5656

Table 6.4 The Fastest Processing Time in Thresholding

6.2.3 Local Averaging

The local averaging algorithm was selected to test the system performance, using 3 × 3, 5 × 5 and 7 × 7 windows. This algorithm was described in Section 5.4.2. The results for each window size are shown in Figure 6.14, Figure 6.15 and Figure 6.16, and Table 6.5 lists the fastest time that could be achieved in the different configurations. The routine for local averaging with a window size of 3 × 3 is shown below.

```
#define LOCAL_AVERAGE3(x)  (char) ((array1[x-1]+array1[x]+array1[x+1]+array2[x-1]+array2[x]
                                    +array2[x+1]+array3[x-1]+array3[x]+array3[x+1])/9)

y = 0;
for (x = 0; x < x_size; x++)
  {
   pix = image[y][x];
   array2[x] = pix;
  }
y = 1;
for (x = 0; x < x_size; x++)
  {
   pix = image[y][x];
   array3[x] = pix;
  }
for (y = 2; y < y_size; y++)
  {
   for (x = 0; x < x_size; x++)
     {
      array1[x] = array2[x];
      array2[x] = array3[x];
     }
   for (x = 0; x < x_size; x++)
     {
      pix = image[y][x];
      array3[x] = pix;
     }
   for (x = 0; x < x_size; x++)
     {
      result = LOCAL_AVERAGE3(x);
      image[y][x] = result;
     }
```



Configurations	Processing Time (seconds)				
	P(3,4,5)	P(3,2,3)	P(2,2,3)	P(2,1,2)	P(1,0,1)
Local Averaging 3x3	0.4145	0.4449	0.5946	0.6345	1.1134
Local Averaging 5x5	0.6088	0.6428	0.8929	0.9450	1.7309
Local Averaging 7x7	0.9025	0.9226	1.3192	1.3577	2.5674

Table 6.5 The Fastest Processing Time in Local Averaging

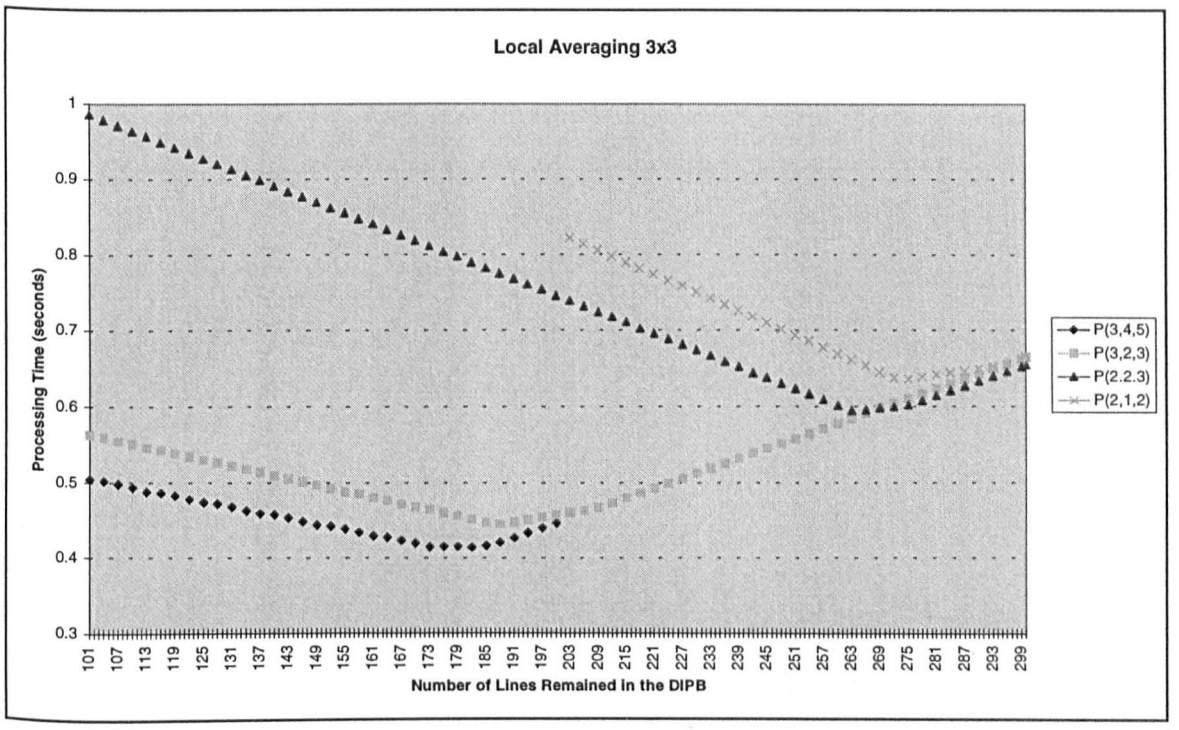


Figure 6.14 Local Averaging 3 × 3 Test Results

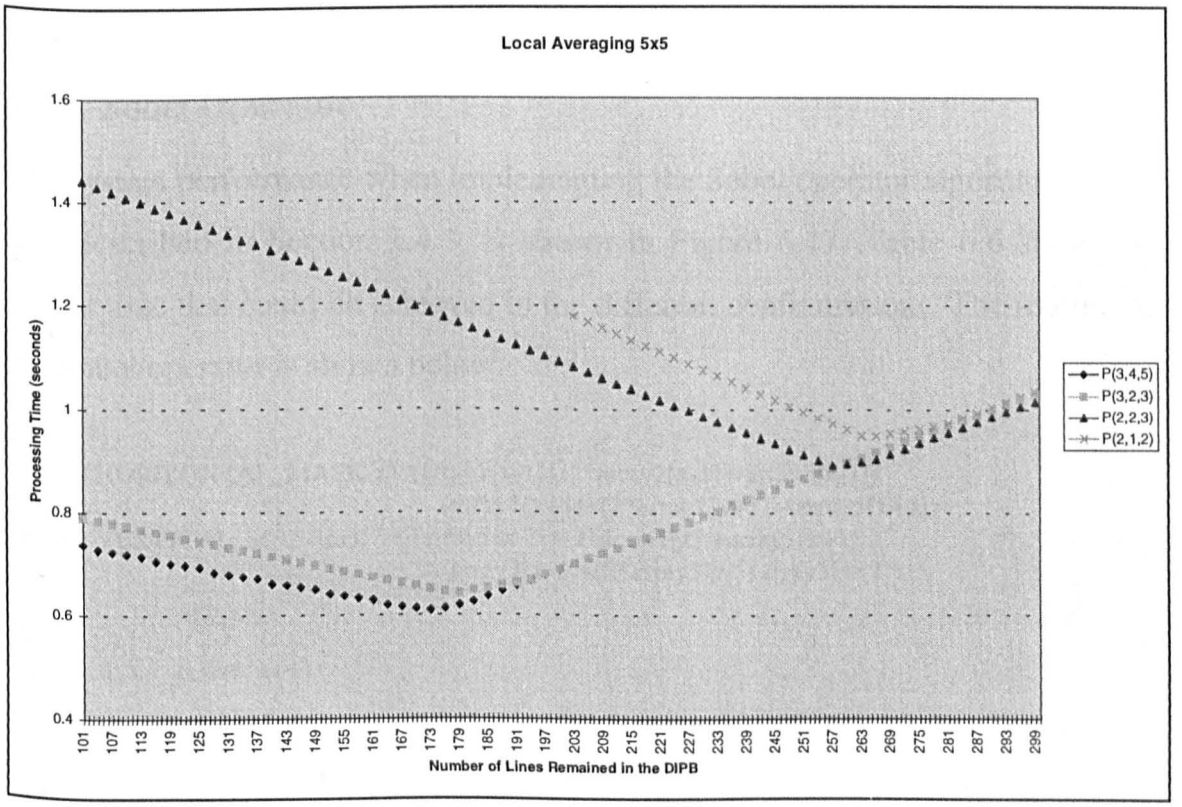


Figure 6.15 Local Averaging 5 × 5 Test Results

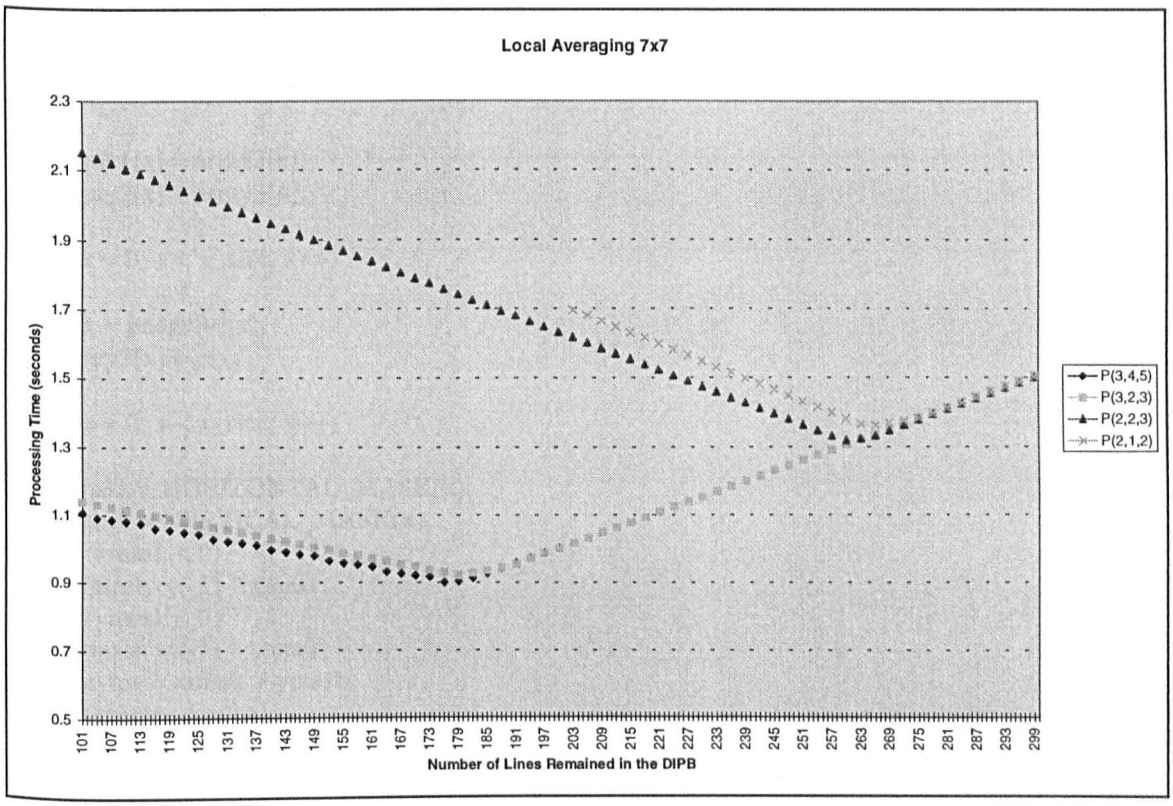


Figure 6.16 Local Averaging 7 × 7 Test Results

6.2.4 Sobel Operator

The system performance when implementing the Sobel operator algorithm, which was described in Section 5.4.3, is shown in Figure 6.17. Table 6.6 shows the fastest time that could be achieved in the different configurations. The routine for the Sobel operator is shown below.

```
#define HORIZONTAL_MASK(x)  (array1[x-1]+(2*array2[x-1])+array3[x-1])
                              - (array1[x+1]+(2*array2[x+1])+array3[x+1])
#define VERTICAL_MASK(x)    (array1[x-1]+(2*array1[x])+array1[x+1])
                              - (array3[x-1]+(2*array3[x])+array3[x+1])

y = 0;
for (x = 0; x < x_size; x++)
  {
    pix = image[y][x];
    array2[x] = pix;
  }
y = 1;
for (x = 0; x < x_size; x++)
  {
    pix = image[y][x];
    array3[x] = pix;
  }
```

```
for (y = 2; y < y_size; y++)
 {
  for (x = 0; x < x_size; x++)
   {
    array1[x] = array2[x];
    array2[x] = array3[x];
   }
  for (x = 0; x < x_size; x++)
   {
    pix = image[y][x];
    array3[x] = pix;
   }
  for (x = 0; x < x_size; x++)
   {
    xmask = HORIZONTAL_MASK(x);
    ymask = VERTICAL_MASK(x);
    if (xmask < 0)
     xmask = (-1) * xmask;
    if (ymask < 0)
     ymask = (-1) * xmask;
    greylev = xmask + ymask;
    if (greylev > edge)
     result = 255;
    else
     result = 0;
    image[y][x] = result;
   }
 }
```

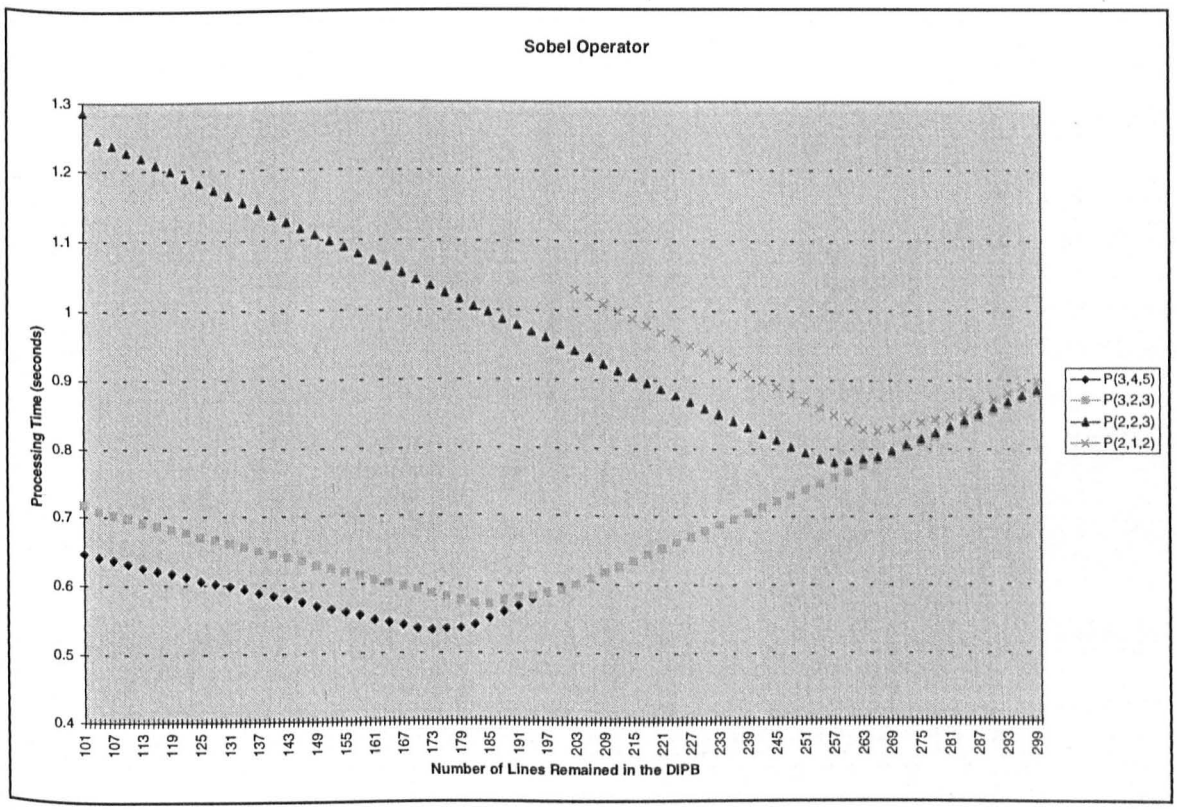


Figure 6.17 Sobel Operator Test Results

Configurations	P(3,4,5)	P(3,2,3)	P(2,2,3)	P(2,1,2)	P(1,0,1)
Processing Time (s)	0.5356	0.5701	0.7792	0.8250	1.4526

Table 6.6 The Fastest Processing Time in Sobel Operator

6.2.5 Seam Tracking

The seam tracking algorithm, which was described in Section 5.4.4, can be implemented in real-time in Single-T9 Mode. The performance of the system when implementing this algorithm was tested for the configuration P(1,0,1) only.

The processing time for seam tracking is related to the size of the window which contains the laser line contour. The window must be large enough to ensure that the area of the seam, which is being tracked, is still inside the window after one frame time. Different window sizes were set to test the processing speed. The width of the window was fixed at 80 pixels but the height of the window was varied from 81 pixels to 250 pixels. Figure 6.18 shows the test results.

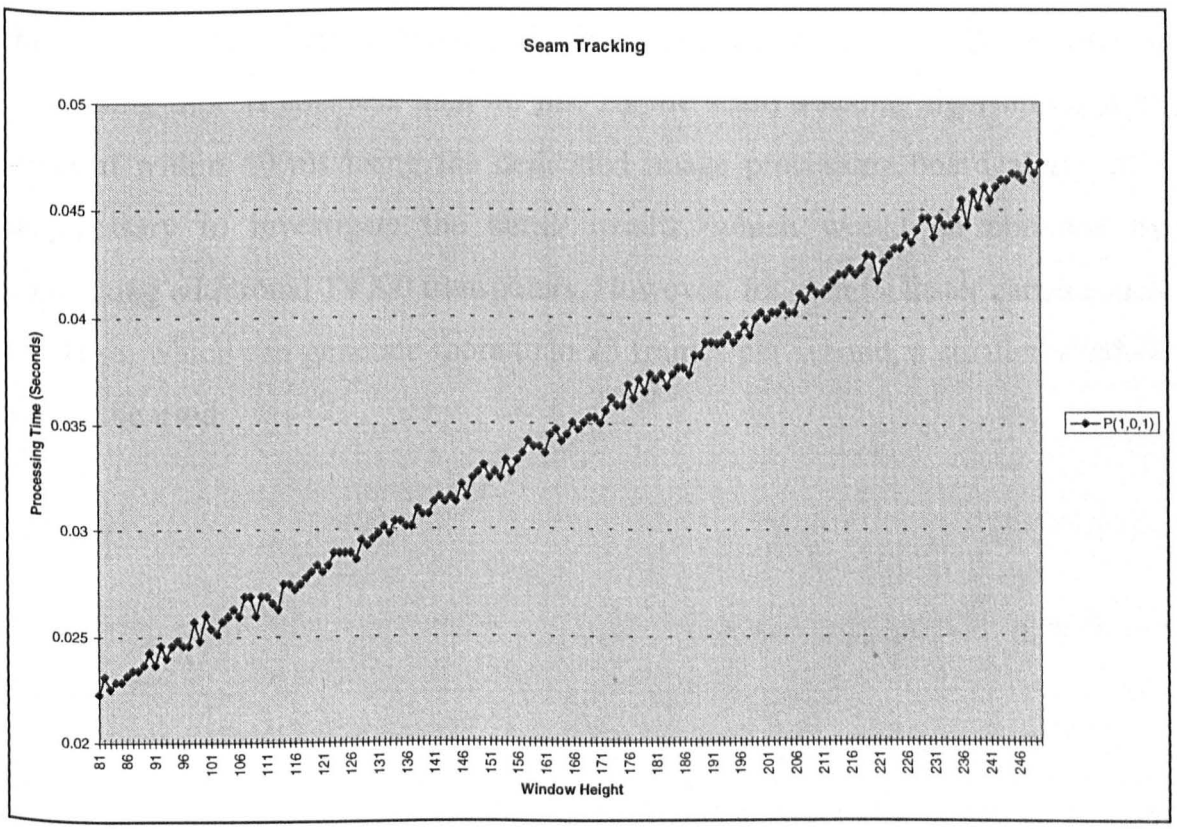


Figure 6.18 Seam Tracking Test Results

6.2.6 Conclusions

The results of thresholding, local averaging and the Sobel operator show a common feature of a diminishing return. The diminishing return is the point where the fastest processing speed is achieved. For applications requiring fast processing time, the diminishing return should be found so that the system processing power is optimised.

For the configurations P(3,4,5) and P(3,2,3), the number of transputers involved in computation is the same, but their processing times are different. The results show that when two more data links are added to support data transmission, they actually reduce the processing time. In addition, at the diminishing return, the number of image lines allocated in the dedicated image processing board in the configuration P(3,2,3) is greater than that in the configuration P(3,4,5). For the configurations P(2,2,3) and P(2,1,2), a similar situation occurs.

For the results of seam tracking, the processing time is roughly proportional to the height of the window. Even if the height of the window is 200 pixels, the processing time is still less than 40 ms. As the seam tracking algorithm can be finished within 40 ms using the dedicated image processing board alone, it is unnecessary to investigate the better results, which would be obtained by connecting additional T9000 transputers. However, for using a faster camera such as Dalsa, which can generate more than 25 frames per second, a smaller window should be used.

Chapter 7

Conclusions and Future Research

This research has led to the design and construction of a generic T9000 transputer based vision system, which is able, in real-time, to capture and process image data, and to display the resulting image. The triple-ported memory interface was successfully designed to integrate the three functions, capturing, processing and displaying an image, onto a single board. From the evaluation results, it has been shown that the processing power of the single board can be enhanced by connecting additional T9000 transputers for shared computation.

By using Altera programmable logic devices, all the necessary combinational logic and state machines have been provided by two EPM7256EGC192-12 chips. As a result, the dedicated image processing board contains only thirty-three integrated circuit chips.

The results in Chapter 6 show that the processing power varied for different system configurations and was dependent on the size of the sub-image assigned to the connected T9000 transputers. In order to optimise the system for a fixed number of T9000 transputers, experiments must be performed to find the configuration and the sub-image size for which the highest processing power is obtained.

The image processing algorithm for seam tracking, is a good example for highlighting the main feature of this image processing system, since this algorithm can be implemented in real-time in Single-T9 mode. As a matter of fact, it is not necessary to involve more than one T9000 transputers for any application.

The result of bidirectional transmission tests show that the dedicated image processing board is not able to send and receive a whole image with size of 640 pixels by 512 lines simultaneously in real-time. For some real-time applications it will be necessary to display a whole processed image, if it is the case an additional dedicated image processing board will be required for the display function.

The performance of the system was evaluated for a non-routed network only, there was no IMS C104 packet routing switch involved. Further evaluation can be done for a routed network with the IMS C104, so that the potential power of the image processing system could be further explored.

The input event channels of the T9000 transputer can be used as interrupt input pins. An interrupt approach can then be employed to synchronise the T9000 transputer with the incoming video signal. Unfortunately, this was not included in the system due to the time restrictions on the project. This, if added at a future date, would result in the polling approach and the interrupt approach both being available for use, and the system performance could be re-evaluated.

At present, the processor in the dedicated image processing board is a 20 MHz T9000 transputer with revision gamma E03, which is the slowest of the available T9000 transputers. For future research, a faster T9000 transputer could be used to increase the processing speed, particularly in Single-T9 mode.

Unfortunately, from the latest Internet news, the future of T9000 transputer is uncertain, the manufacturing of this product has already been terminated by SGS-THOMSON. If the T9000 transputer becomes obsolete, there would be no more technical support for T9000 transputer based systems. In this case, upgrading the present system would mean replacing the T9000 transputer by another parallel processor. In order to avoid considerable modification in the hardware design, the memory interface of the parallel processor must be able to support a mixed

memory system, providing at least three sets of memory-control signals. Since the TMS320C40 digital signal processor can fulfil the above criterion, this could be used to replace the present processor.

References

1 Chambers S P: "TIPS: A Transputer Based Real-Time Vision System", Ph.D Thesis, University of Liverpool, U.K.,1990, pp62.

2 Chambers S P: "TIPS: A Transputer Based Real-Time Vision System", Ph.D Thesis, University of Liverpool, U.K.,1990.

3 Jeremy Hinton & Alan Pinder, "Transputer Hardware and System Design", Prentice Hall, 1993, Chapter One.

4 "The T9000 Transputer Hardware Reference Manual", Inmos Ltd, Almondsbury, Bristol, U.K.,1993.

5 "TMS320C4x User's Guide", Texas Instruments, Houston, Texas, U.S.A.,1996.

6 "Quintek QT9 Multi-Capture and Multi-Display System", Quintek Ltd., 1995.

7 "SMT303 User Guide", Sundance Multiprocessor Technology Ltd, 1995.

8 "SMT304 User Guide", Sundance Multiprocessor Technology Ltd, 1995.

9 "IMS B927" Data Sheet, Inmos Ltd, Almondsbury, Bristol, U.K., 1994.

10 "IMS B926" Data Sheet, Inmos Ltd, Almondsbury, Bristol, U.K., 1994.

11 "IMS B108" Data Sheet, Inmos Ltd, Almondsbury, Bristol, U.K., 1994.

12 "T9000 Toolset Hardware Configuration Manual", Inmos Ltd, Almondsbury, Bristol, U.K., 1994.

13 Martin Bolton, "Digital Systems Design with Programmable Logic", Addition-Wesley, 1990, Chapter 6.

14 John Uffenbeck, "The 8086/8088 Family, Design, Programming, and Intefacing", Prentice-Hall, 1987, pp319-321.

15 Robert C. Hutchison, Steven B. Just, "Programming Using the C Language", McGraw-Hill, 1988.

16 D.A. Protopapas, "Microcomputer Hardware Design", Prentice-Hall International, 1988, pp139-140.

17 "T9000 ANSI C Toolset User Guide", Inmos Ltd, Almondsbury, Bristol, U.K., 1994.

18 "T9000 OCCAM 2 Toolset User Guide", Inmos Ltd, Almondsbury, Bristol, U.K., 1994.

19 "TMS320 Family Development Support Reference Guide", Texas Instruments, Houston, Texas, U.S.A., 1996.

20 "Raytheon Semiconductor 1994 Data Book", Raytheon, Mountain View, U.S.A., pp2.5-2.26.

21 Gordon J. King, "Beginner's Guide to colour Television", Newnes-Butterworths, 1973.

22 "MAX 7000 Programmable Logic Device Family" Data Sheet, Altera Corporation, Bucks, England, 1996.

23 "XC3000 Series Field Programmable Gate Arrays" Data Sheet, Xilinx, 1996.

24 "ELECTROSPEED" Catalogue, Hampshire, U.K., 1996, pp793.

