A cambridge ring node controller

Paul C. Bunn
University of Wollongong, uow_bunnp@uow.edu.au

THE UNIVERSITY OF WOLLONGONG

DEPARTMENT OF COMPUTING SCIENCE

A CAMBRIDGE RING NODE CONTROLLER

Paul C. Bunn

Department of Computing Science
University of Wollongong

Abstract

This report gives a discussion of the design of a peripheral controller chip to interface a Cambridge Ring Node to a MC68000 host machine using VLSI technology. The design of such a chip involves following a design strategy. The strategy used identifies what the interface should be capable of doing, developing a functional description of the interface, mapping the interface onto silicon and finally verifying the design. The characteristics of the interface and the design strategy, along with the software used in the design, will be discussed.

Preprint No 85-8 **January 1985**

P.O. Box 1144, WOLLONGONG, N.S.W. AUSTRALIA
telephone (042)-270-859
telex AA29022

THE UNIVERSITY OF WOLLONGONG

DEPARTMENT OF COMPUTING SCIENCE

A CAMBRIDGE RING NODE CONTROLLER

Paul C. Bunn

Department of Computing Science
University of Wollongong

Abstract

This report gives a discussion of the design of a peripheral controller chip to interface a Cambridge Ring Node to a MC68000 host machine using VLSI technology. The design of such a chip involves following a design strategy. The strategy used identifies what the interface should be capable of doing, developing a functional description of the interface, mapping the interface onto silicon and finally verifying the design. The characteristics of the interface and the design strategy, along with the software used in the design, will be discussed.

Preprint No 85-8 **January 1985**

P.O. Box 1144, WOLLONGONG, N.S.W. AUSTRALIA
telephone (042)-270-859
telex AA29022

A Cambridge Ring Node Controller

The Department of Computing Science
University of Wollongong

Paul C. Bunn

Acknowledgements

I would particularly like to thank my two supervisors, Dr. Richard Hartley for the many hours of entertaining discussions on a wide variety of subjects and the router and pla logic programmes which he wrote, without which the project would have never been completed on time and Mr. Phillip McKerrow for his excellent supervision and valuable suggestions on the different ways of approaching the more difficult aspects of the project.

There are quite a number of people that were involved at some stage during the project particularly John Fulcher, Gary Kelly, Ross Nealon and Michael Milway. I thank you all for your patience and advice.

1. Introduction

This report gives a discussion of the design of a peripheral controller chip to interface a Cambridge Ring Node to a MC68000 host machine using VLSI technology.

The design of such a chip involves following a design strategy. The strategy used identifies what the interface should be capable of doing, developing a functional description of the interface, mapping the interface onto silicon and finally verifying the design.

The characteristics of the interface and the design strategy along with the software used in the design will be discussed in the following sections.

2. The Problem

The problem is to design a peripheral controller chip using VLSI design techniques, which will interface a Cambridge Ring Node to a host machine, such that the host machine can initiate transmission and reception of data to and from the network via the ring node efficiently. There should be three data transmission modes, status handshake, interrupt driven and full duplex DMA. The target machine chosen for the interface is the MC68000.

2.1. The Interface Hardware

Before discussing the characteristics of the controller chip it is necessary to understand the capabilities of and the differences between the hardware that is to be interfaced. This section discusses the Cambridge Ring Node and the MC68000 hardware.

2.1.1. The Cambridge Ring

The Cambridge Ring is a local area communications system. It is a passive ring network based on the empty slot principle. The ring is simply a single cable loop to connect peripherals to a computer. Each peripheral on the ring communicates with other devices by sending a 'mini-packet' containing addressing information and data. For a typical ring configuration see Fig. 1.

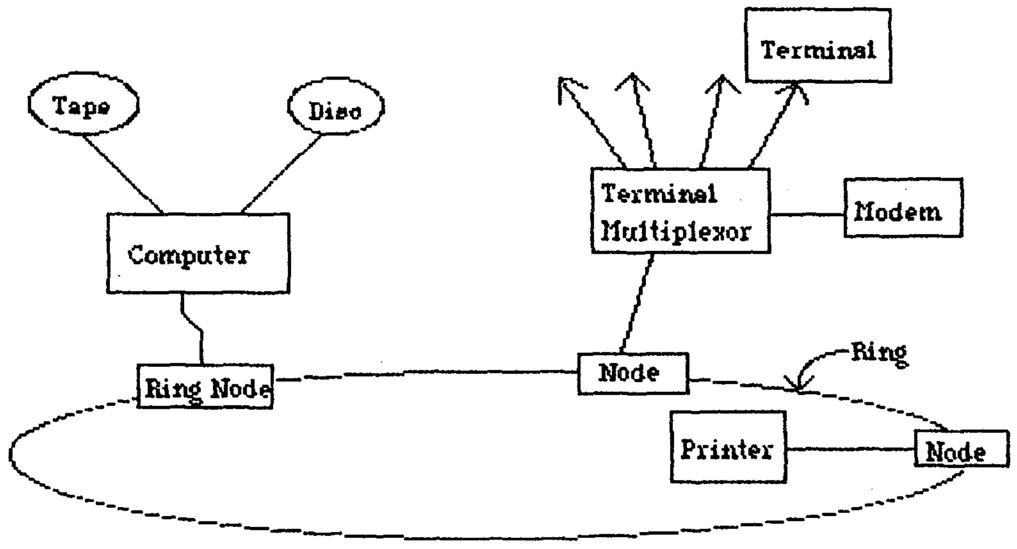


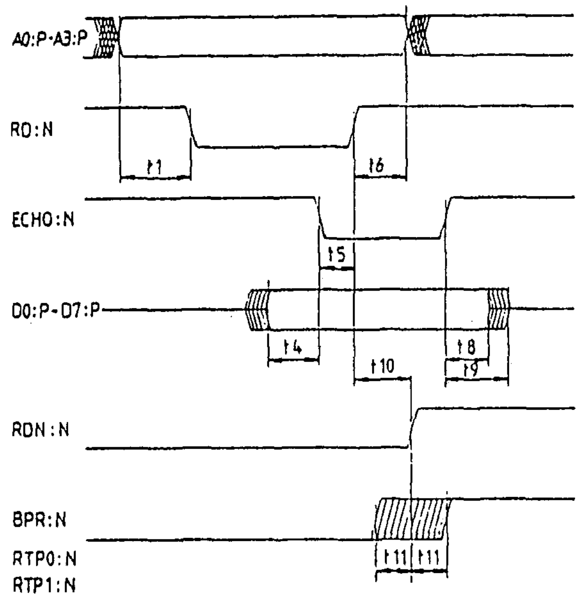
Figure 1. A typical Cambridge Ring configuration.

The peripherals interface to the ring through a 'ring node'. The ring node has an asynchronous bus as does the MC68000. The requests to send and receive data to and from the network are given to the ring node. The ring node has two data transmission request signals, one for reception of data (the read strobe), the other for transmission of data (the write strobe). When data is to be transmitted to the network the transmit signal must be asserted. Similarly, to receive data from the network the reception signal must be asserted. Another signal of importance that the node has, is the Node Read or Write Acknowledge Signal. This signal is asserted by the node to indicate to the device that the ring node is ready to continue with the current operation.

There are two other handshake signals that are necessary to make correct data transmissions and receptions. These transmit done and receive done signals indicate to the device whether the current receive or transmit operation is complete. Thus a device can monitor these signals to determine when a data transfer is complete and therefore when to start the next data transfer (see Fig. 2 for receive request and transmit request timing diagrams).

The ring node also has an internal set of registers. These are accessed by reading or writing to the node. The correct register is selected by placing the appropriate address on the address bus.

The ring node has two signals which enable two error conditions to be monitored by the interface chip. The first error condition uses the Transmit Error signal which indicates that the last mini-packet transmitted was corrupted. The second error condition uses the Transmit Clock signal which can be used to count the number of transmission retries that have been initiated for a single transmission and therefore can detect unsuccessful transmissions (see Fig 3 for ring node interface). For more details on the Cambridge Ring Network see The Polynet network manual [1].



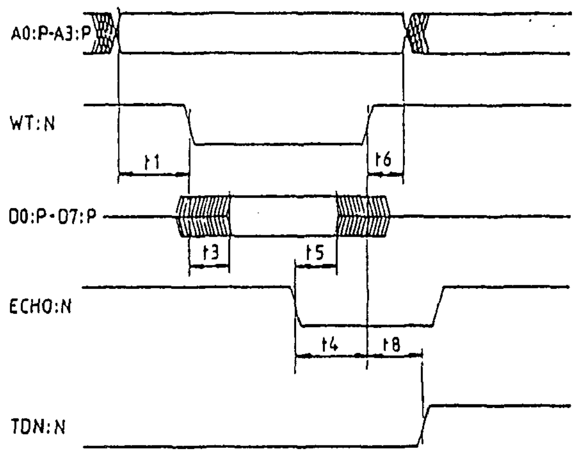


Figure 2. a) Node receive request timing diagram. b) Node transmit request timing diagram.

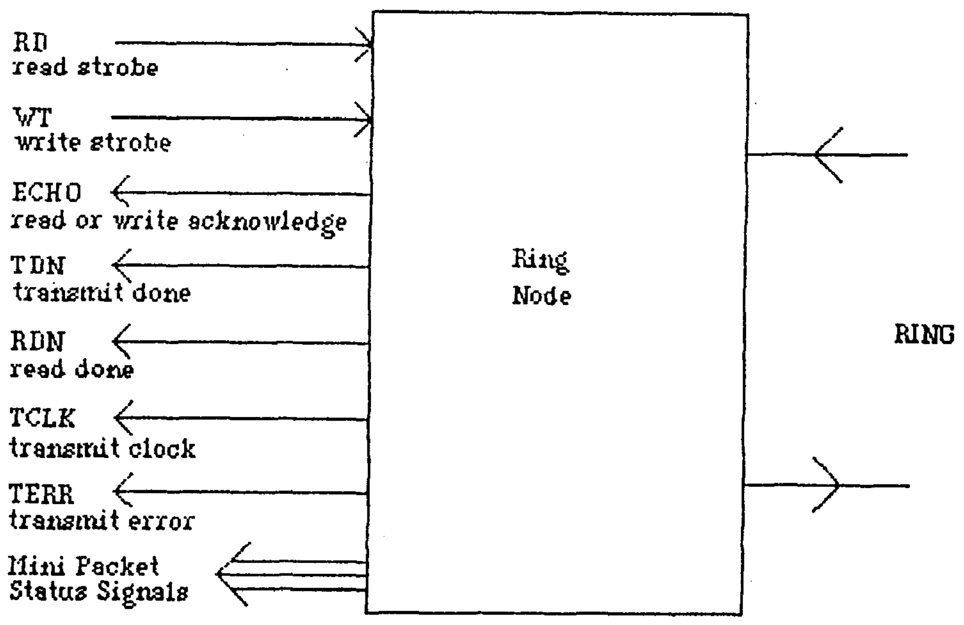
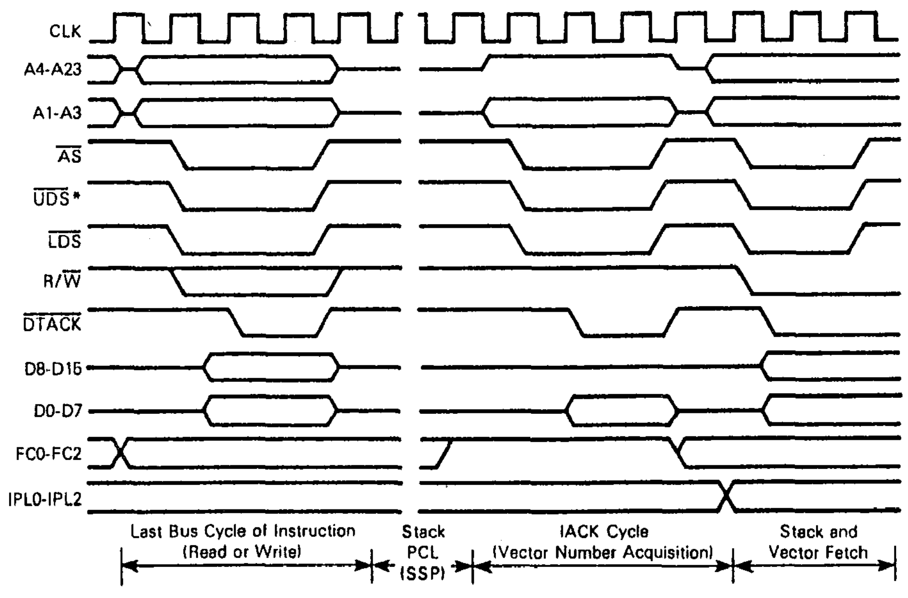


Figure 3. Cambridge ring node device interface.

2.1.2. MC68000 Hardware

The MC68000 is a 16 bit state of the art micro-processor. It has 32 bit registers and versatile address modes. The bus structure is asynchronous with a 24-bit address bus and a 16-bit data bus.

The MC68000 has a priority driven interrupt structure which allows any device to specify an interrupt vector number from which the MC68000 calculates the address of the interrupt service routine. If the priority of the pending interrupt is greater than the current processor priority, the MC68000 will start up an exception processing sequence (See MC68000 Advanced Information [2]). Figure 4 shows the interrupt acknowledge timing diagram. The interrupt acknowledge consists of two parts, these being the Vector Number Acquisition and the Stack and Vector Fetch.



* Although a vector number is one byte, both data strobes are asserted due to the microcode used for exception processing. The processor does not recognize anything on data lines D8 through D15 at this time.

Figure 4. MC68000 Interrupt Acknowledge timing diagram.

For device addressing the MC68000 has four handshake signals. The first, the Address Strobe, indicates to the device that there is a valid address on the address bus. Secondly, the Lower Data Strobe indicates, depending on the state of the Read/Write signal, that data may be placed on or latched from the data bus. The Read/Write signal indicates to the device what the current processor cycle is. The fourth signal, Data Transfer Acknowledge, indicates that the data transfer is completed. When the processor recognises DTACK during a read cycle, data is latched and the bus cycle is terminated. When DTACK is recognised during a write cycle, the bus cycle is terminated. (see Fig 5 for MC68000 interface).