25 "Altera Data Book", Altera Corporation, Bucks, England, June 1996, pp531-593.

26 "Development Systems Products Overview", Xilinx, 1996.

27 John Uffenbeck, "The 8086/8088 Family, Design, Programming, and Intefacing", Prentice-Hall, 1987, pp301.

28 "Hitachi IC Memory No. 3" Data Book, Hitachi, Berkshire, U.K., 1995, pp136-142.

29 "Hitachi IC Memory No. 3" Data Book, Hitachi, Berkshire, U.K., 1995, pp681-703.

30 "Memory ICs Data Book", Winbond Electronics Corporation, Taipei, Taiwan, 1994, pp83-89.

31 "LM1881 Video Sync Separator" Data Sheet, National Semiconductor Corporation, 1986.

32 "Bt481A RAMDAC" Data Sheet, Brooktree Corporation, San Diego, U.S.A.,1993.

33 "The Transputer Data Book", Third Edition, Inmos Ltd, Almondsbury, Bristol, 1992, pp71-396.

34 "The Transputer Data Book", Third Edition, Inmos Ltd, Almondsbury, Bristol, 1992, pp35-36.

35 "Networks, Routers and Transputers: Function, Performance, and Applications", Inmos Ltd, Almondsbury, Bristol, 1993, pp39-46.

36 "41MV, 41MW, and 41MX Dual Differential Transceiver Circuits" Data Sheet, AT&T Microelectronics, 1993.

37 "IMS B108 Installation and User Manual", Inmos Ltd, Almondsbury, Bristol, 1994, pp33-39.

38 Jeremy Hinton & Alan Pinder, "Transputer Hardware and System Design", Prentice Hall, 1993, pp174-179.

39 "MAX+PLUS II AHDL", Altera Corporation, Bucks, England, 1994.

40 "T9000 Toolset Hardware Configuration Manual", Inmos Ltd, Almondsbury, Bristol, 1994, pp155-173.

41 "Raytheon Semiconductor 1994 Data Book", Raytheon, Mountain View, U.S.A., pp2.11-2.12.

42 INMOS Limited, "Occam 2 Reference Manual", Prentice Hall International, 1988.

43 "Networks, Routers and Transputers: Function, Performance, and Applications", Inmos Ltd, Almondsbury, Bristol, 1993, pp46-54.

44 A. Rosenfeld and A.C. Kak, "Digital Picture Processing", Academic Press, 1976, pp258-275.

45 R.C. Gonzalez & P. Wintz, "Digital Image Processing", Second Edition, Addison-Wesley, 1987, pp161-162.

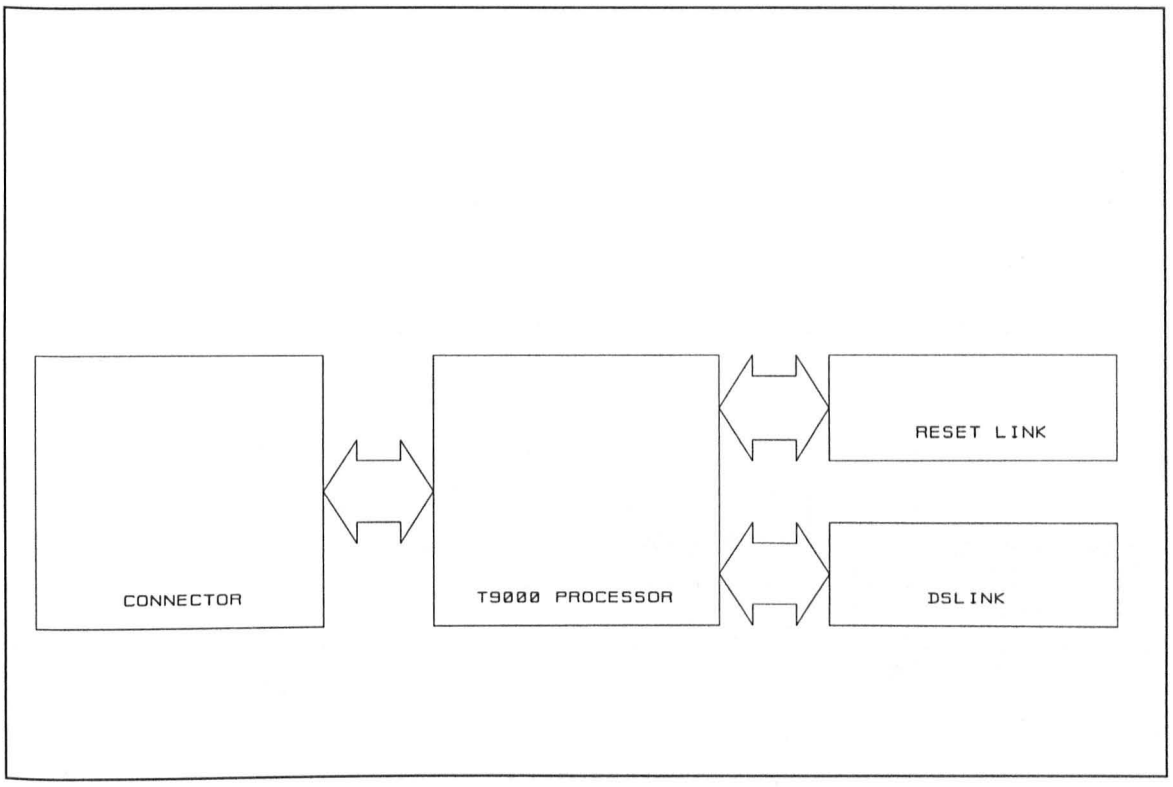
46 R.C. Gonzalez & P. Wintz, "Digital Image Processing", Second Edition, Addison-Wesley, 1987, pp336-338.

47 J.S. Smith, K.J. Howson, R.J. Chinneck and J. Lucas, "Vision systems for high speed seam tracking", Sixth International Conference, Computer Technology in Welding, Lanaken, Belgium, 9-12 June 1996, Paper 46.

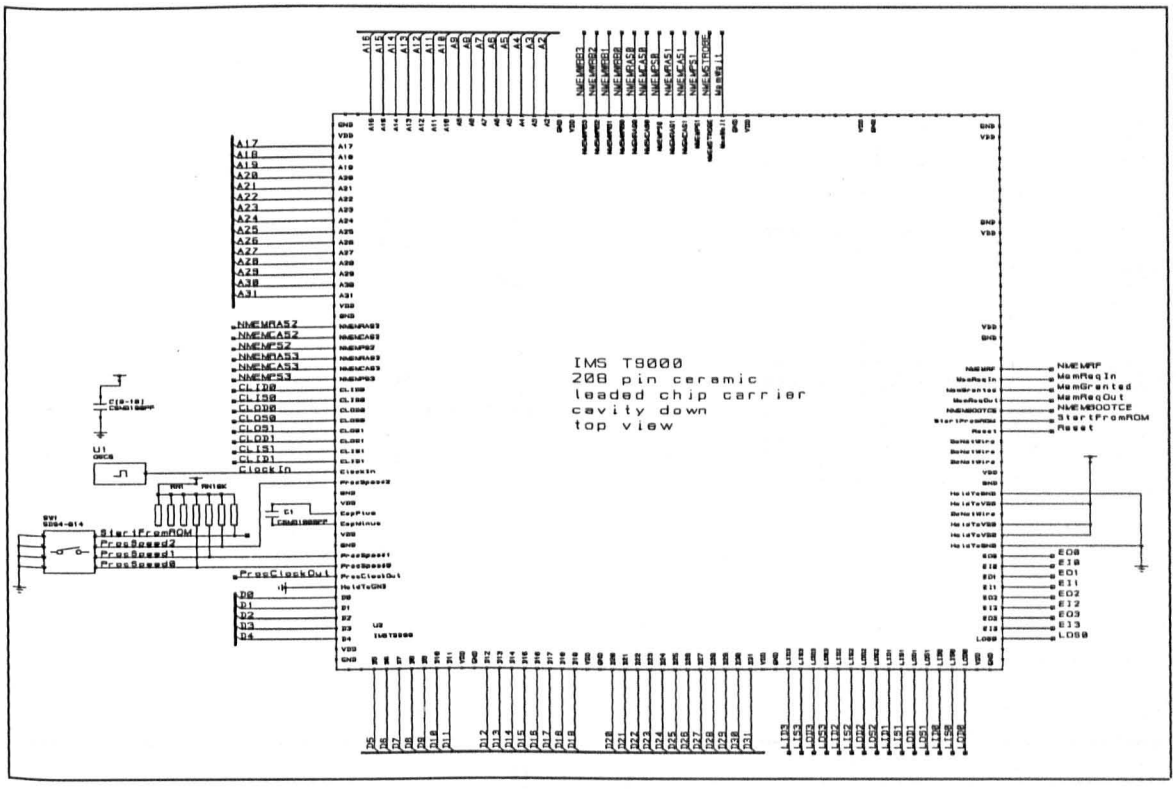
48 A. Rosenfeld and A.C. Kak, "Digital Picture Processing", Academic Press, 1976, pp256-258.

49 "Aserver Programmers' Guide", Inmos Ltd, Almondsbury, Bristol, 1994, pp7-14.

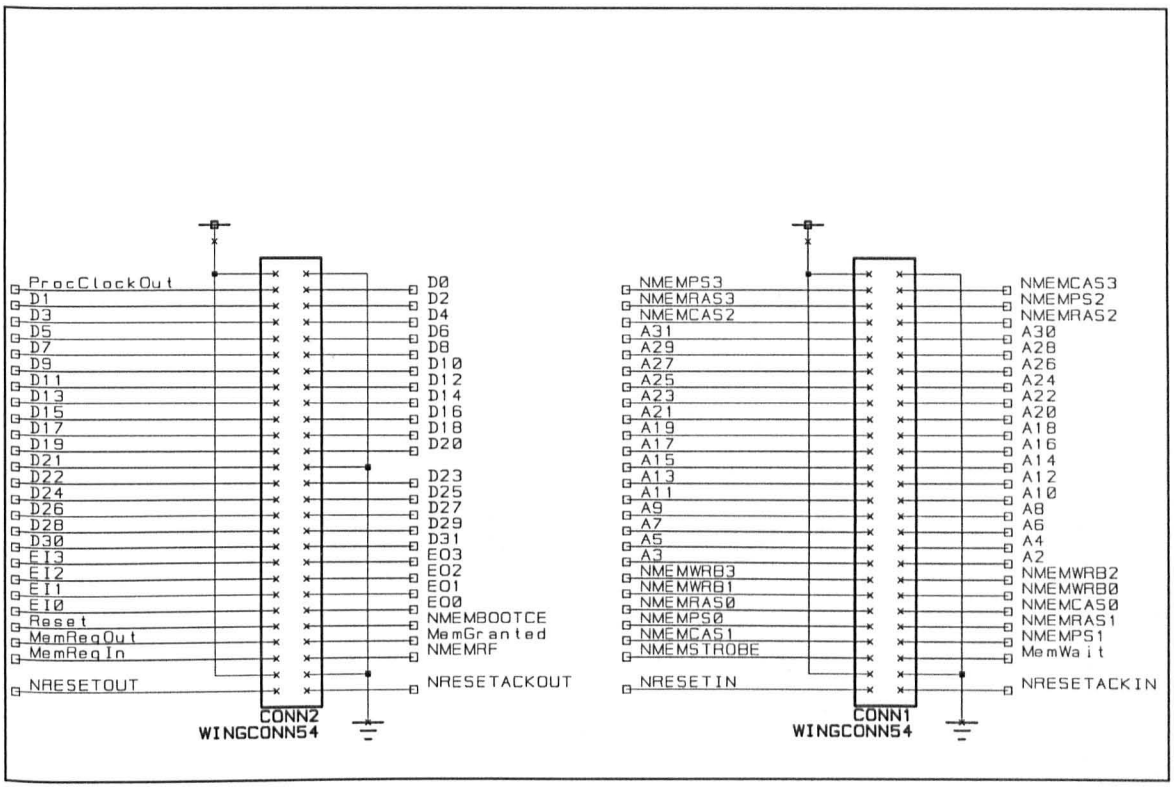
Appendix A - The Schematic Diagram of the T9000 Transputer Adaptor Board



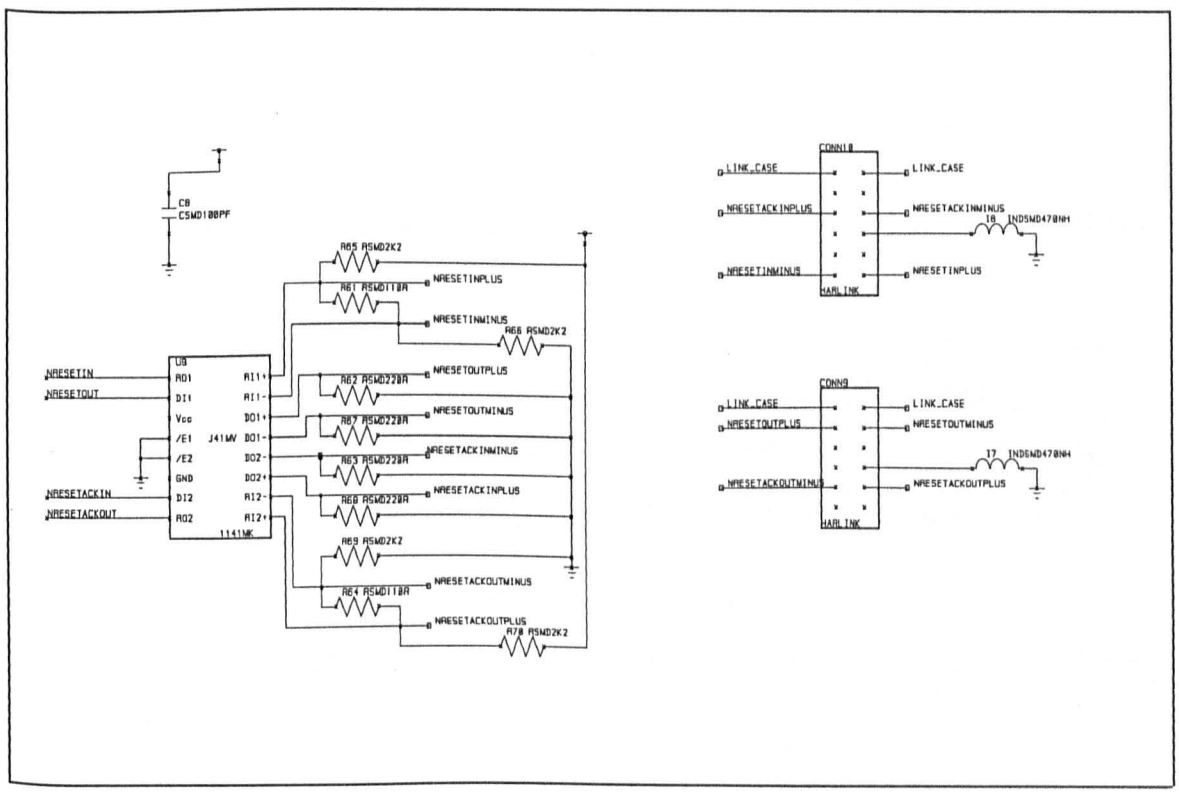
Top Level of the Hierarchy Display



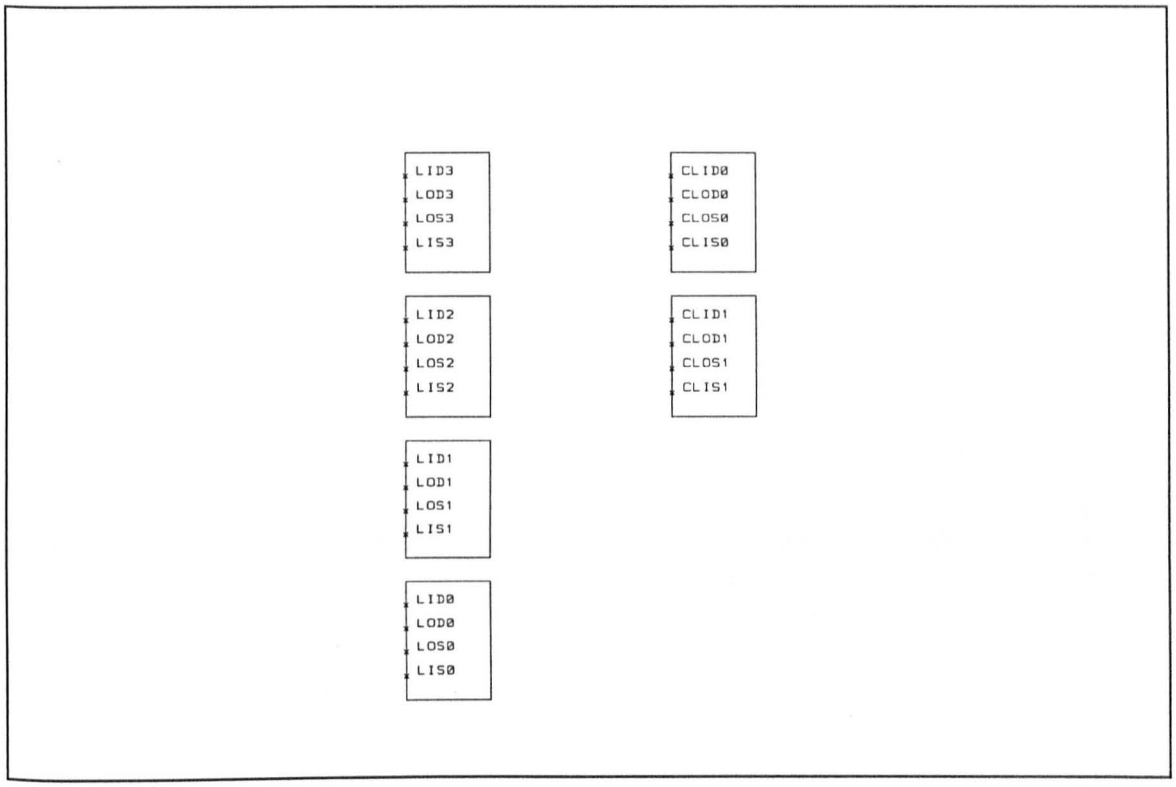
T9000 Processor



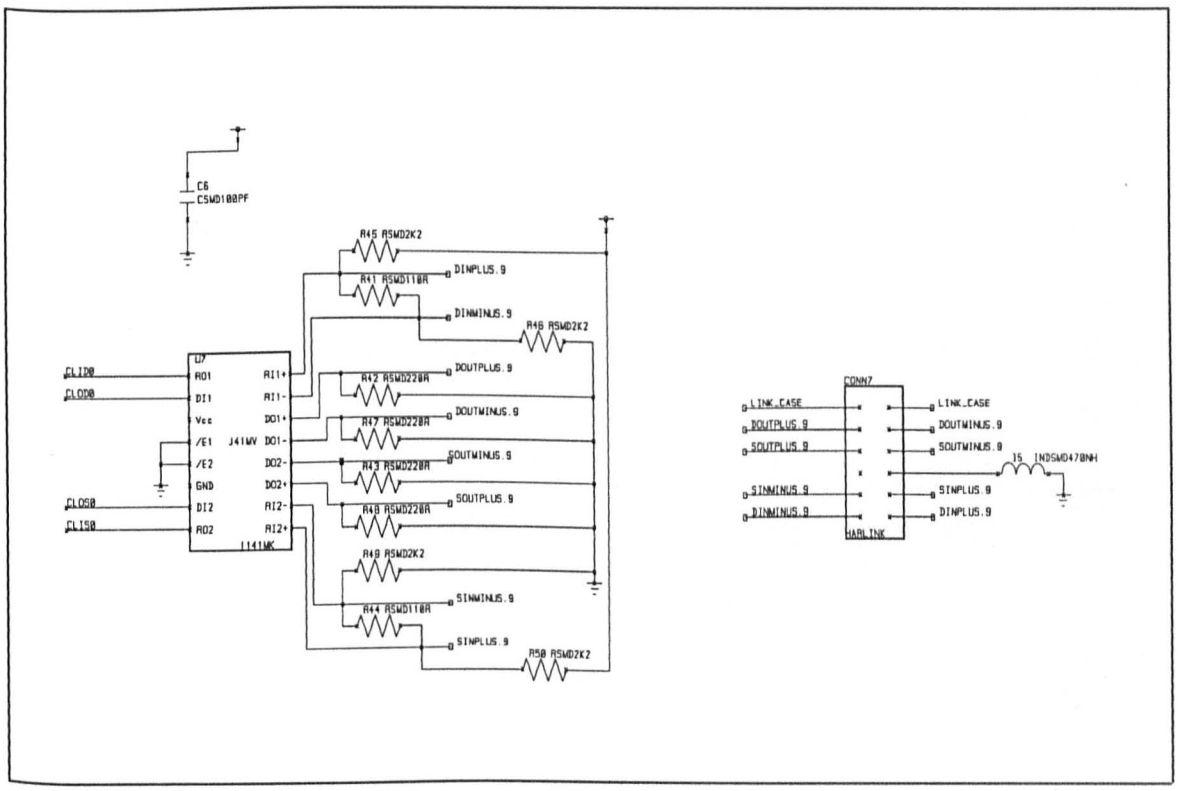
Connectors



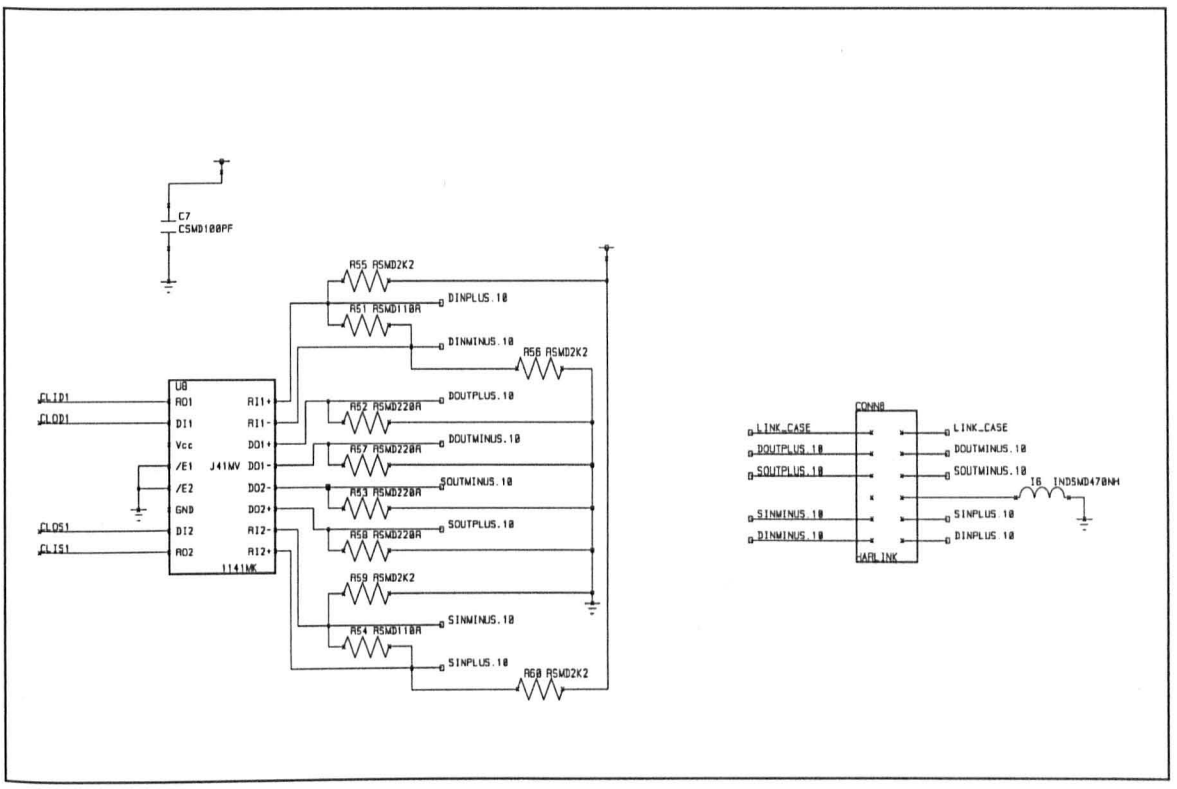
Reset Links



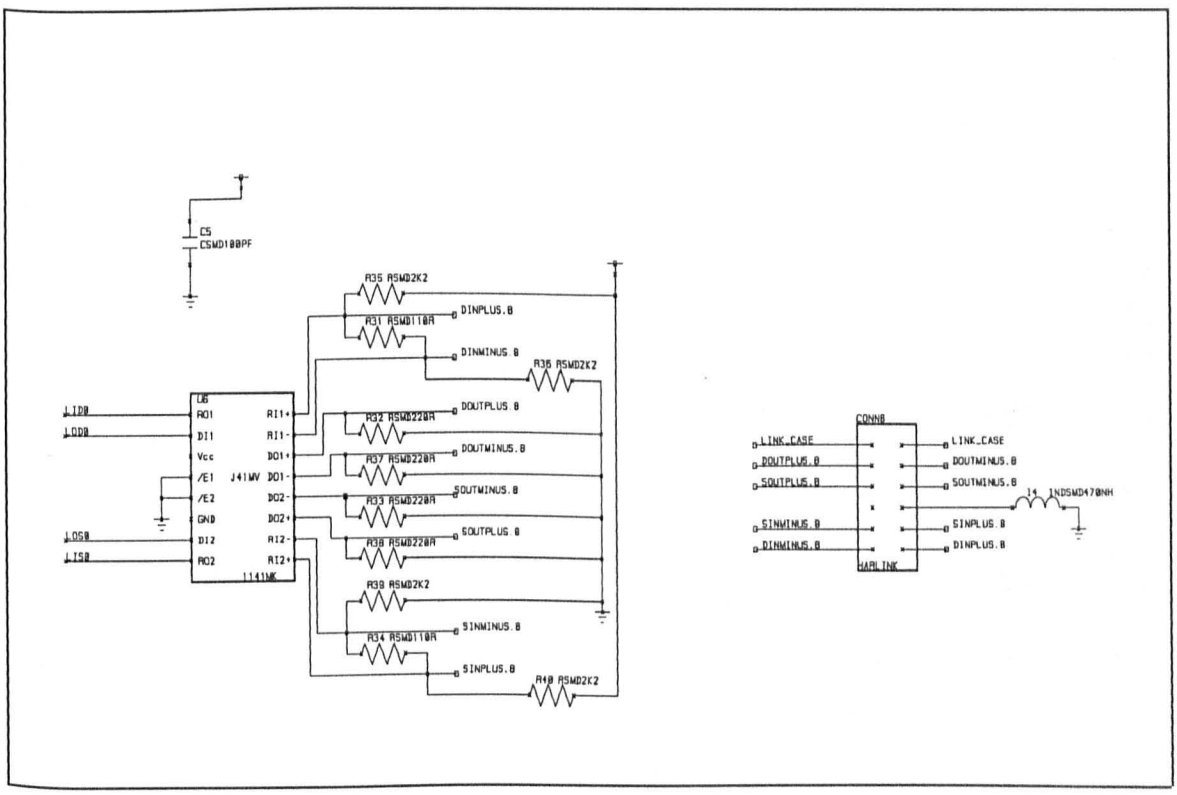
Hierarchical Blocks of DS Links



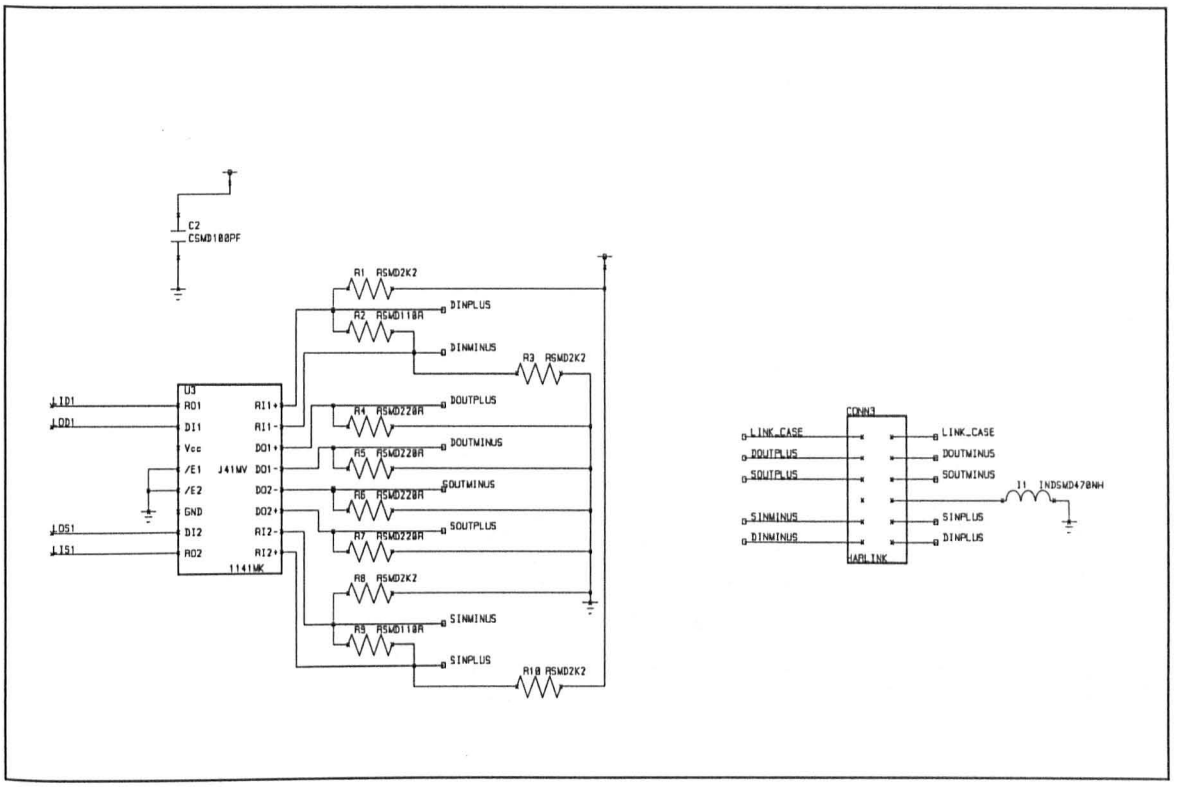
CLink0



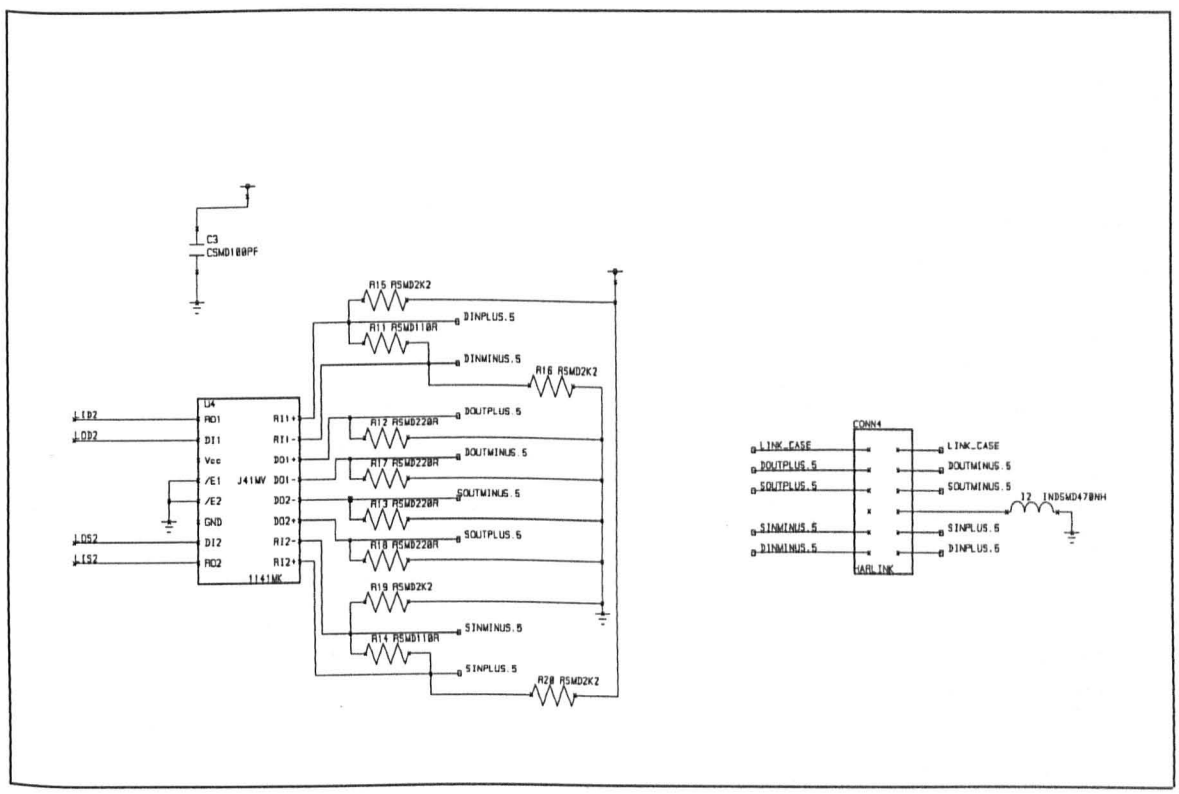
CLink1



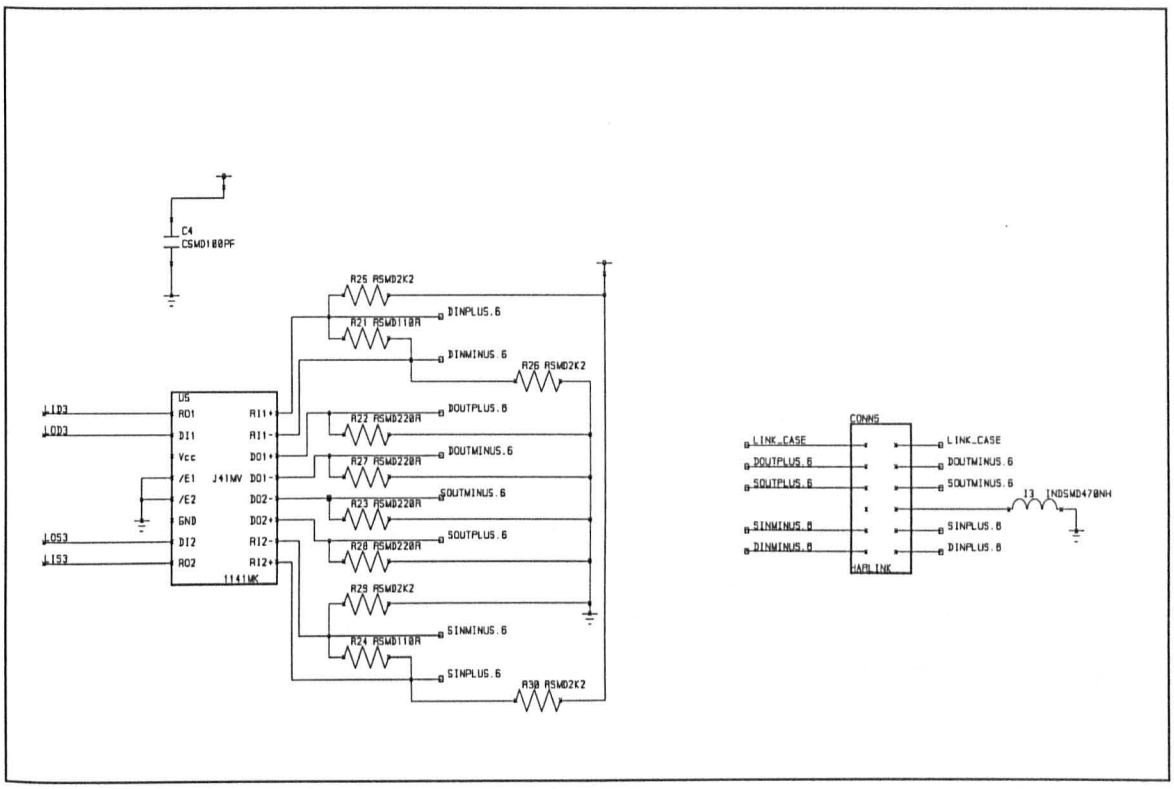
Data Link 0



Data Link 1

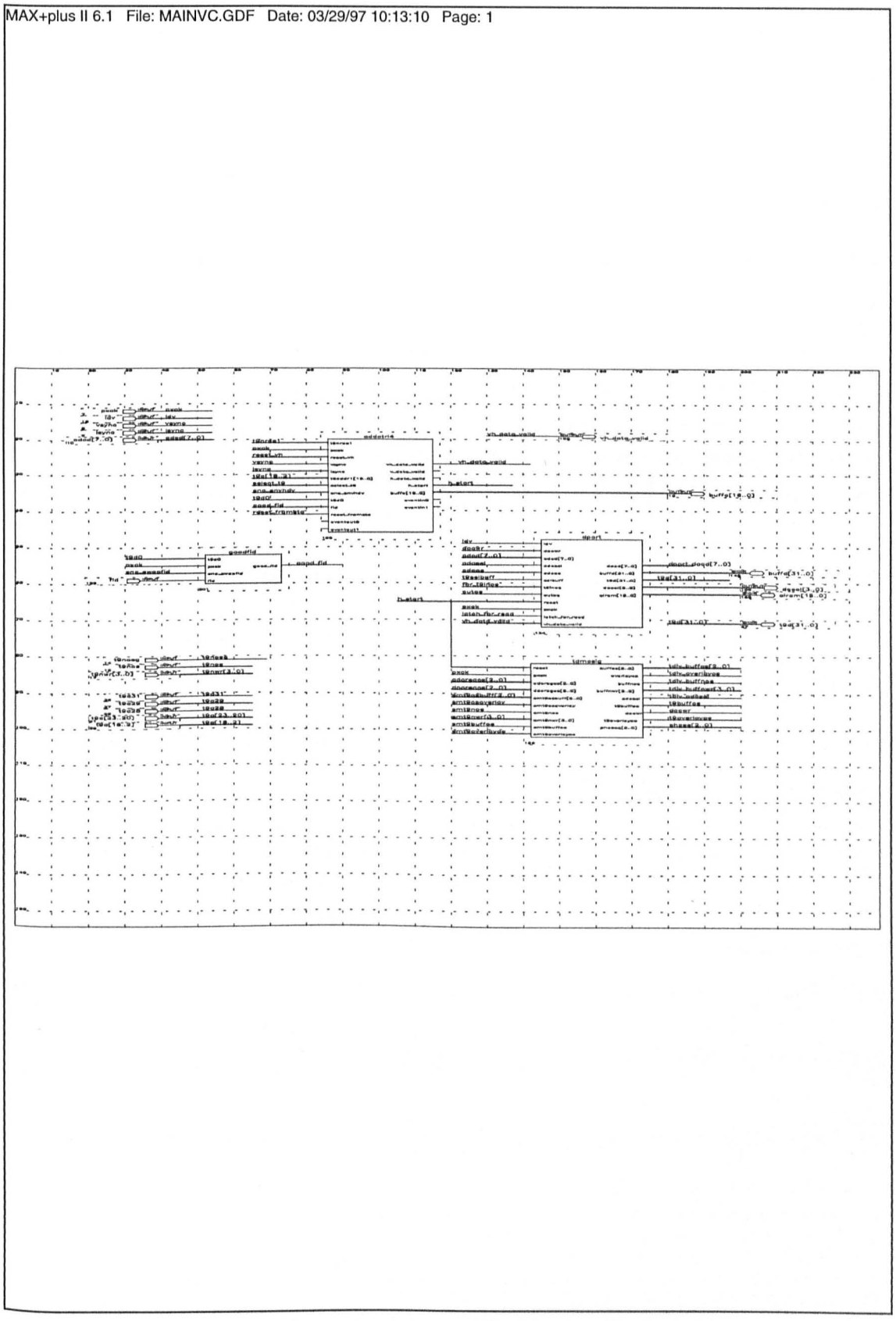


Data Link 2

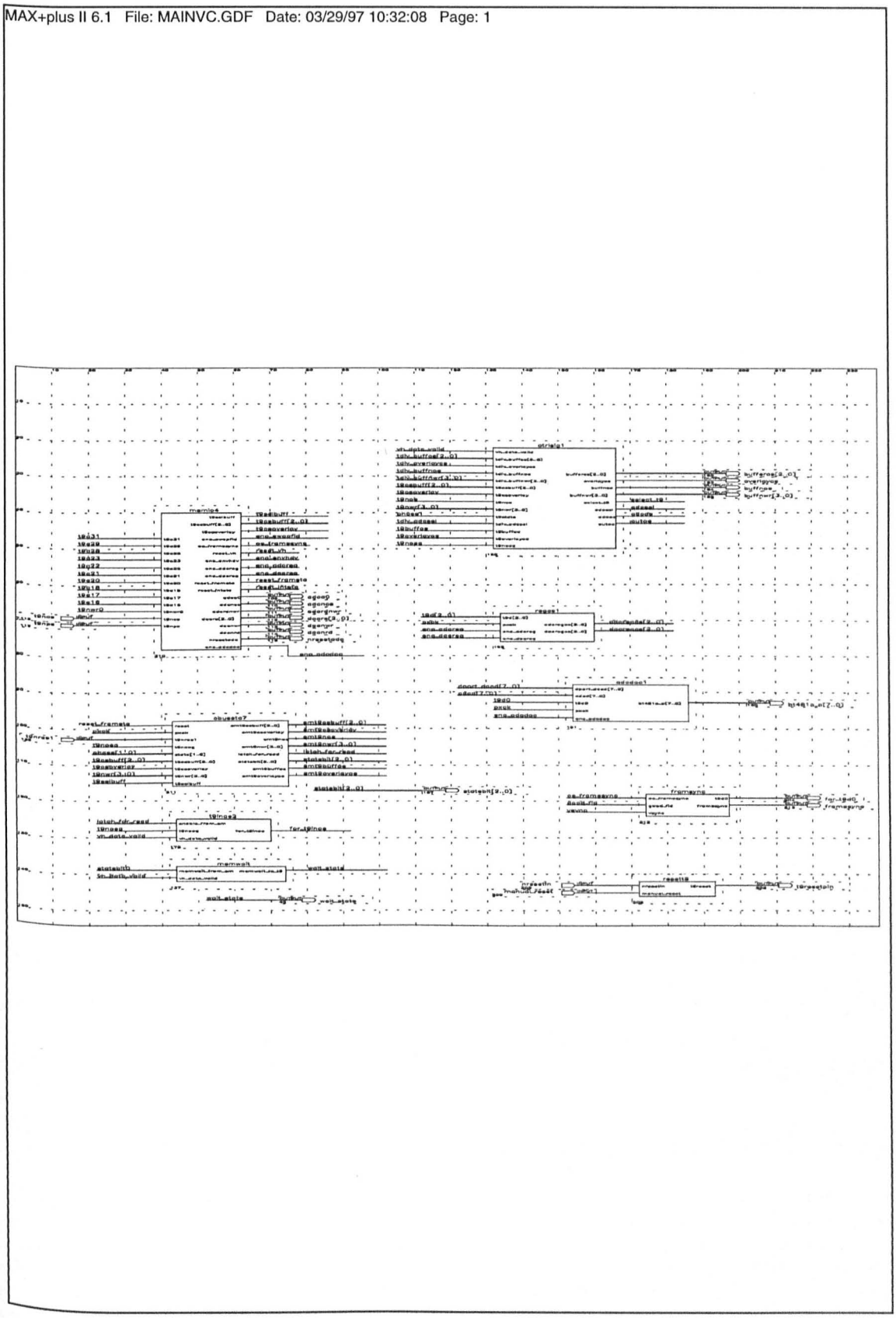

C4
CSMD100PF
R25 RSMD2K2
R21 RSMD110R
DINPLUS.6
DINMINUS.6
R26 RSMD2K2
U5
RD1
DI1
Vcc
/E1
/E2
GND
DI2
RO2
RI1+
RI1-
DO1+
J41MV DO1-
DO2-
DO2+
RI2-
RI2+
1141MK
LID3
LOD3
LOS3
LIS3
R22 RSMD220R
DOUTPLUS.6
R27 RSMD220R
DOUTMINUS.6
R23 RSMD220R
SOUTMINUS.6
R28 RSMD220R
SOUTPLUS.6
R29 RSMD2K2
SINMINUS.6
R24 RSMD110R
SINPLUS.6
R30 RSMD2K2
CONN5
LINK_CASE
DOUTPLUS.6
SOUTPLUS.6
SINMINUS.6
DINMINUS.6
HAPLINK
LINK_CASE
DOUTMINUS.6
SOUTMINUS.6
L3 INDSMD470NH
SINPLUS.6
DINPLUS.6

Data Link 3

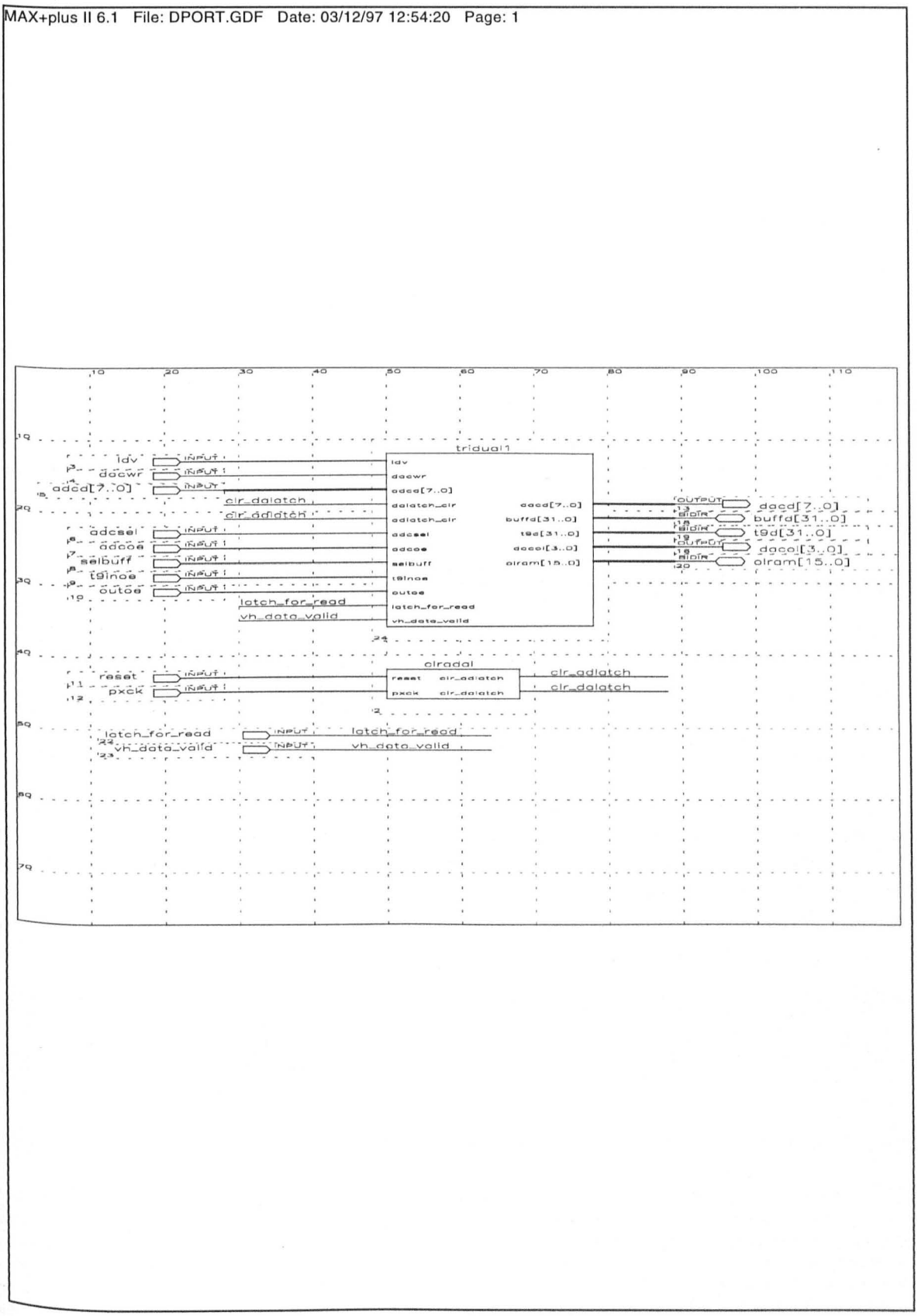
Appendix B - The Altera Design Files



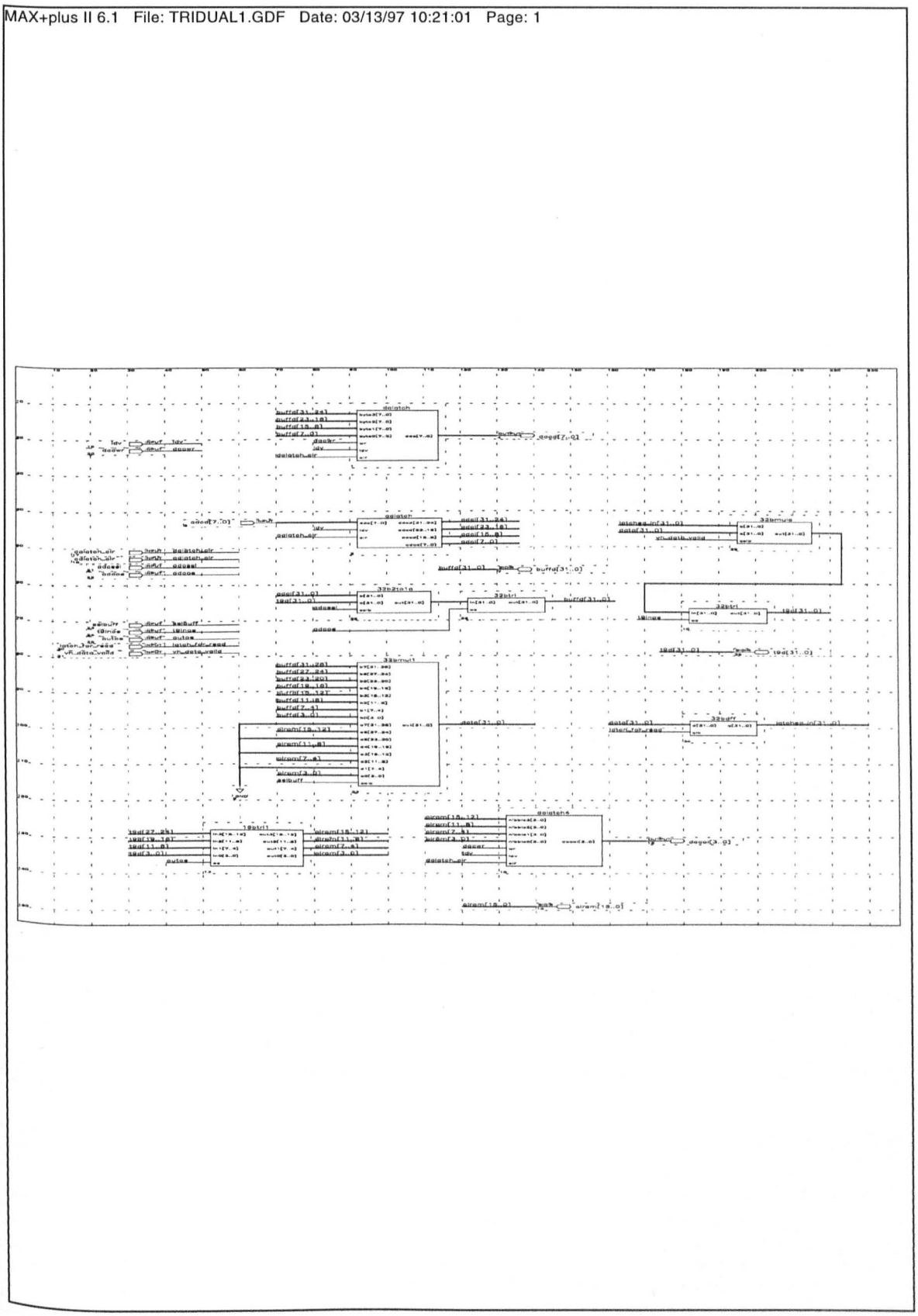
Top Level of the Hierarchy Display (Part One)