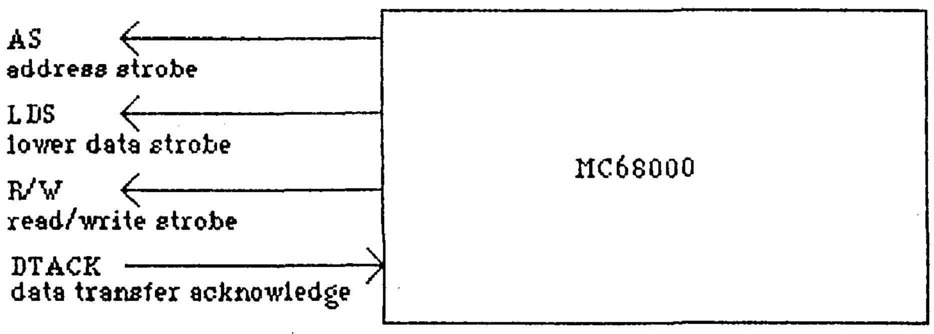


Figure 5. MC68000 device interface.

Another device compatible with the MC68000, which makes data transfers to and from a device faster is, the MC68440 Dual-Channel Direct Memory Access Controller (DDMA). The MC68440 will perform memory-to-memory, memory-to-peripheral and peripheral-to-peripheral data transfers. It has an asynchronous bus structure compatible with the MC68000 microprocessor. The DDMA has two separate device interface channels which have three channel-specific signals and two shared signals. The channel-specific signals are the DMA request, DMA acknowledge and the device ready signals. The shared signals are the Data Transfer Complete signal and DMA done signal (see Fig. 6 for MC68440 interface).

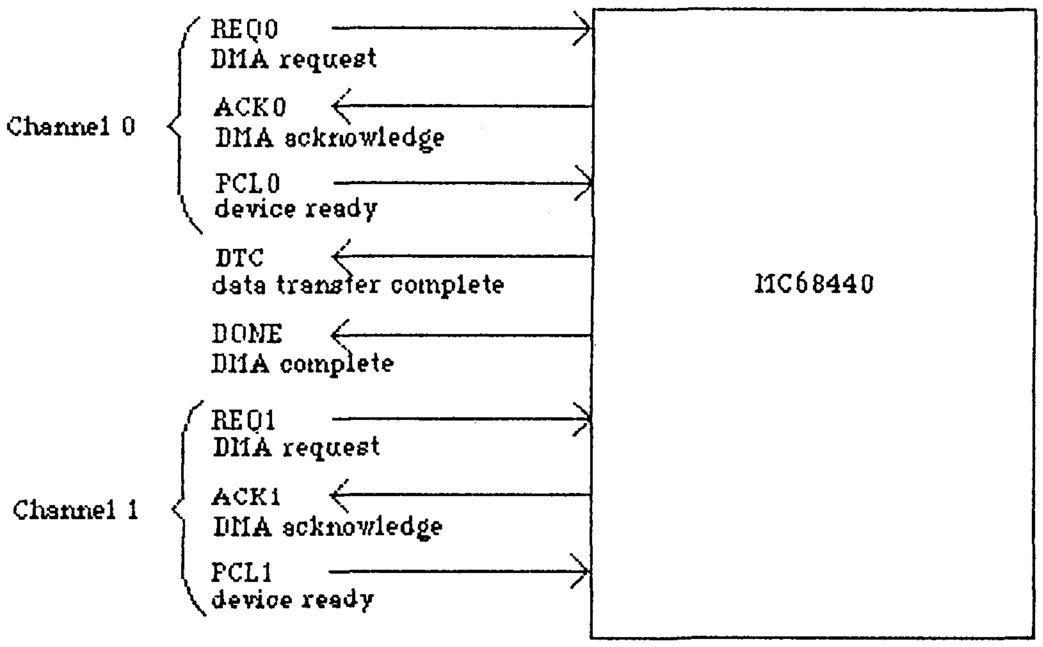


Figure 6. MC68000 deceive interface.

The DDMA is used in cycle steal mode. To initiate a cycle steal transfer the DMA Request signal must be asserted for two falling edges of the DDMA clock. When the request is acknowledged by the DDMA the request line must then be released. (see Fig. 7 for cycle steal timing diagram).

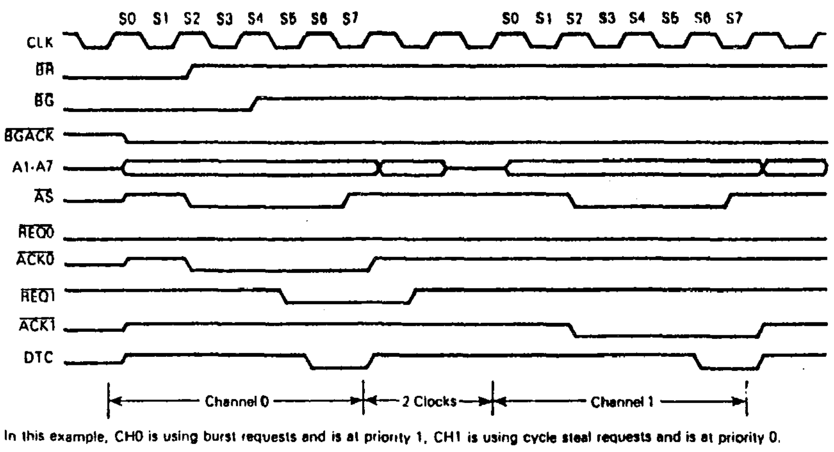

S0 S1 S2 S3 S4 S5 S6 S7 S0 S1 S2 S3 S4 S5 S6 S7
CLK
BR
BG
BGACK
A1-A7
AS
REQ0
ACK0
REQ1
ACK1
DTC
Channel 0
2 Clocks
Channel 1
In this example, CH0 is using burst requests and is at priority 1, CH1 is using cycle steal requests and is at priority 0.

Figure 7. MC68440 DMA request timing diagram. Channel 0 is using *burst* and channel 1 is using *cycle steal*.

2.2. The Interface

The interface chip must co-ordinate data transfers between the ring node and memory using three devices (shown in Fig 8) such that the programmer can initiate data transfers easily and effectively.

The interface chip must be flexible in its data transfer modes and should also do transmission error checking. The error checking utilises the transmit error and transmit clock signals of the node. Error checking will make data transfers more reliable.

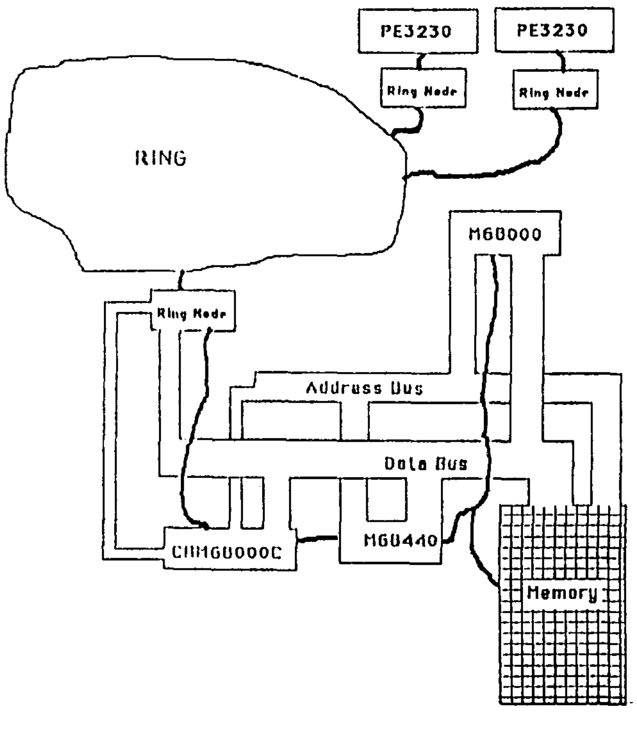


Figure 8. Typical ring configuration encorporating the interface chip.

2.2.1. Data Transfer Modes

The data transfer modes chosen are *status handshake*, *interrupt* and *DMA* driven data transfers. These transfer modes are discussed in the following sections.

Status Handshake should be the simplest of all three data transfer modes. To access an internal register or initiate a data transfer by the ring node, the node must be addressed explicitly through the interface chip. To initiate data transfers using status handshake a ring node address should be placed on the lower two bits of the address bus. The state of the MC68000's Read/Write signal will determine the direction of the data transfer i.e. whether receive or transmit.

The next mode is interrupt driven data transfers. The interface chip should have four different types of interrupts. Two of these interrupts should be mini-packet receive and mini-packet transmission interrupts. The other two interrupts should be the transmission error interrupts. Each interrupt should have a priority associated with it in respect to the other three. The order of priority is receive , retry time-out, transmit error and transmit interrupt.

The interrupt request should be made by simply asserting an interrupt request line and the response to the interrupt acknowledge signal asserted by the host machine should be compatible with the standard MC68000 interrupt acknowledge cycle. The MC68000 acknowledge cycle requires an interrupt vector number to be placed on the data bus. Internal to the interface chip, the upper five bits of the vector number should be software preset to allow flexibility of the position of the interrupt routines in the host machine. The lower three bits of the vector number are specified by the type of interrupt that has occurred (refer to diagrams of register contents in Appendix 3 of the Users Manual).

The third data transfer mode takes advantage of the MC68440 DDMA. When using the DDMA to make a data transfer the interface chip should control transmission and reception of data to and from the ring node without interrupting the processor. After the DDMA and the interface chip have been correctly set up, the interface chip should automatically generate DMA and node data transmission and reception requests to transfer data to and from the host machine and the network. When DMA data transfers are in progress the data transfer interrupts should be automatically disabled so that a transfer interrupt does not occur (N.B. the error interrupts are not disabled). This will allow the interface chip to interrupt the processor as soon as the node is ready to transfer data when a DMA transfer is complete. Note that when doing DMA transfers the interface chip will allow the programmer to access the node's internal registers and will also allow the programmer to initiate a receive or transmit request. The hardware does not stop the programmer doing this.

During any DMA transfer it is necessary for the interface chip to address the node. For this reason the interface chip must have the address of the node stored internally. This means that the address bus for the node must run through the interface chip so it can be multiplexed with the internally stored address during DMA transfers to select the node.

The Error Checking should be interrupt driven or flagged. These two error conditions make up the other two interrupts. Transmit error is set whenever the transmit error signal is asserted by the node. The transmit time-out error involves counting the transmit clock and comparing the number counted with the preset maximum number of retries. When this number is reached the error is set.

To control the interface chip, and its data transfer modes,**four internal registers** are required. The first register should store the current node status, that is, the values of the transmit and receive done signals, values of the mini-packet status signals and two error condition flags. The second register should be reserved for the control of the data transfer modes. The DMA and interrupt data transfer enable bits and the address of the node (which are needed during DMA transfers). The third register should be reserved for the two error condition interrupts and the upper five bits of the interrupt vector number. The fourth should be used to specify the maximum number of retries before a transmit time-out error is set.

In **summary**, the interface chip should allow data transfers in three different modes, Status Handshake, Interrupt and DMA transfers. These may be combined to meet differing host machine configurations. To accompany the data transfer modes there should be three registers for the internal control of the interface chip and one to store the current node status. There should also be two error conditions that the interface chip should detect, transmit and transmit time-out errors.

3. Functional Description

The interface chip can be divided into separate functional blocks (figure 9). These blocks are the interrupt handling, address handling, transmit retry counting, DMA reception and transmission, address multiplexing, status handshake and interface control, and the registers.

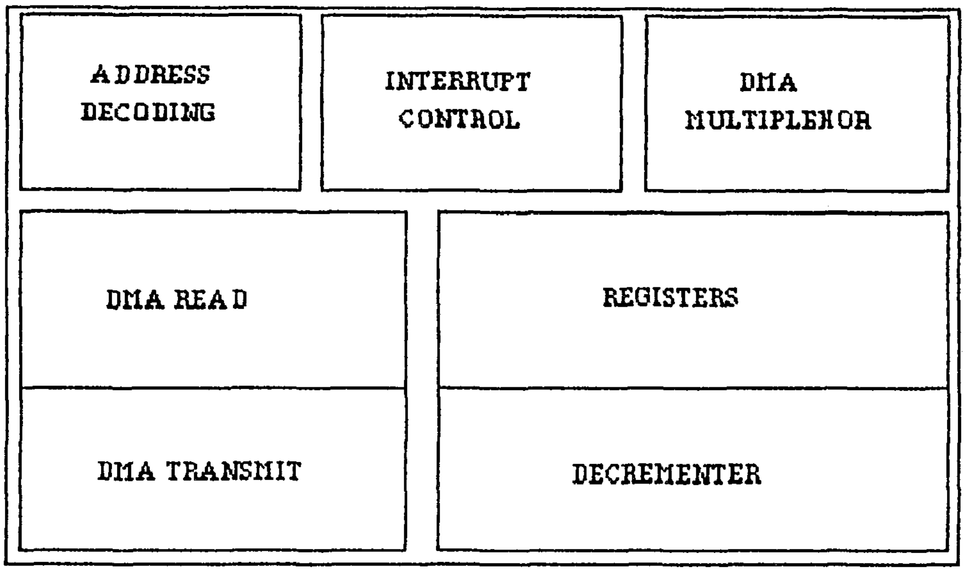


Figure 9. The functional representation of the interface chip.

The **address decoding functional block** (figure 10) determines whether the interface chip is addressed or if the ring node is addressed. If the interface chip has been addressed then the address decoding must determine which of the internal registers is being addressed and whether it is being written or read. If the ring node is being addressed the address selects the node. The address functional block must indicate to the status handshake and interface logic what has been selected. It must also determine when an interrupt acknowledge is asserted and enable the appropriate interrupt vector number onto the data bus.