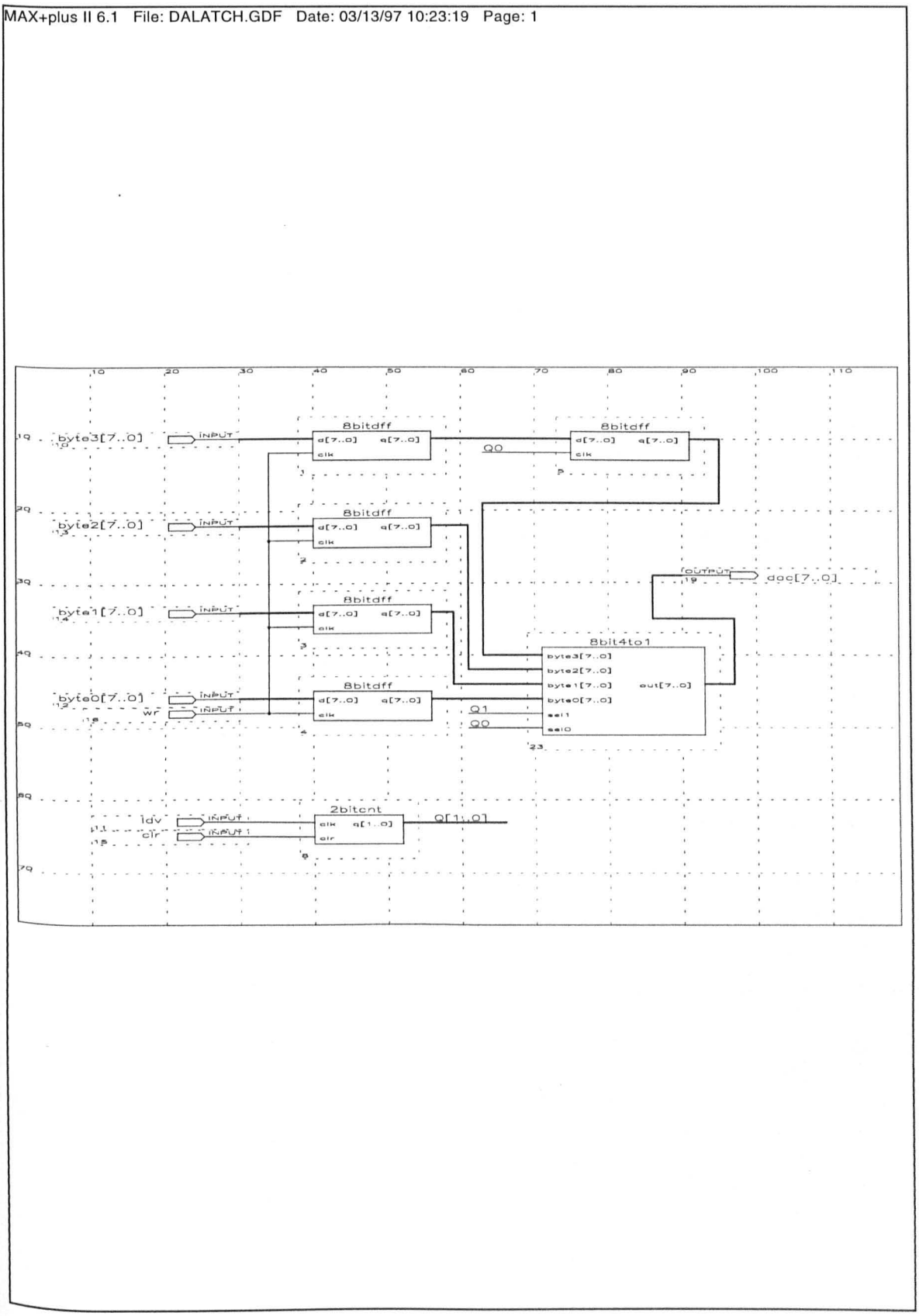
Top Level of the Hierarchy Display (Part Two)



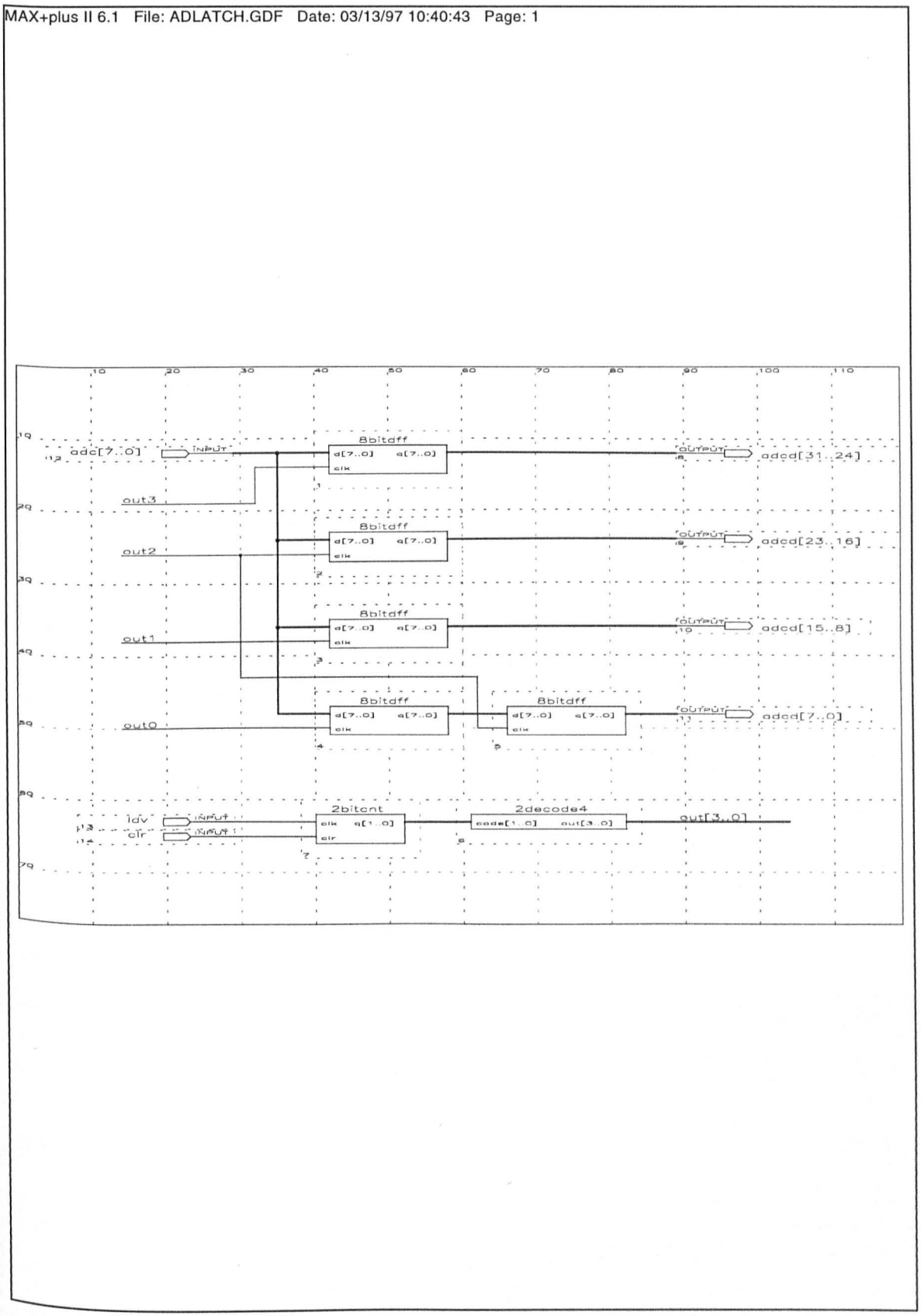
Sub-design Block of DPORT



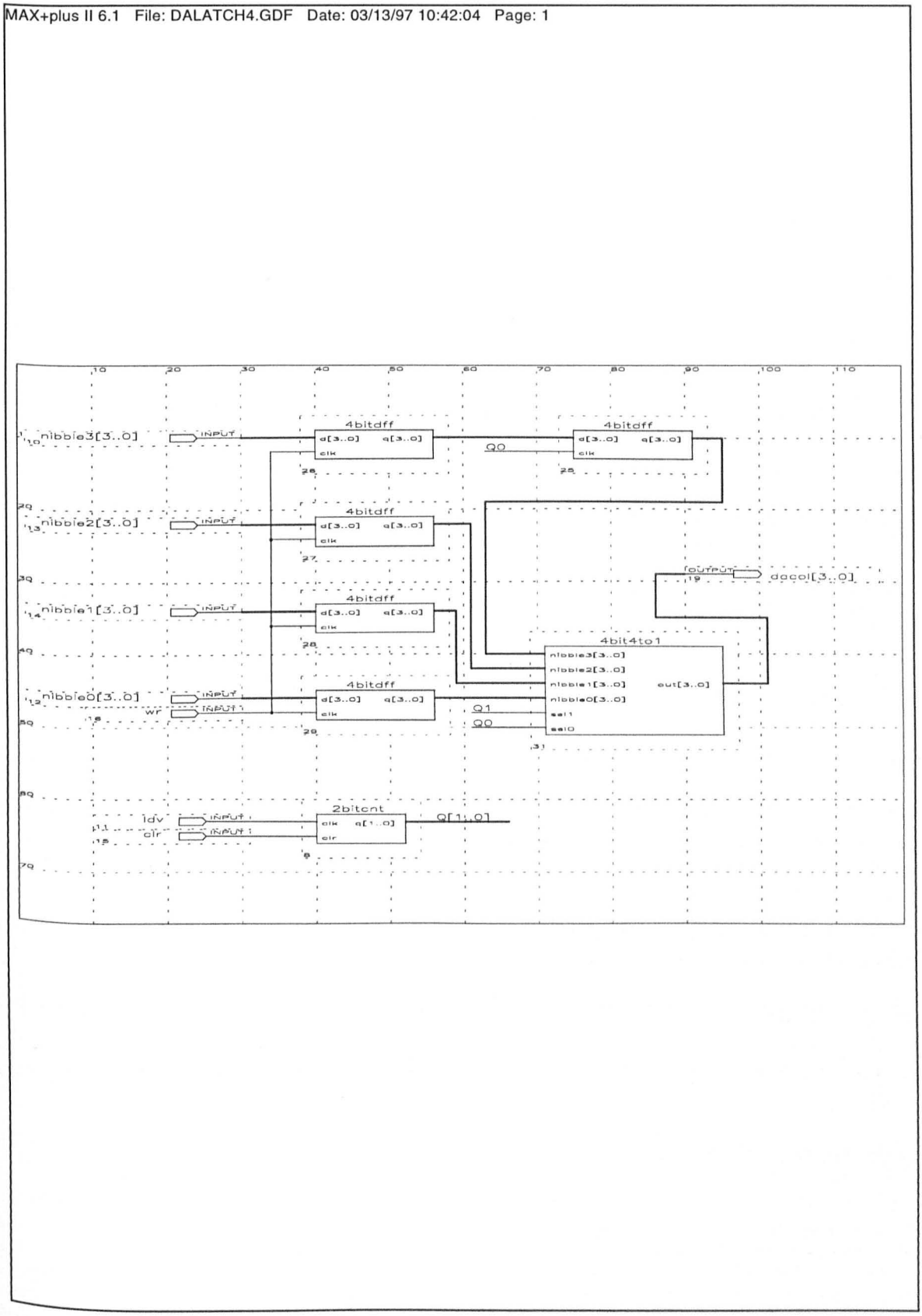
Sub-design Block of TRIDUAL1



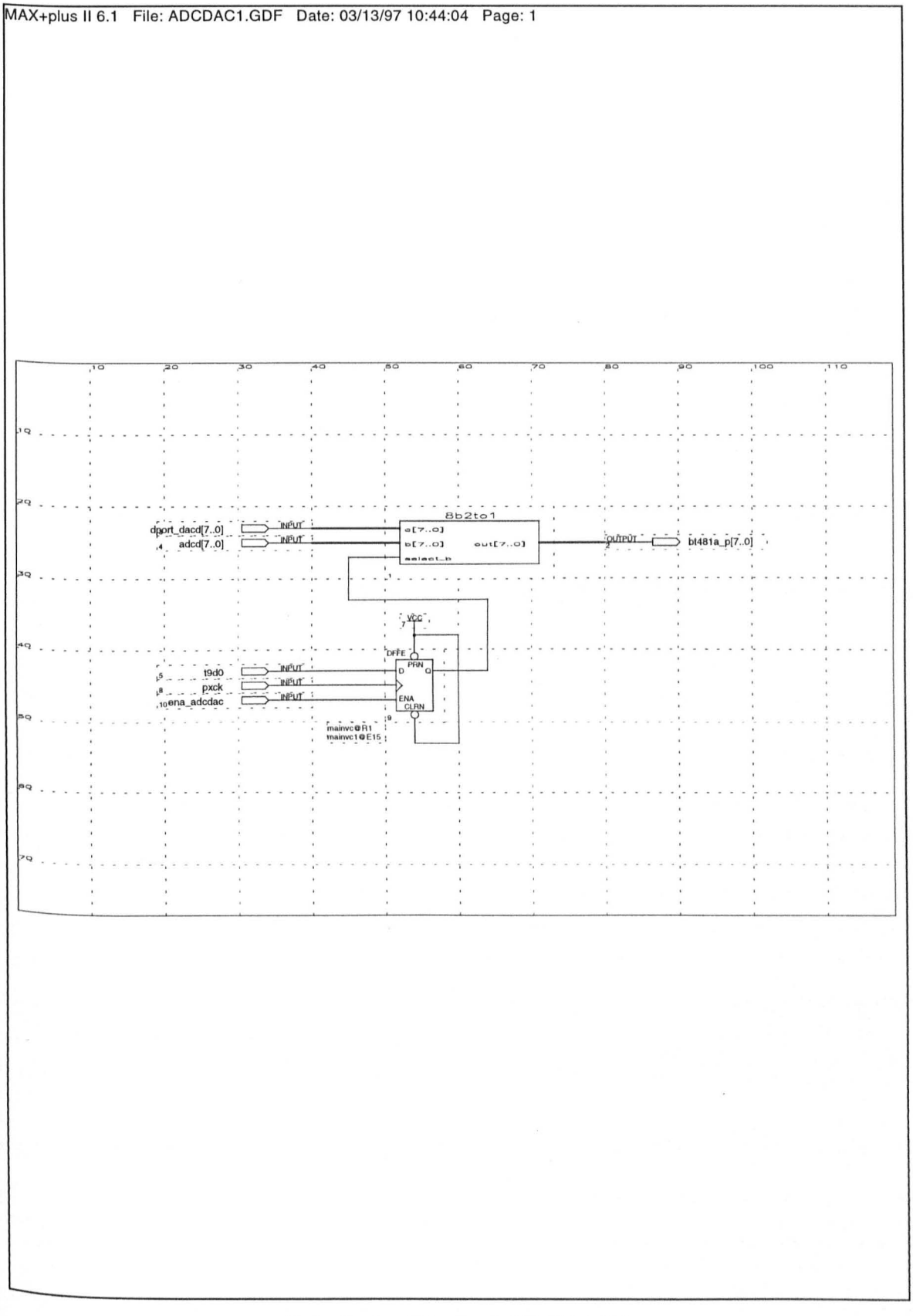
Sub-design Block of DALATCH



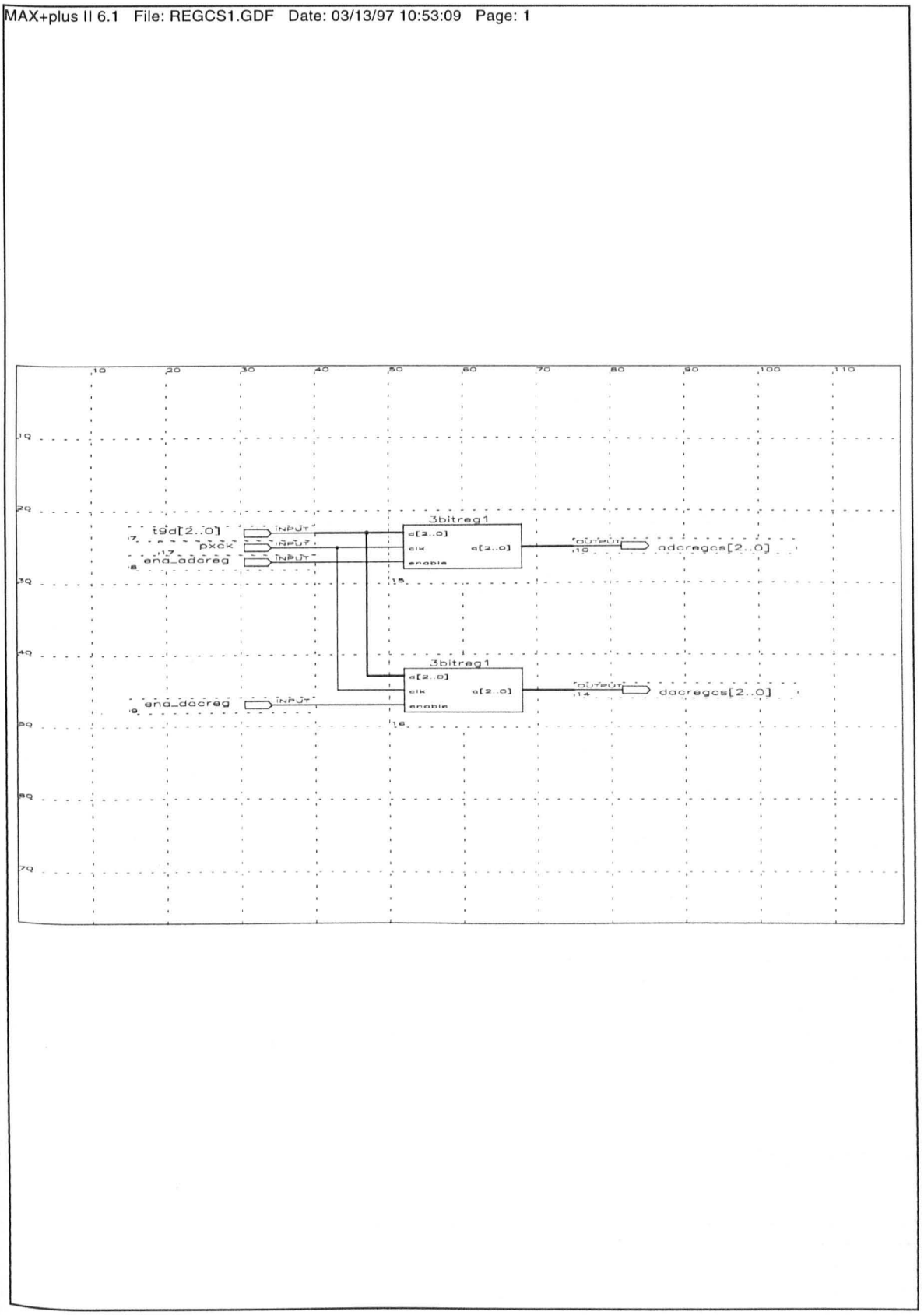
Sub-design Block of ADLATCH



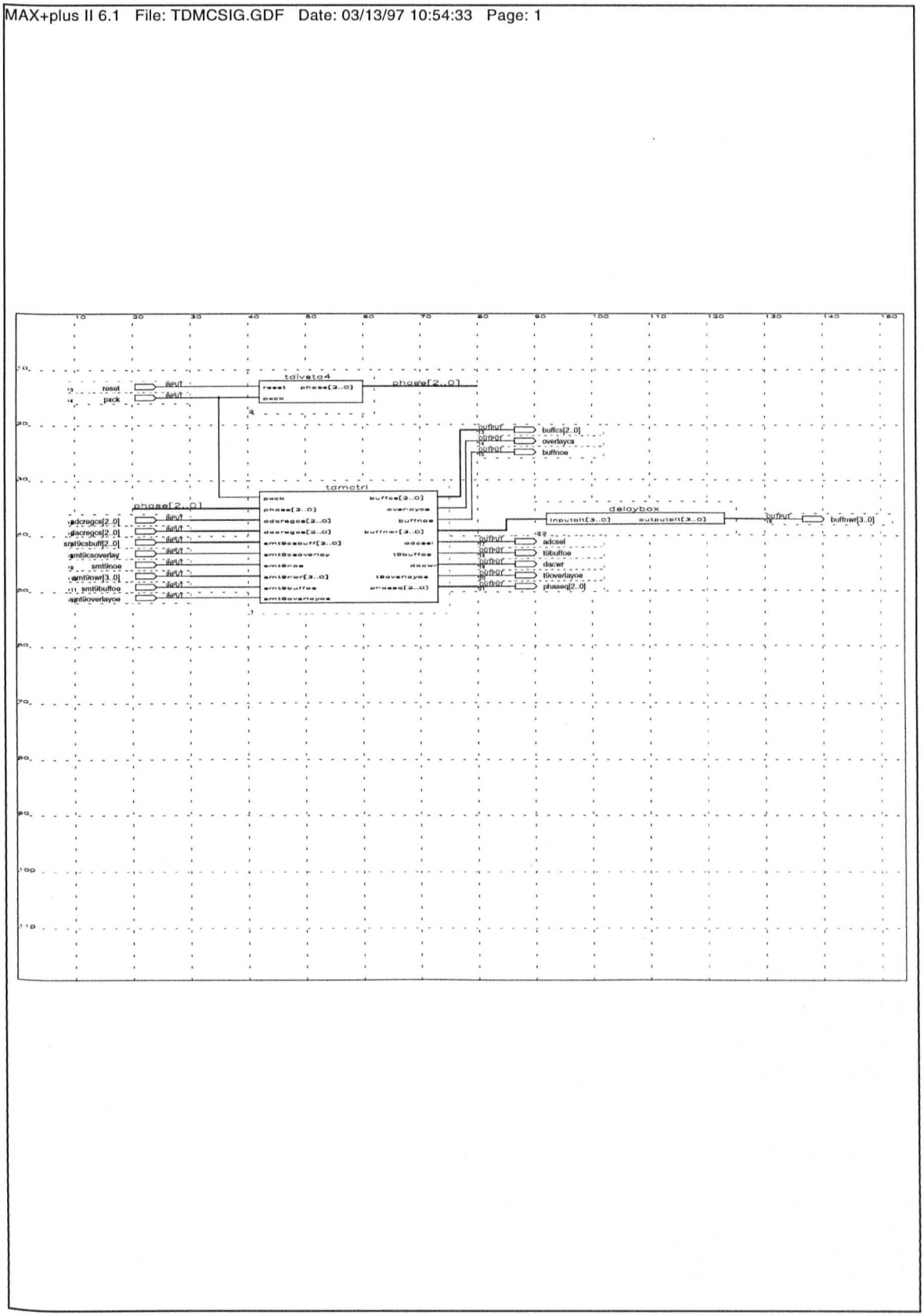
Sub-design Block of DALATCH4



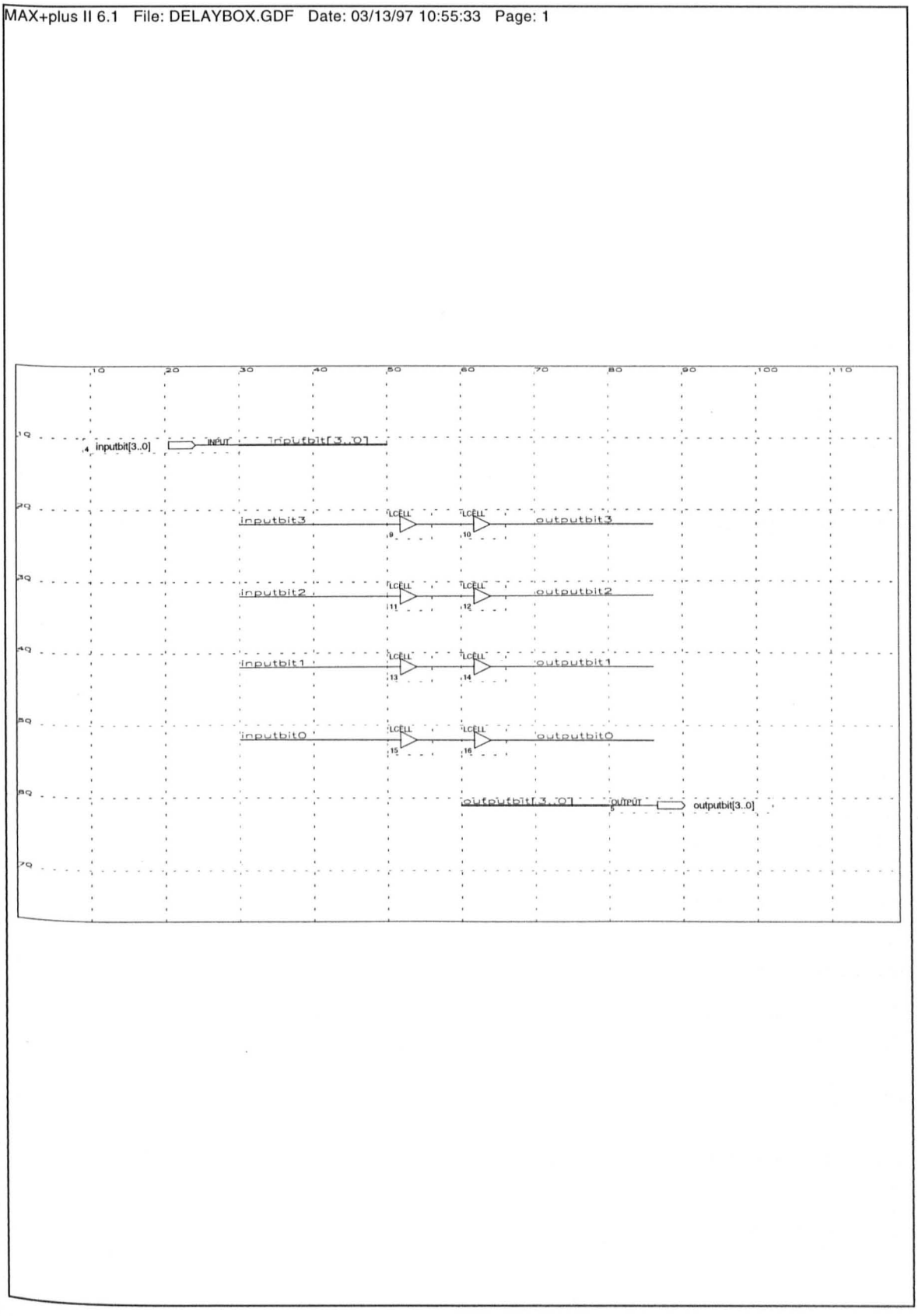
Sub-design Block of ADCDAC1



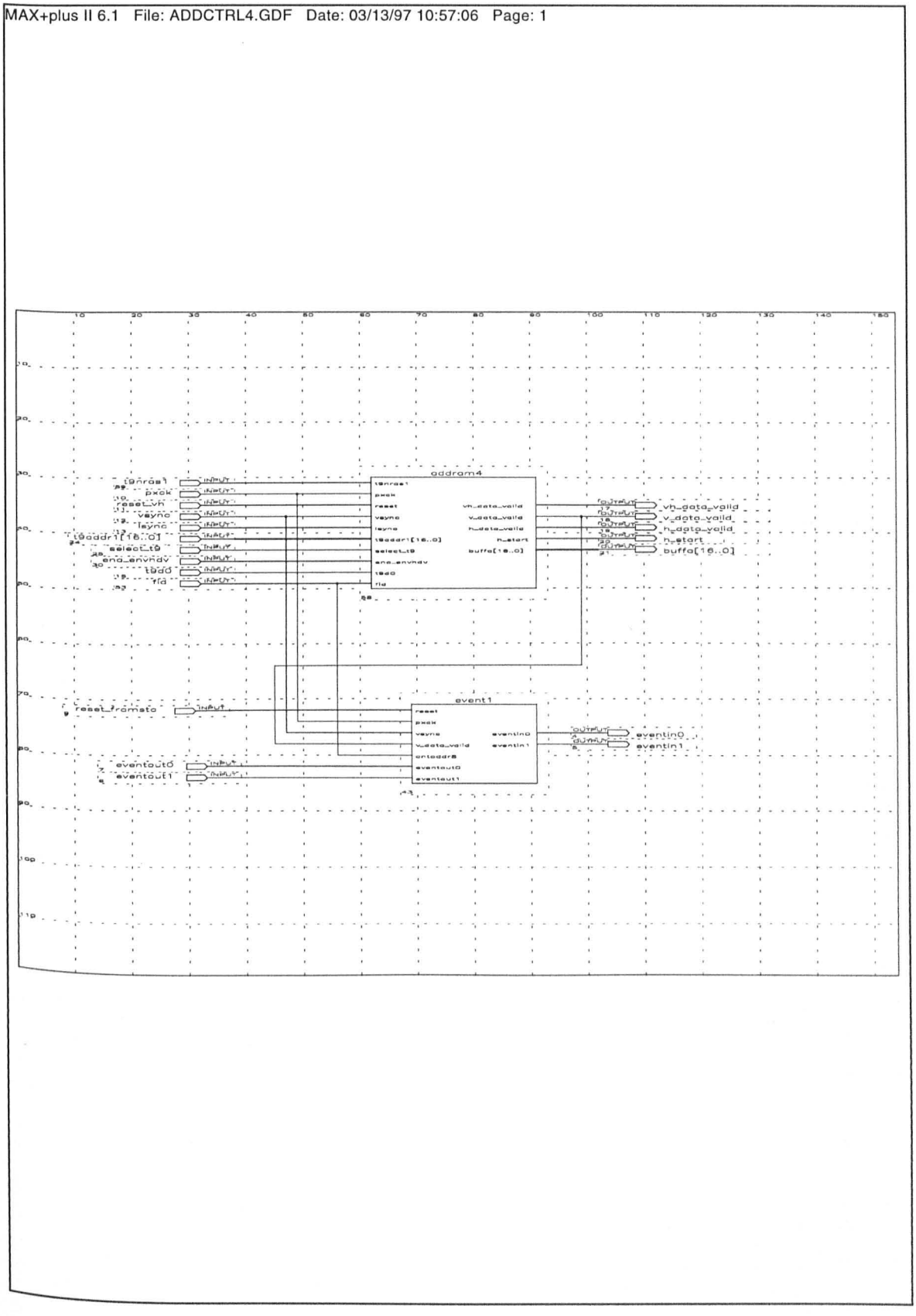
Sub-design Block of REGCS1



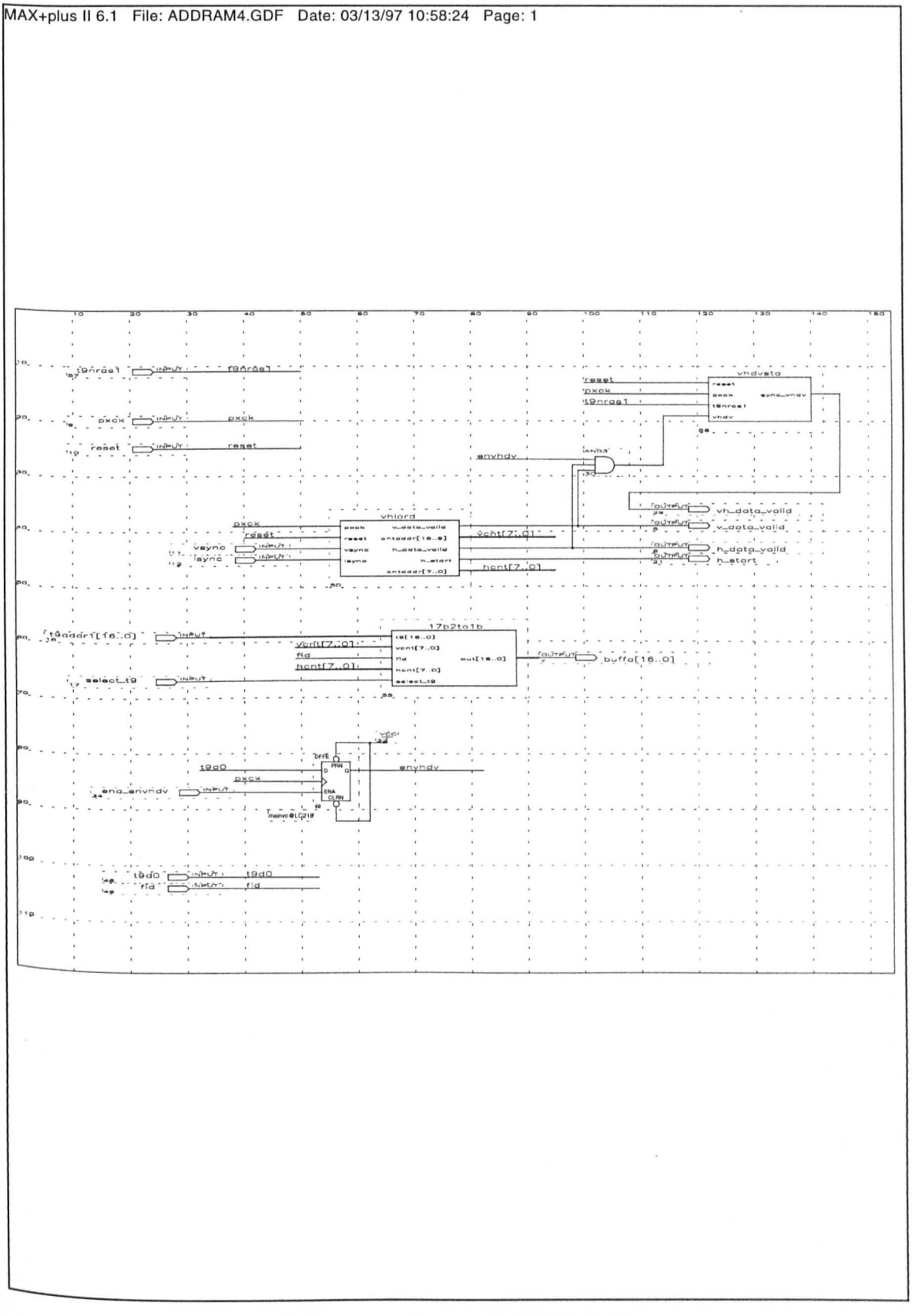
Sub-design Block of TDMCSIG



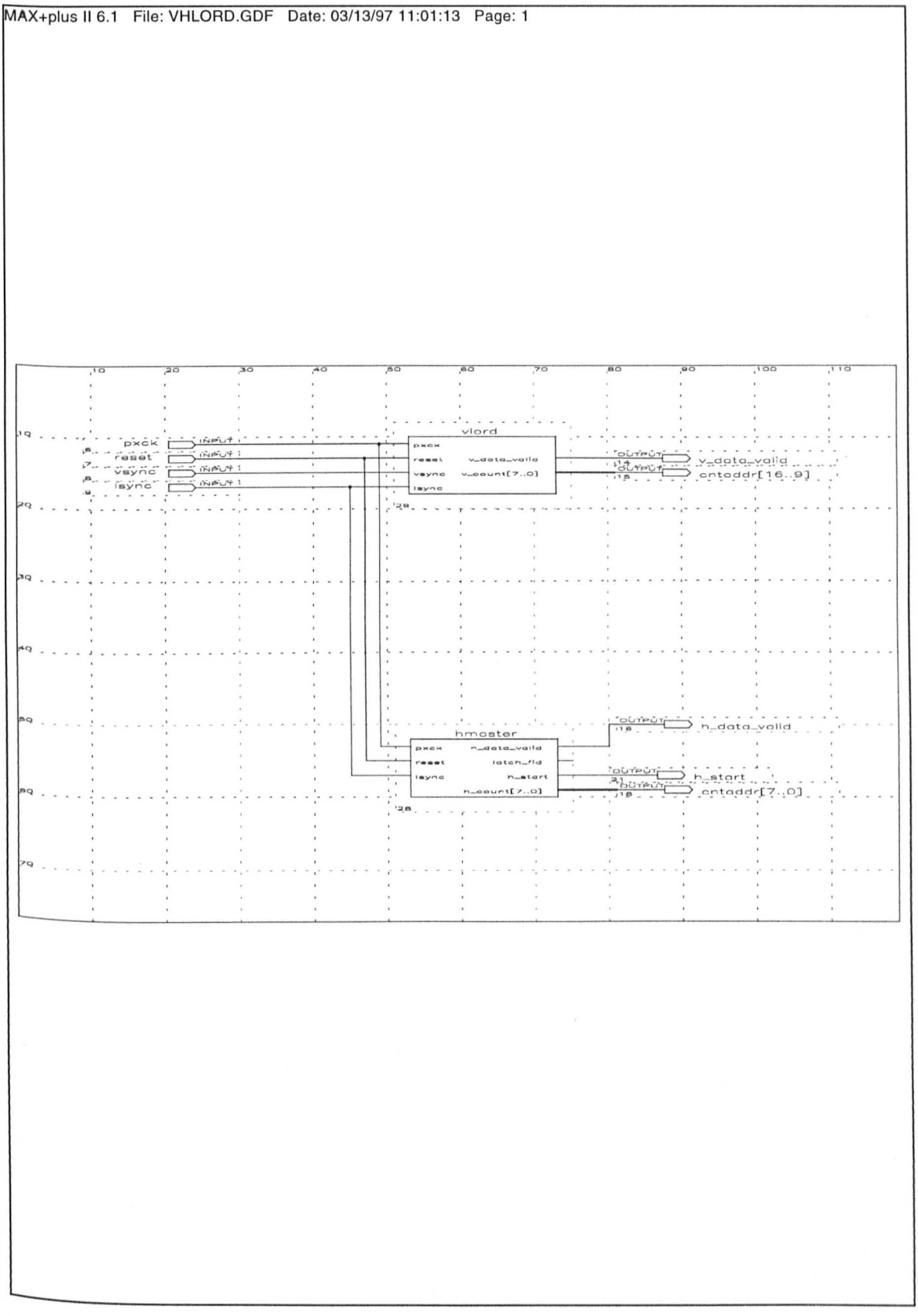
Sub-design Block of DELAYBOX



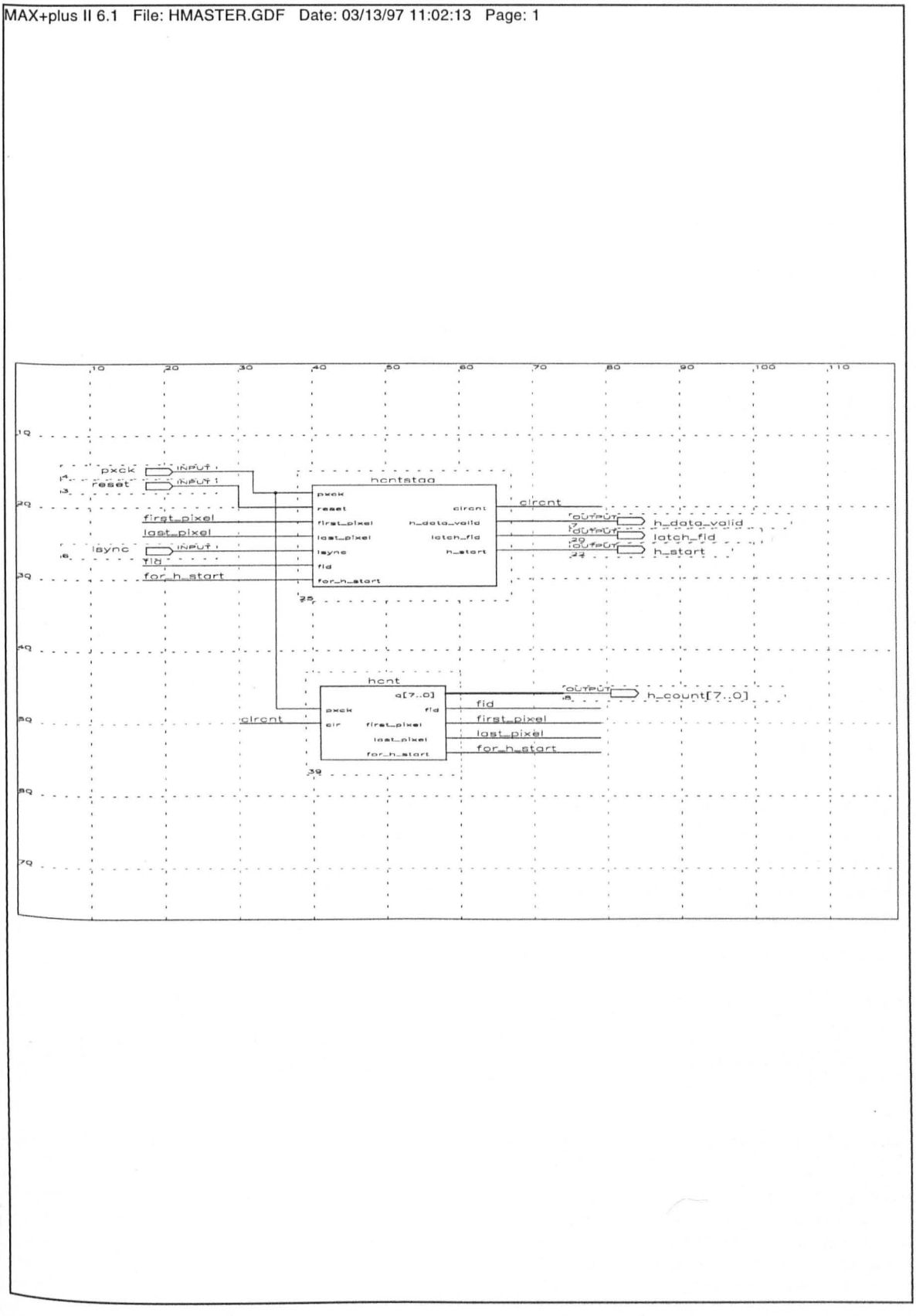
Sub-design Block of ADDCTRL4



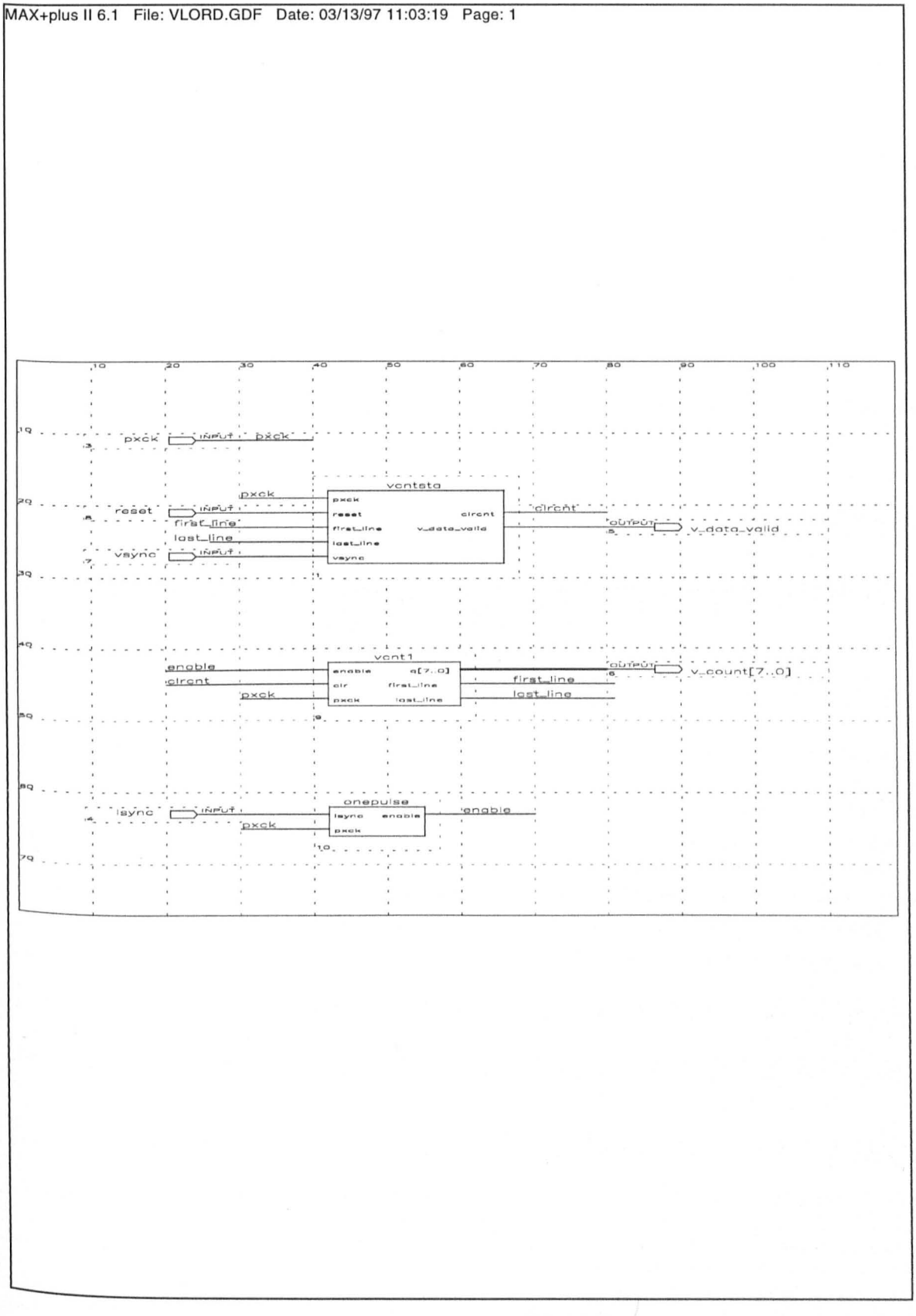
Sub-design Block of ADDRAM4



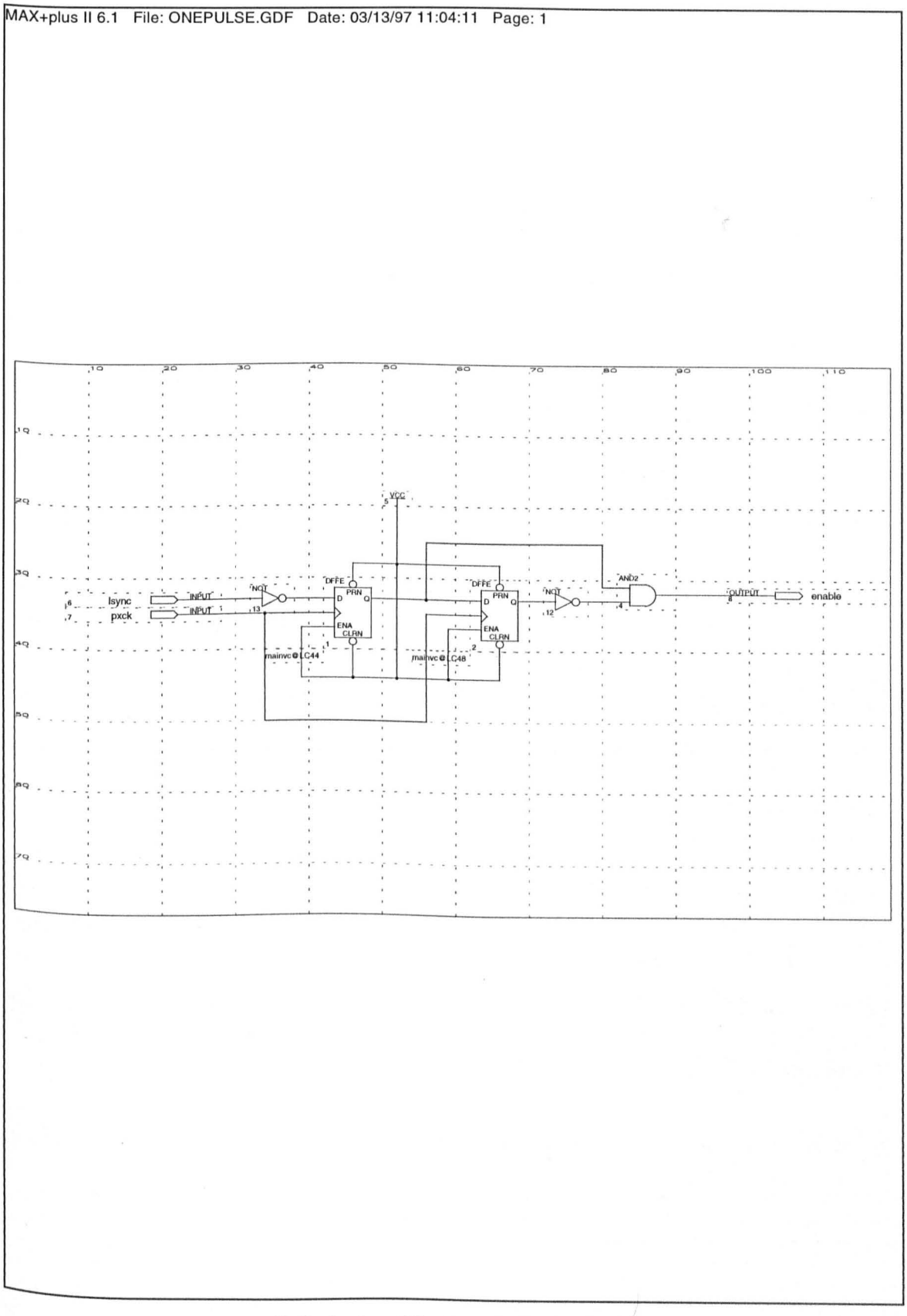
Sub-design Block of VHLORD



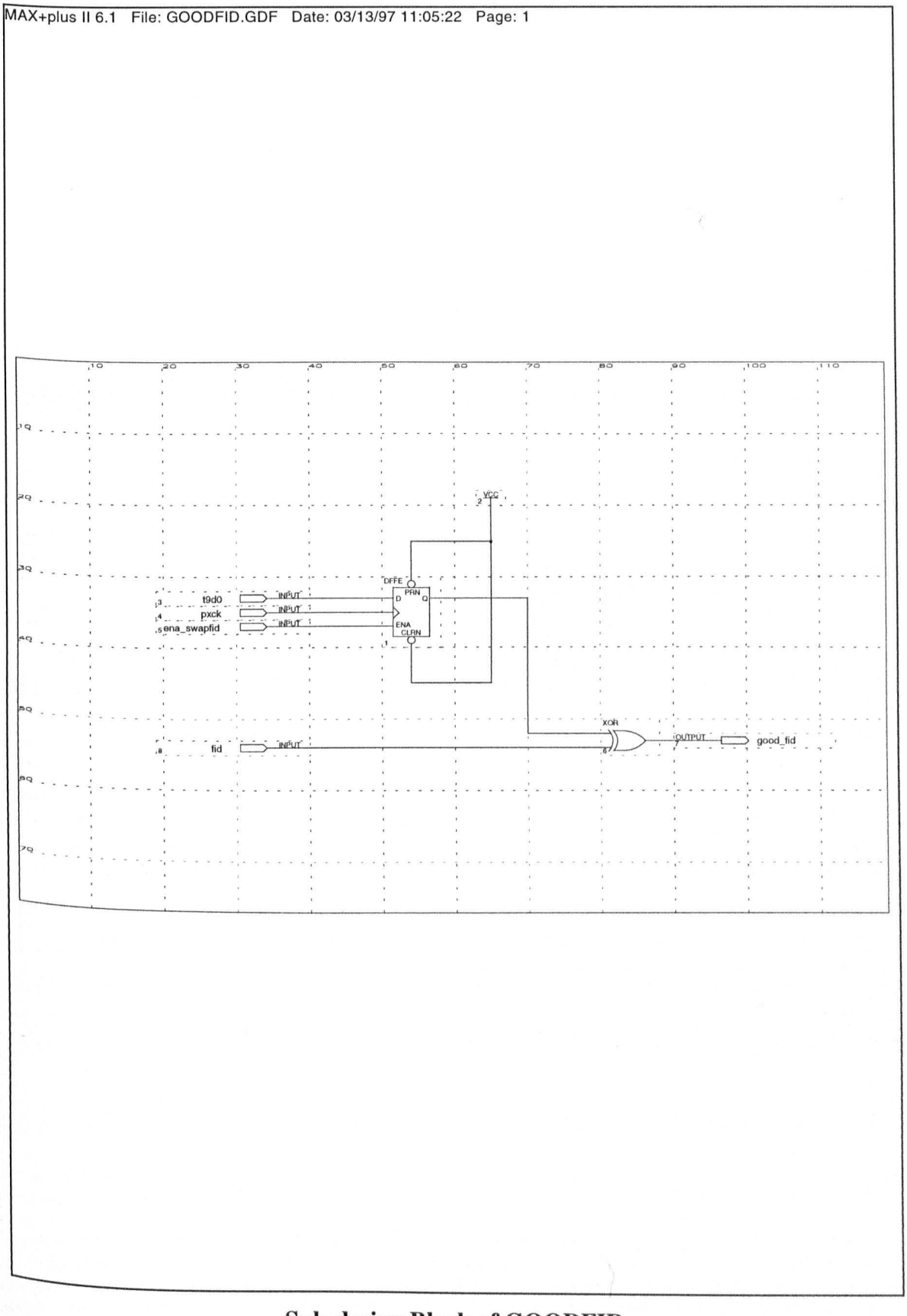
Sub-design Block of HMASTER



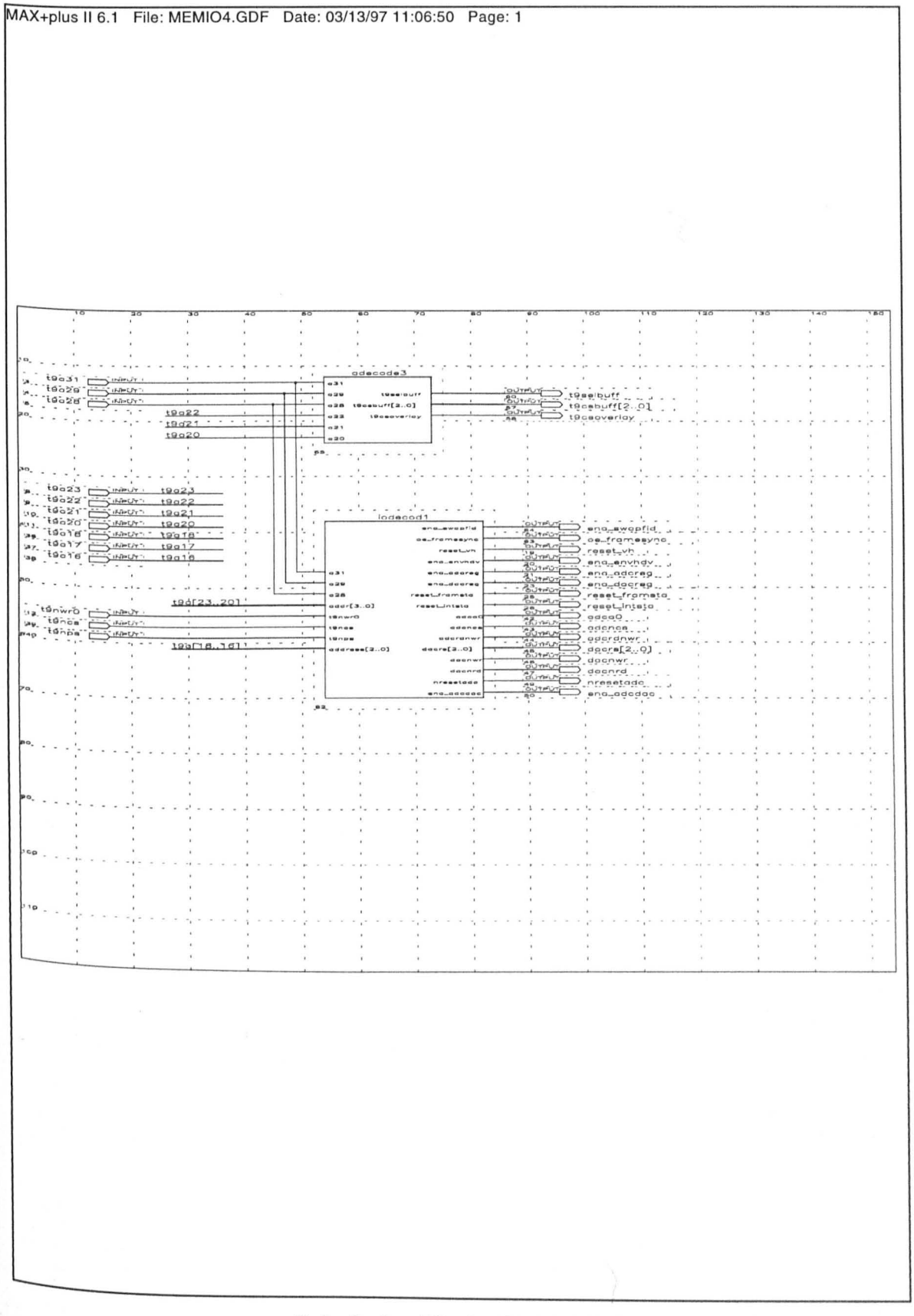
Sub-design Block of VLORD



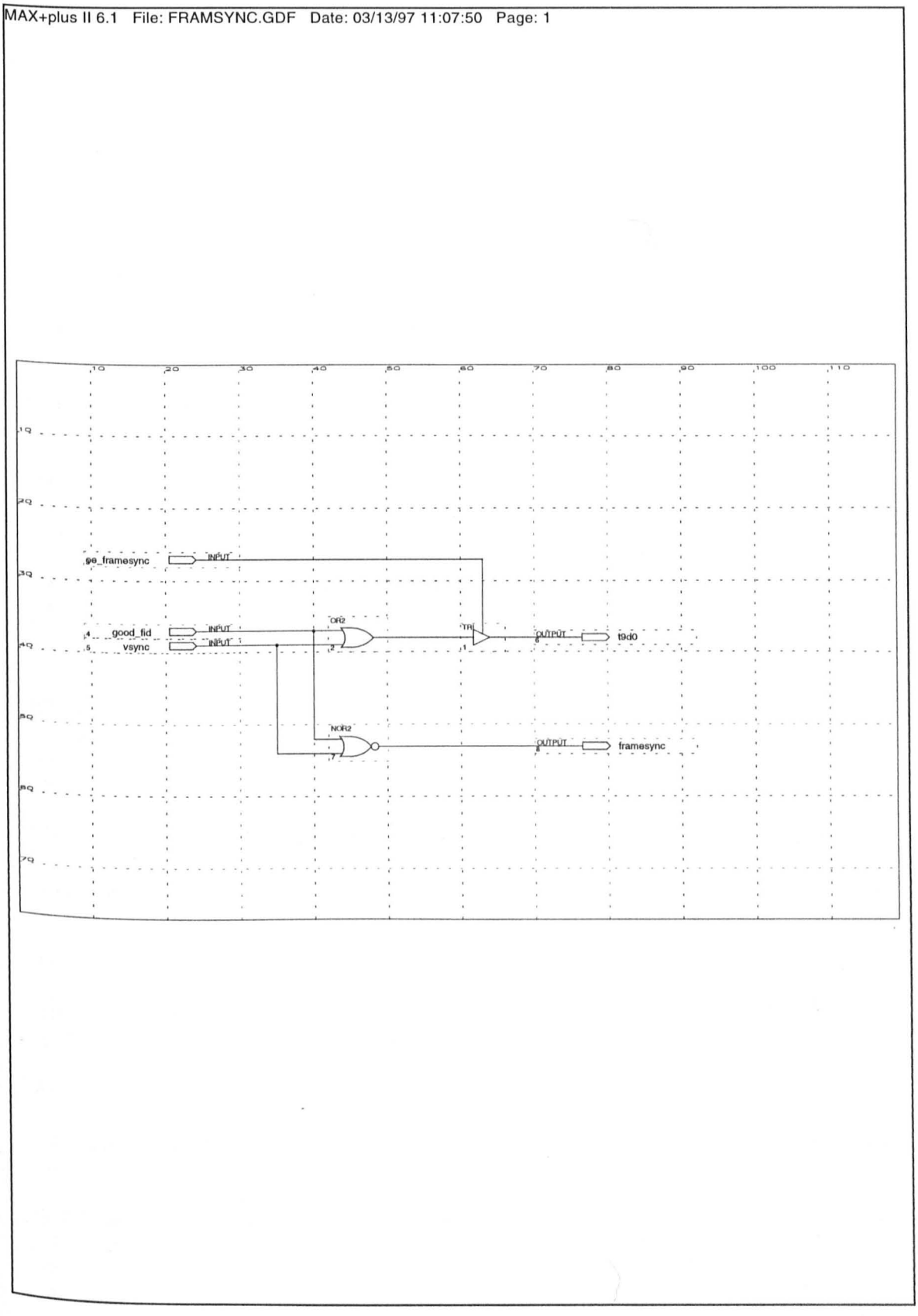
Sub-design Block of ONEPULSE



Sub-design Block of GOODFID



Sub-design Block of MEMIO4



Sub-design Block of FRAMSYNC

(to synchronise the pixel multiplexer and demultiplexer to the system state machine)

```
SUBDESIGN clradal
(
 reset, pxck          : INPUT;
 clr_adlatch, clr_dalatch : OUTPUT;
)
VARIABLE
 ss: MACHINE WITH STATES (s0, s1, s2, s3, s4, s5, s6, s7, s8, s9, s10);
 clr_adlatchd, clr_dalatchd : NODE;

BEGIN
 ss.clk  = pxck;
 ss.reset = reset;
 clr_adlatch = DFF(clr_adlatchd, pxck, VCC, VCC);
 clr_dalatch = DFF(clr_dalatchd, pxck, VCC, VCC);

 CASE ss IS

  % time division multiplexing to access the buffers and overlay rams %
  % this state machine is meaningful only when both v_data and h_data are valid %

  WHEN s0 =>          % first half time of adc %
   clr_adlatchd = b"0";
   clr_dalatchd = b"0";
   ss = s1;

  WHEN s1 =>          % second half time of adc %
   clr_adlatchd = b"0";
   clr_dalatchd = b"0";
   ss = s2;

  WHEN s2 =>          % first half time of t9000 %
   clr_adlatchd = b"0";
   clr_dalatchd = b"1";
   ss = s3;

  WHEN s3 =>          % second half time of t9000 %
   clr_adlatchd = b"0";
   clr_dalatchd = b"1";
   ss = s4;

  WHEN s4 =>          % frist half time of dac %
   clr_adlatchd = b"0";
   clr_dalatchd = b"1";
   ss = s5;

  WHEN s5 =>          % second half time of dac %
   clr_adlatchd = b"0";
   clr_dalatchd = b"1";
   ss = s6;

  WHEN s6 =>          % first half time of t9000 %
   clr_adlatchd = b"0";
   clr_dalatchd = b"0";
   ss = s7;

  WHEN s7 =>          % second half time of t9000 %
   clr_adlatchd = b"0";
```

```
    clr_dalatchd = b"0";
    ss = s8;
   WHEN s8 =>
    clr_adlatchd = b"1";
    clr_dalatchd = b"0";
    ss = s9;

   WHEN s9 =>
    clr_adlatchd = b"1";
    clr_dalatchd = b"0";
    ss = s10;

   WHEN s10 =>
    clr_adlatchd = b"0";
    clr_dalatchd = b"0";
    ss = s10;

   WHEN OTHERS =>
    clr_adlatchd = b"0";
    clr_dalatchd = b"0";
    ss = s10;

  END CASE;
END;
```

(8-bit D-type flipflop)

```
SUBDESIGN 8bitdff
(
  d[7..0], clk  : INPUT;
  q[7..0]       : OUTPUT;
)
VARIABLE
  q[7..0]       : DFF; % D type flip-flop %

BEGIN
  q[].clrn = VCC;
  q[].prn  = VCC;
  q[].clk = clk;
  q[] = d[];
END;
```

(2-bit counter)

```
SUBDESIGN 2bitcnt
(
  clk, clr  : INPUT;
  q[1..0]   : OUTPUT;
)
VARIABLE
  counter[1..0] : DFF;

BEGIN
  counter[].clk = clk;
  counter[].clrn = !clr;
  counter[].d = counter[].q + 1;
  q[] = counter[];
END;
```

(8-bit 4-channel-to-1-channel multiplexer)

```
SUBDESIGN 8bit4to1
(
 byte3[7..0], byte2[7..0], byte1[7..0], byte0[7..0], sel1, sel0  : INPUT;
 out[7..0]  : OUTPUT;
)
BEGIN
 CASE (sel1, sel0) IS
  WHEN (0, 0) => out[] = byte0[];
  WHEN (0, 1) => out[] = byte1[];
  WHEN (1, 0) => out[] = byte2[];
  WHEN (1, 1) => out[] = byte3[];
 END CASE;
END;
```

(2-line-to-4-line decoder)

```
SUBDESIGN 2decode4
(
 code[1..0]  : INPUT;
 out[3..0]   : OUTPUT;
)
BEGIN
 CASE code[] IS
  WHEN 0 => out[] = B"0001";
  WHEN 1 => out[] = B"0010";
  WHEN 2 => out[] = B"0100";
  WHEN 3 => out[] = B"1000";
 END CASE;
END;
```

(32-bit tri-state buffer)

```
SUBDESIGN 32btri
(
 in[31..0], oe  : INPUT;
 out[31..0]     : OUTPUT;
)
BEGIN
 out0 = TRI(in0, oe);
 out1 = TRI(in1, oe);
 out2 = TRI(in2, oe);
 out3 = TRI(in3, oe);
 out4 = TRI(in4, oe);
 out5 = TRI(in5, oe);
 out6 = TRI(in6, oe);
 out7 = TRI(in7, oe);
 out8 = TRI(in8, oe);
 out9 = TRI(in9, oe);
 out10 = TRI(in10, oe);
 out11 = TRI(in11, oe);
 out12 = TRI(in12, oe);
 out13 = TRI(in13, oe);
 out14 = TRI(in14, oe);
 out15 = TRI(in15, oe);
 out16 = TRI(in16, oe);
 out17 = TRI(in17, oe);
 out18 = TRI(in18, oe);
```

```
 out19 = TRI(in19, oe);
 out20 = TRI(in20, oe);
 out21 = TRI(in21, oe);
 out22 = TRI(in22, oe);
 out23 = TRI(in23, oe);
 out24 = TRI(in24, oe);
 out25 = TRI(in25, oe);
 out26 = TRI(in26, oe);
 out27 = TRI(in27, oe);
 out28 = TRI(in28, oe);
 out29 = TRI(in29, oe);
 out30 = TRI(in30, oe);
 out31 = TRI(in31, oe);
END;
```

(16-bit tri-state buffer)

```
SUBDESIGN 16btri1
(
 in3[15..12], in2[11..8], in1[7..4], in0[3..0], oe  : INPUT;
 out3[15..12], out2[11..8], out1[7..4], out0[3..0]  : OUTPUT;
)
BEGIN
 out00 = TRI(in00, oe);
 out01 = TRI(in01, oe);
 out02 = TRI(in02, oe);
 out03 = TRI(in03, oe);
 out14 = TRI(in14, oe);
 out15 = TRI(in15, oe);
 out16 = TRI(in16, oe);
 out17 = TRI(in17, oe);
 out28 = TRI(in28, oe);
 out29 = TRI(in29, oe);
 out210 = TRI(in210, oe);
 out211 = TRI(in211, oe);
 out312 = TRI(in312, oe);
 out313 = TRI(in313, oe);
 out314 = TRI(in314, oe);
 out315 = TRI(in315, oe);
END;
```

(4-bit D-type flipflop)

```
SUBDESIGN 4bitdff
(
 d[3..0], clk  : INPUT;
 q[3..0]       : OUTPUT;
)
VARIABLE
 q[3..0]       : DFF; % D type flip-flop %

BEGIN
 q[].clk = clk;
 q[] = d[];
END;
```

(4-bit 4-channel-to-1-channel multiplexer)

```
SUBDESIGN 4bit4to1
(
 nibble3[3..0], nibble2[3..0], nibble1[3..0], nibble0[3..0], sel1, sel0  : INPUT;
 out[3..0]  : OUTPUT;
)
BEGIN
 CASE (sel1, sel0) IS
  WHEN (0, 0) => out[] = nibble0[];
  WHEN (0, 1) => out[] = nibble1[];
  WHEN (1, 0) => out[] = nibble2[];
  WHEN (1, 1) => out[] = nibble3[];
 END CASE;
END;
```

(32-bit 2-channel-to-1-channel multiplexer)

```
SUBDESIGN 32bmula
(
 b[31..0], a[31..0], selb : INPUT;
 out[31..0]               : OUTPUT;
)

BEGIN
 CASE selb IS
  WHEN 0 =>
   out[] = a[];
  WHEN 1 =>
   out[] = b[];
 END CASE;
END;
```

(32-bit 2-channel-to-1-channel multiplexer)

```
SUBDESIGN 32bmul1
(
 b7[31..28], b6[27..24], b5[23..20], b4[19..16] : INPUT;