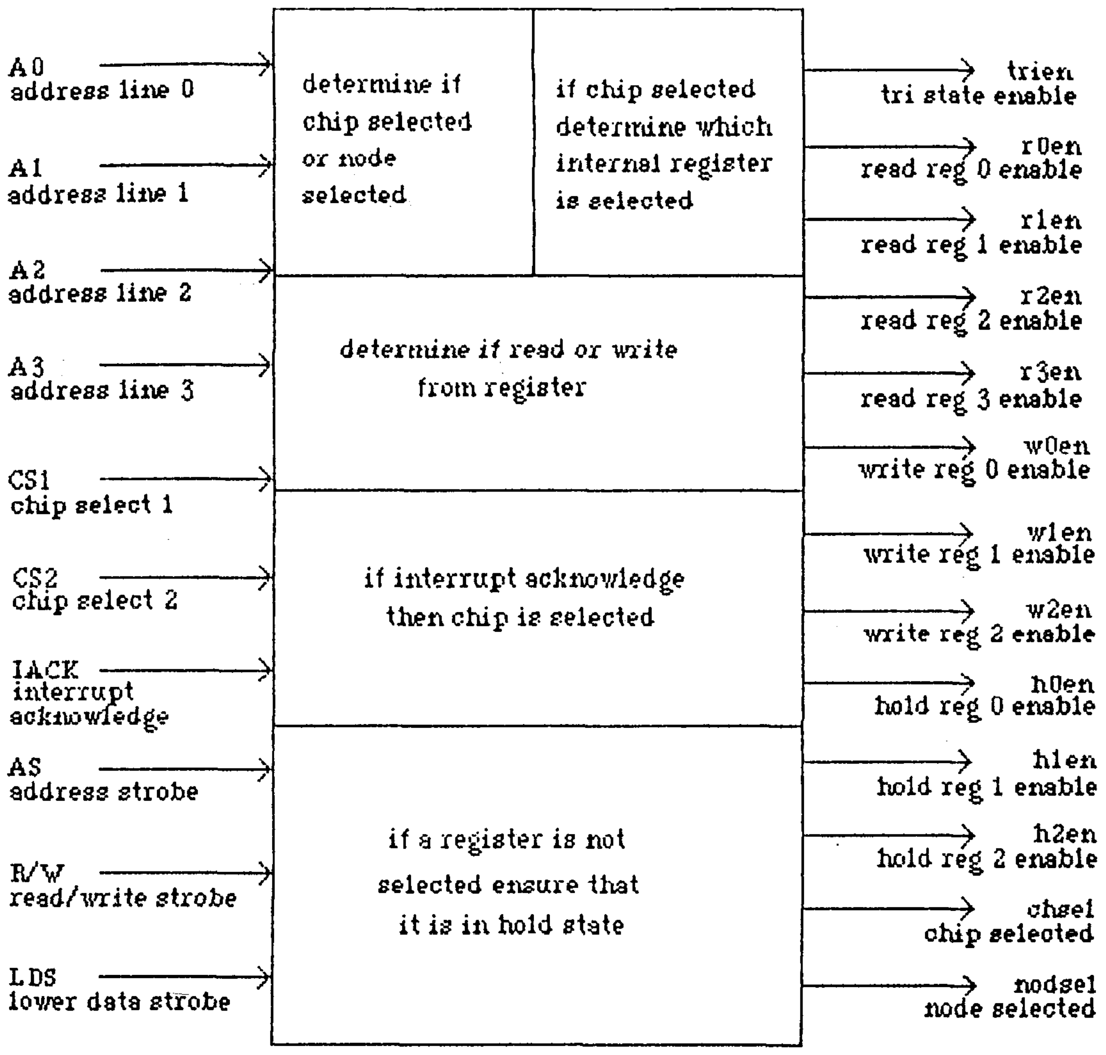


Figure 10. The address decoding functional block.

The **interrupt functional block** (figure 11) determines when an interrupt should be set. If more than one interrupt condition arises, the interrupt logic must determine which interrupt has the highest priority, and then set it. When the interrupt acknowledge is asserted the lower three bits of the interrupt vector number must be ready to be placed on the data bus.

The interrupt functional block is also affected by the DMA functional block. Whenever a receive or transmit DMA transfer is initiated the corresponding interrupt is inhibited until the DMA transfer is complete.

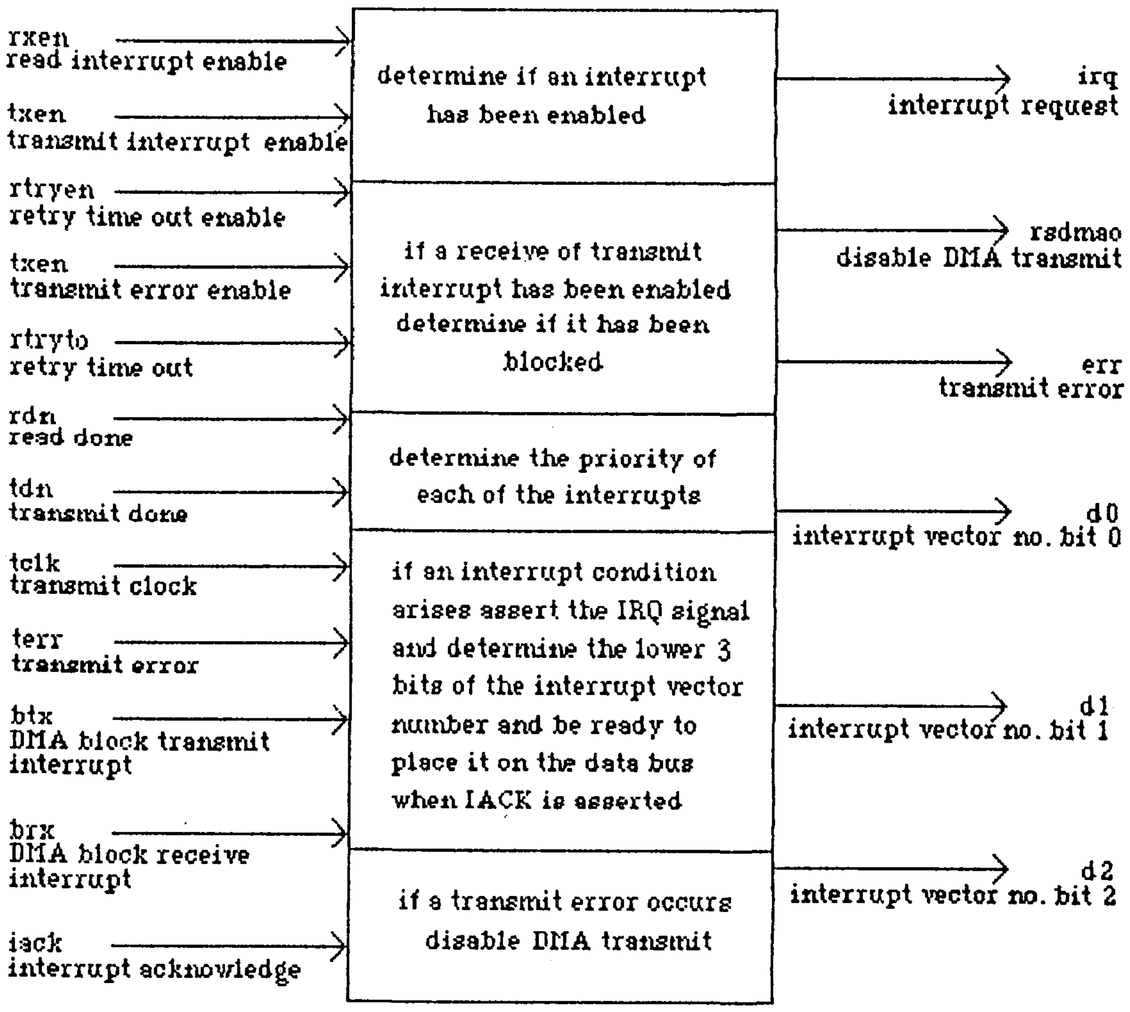

rxen
read interrupt enable
txen
transmit interrupt enable
rtryen
retry time out enable
txen
transmit error enable
rtryto
retry time out
rdn
read done
tdn
transmit done
tclk
transmit clock
terr
transmit error
btx
DMA block transmit interrupt
brx
DMA block receive interrupt
iack
interrupt acknowledge
determine if an interrupt has been enabled
if a receive of transmit interrupt has been enabled determine if it has been blocked
determine the priority of each of the interrupts
if an interrupt condition arises assert the IRQ signal and determine the lower 3 bits of the interrupt vector number and be ready to place it on the data bus when IACK is asserted
if a transmit error occurs disable DMA transmit
irq
interrupt request
rsdmao
disable DMA transmit
err
transmit error
d0
interrupt vector no. bit 0
d1
interrupt vector no. bit 1
d2
interrupt vector no. bit 2

Figure 11. The interrupt functional block.

The **transmit retry counting functional block** (figure 12) consists of three sub-sections, the transmit time-out register, the down counter and the comparator. The value in the retry time-out register must be used to reset the down-counter latch when the current transmission is complete. The down-counter must decrement the value in the latch whenever sixteen retries have been counted for the current transmission. When the value in the down-counter latch is zero the comparator must set the retry time-out signal to the interrupt functional block.

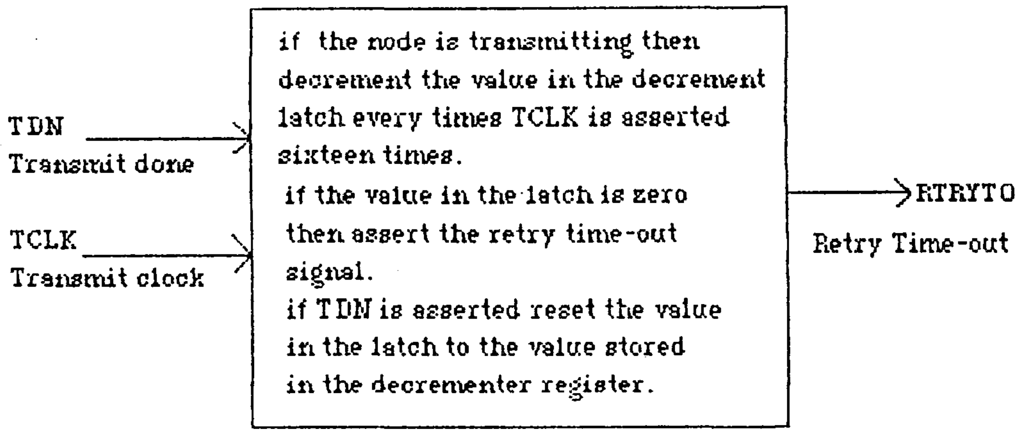

TDN
Transmit done
TCLK
Transmit clock
if the node is transmitting then decrement the value in the decrement latch every times TCLK is asserted sixteen times.
if the value in the latch is zero then assert the retry time-out signal.
if TDN is asserted reset the value in the latch to the value stored in the decrementer register.
RTRYTO
Retry Time-out

Figure 12. Retry counting functional block.

The **DMA functional block** (figure 13) consists of two parts, the DMA receive and DMA transmit logic. This section handles the handshake between the DMA controller and the ring node. One part handles the receive data transfers and the other handles transmission data transfers. The receive logic is much the same as the transmission logic except the transmission logic wait for DTACK to be asserted by the memory.

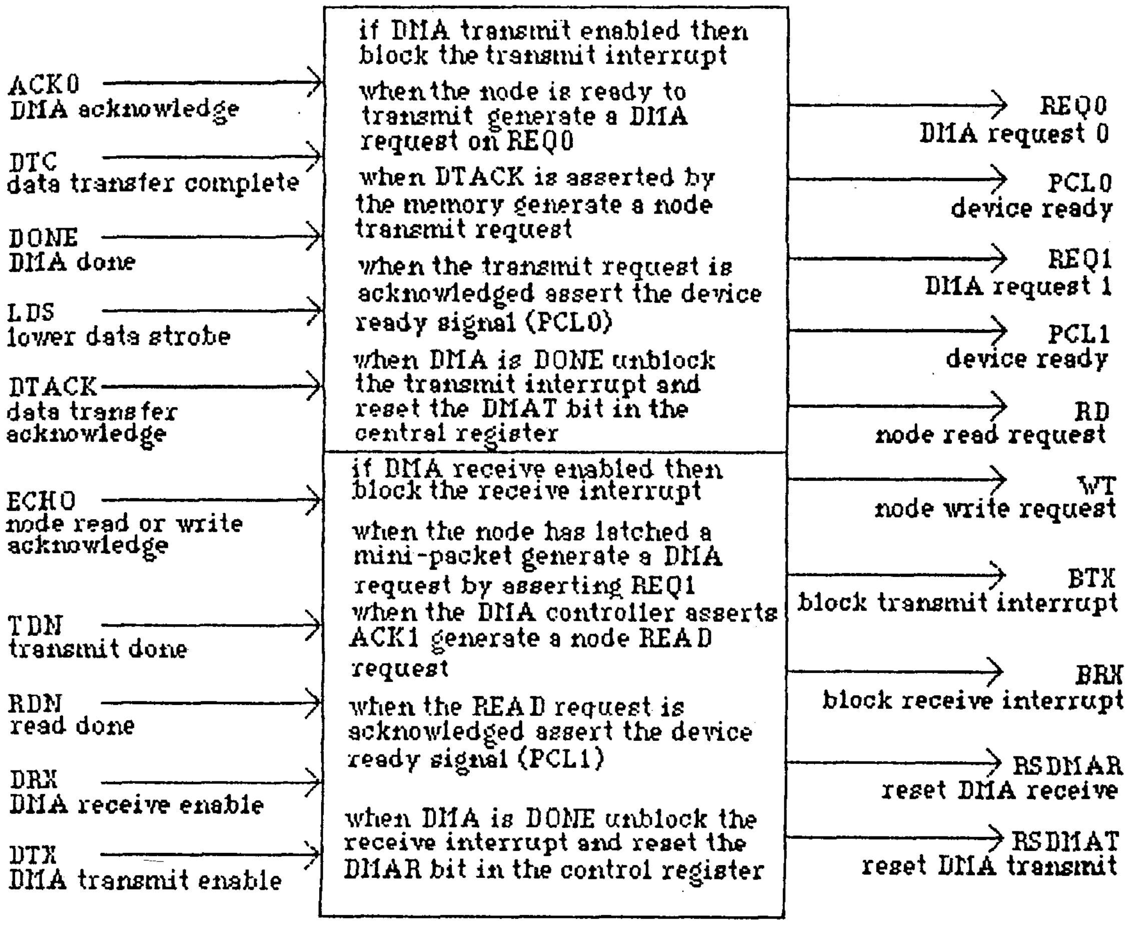

ACK0
DMA acknowledge
DTC
data transfer complete
DONE
DMA done
LDS
lower data strobe
DTACK
data transfer acknowledge
ECHO
node read or write acknowledge
TDN
transmit done
RDN
read done
DRX
DMA receive enable
DTX
DMA transmit enable
if DMA transmit enabled then block the transmit interrupt
when the node is ready to transmit generate a DMA request on REQ0
when DTACK is asserted by the memory generate a node transmit request
when the transmit request is acknowledged assert the device ready signal (PCL0)
when DMA is DONE unblock the transmit interrupt and reset the DMAT bit in the central register
if DMA receive enabled then block the receive interrupt
when the node has latched a mini-packet generate a DMA request by asserting REQ1
when the DMA controller asserts ACK1 generate a node READ request
when the READ request is acknowledged assert the device ready signal (PCL1)
when DMA is DONE unblock the receive interrupt and reset the DMAR bit in the control register
REQ0
DMA request 0
PCL0
device ready
REQ1
DMA request 1
PCL1
device ready
RD
node read request
WT
node write request
BTX
block transmit interrupt
BRX
block receive interrupt
RSDMAR
reset DMA receive
RSDMAT
reset DMA transmit

Figure 13. DMA functional block consists of two sections the DMA receive and DMA transmit functional block.