 b3[15..12], b2[11..8], b1[7..4], b0[3..0]      : INPUT;
 a7[31..28], a6[27..24], a5[23..20], a4[19..16] : INPUT;
 a3[15..12], a2[11..8], a1[7..4], a0[3..0]      : INPUT;
 selb                                  : INPUT;
 out[31..0]                              : OUTPUT;
)
BEGIN
 CASE selb IS
  WHEN 0 =>
   out[31..28] = a7[31..28];
   out[27..24] = a6[27..24];
   out[23..20] = a5[23..20];
   out[19..16] = a4[19..16];
   out[15..12] = a3[15..12];
   out[11..8]  = a2[11..8];
   out[7..4]   = a1[7..4];
   out[3..0]   = a0[3..0];
  WHEN 1 =>
   out[31..28] = b7[31..28];
   out[27..24] = b6[27..24];
   out[23..20] = b5[23..20];
```

```
    out[19..16] = b4[19..16];
    out[15..12] = b3[15..12];
    out[11..8]  = b2[11..8];
    out[7..4]   = b1[7..4];
    out[3..0]   = b0[3..0];
  END CASE;
END;
```

(32-bit D-type flipflop)

```
SUBDESIGN 32bdff
(
  d[31..0], clk : INPUT;
  q[31..0]      : OUTPUT;
)
VARIABLE
  q[31..0] : DFF;  % also declared as outputs %

BEGIN
  q[].clk = clk;
  q[] = d[];
END;
```

(32-bit 2-channel-to-1-channel multiplexer)

```
SUBDESIGN 32b2to1a
(
  b[31..0], a[31..0], selb : INPUT;
  out[31..0]               : OUTPUT;
)
BEGIN
  CASE selb IS
    WHEN 0 => out[] = a[];
    WHEN 1 => out[] = b[];
  END CASE;
END;
```

(memory wait generator)

```
SUBDESIGN memwait
(
  memwait_from_sm, vh_data_valid : INPUT;
  memwait_to_t9                  : OUTPUT;
)
BEGIN
  memwait_to_t9 = memwait_from_sm & vh_data_valid;
END;
```

(programmable strobe of T9000 transputer associated with state machine's signals)

```
SUBDESIGN t9inoe2
(
  enable_from_sm, t9noeg, vh_data_valid : INPUT;
  for_t9inoe                            : OUTPUT;
)
BEGIN
  for_t9inoe = (!t9noeg & !vh_data_valid) # (enable_from_sm & vh_data_valid);
END;
```

(8-bit 2-channel-to-1-channel multiplexer)

```
SUBDESIGN 8b2to1
(
 a[7..0], b[7..0], select_b : INPUT;
 out[7..0]                  : OUTPUT;
)
BEGIN
 out7 = (a7 & !select_b) #
        (b7 & select_b)  #
        (a7 & b7);

 out6 = (a6 & !select_b) #
        (b6 & select_b)  #
        (a6 & b6);

 out5 = (a5 & !select_b) #
        (b5 & select_b)  #
        (a5 & b5);

 out4 = (a4 & !select_b) #
        (b4 & select_b)  #
        (a4 & b4);

 out3 = (a3 & !select_b) #
        (b3 & select_b)  #
        (a3 & b3);

 out2 = (a2 & !select_b) #
        (b2 & select_b)  #
        (a2 & b2);

 out1 = (a1 & !select_b) #
        (b1 & select_b)  #
        (a1 & b1);

 out0 = (a0 & !select_b) #
        (b0 & select_b)  #
        (a0 & b0);
END;
```

(3-bit data latch)

```
SUBDESIGN 3bitreg1
(
 d[2..0], clk, enable : INPUT;
 q[2..0]              : OUTPUT;
)
VARIABLE
 q[2..0]  : DFFE;  % also declared as outputs %

BEGIN
 q[].clk = clk;
 q[].ena = enable;
 q[] = d[];
END;
```

(the time-division-multiplexing controller)

```
SUBDESIGN tdmctrl
(
  pxck, phase[2..0], adcregcs[2..0], dacregcs[2..0]      : INPUT;
  smt9csbuff[2..0], smt9csoverlay, smt9noe, smt9nwr[3..0] : INPUT;
  smt9buffoe, smt9overlayoe                              : INPUT;
  buffcs[2..0], overlaycs, buffnoe, buffnwr[3..0]        : OUTPUT;
  adcsel, t9buffoe, dacwr, t9overlayoe                   : OUTPUT;
  phaseq[2..0]                                           : OUTPUT;
)
VARIABLE

  buffcsd2, buffcsd1, buffcsd0, overlaycsd           : NODE;
  buffnoed, buffnwrd3, buffnwrd2, buffnwrd1, buffnwrd0 : NODE;
  adcseld, t9buffoed, dacwrd, t9overlayoed           : NODE;
BEGIN
  buffcs2    = DFF(buffcsd2, pxck, VCC, VCC);
  buffcs1    = DFF(buffcsd1, pxck, VCC, VCC);
  buffcs0    = DFF(buffcsd0, pxck, VCC, VCC);
  overlaycs  = DFF(overlaycsd, pxck, VCC, VCC);
  buffnoe    = DFF(buffnoed, pxck, VCC, VCC);
  buffnwr3   = DFF(buffnwrd3, pxck, VCC, VCC);
  buffnwr2   = DFF(buffnwrd2, pxck, VCC, VCC);
  buffnwr1   = DFF(buffnwrd1, pxck, VCC, VCC);
  buffnwr0   = DFF(buffnwrd0, pxck, VCC, VCC);
  adcsel     = DFF(adcseld, pxck, VCC, VCC);
  t9buffoe   = DFF(t9buffoed, pxck, VCC, VCC);
  dacwr      = DFF(dacwrd, pxck, VCC, VCC);
  t9overlayoe = DFF(t9overlayoed, pxck, VCC, VCC);
  phaseq2    = DFF(phase2, pxck, VCC, VCC);
  phaseq1    = DFF(phase1, pxck, VCC, VCC);
  phaseq0    = DFF(phase0, pxck, VCC, VCC);

  buffcsd[2..0]  = (adcregcs[2..0] & !phase2 & !phase1)
                 # (dacregcs[2..0] & phase2 & !phase1)
                 # (smt9csbuff[2..0] & phase1);
  overlaycsd     = (phase2 & !phase1) # (smt9csoverlay & phase1);
  !buffnoed      = (phase2 & !phase1) # (!smt9noe & phase1);
  !buffnwrd[3..0] = (!phase2 & !phase1 & !phase0) # (!smt9nwr[3..0] & phase1);
  adcseld        = (!phase2 & !phase1);
  t9buffoed      = (!phase2 & !phase1) # (smt9buffoe & phase1);
  dacwrd         = phase2 & !phase1 & phase0;
  t9overlayoed   = smt9overlayoe & phase1;

END;
```

(system state machine)

```
SUBDESIGN tdivsta4
(
  reset, pxck : INPUT;
  phase[2..0] : OUTPUT;
)
VARIABLE
  ss: MACHINE OF BITS (phase[2..0])
     WITH STATES (
       s0 = B"000",
       s1 = B"001",
```

```
    s2 = B"010",
    s3 = B"011",
    s4 = B"100",
    s5 = B"101",
    s6 = B"110",
    s7 = B"111");

BEGIN
 ss.clk   = pxck;
 ss.reset = reset;
 CASE ss IS
   WHEN s0 => ss = s1;
   WHEN s1 => ss = s2;
   WHEN s2 => ss = s3;
   WHEN s3 => ss = s4;
   WHEN s4 => ss = s5;
   WHEN s5 => ss = s6;
   WHEN s6 => ss = s7;
   WHEN s7 => ss = s0;
 END CASE;
END;
```

(control signals associated with the vh_data_valid signal)

```
SUBDESIGN ctrlsig1
(
 vh_data_valid                                          : INPUT;
 tdiv_buffcs[2..0], tdiv_overlaycs, tdiv_buffnoe, tdiv_buffnwr[3..0] : INPUT;
 t9csbuff[2..0], t9csoverlay, t9noe, t9nwr[3..0]                : INPUT;
 t9state, tdiv_adcsel, t9buffoe, t9overlayoe, t9noeg            : INPUT;
 buffercs[2..0], overlaycs, buffnoe, buffnwr[3..0]              : OUTPUT;
 select_t9, adcsel, adcoe, outoe                                : OUTPUT;
)
BEGIN
 buffercs[2..0] = (tdiv_buffcs[2..0] & vh_data_valid)
            # (t9csbuff[2..0] & !vh_data_valid);
 overlaycs = (tdiv_overlaycs & vh_data_valid)
          # (t9csoverlay & !vh_data_valid);
 !buffnoe = (!tdiv_buffnoe & vh_data_valid)
          # (!t9noe & !vh_data_valid);
 !buffnwr[3..0] = (!tdiv_buffnwr[3..0] & vh_data_valid)
            # (!t9nwr[3..0] & !vh_data_valid);
 select_t9 = (t9state & vh_data_valid) # !vh_data_valid;
 adcsel = tdiv_adcsel & vh_data_valid;
 adcoe = (t9buffoe & vh_data_valid)
       # (t9noeg & !vh_data_valid);
 outoe = (t9overlayoe & vh_data_valid)
       # (t9noeg & !vh_data_valid);
END;
```

(horizontal state machine)

```
SUBDESIGN hcntstaa
(
 pxck, reset, first_pixel, last_pixel, lsync, fid, for_h_start  : INPUT;
 clrcnt, h_data_valid, latch_fid, h_start                       : OUTPUT;
)
 VARIABLE
```

```
 ss: MACHINE WITH STATES (s0, s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11, s12);
BEGIN
 ss.clk  = pxck;
 ss.reset = reset;

 CASE ss IS

  % revision a : h_start is a pulse to indicate the start of h_data_valid %
  % detect the falling edge of lsync %
  WHEN s0 =>          % stay in s0 until lsync is high %
   h_start = b"0";
   latch_fid = b"0";
   h_data_valid = b"0";
   clrcnt = b"0";
   IF lsync THEN
    ss = s1;
   ELSE ss = s0;
   END IF;
  WHEN s1 =>          % stay in s1 until lsync is low %
   h_start = b"0";
   latch_fid = b"0";
   h_data_valid = b"0";
   clrcnt = b"0";
   IF !lsync THEN
    ss = s2;
   ELSE ss = s1;
   END IF;

  WHEN s2 =>
   h_start = b"0";
   latch_fid = b"0";
   h_data_valid = b"0";
   clrcnt = b"1";      % clear counter %
   ss = s3;

  % detect the fid %
  WHEN s3 =>
   h_start = b"0";
   latch_fid = b"0";
   h_data_valid = b"0";
   clrcnt = b"0";
   IF fid THEN
    ss = s4;
   ELSE ss = s3;
   END IF;

  % latch the fid %
  WHEN s4 =>
   h_start = b"0";
   latch_fid = b"1";
   h_data_valid = b"0";
   clrcnt = b"0";
   ss = s5;

  % set latch_fid to b"0" %
  WHEN s5 =>
   h_start = b"0";
   latch_fid = b"0";
```

```
 h_data_valid = b"0";
 clrcnt = b"0";
 ss = s6;

% detect the rising edge of lsync %
  WHEN s6 =>          % stay in s6 until lsync is high %
 h_start = b"0";
 latch_fid = b"0";
 h_data_valid = b"0";
 clrcnt = b"0";
 IF lsync THEN
   ss = s7;
 ELSE ss = s6;
 END IF;

WHEN s7 =>
 h_start = b"0";
 latch_fid = b"0";
 h_data_valid = b"0";
 clrcnt = b"1";
 ss = s8;

WHEN s8 =>          % stay in s8 until for_h_start is found %
 h_start = b"0";
 latch_fid = b"0";
 h_data_valid = b"0";
 clrcnt = b"0";
 IF for_h_start THEN
   ss = s9;
 ELSE ss = s8;
 END IF;

WHEN s9 =>          % h_start is set to high in this state %
 h_start = b"1";
 latch_fid = b"0";
 h_data_valid = b"0";
 clrcnt = b"0";
 ss = s10;

WHEN s10 =>         % stay in s10 until first pixel is found %
 h_start = b"0";
 latch_fid = b"0";
 h_data_valid = b"0";
 clrcnt = b"0";
 IF first_pixel THEN
   ss = s11;
 ELSE ss = s10;
 END IF;

WHEN s11 =>         % the horizontal counter is cleared in this state %
 h_start = b"0";
 latch_fid = b"0";
 h_data_valid = b"1";
 clrcnt = b"1";
 ss = s12;

WHEN s12 =>         % data is valid in s10 %
 h_start = b"0";
```

```
    latch_fid = b"0";
    h_data_valid = b"1";
    clrcnt = b"0";
    IF last_pixel THEN
      ss = s0;
    ELSE ss = s12;
    END IF;

   WHEN OTHERS =>
      h_start = b"0";
    latch_fid = b"0";
    h_data_valid = b"0";
    clrcnt = b"0";
    ss = s0;

   END CASE;
END;
```

(horizontal counter)

```
CONSTANT true_field_id    = 57;  % it is counter[10..0] when the FID turns up %
CONSTANT true_for_h_start = 40;  % it is counter[10..2] when h_start should be triggered %
CONSTANT true_first_pixel = 42;  % it is counter[10..2] when the first pixel turns up %
CONSTANT true_last_pixel  = 160; % it is counter[10..3] when the last pixel turns up %
                                 % 640/4 = 160 %
CONSTANT sim_field_id     = 3;  % used for simulation only %
CONSTANT sim_for_h_start  = 5;  % used for simulation only %
CONSTANT sim_first_pixel  = 8;  % used for simulation only %
CONSTANT sim_last_pixel   = 12; % used for simulation only %
CONSTANT sim_last_pixel_1 = 40; % used for simulation only %
SUBDESIGN hcnt
(
 pxck, clr                                : INPUT;
 q[7..0], fid, first_pixel, last_pixel, for_h_start : OUTPUT;
)
VARIABLE
 counter[10..0] : DFF;   % using counter[10..3] to generate horizontal address %

BEGIN
 counter[].clk = pxck;
 counter[].clrn = !clr;
 counter[].d = counter[].q + 1;
 q[] = counter[10..3];                    % the address rate = pxck/8 %
 fid = (counter[7..0] == true_field_id);
 for_h_start = (counter[8..1] == true_for_h_start);
 first_pixel = (counter[10..2] == true_first_pixel);  % counter[10..1] = no. of pixels %
 last_pixel = (counter[10..3] == true_last_pixel);
END;
```

(vertical state machine)

```
SUBDESIGN vcntsta
(
 pxck, reset, first_line, last_line, vsync : INPUT;
 clrcnt, v_data_valid                      : OUTPUT;
)
VARIABLE
 ss: MACHINE WITH STATES (s0, s1, s2, s3, s4, s5, s6, s7);
```

```
BEGIN
 ss.clk  = pxck;
 ss.reset = reset;

 CASE ss IS
  % find the start of a field by means of detecting the falling edge of vsync %
  WHEN s0 =>          % stay in s0 until vsync is high %
   clrcnt = b"0";
   v_data_valid = b"0";
   IF vsync THEN
    ss = s1;
   ELSE ss = s0;
   END IF;

  WHEN s1 =>          % stay in s1 until vsync is low %
   clrcnt = b"0";
   v_data_valid = b"0";
   IF !vsync THEN
    ss = s2;
   ELSE ss = s1;
   END IF;
  WHEN s2 =>          % reset vcnt %
   clrcnt = b"1";
   v_data_valid = b"0";
   ss = s3;

  WHEN s3 =>
   clrcnt = b"0";
   v_data_valid = b"0";
   ss = s4;

  WHEN s4 =>          % the falling edge of vsync has been found %
   clrcnt = b"0";
   v_data_valid = b"0";
   IF first_line THEN
    ss = s5;
   ELSE ss = s4;
   END IF;

  WHEN s5 =>          % clear vcnt %
   clrcnt = b"1";
   v_data_valid = b"1";
   ss = s6;

  WHEN s6 =>
   clrcnt = b"0";
   v_data_valid = b"1";
   ss = s7;

  WHEN s7 =>          % v_data_valid in s7 %
   clrcnt = b"0";
   v_data_valid = b"1";
   IF last_line THEN
    ss = s0;
   ELSE ss = s7;
   END IF;

  END CASE;
```

```
END;


(vertical counter)
% There are 23 lsync pulses in the specified interval %
CONSTANT true_first_line =  35;      % It is a number used to indicate the first line %
CONSTANT true_last_line  = 255;      % It is a number used to indicate the last line  %
CONSTANT sim_first_line  =  10;      % It is a number used to indicate the first line in simulation %
CONSTANT sim_last_line   =  10;      % It is a number used to indicate the last line in simulation %
SUBDESIGN vcnt1
(
 enable, clr, pxck             : INPUT;
 q[7..0], first_line, last_line : OUTPUT;
)
VARIABLE
 counter[7..0] : DFFE;  % using counter[7..0] to generate vertical address %

BEGIN
 counter[].clk = pxck;
 counter[].clrn = !clr;
 counter[].ena = enable;
 counter[].d = counter[].q + 1;
 q[] = counter[7..0];
 first_line = (q[] == true_first_line);
 last_line = (q[] == true_last_line);
END;


(17-bit 2-channel-to-1-channel multiplexer)
SUBDESIGN 17b2to1b
(
 t9[16..0], vcnt[7..0], fid, hcnt[7..0], select_t9 : INPUT;
 out[16..0]                                : OUTPUT;
)
BEGIN
 out16 = (t916 & select_t9) #
       (vcnt7 & !select_t9) #
       (t916 & vcnt7);

 out15 = (t915 & select_t9) #
       (vcnt6 & !select_t9) #
       (t915 & vcnt6);

 out14 = (t914 & select_t9) #
       (vcnt5 & !select_t9) #
       (t914 & vcnt5);

 out13 = (t913 & select_t9) #
       (vcnt4 & !select_t9) #
       (t913 & vcnt4);

 out12 = (t912 & select_t9) #
       (vcnt3 & !select_t9) #
       (t912 & vcnt3);

 out11 = (t911 & select_t9) #
       (vcnt2 & !select_t9) #
       (t911 & vcnt2);
```

```
out10 = (t910 & select_t9) #
        (vcnt1 & !select_t9) #
        (t910 & vcnt1);

out9 =  (t99 & select_t9)  #
        (vcnt0 & !select_t9) #
        (t99 & vcnt0);

out8 =  (t98 & select_t9)  #
        (fid & !select_t9)   #
        (t98 & fid);

out7 =  (t97 & select_t9)  #
        (hcnt7 & !select_t9) #
        (t97 & hcnt7);

out6 =  (t96 & select_t9)  #
        (hcnt6 & !select_t9) #
        (t96 & hcnt6);

out5 =  (t95 & select_t9)  #
        (hcnt5 & !select_t9) #
        (t95 & hcnt5);

out4 =  (t94 & select_t9)  #
        (hcnt4 & !select_t9) #
        (t94 & hcnt4);

out3 =  (t93 & select_t9)  #
        (hcnt3 & !select_t9) #
        (t93 & hcnt3);

out2 =  (t92 & select_t9)  #
        (hcnt2 & !select_t9) #
        (t92 & hcnt2);

out1 =  (t91 & select_t9)  #
        (hcnt1 & !select_t9) #
        (t91 & hcnt1);

out0 =  (t90 & select_t9)  #
        (hcnt0 & !select_t9) #
        (t90 & hcnt0);

END;
```

(to synchronize the bus-multiplexing switching to the current buffer read/write cycle of T9000)

```
SUBDESIGN vhdvsta
(
  reset, pxck, t9nras1, vhdv : INPUT;  % t9nras1 showing chip enable %
  sync_vhdv                  : OUTPUT;
)
VARIABLE
  ss: MACHINE OF BITS (sync_vhdv)
     WITH STATES (s0 = b"0",
                  s1 = b"1");
```

```
BEGIN
 ss.clk   = pxck;
 ss.reset = reset;

 CASE ss IS
  WHEN s0 =>
   IF vhdv & t9nras1 THEN
    ss = s1;
   ELSE
    ss = s0;
   END IF;

  WHEN s1 =>
   IF !vhdv & t9nras1 THEN
    ss = s0;
   ELSE
    ss = s1;
   END IF;
 END CASE;
END;


(reset signal for T9000 transputer)
SUBDESIGN resett9
(
 nresetin, manual_reset : INPUT;
 t9reset                : OUTPUT;
)

BEGIN
 t9reset = !nresetin # manual_reset;
END;


(input/output decoder)
SUBDESIGN iodecod1
(
 a31, a29, a28, addr[3..0], t9nwr0, t9ncs, t9nps, address[2..0]   : INPUT;
 ena_swapfid, oe_framesync, reset_vh, ena_envhdv, ena_adcreg      : OUTPUT;
 ena_dacreg, reset_framsta, reset_intsta, adca0, adcncs, adcrdnwr : OUTPUT;
 dacrs[2..0], dacnwr, dacnrd, nresetadc, ena_adcdac               : OUTPUT;
)
VARIABLE
 t9selio : node;

BEGIN
 t9selio = a31 & a29 & !a28;                % 1010 = A %
 ena_swapfid = t9selio & ( addr[3..0] == h"0") & !t9nwr0;
 oe_framesync = t9selio & ( addr[3..0] == h"1");
 reset_vh = t9selio & ( addr[3..0] == h"2" );
 ena_envhdv = t9selio & ( addr[3..0] == h"3" ) & !t9nwr0;
 ena_adcreg = t9selio & ( addr[3..0] == h"4" ) & !t9nwr0;
 ena_dacreg = t9selio & ( addr[3..0] == h"6" ) & !t9nwr0;
 reset_framsta = t9selio & ( addr[3..0] == h"8" );
 reset_intsta = t9selio & ( addr[3..0] == h"9" );
 adca0 = t9selio & ( addr[3..0] == h"A" ) & address0;
 !adcncs = t9selio & ( addr[3..0] == h"A" ) & !t9ncs;
 adcrdnwr = t9nwr0;
```

```
 dacrs[2..0] = t9selio & ( addr[3..0] == h"B" ) & address[2..0];
 !dacnwr = t9selio & ( addr[3..0] == h"B" ) & !t9nwr0;
 !dacnrd = t9selio & ( addr[3..0] == h"B" ) & !t9nps;
 !nresetadc = t9selio & ( addr[3..0] == h"C" ) & !t9ncs;
 ena_adcdac = t9selio & ( addr[3..0] == h"D" ) & !t9nwr0;
END;
```

(address decoder)

```
SUBDESIGN adecode3
(
 a31, a29, a28, a22, a21, a20          : INPUT;
 t9selbuff, t9csbuff[2..0], t9csoverlay : OUTPUT;
)
BEGIN
 t9selbuff = a31 & !a29 & a28 & !a22;
 t9csbuff2 = a31 & !a29 & a28 & !a22 & a21 & !a20;
 t9csbuff1 = a31 & !a29 & a28 & !a22 & !a21 & a20;
 t9csbuff0 = a31 & !a29 & a28 & !a22 & !a21 & !a20;
 t9csoverlay = a31 & !a29 & a28 & a22;
END;
```

(state machine for control bus)

```
SUBDESIGN cbussta7
(
 reset, pxck, t9nras1, t9noeg, state[1..0]              : INPUT;  % t9nras1 showing chip enable %
 t9csbuff[2..0], t9csoverlay, t9nwr[3..0], t9selbuff    : INPUT;  % t9noeg showing read operation %
 smt9csbuff[2..0], smt9csoverlay, smt9noe, smt9nwr[3..0] : OUTPUT;
 latch_for_read, statebit[2..0]                         : OUTPUT;
 smt9buffoe, smt9overlayoe                              : OUTPUT;
)
VARIABLE
 ss: MACHINE OF BITS (statebit[2..0])
    WITH STATES (
        s0  = B"000",
        s10 = B"001",
        s20 = B"011",
        s21 = B"010",
        s22 = B"100",
        s30 = B"101",
        s31 = B"111",
        s32 = B"110");

% statebit 0 is used for t9memwait %

 code[2..0] : NODE;
 latch_for_read_d : NODE;

BEGIN
 ss.clk   = pxck;
 ss.reset = reset;
 code2 = t9noeg;
 code1 = state1;
 code0 = state0;
 latch_for_read = DFF(latch_for_read_d, pxck, VCC, VCC);

 CASE ss IS
```

```
% control signals generated for t9 read and write operation %
WHEN s0 =>                          % t9memwait = 0 %
 smt9csbuff[2..0] = b"000";
 smt9csoverlay = b"0";
 smt9noe = b"1";
 smt9nwr[3..0] = b"1111";
 smt9buffoe = b"0";
 smt9overlayoe = b"0";
 latch_for_read_d = b"0";
 IF !t9nras1 THEN
  ss = s10;
 ELSE
  ss = s0;
 END IF;

WHEN s10 =>                         % t9memwait = 1 %
 smt9csbuff[2..0] = b"000";
 smt9csoverlay = b"0";
 smt9noe = b"1";
 smt9nwr[3..0] = b"1111";
 smt9buffoe = b"0";
 smt9overlayoe = b"0";
 latch_for_read_d = b"0";
 CASE code[] IS
  WHEN B"100" =>                    % write operation %
   ss = s20;
  WHEN B"000" =>                    % read operation %
   ss = s30;
  WHEN OTHERS =>
   ss = s10;
 END CASE;

WHEN s20 =>          % start of t9 write operation, t9memwait = 1 %
 smt9csbuff[2..0] = t9csbuff[2..0];
 smt9csoverlay = t9csoverlay;
 smt9noe = b"1";
 smt9nwr[3..0] = t9nwr[3..0];
 smt9buffoe = t9selbuff;
 smt9overlayoe = !t9selbuff;
 latch_for_read_d = b"0";
 ss = s21;

WHEN s21 =>                         % t9memwait = 0 %
 smt9csbuff[2..0] = t9csbuff[2..0];
 smt9csoverlay = t9csoverlay;
 smt9noe = b"1";
 smt9nwr[3..0] = b"1111";
 smt9buffoe = t9selbuff;
 smt9overlayoe = !t9selbuff;
 latch_for_read_d = b"0";
 ss = s22;

WHEN s22 =>                         % t9memwait = 0 %
 smt9csbuff[2..0] = b"000";
 smt9csoverlay = b"0";
 smt9noe = b"1";
 smt9nwr[3..0] = b"1111";
 smt9buffoe = b"0";
```

```
    smt9overlayoe = b"0";
    latch_for_read_d = b"0";
    IF t9nras1 THEN
      ss = s0;
    ELSE ss = s22;
    END IF;              % end of t9 write operation %

  WHEN s30 =>            % start of t9 read operation,  t9memwait = 1 %
    smt9csbuff[2..0] = t9csbuff[2..0];
    smt9csoverlay = t9csoverlay;
    smt9noe = b"0";
    smt9nwr[3..0] = b"1111";
    smt9buffoe = b"0";
    smt9overlayoe = b"0";
    latch_for_read_d = b"0";
    ss = s31;

  WHEN s31 =>                        % t9memwait = 1 %
    smt9csbuff[2..0] = t9csbuff[2..0];
    smt9csoverlay = t9csoverlay;
    smt9noe = b"0";
    smt9nwr[3..0] = b"1111";
    smt9buffoe = b"0";
    smt9overlayoe = b"0";
    latch_for_read_d = b"1";
    ss = s32;

  WHEN s32 =>                        % t9memwait = 0 %
    smt9csbuff[2..0] = b"000";
    smt9csoverlay = b"0";
    smt9noe = b"1";
    smt9nwr[3..0] = b"1111";
    smt9buffoe = b"0";
    smt9overlayoe = b"0";
    latch_for_read_d = b"1";
    IF t9noeg THEN
      ss = s0;
    ELSE ss = s32;
    END IF;              % end of t9 read operation %
  END CASE;
END;
```

Appendix C - The Memory Configuration File for the DIPB

```
Processor.Type      := T9000
Dram.Refresh.Interval := 300 Cycles
Dram.Refresh.Time    := 2 Cycles
Dram.Refresh.RAS.High := 2 Phases

Bank 2 "DRAM BANK 2"
   Device.Type          := Dram
   Cacheable
   Wait.Pin             := disabled
   Bank.Base.Address    := 80000000
   Bank.Address.Mask    := FF800000
   Port.Size            := 32 bits
   --Page.Address.Bits     := 007FF000  -- this value is not working
   Page.Address.Bits    := 001FFFFC   -- this value is working well
   Page.Address.Shift   := 10 bits    -- 11 refresh address bits

   Ras.Strobe "RAS2"
      Active.during      := Read & Write
      Time.to.falling.edge := 2 Phases
      Time.to.rising.edge  := 8 Phases
   End.Strobe

   Cas.Strobe "CAS2"
      Active.during      := Read & Write
      Time.to.falling.edge := 2 Phases
      Time.to.rising.edge  := 7 Phases
   End.Strobe

   Programmable.Strobe "OE2"
      Active.during      := Read
      Time.to.falling.edge := 0 Phases
      Time.to.rising.edge  := 7 Phases
   End.Strobe

   Write.Strobe "WRITE2"
      Active.during      := Write
      Time.to.falling.edge := 0 Phases
      Time.to.rising.edge  := 7 Phases
   End.Strobe

   Ras.Precharge.Time   := 1 Cycles
   Ras.Edge.Time        := 2 Phases    -- 1 Phases is not working
   Ras.Cycle.Time       := 1 Cycles
   Cas.Cycle.Time       := 2 Cycles
   Bus.Release.Time     := 1 Cycles
End.Bank

Bank 1 "SRAM BANK 1"

   Device.Type          := Non-Dram
   Wait.Pin             := enabled   -- enabled normally working
   --Cacheable
   Bank.Base.Address    := 90000000
   Bank.Address.Mask    := FF000000
   Port.Size            := 32 bits
```

```
  Ras.Strobe "RAS1"
    Active.during       := Read & Write
    Time.to.falling.edge := 2 Phases  -- 2 Phases normally working
    Time.to.rising.edge  := 14 Phases -- 14 Phases normally working
  End.Strobe

  Cas.Strobe "CAS1" -- /OEG
    Active.during       := Read
    Time.to.falling.edge := 2 Phases  -- 2 Phases normally working
    Time.to.rising.edge  := 14 Phases -- 14 Phases normally working
  End.Strobe

  Programmable.Strobe "PS1"  -- /OE
    Active.during       := Read
    Time.to.falling.edge := 2 Phases  -- 2 Phases normally working
    Time.to.rising.edge  := 13 Phases -- 13 Phases normally working
  End.Strobe

  Write.Strobe "WRITE1"  -- /WE
    Active.during       := Write
    Time.to.falling.edge := 2 Phases  -- 2 Phases normally working
    Time.to.rising.edge  := 13 Phases -- 13 Phases normally working
  End.Strobe

  Cycle.Time            := 4 Cycles  -- 4 Cycles normally working
  Bus.Release.Time      := 2 Cycles  -- 2 Cycles normally working
End.Bank

Bank 3 "I/O BANK 3"

  Device.Type           := Non-Dram
  Device.Only
  Wait.Pin              := disabled
  Bank.Base.Address     := A0000000
  Bank.Address.Mask     := FF000000
  Port.Size             := 32 bits

  Cas.Strobe "CAS3" -- /CS
    Active.during       := Read & Write
    Time.to.falling.edge := 2 Phases      -- 8, 2 Phases or higher set for adc
    Time.to.rising.edge  := 6 Phases      -- 15, 6 Phases
  End.Strobe

  Programmable.Strobe "PS3"  -- /OE
    Active.during       := Read
    Time.to.falling.edge := 1 Phases      -- 4, 1 phases set for dac
    Time.to.rising.edge  := 5 Phases      -- 12, 5 phases set for dac
  End.Strobe

  Write.Strobe "WRITE3"  -- /WE
    Active.during       := Write
    Time.to.falling.edge := 1 Phases      -- 4, 1 phases set for dac
    Time.to.rising.edge  := 5 Phases      -- 12, 5 phases set for dac
  End.Strobe

  Cycle.Time            := 2 Cycles      -- 4, 2 normally working
  Bus.Release.Time      := 1 Cycles      -- 1, 1 normally working
End.Bank
```

Appendix D - The Memory Configuration File for the IMS B927

```
-- -----
--
--    Memory configuration file produced
--    by the MEM package.
--
--
--    This file is intended for use on a 20MHz, 8Mbyte, 64bit Falcon HTRAM.
-- -----

Processor.Type       := T9000
Dram.Refresh.Interval := 300 Cycles
Dram.Refresh.Time    := 2 Cycles
Dram.Refresh.RAS.High := 2 Phases

Bank 0 "DRAM BANK 0"
   Device.Type          := Dram
   Cacheable
   Wait.Pin             := disabled
   Bank.Base.Address    := 80000000
   Bank.Address.Mask    := FFE00000
   Port.Size            := 64 bits
   --Page.Address.Bits    := 000FFC00
   Page.Address.Bits    := 001FFFFC
   Page.Address.Shift   := 9 bits

   Ras.Strobe "RAS0"
      Active.during      := Read & Write
      Time.to.falling.edge := 0 Phases
      Time.to.rising.edge  := 16 Phases
   End.Strobe

   Cas.Strobe "CAS0"
      Active.during      := Read & Write
      Time.to.falling.edge := 2 Phases
      Time.to.rising.edge  := 15 Phases
   End.Strobe

   Programmable.Strobe "PS0"
   End.Strobe

   Write.Strobe "WRITE0"
      Active.during      := Write
      Time.to.falling.edge := 0 Phases
      Time.to.rising.edge  := 15 Phases
   End.Strobe

   Ras.Precharge.Time   := 1 Cycles
   Ras.Edge.Time        := 2 Phases
   Ras.Cycle.Time       := 1 Cycles
   Cas.Cycle.Time       := 4 Cycles
   Bus.Release.Time     := 1 Cycles
End.Bank


Bank 1 "DRAM BANK 1"
```

```
  Device.Type          := Dram
  Cacheable
  Wait.Pin             := disabled
  Bank.Base.Address    := 80200000
  Bank.Address.Mask    := FFE00000
  Port.Size            := 64 bits
  Page.Address.Bits    := 001FFFFC
  Page.Address.Shift   := 9 bits

  Ras.Strobe "RAS1"
    Active.during      := Read & Write
    Time.to.falling.edge := 0 Phases
    Time.to.rising.edge  := 16 Phases
  End.Strobe

  Cas.Strobe "CAS1"
    Active.during      := Read & Write
    Time.to.falling.edge := 2 Phases
    Time.to.rising.edge  := 15 Phases
  End.Strobe

  Programmable.Strobe "PS1"
  End.Strobe

  Write.Strobe "WRITE1"
    Active.during      := Write
    Time.to.falling.edge := 0 Phases
    Time.to.rising.edge  := 15 Phases
  End.Strobe

  Ras.Precharge.Time   := 1 Cycles
  Ras.Edge.Time        := 2 Phases
  Ras.Cycle.Time       := 1 Cycles
  Cas.Cycle.Time       := 4 Cycles
  Bus.Release.Time     := 1 Cycles
End.Bank


Bank 2 "DRAM BANK 2"
  Device.Type          := Dram
  Cacheable
  Wait.Pin             := disabled
  Bank.Base.Address    := 80400000
  Bank.Address.Mask    := FFE00000
  Port.Size            := 64 bits
  Page.Address.Bits    := 001FFFFC
  Page.Address.Shift   := 9 bits

  Ras.Strobe "RAS2"
    Active.during      := Read & Write
    Time.to.falling.edge := 0 Phases
    Time.to.rising.edge  := 16 Phases
  End.Strobe

  Cas.Strobe "CAS2"
    Active.during      := Read & Write
    Time.to.falling.edge := 2 Phases
    Time.to.rising.edge  := 15 Phases
```

```
  End.Strobe

  Programmable.Strobe "PS2"
  End.Strobe

  Write.Strobe "WRITE2"
    Active.during       := Write
    Time.to.falling.edge := 0 Phases
    Time.to.rising.edge  := 15 Phases
  End.Strobe

  Ras.Precharge.Time    := 1 Cycles
  Ras.Edge.Time         := 2 Phases
  Ras.Cycle.Time        := 1 Cycles
  Cas.Cycle.Time        := 4 Cycles
  Bus.Release.Time      := 1 Cycles
End.Bank


Bank 3 "DRAM BANK 3"
  Device.Type           := Dram
  Cacheable
  Wait.Pin              := disabled
  Bank.Base.Address     := 80600000
  Bank.Address.Mask     := 01000000
  Port.Size             := 64 bits
  Page.Address.Bits     := 001FFFFC
  Page.Address.Shift    := 9 bits

  Ras.Strobe "RAS3"
    Active.during       := Read & Write
    Time.to.falling.edge := 0 Phases
    Time.to.rising.edge  := 16 Phases
  End.Strobe

  Cas.Strobe "CAS3"
    Active.during       := Read & Write
    Time.to.falling.edge := 2 Phases
    Time.to.rising.edge  := 15 Phases
  End.Strobe

  Programmable.Strobe "PS3"
  End.Strobe

  Write.Strobe "WRITE3"
    Active.during       := Write
    Time.to.falling.edge := 0 Phases
    Time.to.rising.edge  := 15 Phases
  End.Strobe

  Ras.Precharge.Time    := 1 Cycles
  Ras.Edge.Time         := 2 Phases
  Ras.Cycle.Time        := 1 Cycles
  Cas.Cycle.Time        := 4 Cycles
  Bus.Release.Time      := 1 Cycles
End.Bank
```