The **status handshake and interface control functional block** (figure 14) is reasonably simple and is responsible for asserting the Data Transfer Acknowledge Signal.

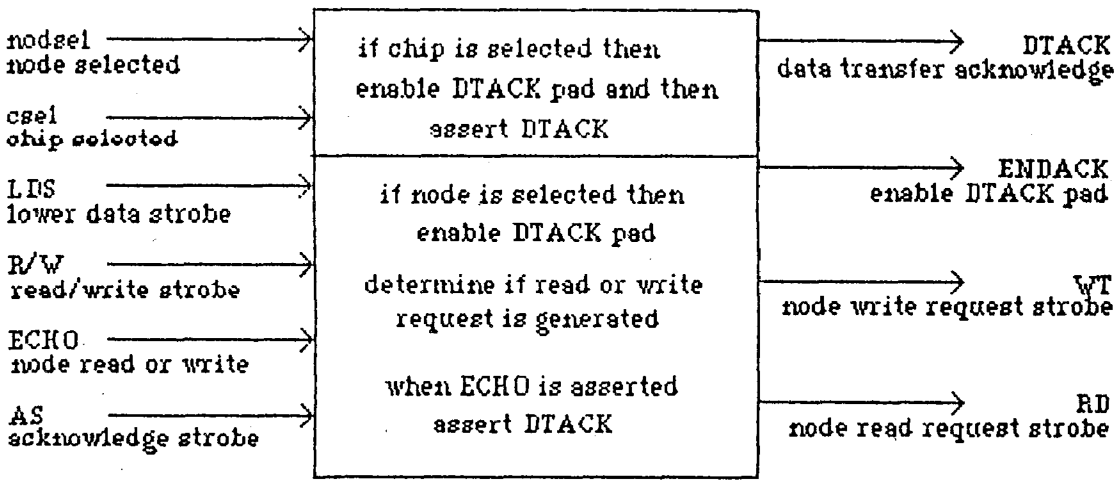

nodsel
node selected
csel
chip selected
LDS
lower data strobe
R/W
read/write strobe
ECHO
node read or write
AS
acknowledge strobe
if chip is selected then enable DTACK pad and then assert DTACK
if node is selected then enable DTACK pad
determine if read or write request is generated
when ECHO is asserted assert DTACK
DTACK
data transfer acknowledge
ENDACK
enable DTACK pad
WT
node write request strobe
RD
node read request strobe

Figure 14. Status handshake and interface control functional block.

a)

PROCESSOR - RING NODE RECEIVE/TRANSMIT HANDSHAKING SEQUENCE

PROCESSOR	INTERFACE	NODE
1. Processor Asserts AS		
2. Processor Aserts LDS		
	3. Interface initates Node transmission.	
		4. Node asserts ECHO
	5. Interface asserts DTACK	

b) PROCESSOR - INTERFACE CHIP READ/WRITE HANDSHAKE SEQUENCE

PROCESSOR	INTERFACE
1. Processor Asserts AS	
2. Processor Asserts LDS	
	3. Interface Asserts DTACK

c) DMA-NODE DATA TRANFER HANDSHAKING SEQUENCE

Receive

DMA CONTROLLER	INTERFACE CHIP	NODE
		1. Tranmission Complete RDN asserted
	2. Initiate a DMA request assert REQ1	
3. Acknowledge Request ACK1 asserted		
	4. Generate a receive request assert RD	
		5. Assert ECHO
	6. Assert DMA device ready PCL1	
7. DMA complete assert DTC.		

DTACK

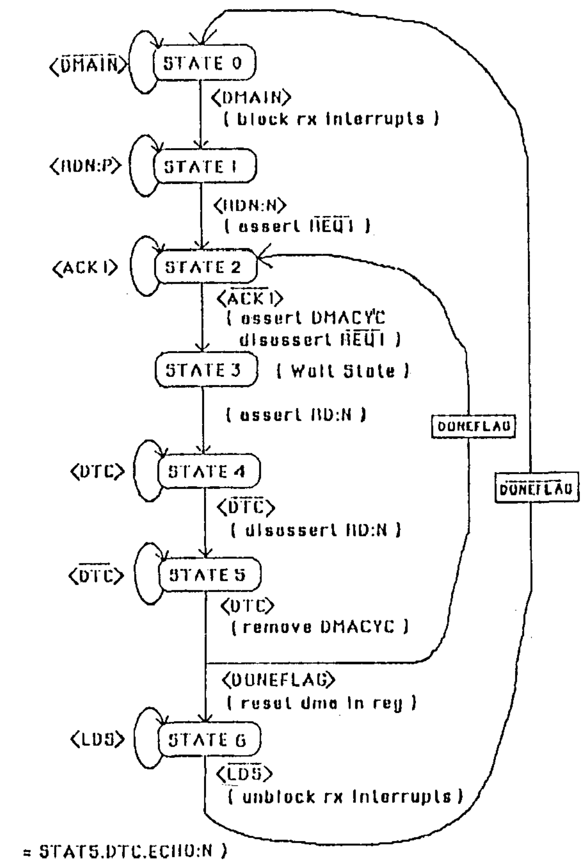
DMA-NODE DATA TRANFER HANDSHAKING SEQUENCE

Transmit

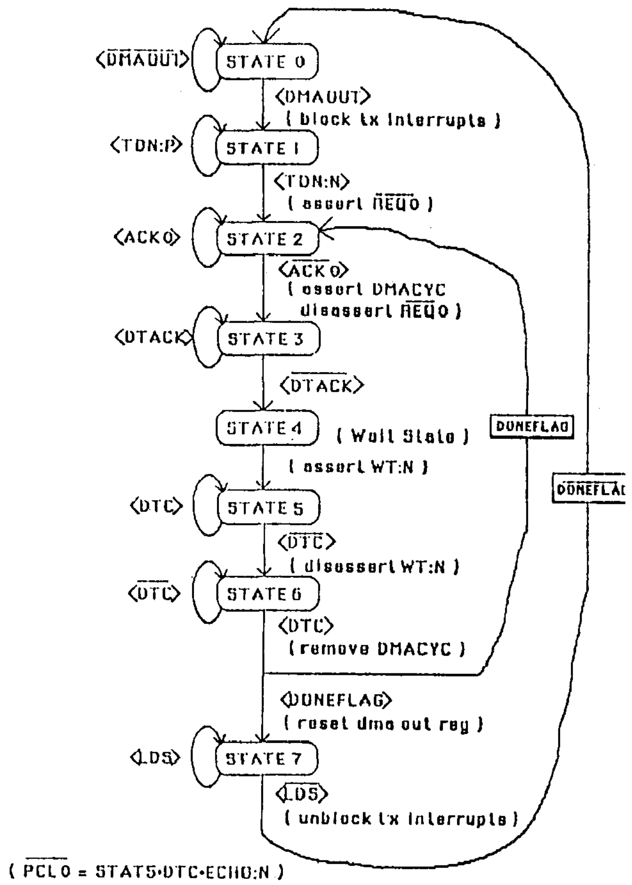
DMA CONTROLLER	INTERFACE CHIP	NODE
		1. Tranmission Complete TDN asserted
	2. Initiate a DMA request assert REQ0	
3. Acknowledge Request ACK0 asserted		
	4. Generate a transmit request assert WT	
		5. Assert ECHO
	6. Assert DMA device ready PCL0	
7. DMA complete assert DTC.		

Figure 15. Handshaking for a) Status handshake transfers, b) Read/ Write t Interface chip, c) DMA transfers

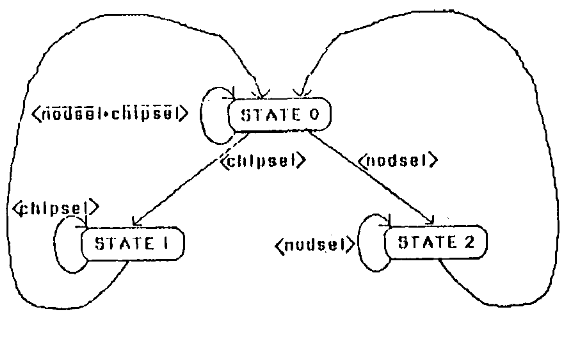
Figure 16. a) DMA receive state machine, b) DMA transmit state machine, c) Status handshake and interface control state machine.



= STAT5.DTC.ECHO:N)
FLAG = DONE + DONEFLAG.LDS)



(PCLO = STAT5•DTC•ECHO:N)
(DONEFLAG = DONE + DONEFLAG•LDS)



(WT:N = LDS•R/W•STATE2)
(RD:N = LDS R/W STATE2)
(DTACK = ECHO:N•STATE2•LDS•STATE1)
(ENDTACK = STATE2 + STATE1)

The **multiplexor** is a 4 by 2 to 1 multiplexor.It has one input control signal that determines if the DMA address is jammed onto the address bus. This signal is asserted by the DMA logic when the node is addressed for a data transmission.

The last functional block consists of the other registers. The functions of these were discussed earlier.

External to the interface chip are the **data bus** interconnections. The ring node data bus is external to the interface chip. This bus has a bi-directional buffer which controls the direction of the data transfer. This was made external to reduce chip area. Data on the bus is of no interest so with respect to the interface it is better if it is external.

3.1. External Timing

Another important stage in the functional specification is the checking of the event sequence during a particular data transfer. These sequences were checked by drawing the timing diagrams for the interface chip with the constraints of the external hardware placed on it. The bus handshaking for the data transfer modes are shown in Figure 15 with the chip and node access and the interrupt acknowledge handshaking. Timing diagrams for these sequences can be found in the interface chip users manual.

3.2. State Diagrams

The DMA, status handshake and interface control functional blocks are low level handshake sequencing logics (or sequential machines). These are known as finite-state machines and consist of a finite number of states with 'rules' for transitions between the states.

The DMA functional block is really two separate state machines. These state machines keep track of the current state in the DMA and data transfer handshaking. These two machines are quite similar except that the DMA transmit state machine waits on DTACK to be asserted by the memory before allowing the cycle to complete.

The status handshake and interface control state machine is reasonably simple. Its task is to assert DTACK when either one of the interface chip's internal registers is accessed or during a status handshake data transfer. These state machines are shown in figure 16a, 16b and 16c.

4. Chip Design

The design of the chip really consists of mapping the functional description onto silicon. However, before proceeding it is necessary to formulate a layout strategy. The strategy used in this design is the same as that used in PLAs. All control runs at right angles to data (Fig. 17 illustrates this strategy). This allows data to travel in one direction in one layer and data to travel in the other direction in a different layer. Thus control can either affect data or 'pass over it'.

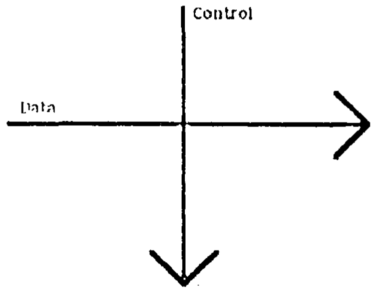


Figure 17. The layout strategy. Data runs at right angles to control.

Keeping this strategy in mind, the leaf cells developed for each functional block should reflect this. In the following sections each functional block will be discussed in terms of its 'silicon representation'.

Figure 18. The logical representation of the state machines. a) DMA receive, b) DMA transmit, c) status handshake and interface control.

rstate0 = rstate0·$\overline{\text{DMAIN}}$ + rstate6·$\overline{\text{LDS}}$;

rstate1 = rstate0·DMAIN + rstate1·RDN:P; + rstate5·DTC·$\overline{\text{DONEFLAG}}$;

rstate2 = rstate1·RDN:N + rstate2·ACK1;

rstate3 = rstate2·$\overline{\text{ACK1}}$;

rstate4 = rstate3 + rstate4·DTC;

rstate5 = rstate4·$\overline{\text{DTC}}$ + rstate5·$\overline{\text{DTC}}$;

rstate6 = rstate5·DTC·DONEFLAG + rstate6·LDS;

BRX = rstate0·DMAIN + rstate1·RDN:P + rstate1·RDN:N + rstate2·ACK1 + rstate2·$\overline{\text{ACK1}}$ + rstate3 + rstate4·DTC + rstate4·$\overline{\text{DTC}}$ + rstate5·$\overline{\text{DTC}}$ + rstate5·DTC + rstate5·DTC·$\overline{\text{DONEFLAG}}$ + rstate6·LDS;

REQ1 = rstate1·RDN:N + rstate2·ACK1;

DMACYC = rstate2·$\overline{\text{ACK1}}$ + rstate3 + rstate4·DTC + rstate4·$\overline{\text{DTC}}$ + rstate5·$\overline{\text{DTC}}$;

RD:N = rstate3 + rstate4·DTC;

PCL1 = rstate4·DTC·ECHO:N + rstate5·$\overline{\text{DTC}}$;

RSDMAR = rstate5·DTC·DONEFLAG;

DONEFLAG = $\overline{\text{DONE}}$ + DONEFLAG·LDS;

b) state0 = state0·$\overline{\text{DMAOUT}}$ + state7·$\overline{\text{LDS}}$ +

state1 = state0·DMAOUT + state1·TDN:P + state6·DTC·$\overline{\text{DONEFLAG}}$;

state2 = state1·TDN:N + state2·ACK0;

state3 = state2·$\overline{\text{ACK0}}$ + state3·DTACK;

state4 = state3·$\overline{\text{DTACK}}$;

state5 = state4 + state5·DTC;

state6 = state5·$\overline{\text{DTC}}$ + state6·$\overline{\text{DTC}}$;

state7 = state6·DTC·DONEFLAG + state7 LDS;

BTX = state0·DMAOUT + state1·TDN:P + state1·TDN:N + state2·ACK0 + state2·$\overline{\text{ACK0}}$ + state3·DTACK + state3·$\overline{\text{DTACK}}$ + state4 + state5·DTC + state5·$\overline{\text{DTC}}$ + state6·$\overline{\text{DTC}}$ + state6·DTC·DONEFLAG + state6·DTC + state7 + LDS;

$\overline{\text{REQ0}}$ = state1·TDN:N + state2·ACK0;

DMACYC = state2·$\overline{\text{ACK0}}$ + state3·DTACK + state3·$\overline{\text{DTACK}}$ + state4 + state5·DTC + state5·$\overline{\text{DTC}}$ + state6·$\overline{\text{DTC}}$;

WT:N = state4 + state5·DTC;

$\overline{\text{PCL0}}$ = state5·DTC·ECHO:N;

RSDMAO = state6·DONEFLAG·DTC;

DONEFLAG = $\overline{\text{DONE}}$ + DONEFLAG·LDS;

c) state0 = ($\overline{\text{nodsel}}$·$\overline{\text{chipsel}}$ + state1·$\overline{\text{nodsel}}$ + state2·$\overline{\text{chipsel}}$)·reset

state1 = (state0·nodsel + state1·nodsel)·reset

state2 = (state0·chipsel + state2·chipsel)·reset

WT:N = $\overline{\text{LDS}}$·$\overline{\text{R}/\overline{\text{W}}}$·STATE2

RD:N = $\overline{\text{LDS}}$ R/$\overline{\text{W}}$ STATE2

DTACK = ECHO:N·STATE2·$\overline{\text{LDS}}$·STATE1

ENDTACK = STATE2 + STATE1

The **Interrupt functional block** and **address decoding functional blocks** are implemented using Programmable Logic Arrays (PLA). A PLA was chosen since the two sections are simply combinatoric logic and therefore are suited to a PLA. A PLA is also easily modified and regular in shape and complies to the overall layout strategy. The logic for a PLA (see Appendix 2) is passed to a logic simplification programme `simpl'. Simpl produces as output, input to `plalog' which generates a truth table to be used as input to `plagen' (`simpl', `simpl2' and `plalog' were written by Dr R.I Hartley).

DMA receive and transmit and the **status handshake and interface control** functional blocks consist of three state machines. The best way of implementing state machines in nMOS technology is with a PLA. The three state machines are shown in Figure 16. If these three state machines are examined carefully it can be seen that there are quite a number of common inputs and outputs. Therefore, if these state machines are merged into one PLA only one signal line is needed for a common signal, thus eliminating a large amount of routing.

The state machines must be converted from their graphical representation into a logical description suitable for input into the logic simplification programmes. To do this each state machine is considered separately. The first step is to give each state a binary representation which is unique in the minimum number of bits possible. The logic for each state and the logic for state transitions is then extracted from the diagram. Each logical description is shown in figure 18 (see also Appendix 2). These logical descriptions are then merged into one logical description keeping the state bits of each separate.

So far all the functional blocks have been implemented using PLAs. However a PLA cannot be used as a register. The register cells need to readable and writable and sit neatly on the data bus. Further, these register cells must comply with the overall layout strategy. The best approach is to firstly consider the register in a functional form and then proceed to the layout details later. Figure 19 shows a functional representation which allows the register to stack such that control runs a right angles to data. Thus each register consists of the correct number of register cells stacked horizontally.

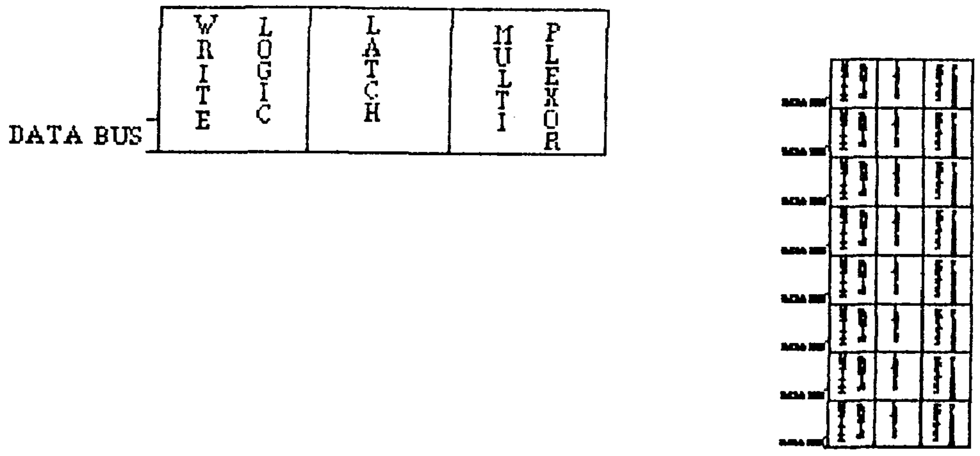


Figure 19. Functional representation of a register.

As shown in figure 19 the register can be divided into three parts, the load control multiplexor, the basic storage latch and the write data bus logic. The following sections will discuss each of the registers.

The **node status register** simply stores the current values of the ring node signal discussed earlier and the error flags. Therefore no multiplexor is required and the basic register cell with the write bus logic is needed (note the write bus is a pull up bus). The circuit diagram is shown in figure 20.

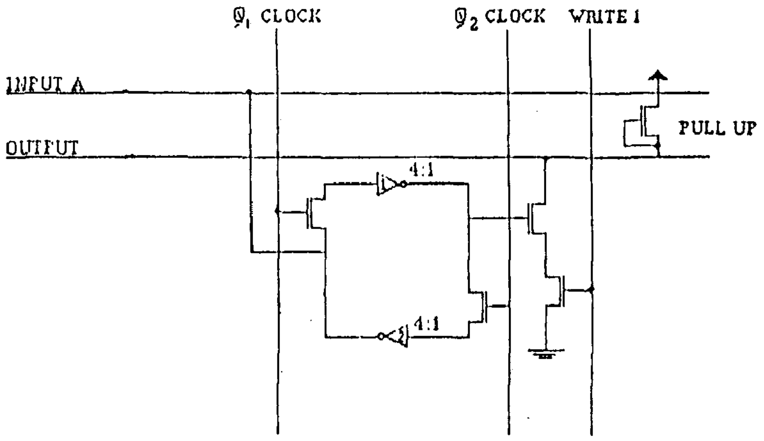
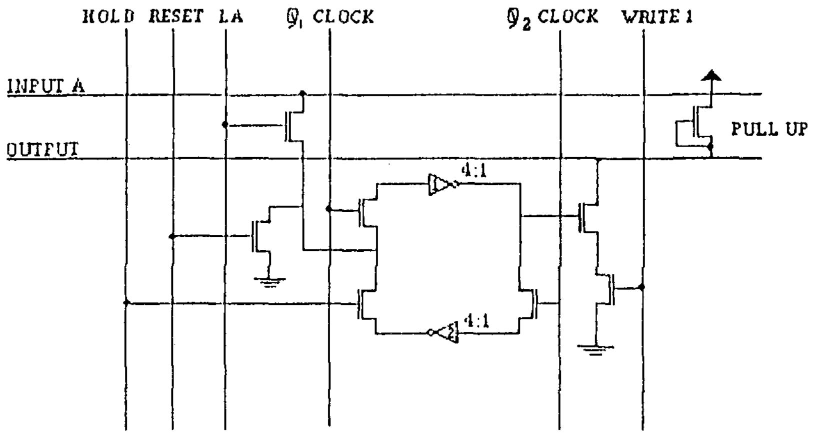
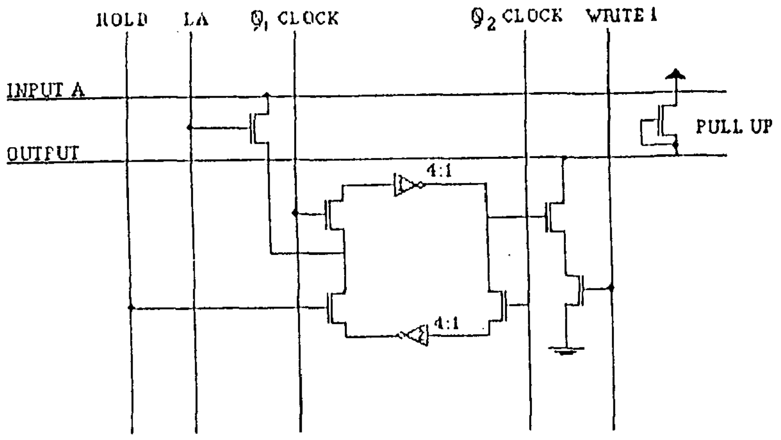


Figure 20. Circuit diagram for the node status register.

The **control register** has to latch the value that is written to it from the input data bus. Therefore the register must recycle the value stored, and must be able to load a new value. The DMA control bits in this register must be capable of being reset by the DMA logic. To satisfy this the registers must have two different multiplexors with reset and six without. Figure 21 shows the circuit diagrams of the two register cells that make up the control register.



(a)



(b)

Figure 21. The two register cells that make up the control register. a) register with reset and b) register without reset.

The **interrupt vector register** is a read/write register and no bit is reset by internal logic as in the control register. Therefore, only a simple multiplexor is required that will allow the register to be loaded and recycle the stored value. However since the upper five bits of the register stores the upper five bits of the interrupt vector number, it must be possible to enable these five bits without enabling the lower three during an interrupt acknowledge cycle. To meet this criteria the register must have two write control lines to the upper five bits (see Fig 22 for circuit diagrams and Fig 21b).

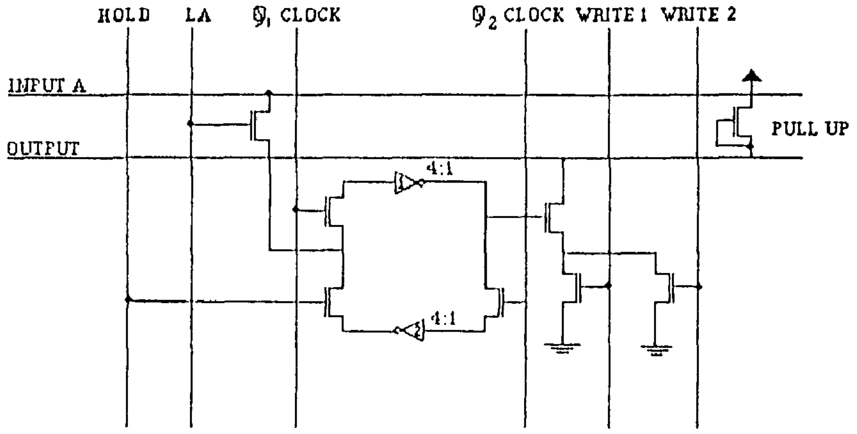


Figure 22.One of the two register cells that make up the interrupt register, note the two output bus pull downs.

The counter register is a normal read/write register. The value in this register must be used to reset the decrementer latch. Therefore the multiplexor must facilitate this (see Fig 21b for circuit diagram).

The **decrementer** consists of three parts, the latch, the comparator and the decrementer. The latch must be situated next to the counter register's multiplexor so that it can be reset. The decrementer must be situated next to the latch so the values in the latch can be decremented. For these reasons the best approach to the design of the decrementer is to design it so that it will fit across the data bus with the other registers. The comparator should also be designed in this way since it must detect when all the values in each of the eight latches are zero.

The latch must be decremented every time TCLK is asserted for each transmission and be reset when the transmission is complete. The logic to count TCLK, to reset the decrement latch and set the retry-time out signal was implemented with a PLA. The logical representation is shown in Appendix 2.

The functional block specifications must now be placed on silicon. The functional blocks cannot be randomly placed since they need to be routed together and to the input and output pads. To have a reasonably good guide of the placement of each block a floor plan must be developed.

The **floor plan** must show the relative sizes of each of the functional blocks with respect to the others. The positions of the input and output pads must also be decided. Figure 23 shows the floor plan for the interface chip. A major part of the floor plan is taken up by routing channels which shows the proportion of routing that has to be done. All the functional blocks were placed so that the routing could be done automatically.