Appendix E - The Initialisation Procedures for the DIPB

```
#INCLUDE "hostio.inc"   -- contains SP protocol
#INCLUDE "linkaddr.inc" -- contains event channel addresses

PROC WorldP (CHAN OF SP HostInput, HostOutput, CHAN OF [64]INT64 ImageData1, CHAN OF
[64]INT64 ImageData2, VAL INT32 Count)

  #USE "hostio.lib" -- iserver libraries

  --{{{  Frame buffer locations
  [512][256]INT buffer0data, buffer1data, buffer2data:
  PLACE buffer0data AT #4000000:
  PLACE buffer1data AT #4040000:
  PLACE buffer2data AT #4080000:
  --}}}

  --{{{  Overlay buffer locations
  [512][256]INT overlaydata:
  PLACE overlaydata AT #4100000:  -- #9040 0000 in machine map
  --}}}

  --{{{  I/O port locations
  PORT OF INT TMC22071notA0, TMC22071A0, TMC22071notRESET:
  PLACE TMC22071notA0       AT #8280000:  -- #A0A00000 in machine map
  PLACE TMC22071A0          AT #8284000:  -- #A0A10000 in machine map
  PLACE TMC22071notRESET AT #8300000:  -- #A0C00000 in machine map

  PORT OF INT changefid, framesync:
  PORT OF INT envhdv, resetvh, adcreg, dacreg, resetframsta:
  PLACE changefid      AT #8000000:  -- #A0000000 in machine map
  PLACE framesync      AT #8040000:  -- #A0100000 in machine map
  PLACE envhdv         AT #80C0000:  -- #A0300000 in machine map
  PLACE resetvh        AT #8080000:  -- #A0200000 in machine map
  PLACE adcreg         AT #8100000:  -- #A0400000 in machine map
  PLACE dacreg         AT #8180000:  -- #A0600000 in machine map
  PLACE resetframsta   AT #8200000:  -- #A0800000 in machine map

  PORT OF INT adcdac:
  PLACE adcdac         AT #8340000:  -- #A0D00000 in machine map

  PORT OF INT nrs2.nrs1.nrs0:
  PLACE nrs2.nrs1.nrs0  AT #82C0000:  -- #A0B00000 in machine map

  PORT OF INT nrs2.nrs1.rs0:
  PLACE nrs2.nrs1.rs0   AT #82C4000:  -- #A0B10000 in machine map

  PORT OF INT nrs2.rs1.nrs0:
  PLACE nrs2.rs1.nrs0   AT #82C8000:  -- #A0B20000 in machine map

  PORT OF INT nrs2.rs1.rs0:
  PLACE nrs2.rs1.rs0    AT #82CC000:  -- #A0B30000 in machine map

  PORT OF INT rs2.nrs1.nrs0:
  PLACE rs2.nrs1.nrs0   AT #82D0000:  -- #A0B40000 in machine map

  PORT OF INT rs2.nrs1.rs0:
```

```
  PLACE rs2.nrs1.rs0    AT #82D4000:  -- #A0B50000 in machine map
  PORT OF INT rs2.rs1.nrs0:
  PLACE rs2.rs1.nrs0    AT #82D8000:  -- #A0B60000 in machine map

  PORT OF INT rs2.rs1.rs0:
  PLACE rs2.rs1.rs0     AT #82DC000:  -- #A0B70000 in machine map
  --}}}

--{{{  PROC reset.adc ()
  PROC reset.adc ()
    TMC22071notRESET ! INT(#00)   -- write any value to this port
  :
  --}}}
  --{{{  PROC write.control.word ()  -- Write a specific control word
  PROC write.control.word ()  -- Write a specific control word
    --INT i:
    INT CtrlWd0, CtrlWd8, CtrlWd16, CtrlWd24, CtrlWd32, CtrlWd40, Buffer:
    SEQ
      CtrlWd0  := #90          -- 1001 0000, bit 0,1,...,7
      CtrlWd8  := #00          -- 0000 0000, bit 8,9,...,15
      CtrlWd16 := #6F          -- 0110 1111, bit 16,17,...,23
      CtrlWd24 := #03          -- 0000 0011, bit 24,25,...,31
      CtrlWd32 := #60          -- 0110 0000, bit 32,33,...,39
      CtrlWd40 := #C2          -- 1100 0010, bit 40,41,...46, LSB is useless

      --{{{  Write CtrlWd40
      Buffer := CtrlWd40 >> 1    -- shift one bit to right
      SEQ i = 0 FOR 7
        SEQ
          TMC22071notA0 ! Buffer     -- write Buffer to bit 46,45,...,40
          Buffer := Buffer >> 1      -- shift one bit to right
      --}}}
      --{{{  Write CtrlWd32
      Buffer := CtrlWd32
      SEQ i = 0 FOR 8
        SEQ
          TMC22071notA0 ! Buffer     -- write Buffer to bit 39,38,...,32
          Buffer := Buffer >> 1      -- shift one bit to right
      --}}}
      --{{{  Write CtrlWd24
      Buffer := CtrlWd24
      SEQ i = 0 FOR 8
        SEQ
          TMC22071notA0 ! Buffer     -- write Buffer to bit 31,30,...,24
          Buffer := Buffer >> 1      -- shift one bit to right
      --}}}
      --{{{  Write CtrlWd16
      Buffer := CtrlWd16
      SEQ i = 0 FOR 8
        SEQ
          TMC22071notA0 ! Buffer     -- write Buffer to bit 23,22,...,16
          Buffer := Buffer >> 1      -- shift one bit to right
      --}}}
      --{{{  Write CtrlWd8
      Buffer := CtrlWd8
      SEQ i = 0 FOR 8
        SEQ
          TMC22071notA0 ! Buffer     -- write Buffer to bit 15,14,...,8
```

```
      Buffer := Buffer >> 1      -- shift one bit to right
    --}}}
    --{{{  Write CtrlWd0
    Buffer := CtrlWd0
    SEQ i = 0 FOR 7
      SEQ
        TMC22071notA0 ! Buffer     -- write Buffer to bit 7,6,...,1
        Buffer := Buffer >> 1      -- shift one bit to right

    TMC22071A0 ! Buffer            -- write Buffer to bit 0 and transfer$
    so.write.nl(HostInput,HostOutput)            -- data to Control Register
    so.write.string.nl(HostInput,HostOutput,"...Done")
    --}}}
  :
  --}}}

--{{{  PROC reset.epld()
  PROC reset.epld()
    SEQ
      resetvh ! INT(#00)
      resetframsta ! INT(#00)
  :
  --}}}

--{{{  PROC write.color.palette ()
  PROC write.color.palette ()
    SEQ
      nrs2.nrs1.nrs0 ! #00  -- address register (palette write mode)
      SEQ i = 0 FOR 256
        SEQ
          nrs2.nrs1.rs0 ! i
          --so.write.string(HostInput, HostOutput,"Writing R ")
          nrs2.nrs1.rs0 ! i
          --so.write.string(HostInput, HostOutput,"Writing G ")
          nrs2.nrs1.rs0 ! i
          --so.write.string(HostInput, HostOutput,"Writing B ")

  :
  --}}}

  --{{{  PROC write.overlay ()
  PROC write.overlay ()
    SEQ
      rs2.nrs1.nrs0 ! #00  -- address register (overlay write mode)
      SEQ i = 1 FOR 15
        SEQ
          rs2.nrs1.rs0 ! i
          rs2.nrs1.rs0 ! i
          rs2.nrs1.rs0 ! i
  :
  --}}}

--{{{  PROC clear.overlay.buffer()
  PROC clear.overlay.buffer()
    SEQ
      so.write.string(HostInput, HostOutput, "Write starts *n")
      SEQ i = 0 FOR 512
        SEQ j=0 FOR 256
```

```
        SEQ
          overlaydata[i][j] := #00
          --so.write.hex.int(HostInput, HostOutput, i, 8)
          --so.write.string(HostInput, HostOutput, " ")
    so.write.string(HostInput, HostOutput, "Write finished *n")
  :
  --}}}

--{{{  PROC initialisation()
  PROC initialisation()
    SEQ
      reset.adc()
      write.color.palette()
      write.overlay()
      clear.overlay.buffer()
      adcreg ! #01
      dacreg ! #01
      reset.adc()
      write.control.word()
      envhdv ! #01
      reset.epld()
  :
  --}}}


  initialisation () ---- Main Program
:
```

Appendix F - The Seam Tracking Source Programme

```
#include <misc.h>
#include <stdlib.h>
#include <time.h>
#include <process.h>
#include <stdio.h>
#include <process.h>
#include <iocntrl.h>

#include "hardware.h"
#include "filters.h"
#include "wings.h"

void tracking(unsigned char data[ysize][xsize], int WINDOW_HEIGHT);

int window_top = 0, window_bot = 511, window_left = 0 , window_right = 639;
int pos_disp, diff_disp;

/* ACTIVE WINDOW VARIABLES */
int pos_points, half_window_width, half_window_height;

/* ARRAY POINTERS */
int x, y, i, j, c, image_bot, image_end;

/* AVERAGING AND DIFERENTIAL COREECTION OFFSETS */
int av_offset, diff_offset, final_offset, x_offset;

/* VARIABLES FOR PRODUCING POSITIONAL ARRAY */
int average_val, array_index, biggest_val, biggest_loc;
int averaging_array[20], av_intensity[xwindow], pos[ywindow];

/* ADDITIONAL VARIABLES FOR PRODUCING SMOOTHED POSITIONAL ARRAY */
int smoothing_array[20], av_pos[ywindow];

/* ADDITIONAL VARIABLES FOR PRODUCING DIFFERENTIAL ARRAY */
int diff_points, diff[ywindow], av_diff[ywindow];

/* ADDITIONAL VARIABLES FOR FINDING THE SEAM EDGES ON THE PROFILE */
int final_points, bot_diff, top_diff, top_loc, bot_loc;
int valid_top_peak, valid_bot_peak;

/* ADDITIONAL VARIABLES FOR VALIDATING THE SEAM EDGES */
int max_hump, edge_threshold, top_edge, bot_edge, width, last_width;
int image_data_valid;

/* ADDITIONAL VARIABLES FOR EXTRACTING SEAM AND LASER POSITIONS */
int seam_pos, laser_pos, line_pos_total, line_points;


Channel *in, *out;

int main()
{
int   buffer, xstart, xstop, ystart, ystop;
int WINDOW_HEIGHT;
```

```
    printf("Process to do local seam tracking runing\n");
    printf("About to get channels\n");
    in = get_param(3);
    out = get_param(4);
    printf("Got other channels \n");

  /******** GET IMAGE PROCESSING OPERATIONAL VALUES FROM CONTROL ********/
  #define SAMPLE_SPACE          2
  #define VER_AVERAGING         10
  #define HOR_AVERAGING         10
  #define INTENSITY_THRESHOLD   10
  #define WINDOW_WIDTH          80
  #define DIFF_SPACING          5
  #define TOPEDGE_THRESHOLD     2
  #define BOTEDGE_THRESHOLD     -2
  #define HUMP_THRESHOLD        25
  #define SEAM_WIDTH            50
  #define MIN_WIDTH             20
  #define MAX_WIDTH             150
/*  #define WINDOW_HEIGHT       150*/

  image_bot = ywindow - 1;
  image_end = xwindow - 1;
  av_offset  = VER_AVERAGING/2;
  x_offset   = HOR_AVERAGING/2;
  half_window_width  = WINDOW_WIDTH/2;
/*  half_window_height = WINDOW_HEIGHT/2;*/
  diff_offset    = VER_AVERAGING + DIFF_SPACING;
  final_offset   = diff_offset + VER_AVERAGING;
  edge_threshold = HUMP_THRESHOLD/2;
  last_width = SEAM_WIDTH;

    do  {
            buffer = ChanInInt(in);

            WINDOW_HEIGHT = ChanInInt(in);
            half_window_height = WINDOW_HEIGHT/2;
            if (buffer == 0)
                tracking(buffer0data, WINDOW_HEIGHT);
            else if (buffer == 1)
                tracking(buffer1data, WINDOW_HEIGHT);
            else if (buffer == 2)
                tracking(buffer2data, WINDOW_HEIGHT);
            ChanOutInt(out, 1);

        }while (1);
    exit(EXIT_SUCCESS);
    return(0);
}


void tracking(unsigned char data[ysize][xsize], int WINDOW_HEIGHT)
{
  /************************ IMAGE PROCESSING ALGORITHM ************************/

    /************** GET START INSTRUCTION, START TIMING **************/
    /*
    proc_start = ChanInInt(fControl);
```

```
start_time = clock();
*/
/********************** GET LASER LINE PROFILE *****************/
pos_points = (window_bot-window_top) / SAMPLE_SPACE;
/*printf("pos_points = %d\n", pos_points);*/
/* FILL ARRAY POS WITH LINE CENTRE POSITIONS */
for (i = 0; i <= pos_points; i++) {
   y = window_top + (i * SAMPLE_SPACE);
   for (j = 0; j <= HOR_AVERAGING; j++)  averaging_array[j] = 0;
   average_val = 0;
   array_index = 0;
   biggest_val = INTENSITY_THRESHOLD;
   biggest_loc = 0;
   for (x = window_left; x <= window_right; x++) {
      average_val = average_val - averaging_array[array_index];
      averaging_array[array_index] = data[y][x];
      average_val = average_val + averaging_array[array_index];
      av_intensity[x] = average_val  / HOR_AVERAGING ;
      if (av_intensity[x] > biggest_val) {
        biggest_val = av_intensity[x];
        biggest_loc = x;
       }
      if (array_index >= (HOR_AVERAGING-1)) array_index = 0;
      else                                  array_index++;
    }
   /* CHECK FOR VALID INTENSITY LEVELS */
   if (biggest_val > INTENSITY_THRESHOLD) {
     window_left = biggest_loc - half_window_width;
     if (window_left < 0)          window_left  = 0;
     window_right = biggest_loc + half_window_width;
     if (window_right > image_end)  window_right = image_end;
     pos[i] = biggest_loc;
     /*printf("window_left = %d\n", window_left);*/
    }
   else {              /* no pixels bright enough (set to -1) */
     window_left  = 0;          /* reset scan limits */
     window_right = image_end;
     pos[i] = -1;
    }
}
/* FILL (-1) HOLES IN Y_POS ARRAY */
c = 1;
while (pos[0] == (-1)) {
    if (pos[c] == (-1)) {
       c++;
       if (c >= pos_points)  pos[0] = (xwindow/2);
      }
    else pos[0] = pos[c];
}
for (i = 1; i <= pos_points; i++) {
   if (pos[i] == (-1))       pos[i] = pos[i-1];
}

/******** SMOOTH THE POS ARRAY TO PRODUCE THE AVPOS ARRAY *******/
for (j = 0; j <= VER_AVERAGING; j++) smoothing_array[j] = 0;
average_val = 0;
array_index = 0;
for (i = 0; i <= pos_points; i++) {
```

```
    average_val = average_val - smoothing_array[array_index];
    smoothing_array[array_index] = pos[i];
    average_val = average_val + smoothing_array[array_index];
    av_pos[i] = average_val / VER_AVERAGING;
    if     (av_pos[i] < 0)    av_pos[i] = 0;
    else if (av_pos[i] > image_bot)  av_pos[i] = image_bot;
    if (array_index == (VER_AVERAGING-1)) array_index = 0;
    else                              array_index++;
  }

  /****************** GET DIFFERENTIAL PROFILE ******************/
  /* GET DIFFERENTIAL PROFILE DIFF FROM THE SMOOTHED ARRAY */
  diff_points  = pos_points - (VER_AVERAGING + (2*DIFF_SPACING));
  for (i = diff_offset; i <= (diff_offset+diff_points); i++) {
    diff[i] = av_pos[i+DIFF_SPACING] - av_pos[i-DIFF_SPACING];
  }
  for (i = 0; i < VER_AVERAGING; i++)  smoothing_array[i] = 0;
  average_val = 0;
  array_index = 0;
  for (i = diff_offset; i <= (diff_offset+diff_points); i++) {
    average_val = average_val - smoothing_array[array_index];
    smoothing_array[array_index] = diff[i];
    average_val = average_val + smoothing_array[array_index];
    av_diff[i]  = average_val / VER_AVERAGING;
    if (array_index == (VER_AVERAGING-1))  array_index = 0;
    else                              array_index++;
  }

  /************************** FIND SEAM EDGES **********************/
  final_points = diff_points - (2*VER_AVERAGING);
  valid_top_peak = 0;
  valid_bot_peak = 0;
  top_diff = TOPEDGE_THRESHOLD;
  bot_diff = BOTEDGE_THRESHOLD;
  for (i = final_offset; i <= (final_offset+final_points); i++) {
    if (av_diff[i] >= top_diff) {                /* +ve peak */
      top_diff = av_diff[i];
      top_loc  = i;
      valid_top_peak = 1;
    }
    else if (av_diff[i] <= bot_diff) {           /* -ve peak */
      bot_diff = av_diff[i];
      bot_loc = i;
      valid_bot_peak = 1;
    }
  }

  /**************** COMBINE PEAKS AND VALIDATE EDGES *************/
  image_data_valid = 0;

  /** BOTH EDGES ARE VALID, THRESHOLD HUMP **/
  if (valid_top_peak && valid_bot_peak) {
    max_hump = top_diff - bot_diff;
    if (max_hump >= HUMP_THRESHOLD) {
      /* HUMP MARKING SEAM IS FOUND */
      top_edge = window_top + (top_loc * SAMPLE_SPACE);
      bot_edge = window_top + (bot_loc * SAMPLE_SPACE);
      width    = bot_edge - top_edge;
```

```
    if ((width>MIN_WIDTH) && (width<MAX_WIDTH)) {
      image_data_valid = 1;
      last_width = width;
              /*printf("first\n");*/
    }
    else {
      /* INVALID SEAM WIDTH */
      if (width > 0) {
        /* CORRECT THE WEAKER EDGE */
        if ((0 - bot_diff) >= top_diff) top_edge = bot_edge - last_width;
        else  bot_edge = top_edge + last_width;
        image_data_valid = 1;
        printf("second\n");
       }
      else image_data_valid = 0;
    }
  }
  else {
    /* CAN'T FIND HUMP - TRY AND FIND ONE EDGE */
    if (top_diff > edge_threshold) {
      /* TOP EDGE IS OK - CORRECT THE BOTTOM EDGE */
      top_edge = window_top + (top_loc * SAMPLE_SPACE);
      bot_edge = top_edge + last_width;
      image_data_valid = 1;
      printf("third\n");
    }
    else if ((0 - bot_diff) > edge_threshold) {
      /* BOTTOM EDGE IS OK - CORRECT THE TOP EDGE */
      bot_edge = window_top + (bot_loc * SAMPLE_SPACE);
      top_edge = bot_edge - last_width;
      image_data_valid = 1;
      printf("fourth\n");
    }
    else image_data_valid = 0;
  }
}
/********** TOP EDGE IS VALID - NO BOTTOM EDGE **********/
else if (valid_top_peak && (!valid_bot_peak)) {
  if (top_diff > edge_threshold) {
    /* GOOD TOP EDGE - ADD BOTTOM EDGE */
    top_edge = window_top + (top_loc * SAMPLE_SPACE);
    bot_edge = top_edge + last_width;
    image_data_valid = 1;
    printf("fifth\n");
  }
  else image_data_valid = 0;
}
/********** BOTTOM EDGE IS VALID - NO TOP EDGE **********/
else if (valid_bot_peak && (!valid_top_peak)) {
  if ((0 - bot_diff) > edge_threshold) {
    /* GOOD BOTTOM EDGE - ADD TOP EDGE */
    bot_edge = window_top + (bot_loc * SAMPLE_SPACE);
    top_edge = bot_edge - last_width;
    image_data_valid = 1;
    printf("sixth\n");
  }
  else image_data_valid = 0;
}
```

```
else image_data_valid = 0;

/*************** DISPLAY AND EXTRACT DATA ***************/
if (image_data_valid) {
  top_edge = top_edge - (VER_AVERAGING * (SAMPLE_SPACE+1));
  bot_edge = bot_edge - ((VER_AVERAGING/2) * SAMPLE_SPACE);
  if (top_edge < 0)          top_edge = 0;
  if (bot_edge > image_bot)   bot_edge = image_bot;
  seam_pos = (top_edge + bot_edge)/2;
}
else {
  top_edge = 200;
  bot_edge = 300;
  seam_pos = 250;
}
/* FIND LASER LINE POSITION AT SEAM EDGE */
if (image_data_valid) {
  line_pos_total = 0;
  line_points = pos_points - VER_AVERAGING;
  for (x = VER_AVERAGING; x <= pos_points; x++) {
     line_pos_total = line_pos_total + av_pos[x];
     laser_pos = (line_pos_total/line_points) - x_offset;
     if (laser_pos > image_end)    laser_pos = image_end;
     else if (laser_pos < 0)       laser_pos = 0;
  }
}
/******************* DISPLAY FEATURES *******************/
/* DISPLAY LASER LINE POSITION PROFILE */

for (y = window_top, i = 0; y < window_bot; y = y + SAMPLE_SPACE, i++)
{
   pos_disp = av_pos[i];
   if (pos_disp > 639)     pos_disp = 639;
   else if (pos_disp < 0)  pos_disp = 0;
   data[y][pos_disp] = (unsigned char) 0x00;
   data[y+1][pos_disp] = (unsigned char) 0x00;

}


/* DISPLAY DIFFERENTIAL PROFILE */

for (y =window_top, i = 0; y < window_bot; y = y + SAMPLE_SPACE, i++)
{
   diff_disp = (2 * av_diff[i]) + (laser_pos+75);
   if (diff_disp > 639)    diff_disp = 639;
   else if (diff_disp < 0) diff_disp = 0;
   data[y][diff_disp] = (unsigned char) 0xff;
   data[y+1][diff_disp] = (unsigned char) 0xff;
}


/* PROCESSING COMPLETE - RETURN RESULTS */
for (i=0; i < 100; i++)
{
 data[top_edge][laser_pos+i] = (unsigned char) 0;
 data[top_edge+1][laser_pos+i] = (unsigned char) 0;
        data[bot_edge][laser_pos+i] = (unsigned char) 0;
```

```
                data[bot_edge+1][laser_pos+i] = (unsigned char) 0;
        }
        /*
        ChanOutInt(tControl, image_data_valid);
        if (image_data_valid) {
          ChanOutInt(tControl, top_edge);
            ChanOutInt(tControl, bot_edge);
            ChanOutInt(tControl, laser_pos);
        }
        */
        /***************** SET NEW WINDOW VALUES **************/
        if (image_data_valid && (window_top < window_bot)) {
          /*printf("top_edge = %d bot_edge = %d\n", top_edge, bot_edge);*/
          window_top = top_edge - half_window_height;
          if (window_top < 0)                          window_top = 0;
          else if (window_top > (image_bot-WINDOW_HEIGHT))  window_top = image_bot -
WINDOW_HEIGHT;
          window_bot = bot_edge + half_window_height;
          if (window_bot < WINDOW_HEIGHT)    window_bot = WINDOW_HEIGHT;
          else if (window_bot > image_bot)   window_bot = image_bot;
                /*printf("window_top = %d window_bot = %d\n", window_top, window_bot);*/
        }
        else {
                printf("Reset Window size\n");
          window_top = 0;
          window_bot = image_end;
        }
        /*
        stop_time = clock();
        proc_time = (1000 * (stop_time - start_time)) / 15625;
        ChanOutInt(tControl, proc_time);
        */

}
```