Since the distance from the control PLA (i.e. interrupt and address PLAs) to the DMA and status handshake PLA and the distance from the end of the data path to the pads is considerable, buffers are encorporated to drive the signals through the full distance of the wire. This makes the considerably faster and more efficient. At this stage the routing of power and the clock should be considered. As can be seen in Figure 23, Vdd and Vgnd are diametrically opposite each other. This allows Vdd and Vgnd to run to all parts of the circuit without crossing each other. Similarly with the clocks,

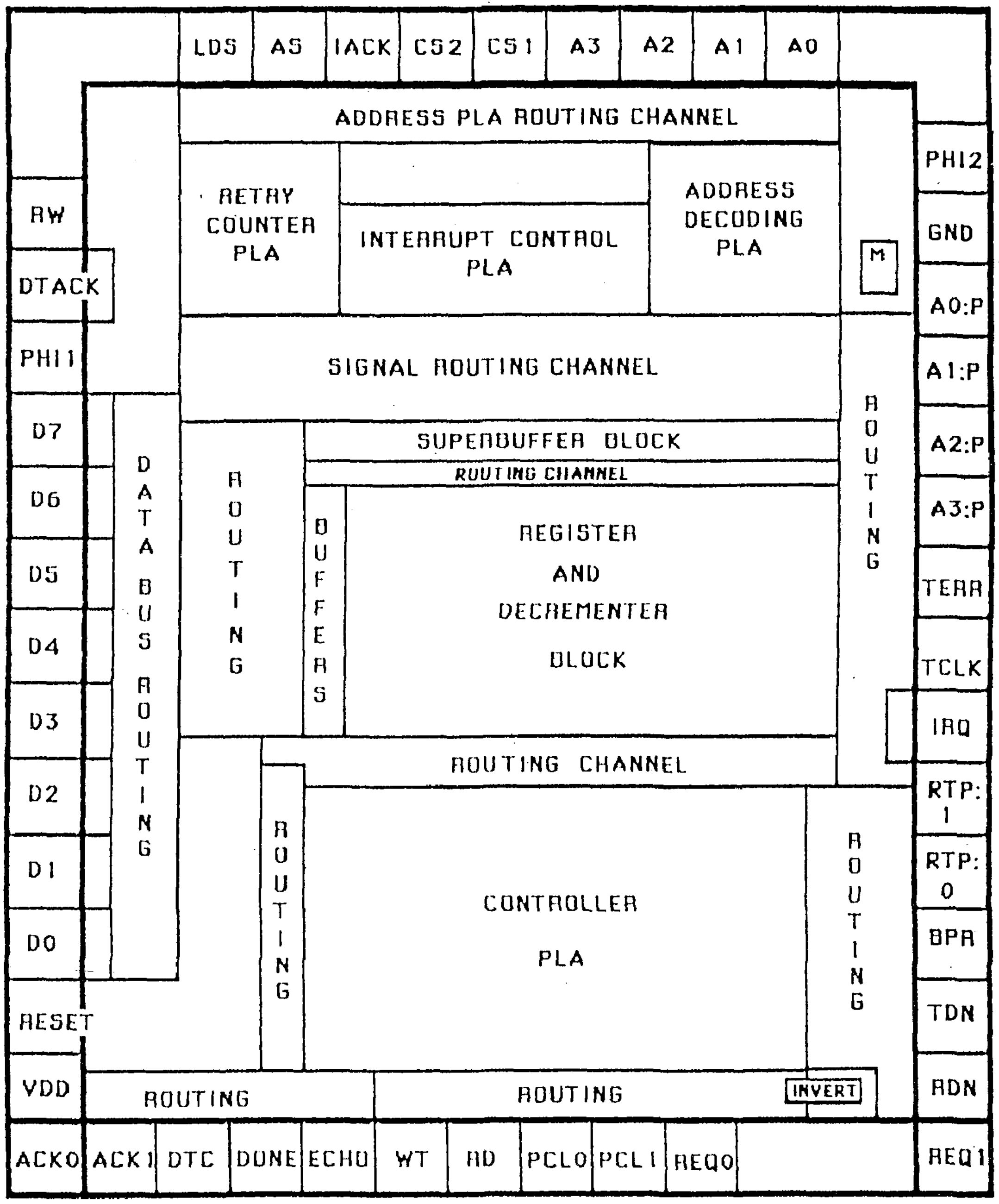


Figure 23. Floor plan for the interface chip.

the pads for Phi1 and Phi2 were placed opposite each other.

Before the floor plan is complete the positions of the registers and decrementer in the data path must be decided. Various signals must be routed from the register to other functional blocks, therefore adequate space must be left between the various registers so that these signal can be routed. Not only do signals have to be routed from the register cell but they also have to be routed from the control section across the data path to the DMA and status handshake section. Figure 24 shows the block diagram for the data path.

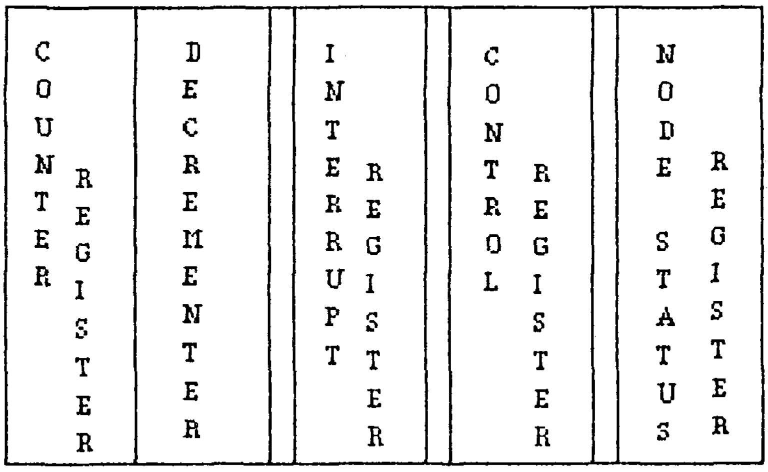


Figure 24. Block diagram of the data path.

The floor plan is a major part of the whole design process. A badly designed floor plan will make the routing difficult and thus increase the amount of time spent on the layout.

The last important stage that must be considered before the final details of the layout can be completed is the **internal timing sequences**. These sequences are important since it is essential that data arrives at its destination at the right time and on the right phase of the clock. Figure 25 shows the internal timing diagrams for the interface chip. The method used in this diagram is the same as that used by Hartley [4]. The timing sequences are expressed using modified petri-nets. The horizontal lines are the events, the green line represents phi1 and the red line phi2. The length of the blue line indicates the relative time taken for the signal to reach it destination.

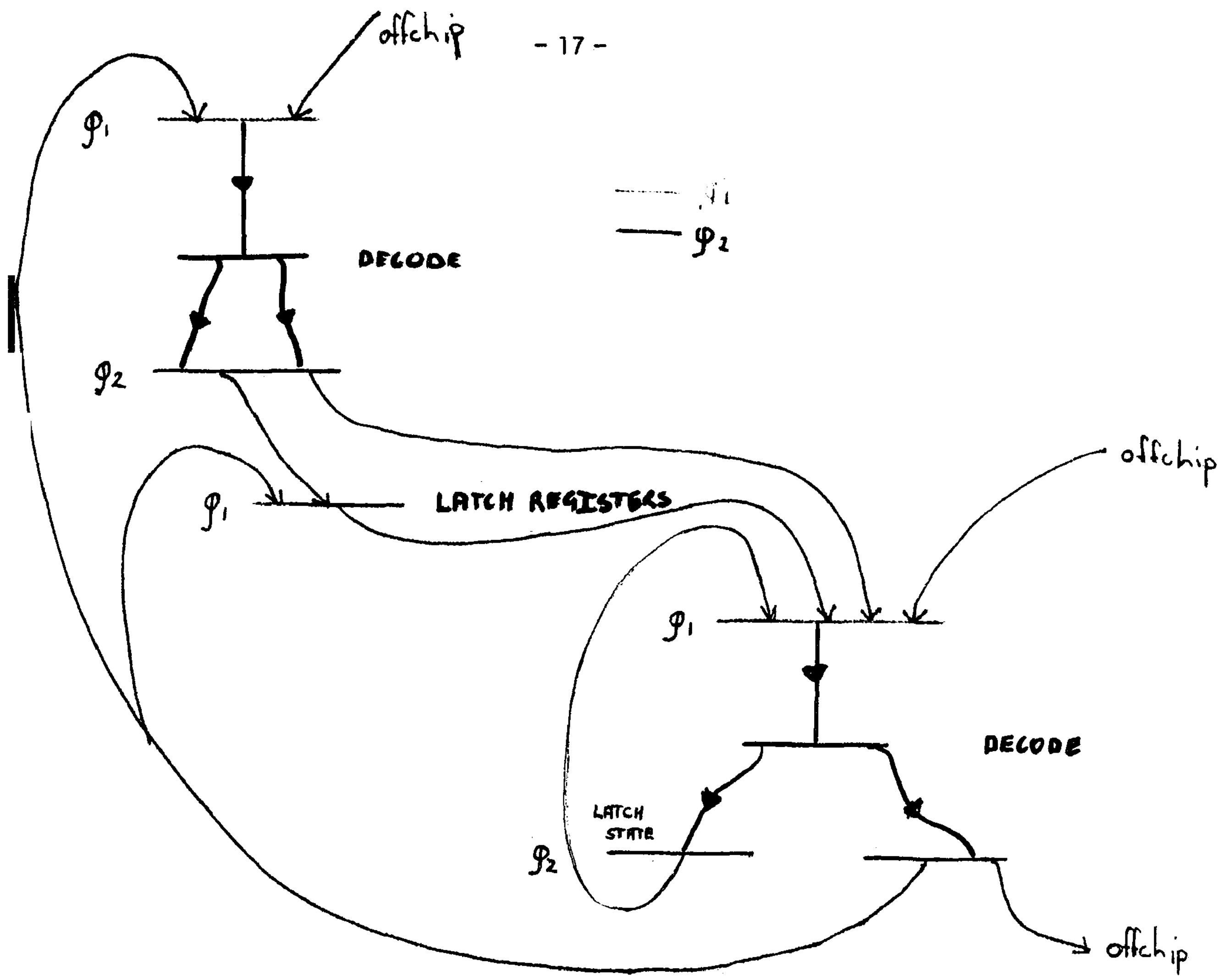


Figure 25. Internal timing diagram for the interface chip.

Now that the positions of the functional blocks have been finalised so too can the layouts for the various functional blocks. The layout for each of the above functional blocks is shown in figure 26. These layout are arranged to reflect the floor plan and shown the relative sizes of each functional block. The **routing1 between the functional block was done automatically using a routing programme written by Dr R.I Hartley.**

Before the design can be fabricated it must be verified using software tools. The PLA were easy to verify using a programme 'simul' which allows values to be assigned to the inputs of the PLA and shows the outputs. The address decoding, interrupt, DMA-status handshake and interface control PLAs were verified using this programme.

Verifying the PLAs by themselves does not give complete verification that they will work since the have to work when connected to the pads, clocks and power in the actual circuit. Not only do the PLAs have to verified under these conditions but this is also the case with the register block and decrementer must be. When the layout was completely finalised the programme 'unswim' was used to simulate the whole interface chip from the pads.

5. Design Procedure

The design procedure used consists of twelve important steps. These are:

- specification
- functional description
- external timing diagrams
- state diagrams
- layout strategy
- silicon representation of functional blocks

- floor plan
- final layout of functional blocks
- layout composition
- design rule checking
- functional block simulation
- complete layout simulation

The **specification** is a description of what the capabilities of the interface chip should be. The specification covers the data transfer modes and the error conditions that should be considered along with the hardware interfaces. The details of the internal register contents and addressing considerations are also decided upon. External bus connections also need to be considered since it should be decided if the various buses really need to run through the interface chip.

The **functional description** addresses the individual functions of the interface chip. For each individual function a functional representation is developed and is expressed in a diagram. The interactions between these functional blocks are also considered. It is important that the functional separation is clean and logic in the individual functional block is in its simplest form. This means any control sequences must be identified and represented as **state diagrams**.

Before the functional design can be mapped onto silicon it is necessary to have some sort of **layout strategy** that guides the way in which each functional description is mapped onto silicon. The layout strategy should also dictate how the silicon functional blocks are put together. It is important that the strategy is well understood and that each functional block complies to the strategy.

Once the layout strategy has be decided upon the individual functional blocks can be mapped into their **silicon representation**. The approach taken in this design was to identify combinatoric logic and state machines and implement them with PLAs which proved to be a good idea. The only other considerations were the registers. The register cells must comply with the overall strategy and be easily butted together to give a neat register. The individual register cells were first represented in a block form showing the positions of the input and output signals. After the block representations are finalised the internal circuitry is specified.

Now that a relative idea of the shape and size of the individual functional blocks silicon representation has been gained a **floor plan** needs to be drawn up. The floor plan should consist only of blocks representing each of the functional blocks and their relative sizes. The functional blocks should be arranged so that they comply to the layout strategy. Not only should the position of the internal functional block be decided upon, but also the routing of the signal between them and most importantly the routing of the two phase clock, Vdd and Vgnd. The position of the functional blocks decides the position of the input and output pads. Obviously, if a signal to a functional block comes from or goes to a pad then it is desirable for that pad to be as close as possible to the functional block to make routing easier and reduce delay times.

Now that the position of the functional blocks relative to each other is defined, which therefore defines the position of the input and output signals for each functional block, the **final layout** for each of the functional blocks can be decided upon. Each of the state machines and combinatoric logic blocks are easily generated with their input and output signals in the right places since they are PLAs and are automatically generated from a logic specification. However, the register cells, decrementer and multiplexor are different. These need to be considered carefully, the layouts for each of these should take up as little area as possible. The layouts are first represented as stick diagrams and are the drawn as complete layouts. These layouts are then mapped into the layout language BELLE as separate definitions.

Once all the layouts for the functional blocks have been completed all that remains is to connect them together. The **composition of the layout** is relatively straight forward. The approach taken is firstly to lay the bottom row of pads, then the left side pads followed by the state machine PLA and the data path. The data path and state machine PLA were routed together and then to their pads. When this was completed the rest of the pads, counter, address and interrupt PLAs were laid and then finally the multiplexor.

After the layout had been completed it was then tested for design rule violations with the design rule checker. Any design rule violations are fixed with the graphics editor.

The next step was then to simulate the pieces. This step should have been done before the layout composition. The reason it was not done at this stage was because the software was not available for use. If the simulation was unsuccessful then the functional block was corrected with the graphics editor.

When all the pieces simulated successfully the whole layout was then simulated from the pads to ensure that all of the routing was correct. If any mistakes were encounted then the mistakes were fixed using the graphics editor.

6. Software Tools

There are a number of software tools that were used in the composition of the layout. The tools consisted of CSIRO's VLSI software, developed during the CSIRO VLSI programme, PLA logic simplification and simulation programmes and routing programmes developed by Dr. R.I Hartley for this project. Graphics editor and verification tools at NSW JMRC were also used in the later part of the design.

The CSIRO VLSI design suit consists of a layout language *BELLE*, a CIF file summary programme for use with BELLE called *GETSYMBOL*, a PLA generation programme *PLAGEN* and a CIF plotting programme *VIEWCIF*. BELLE proved to be very useful. It is a reasonably powerful layout language for describing leaf cells. It is a language which is embedded in *pascal* and therefore has all the programming constructs that that pascal offers (see Appendix 3). All the leaf cells for the layout are described in BELLE. The difficulty arises with BELLE when the leaf cells have to be merged into the complete layout. This was solved by the development of a router which was intergrated into BELLE by Dr. R.I.Hartley. GETSYMBOL is more or less part of BELLE.

The PLA generation programme, *PLAGEN*, proved to be useful since the PLAs are automatically generated free of design rule errors. This saves vast amounts of time and allows the PLA to be easily modified without much worry. There are however, a number of undesirable features with PLAGEN. The low level PLA specification, i.e a truth table, can lead to incorrect specifications for large PLA's, due to the number of 1's and 0's that are needed to specify the PLA (see PLAGEN users manual). This problem was solved by Hartley by developing the PLA logic specification programmes *simpl* and *PLAlog*. These take a high level description of a PLA and produce a truth table suitable for input to PLAGEN (see PLA tutorial). PLAGEN also produces a standard PLA and does not allow for the user to easily specify different input and output pull up ratios to cater for large delay times across the *and* and *or* planes in large PLAs. Another feature which is undesirable concerns the fact that when PLAs become large PLAGEN does not change the metal thickness for Vdd and GND power lines. These features could be easily encorporated in PLAGEN without too much worry and hence produce better PLAs.

VIEWCIF is a necessary piece of software, without which the design could not be checked visually.

Simul the PLA simulation programme was very useful. It enabled all the PLAs to be verified before they were finally encorporated into the layout.

The software used at JMRC consisted of a design rule checker, a graphics editor and a simulator. The design rule checker, *lrc*, identified design rule violations that went unnoticed during visual checking. Lrc proved to be be very useful since it detected a number of errors. The graphics editor *KIC* was also very useful since it allowed all the design rule violations to be fixed quickly. The simulator *unswim* used to simulate was fairly effective. It was not excessively slow and gave good results. The only undesirable feature is its input format. The input is specified with 1's and 0's which are used to indicate a signals' state. This becomes confusing and messy with large simulations. The interface could probably be improved by either developing a simulation language or some sort of graphics interface that allows timing diagram like specifications.

Generally, I found the software to be reasonably adequate but BELLE became more and more tedious as the layout grew. The routing, PLA logic and PLA simulation software was particularly useful. For a layout of this size, a good graphics editor and simulator are essential from the start.

7. Testing

After the chip has been fabricated it is necessary to test that it will function correctly. Firstly, the CRM68000C users manual in Appendix 3 should be read. The simple operations should be tested first. An important point to note is that the interface chip should always be reset by asserting the RESET signal before each test. This will ensure that all the state machines are in their initial states.

The first function to test is writing to an internal register of the interface chip. The control register would be the best, just be sure not to enable interrupts or DMA, just set the Node address . If this seems successful then read from the register and compare it with the value stored. If this works then do it with the interrupt register and the counter register. The Node Status register is read-only so just read it.

The next function to test is initiating a data transfer by the ring node, using status handshake driven data transfers. Test both transmit and receive operations (see Users Manual Sect. 2.1).

The next function to test is the interrupts. Firstly, determine the position of the interrupt vector addresses (see MC68000 Advanced Information). Next, store up the five most significant bits of the interrupt vector address in the five most significant bits of the interrupt vector register (see user manual Sect. 2.1). Next, enable the interrupt control bits in the control register. The interface chip will then interrupt when a packet is latched into the node or if the node is ready to initiate a transmission.

If the MC68440 Dual-Channel Direct Memory Access Controller is available then test the DMA transfer modes. To do this the correct address to access the node must be stored in the four most significant bits of the control register. Then enable the DMA data transfer bits in the control register. Next initialise the DMA controller so that it will transfer x bytes from a given memory location to the transmission channel. This will test DMA transmission. To test DMA reception initialise the DMA controller such the x bytes is transferred from the reception channel of the DMA controller.

If all these modes of operation function correctly then combine the different transfer modes. For example, transmit data using status handshake and receive data using DMA, receive data using interrupts and transmit data using DMA (see also Sect 2.1 in the Users Manual).

The final checking necessary consists of testing the two error conditions. These must be tested in two ways, firstly by simply reading the node status register (see Sect. 7.1 in the Users Manual) and secondly with the interrupts enabled (see Sect 7.3 in the Users Manual). To test the mini-packet corruption error hold the TDN pin high and pull the TERR pin low. Similary to test the retry time-out error, load the counter register with one and hold the TDN pin high and pull the TCLK signal low sixteen times.

As far as basic error checking goes this is about it. More complicated data transfer mode switching and special case testing should be done.

Conclusion

The design approach used in the design of the interface chip proved to be reasonably successful. Starting with the development of a functional description, and then verifying the functional description by drawing the timing diagrams is a good method of verifying that the design, when implemented, will work. Choosing a layout strategy and developing an internal timing model for the layout is essential. The composition of the layout is particularly dependent upon the software tools and hardware available. A good graphics editor, layout language, router, PLA generator and simulator are about the minimum software requirements needed to design a reasonable chip. A good graphics computer or CAD system would not go astray.

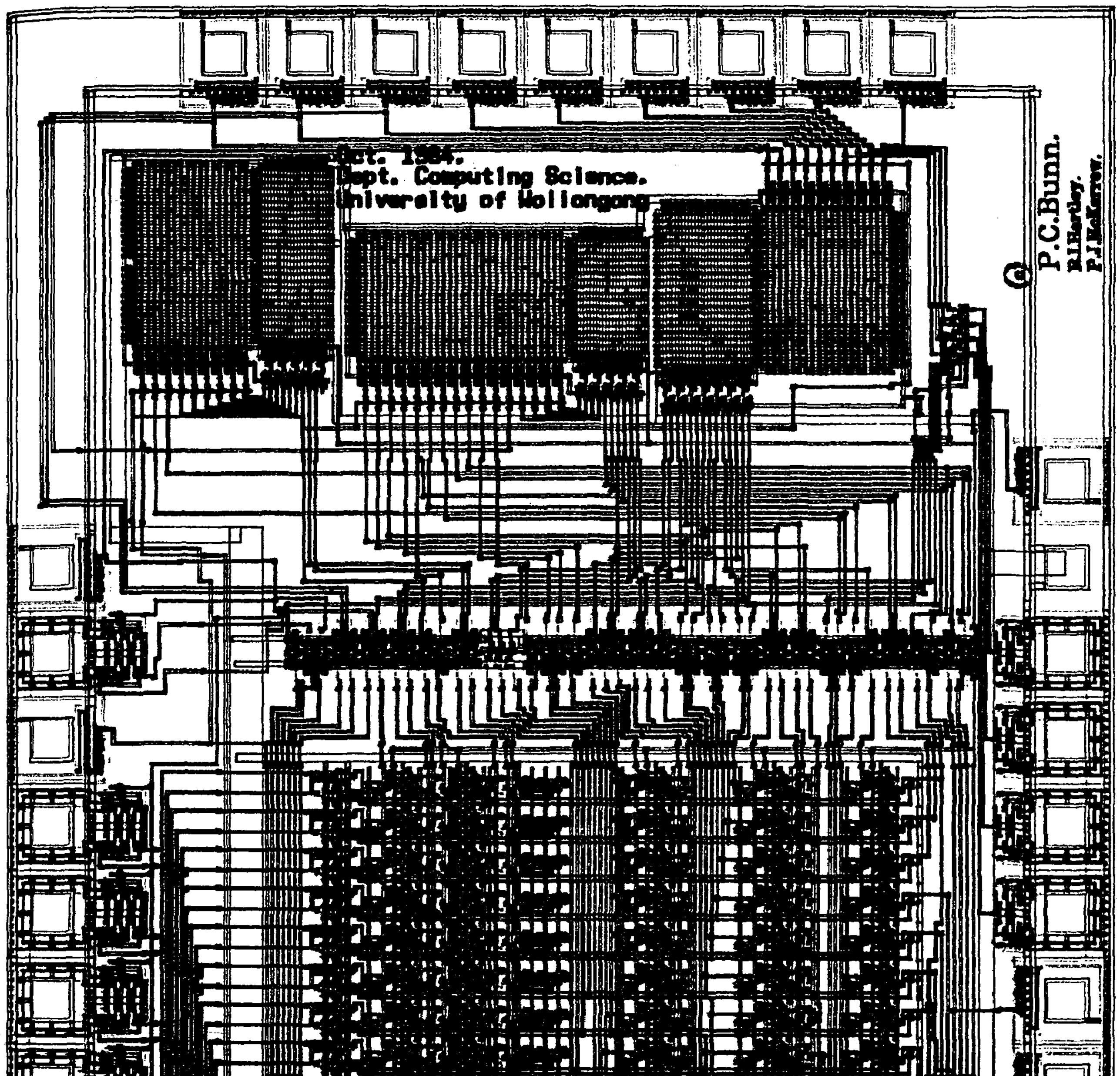


Figure 26. The interface chip layout.

REFERENCES

(1) Logica VTS Limited (1981) *Cambridge Ring Network Manual*

(2) Motorola (1983) *MC68000 16-Bit Microprocessor, Advanced Information*

(3) Motorola (1984) *MC68440 Dual-Channel Direct Memory Access Controller, Advanced Information*

(4) Hartley,R.I. (1983) *Z-80 Multiplier Chip*

(5) Mead, C. Conway, L. (1980) *Introduction to VLSI Systems* 1st ed. Addison-Wesley Inc. Phillippines.

Appendix 1

Layouts of standard cells.

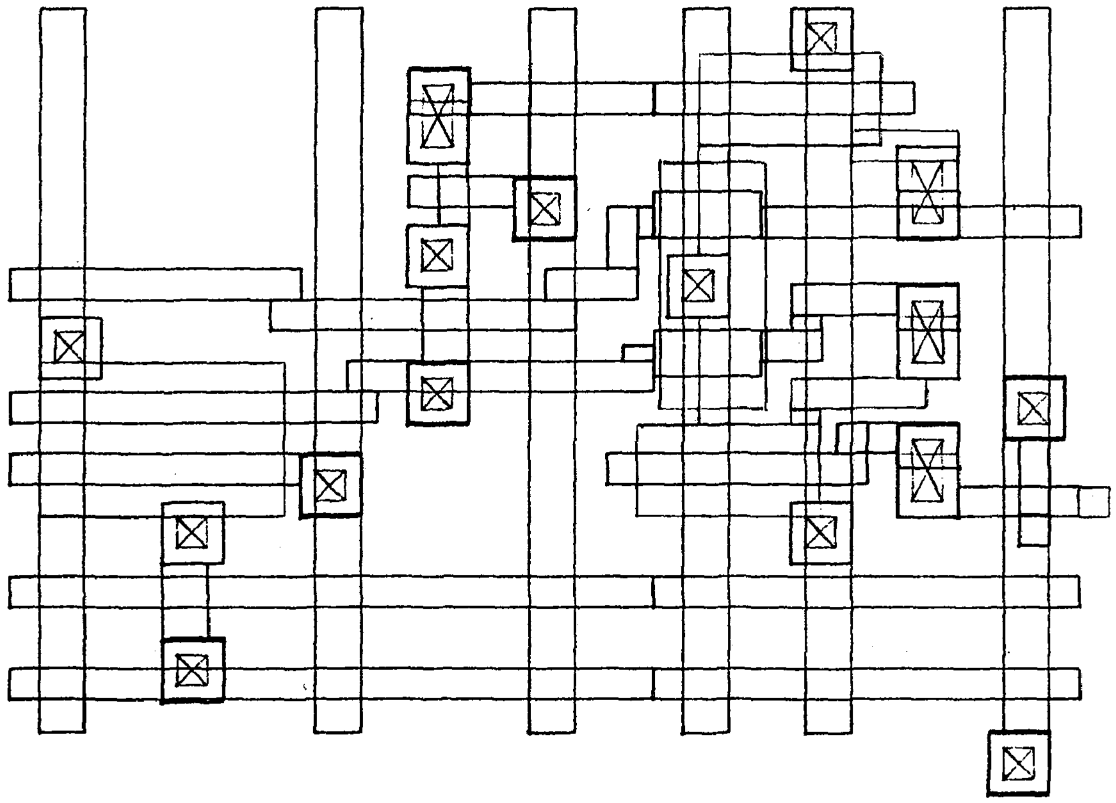
NAME

regcell – general purpose register cell

DESCRIPTION

Regcell is a general purpose static register cell which is Phi1–Phi2 clocked. It is designed so the input can be multiplexed. Thus the register can have many inputs form a number of sources one of which must be the register itself, since the value in the register must be recycled. The register cell has two buses, the input bus and the output bus. The output bus must be a pull up bus since the write logic is dependent upon this.

LAYOUT



CHARACTERISTICS

size: 72 by 47 Lambda.

inverter raitios:

inverter 1	4:1
inverter 2	4:1

bus pull down raito: 2:1

input capacitance:
7 square Cg.

resistence of bus pull down:
8 square Rg

TESTING

Fully simulated with *unswim 2.0* and design rule checked with *lrc*.

AUTHOR

Paul C Bunn
University of Wollongong

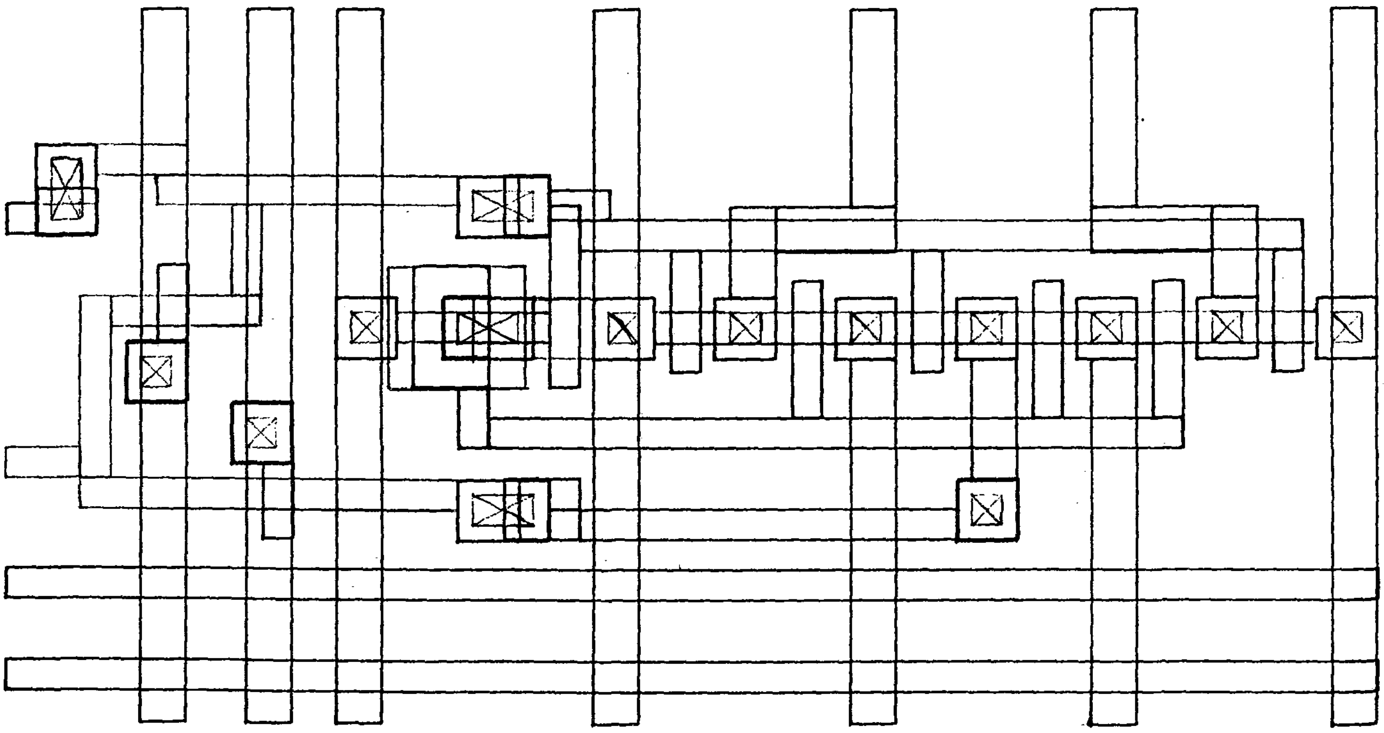
NAME

decrementer – ripple carry decrementer

DESCRIPTION

Decrementer is a ripple carry decrementer. Its is designed to be compatible with **regcell**. An appropriate multiplexor should be designed to interface the decrementer to the register cell or latch. It is important to note that buffers should be placed between each decrementer cell since the time taken to ripple accross the can data bus is quite substantial espically for more than about 4 bits.

LAYOUT



CHARACTERISTICS

size: 91 by 47 Lambda.

Inverter raitios:

Inverter 1	4:1

Input capacitance:

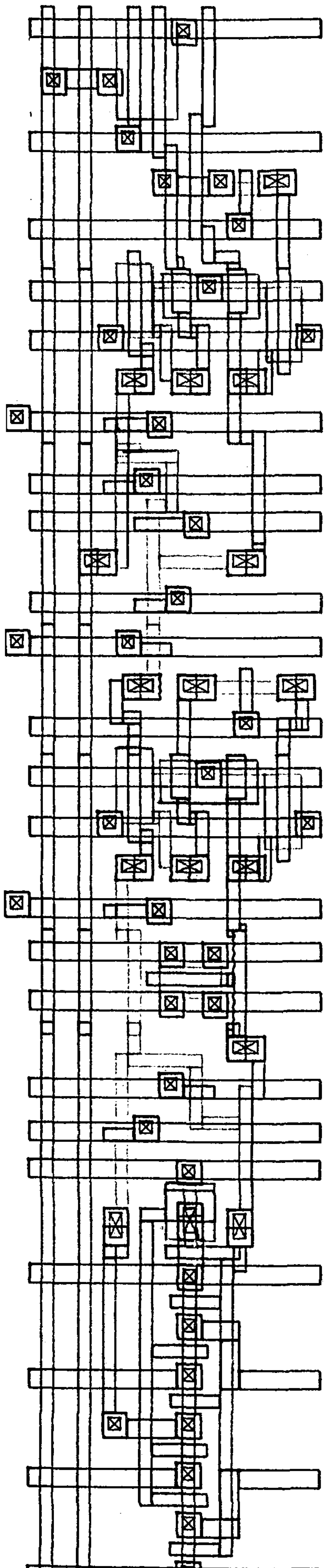
7 square Cg.

TESTING

Fully simulated with *unswim 2.0* and design rule checked with *lrc.*

AUTHOR

Paul C Bunn
University of Wollongong



Appendix 2

Logical descriptions of PLAs.

Interrupt PLA logic.

```
%input
rxen            /* receive interrupt enable */
rtryen          /* retry checking interrupt enable */
txen            /* transmit interrupt enable */
terren          /* transmit error interrupt enable */
rtryto          /* retry time out */
rdn             /* receive done */
tdn             /* transmit done */
tclk            /* transmit clock */
terr            /* transmit error */
brx             /* DMA block receive interrupt */
btx             /* DMA block transmit interrupt */
iack            /* interrupt acknowledge */
int1            /* interrupt state bits */
int2
int3
int4

%output
int4
int3
int2
int1
irq             /* interrupt request */
rsdmao          /* reset DMA transmit bit */
err             /* transmit error */
d0              /* interrupt vector number bit 0 */
d1              /* interrupt vector number bit 1 */
d2              /* interrupt vector number bit 2 */

%specification
int1 = -rdn.rxen.-brx;
int2 = rtryto.rtryen;
int3 = -tclk.-terr.terren;
int4 = -tdn.txen.-btx;
irq = int1 + int2 + int3 + int4;
d0 = -iack.int3.-int1.-int2 + -iack.int4.-int1.-int2.-int3;
d1 = -iack.int2.-int1 + -iack.int4.-int1.-int2.-int3;
d2 = -iack.int1 + -iack.int2 + -iack.int3 + -iack.int4;
err = -tclk.-terr;
rsdmao = int3.-int2.-int1;
```

Address decoding PLA logic.

```
%input
a0              /* address line 0 */
a1              /* address line 1 */
a2              /* address line 2 */
a3              /* address line 3 */
cs1             /* chip select 1  */
cs2             /* chip select 2  */
iack            /* interrupt acknowledge */
as              /* address strobe */
rw              /* read/write address strobe */
lds             /* lower data strobe */

%output
trien           /* tristate pads enable */
r0en            /* nodes status register read enable */
r1en            /* control register read enable */
r2en            /* interrupt vector register read enable */
r3en            /* decrementer register read enable */
w1en            /* control register write enable */
w2en            /* interrupt vector register write enable */
w3en            /* decrementer register write enable */
h0en            /* control register recycle enable */
h1en            /* interrupt vector register recycle enable */
h2en            /* decrementer register recycle enable */
chsel           /* chip selected signal */
nodsel          /* node selected signal */

%definitions
csel = -as.cs1.-cs2.a2.a3;
nsel = -as.cs1.-cs2.-a2 + -as.cs1.-cs2.-a3;
reg0 = csel.-a0.-a1.-lds;
reg1 = csel.a0.-a1.-lds;
reg2 = csel.-a0.a1.-lds;
reg3 = csel.a0.a1.-lds;
read0en = reg0.rw;
read1en = reg1.rw;
read2en = reg2.rw;
read3en = reg3.rw;
write0en = reg0.-rw;
write1en = reg1.-rw;
write2en = reg2.-rw;

%specification
r0en = read0en;
r1en = read1en;
r2en = read2en;
r3en = read3en;
w0en = write0en;
w1en = write1en;
w2en = write2en;
h0en = -write0en;
h1en = -write1en;
h2en = -write2en;
trien = csel.rw.-lds + -iack;
chsel = csel + -iack;
nodsel = nsel;
```

Counter PLA Logic

```
%inputs
tdn             /* transmit done */
tclk            /* transmit clock */
x0              /* count state bits */
x1
x2
x3
oldtdn          /* tdn flag */
oldtclk         /* tclk flag */

%output
oltclko         /* tclk flag */
oltdno          /* tdn flag */
x3o             /* count state bits */
x2o
x1o
x0o
resetc          /* reset decrementer latch */
count           /* decrement decrementer latch */
chold           /* hold decrementer latch */

%definitions
cnt = oldtclk.-tclk;

%specification
x0o = -x0.cnt + x0.-cnt;
x1o = -x1.x0.cnt + x1.-(x0.cnt);
x2o = -x2.x1.x0.cnt + x2.-(x1.x0.cnt);
x3o = -x3.x2.x1.x0.cnt + x3.-(x2.x1.x0.cnt);
count = x3.x2.x1.x0;
oltclko = tclk;
oltdno = tdn;
resetc = oldtdn.-tdn;
chold = -(x3.x2.x1.x0);
```

Appendix 3

Interface chip user manual.

CRM68000C Users Manual

Paul C. Bunn

Department of
Computing Science
University of Wollongong

CRM68000C Users Manual

Paul C. Bunn

Department of
Computing Science
University of Wollongong

1. Introduction

The CRM68000C is a peripheral controller chip designed to interface a Motorola 68000 host machine to a Logica Polynet Cambridge Ring Node. The CRM68000C arbitrates data transfers between the Cambridge Ring Node and the host machine utilising the following features:

- an 8 bit data bus.

- flexible data transfer modes
  * status handshake transfers
  * interrupt driven transfers
  * Direct Memory Access transfers

- four on board registers
  * Node Status register
  * Control register
  * Interrupt Vector register
  * Transmission Time-out register

- transmission error checking
- transmission retry checking
- reasonable transfer rates *(yet to be determined)*
- a programmable interrupt vector address
- programmable ring node address for DMA transfers

2. General Operation

The purpose of the CRM68000C is to make the control of data transfers between the host machine and the Cambridge Ring Node faster and easier. To do this the CRM68000C has three separate data transfer modes and two different error condition interrupts. These features enable compatability with different host machines (see Fig.1 for typical system configuration). This section discusses the three modes of operation and each of the two error conditions.

2.1. Modes of Operation.

Status Handshake is the simplest of all the three modes. When using this mode of operation the ring node must be addressed explicity through the CMR68000C to initiate a data transfer. To initiate a receive operation by the ring node one of the three ring node addresses must be placed on address lines A0 and A1 (see Sect 4.0) and the Read/Write (R/W) signal asserted (note: this is achieved by a simple processor load from the address of the I/O register). When the node has serviced the receive request the CRM68000C will assert the Data Transfer Acknowledge (DTACK) signal to indicate to the processor that data is present on the data bus and the cycle may be terminated. Figure 2 shows the sequence of events between the processor and the

CRM68000C when using status handshake.

Using **Interrupt Driven** data transfers the CRM68000C will interrupt the processor when the ring node is ready to transmit or receive data. When the CRM68000C has set an interrupt the processor will acknowledge the interrupt and the CRM68000C will place an appropriate interrupt vector number(see Sect. 3.0) on the data bus as with all interrupts. The interrupt vector number is used to indicate which interrupt was set. The M68000 uses the interrupt vector number to calculate the interrupt vector address of the appropriate interrupt handler for each interrupt (for interrupt vector numbers see Table 1.). Figure 3 shows the sequence of events for a transmit interrupt.

Direct Memory Access transfers are the fastest of all transfer modes. After the DMA controller and the CRM68000C have been initialised, the CRM68000C and the DMA controller will arbitrate data transfers between the ring node and the host machine. The CRM68000C has two DMA channels; one is for the control of data reception and the other for data transmission (see Fig.4). The DMA controller compatible with the CRM68000C is the Motorola 68440 Dual Channel Direct-memory Access Controller.

For machines that do not have a DMA controller the CRM68000C can be configured so that both handshake and interrupt control can be used to transfer data. For example, transmission of data can be interrupt driven and reception of data handshake driven. This means that the CRM68000C will interrupt the processor when it is ready to transmit data and to receive data the CRM68000C must be explicitly addressed using a processor read.

The CRM68000C can deal with machines that have a DMA controller with only one channel. For example: the DMA controller could be interfaced to the reception channel of the CRM68000C and data transmission could be controlled in either handshake or interrupt mode. This means that the DMA controller will handle all receive data operations and the CRM68000C will interrupt the processor when the ring node is ready to transmit data. **N.B.** When one data transfer direction is under DMA control, the programmer **must** ensure that node accesses involving data transfer in the other direction do not interfere with DMA operations. The address of the register to/from which DMA transfers are taking place is automatically selected by the CRM68000C, protecting the channel against address changes, but not against changes to the register contents.

The flexibility of transfer modes allows the ring mode to be driven in different ways, so that it is easy to interface it to most systems using the CRM68000C.

Retry Checking allows unsuccessful data transmissions to be detected. The ring node will continue sending the same mini-packet until it is acknowledged by the destination node. The number of transmit retries, before the CRM68000C will set a transmit time-out interrupt, can be specified using the interrupt time-out register (this register counts in quantum of 16 retries thus the value in the register must be multipilied by 16 to get the actual number of retries).

When **transmit error** interrupt is enabled the CRM68000C will set the transmit error interrupt when the ring node asserts transmit error signal (TERR).

3. Interrupt Vector Numbers

The interrupt vector number is an eight bit number. The upper five bits are specified by the value stored in the upper five bits of the interrupt vector register. This gives the user the ability to select the vector number base value under programme control. The lower three bits are specified by the interrupt that occurs. Table 1 summarises the interrupt vector numbers and the priority of each interrupt, where 3 is highest priority and 0 is the lowest priority.

Table 1: Interrupt Summary.

Interrupt	priority	Interrupt Vector		
		D0	D1	D2
Receive	3	1	1	0
Trans. Time-out	2	0	1	0
Trans. Error	1	1	0	0
Transmit	0	0	0	0
Spurious	X	0	0	1

X : not applicable.

4. Addressing Modes

There are two addressing modes. The first involves addressing the ring node's registers through the CRM68000C, the second involves addressing the CRM68000C's internal registers. When addressing either the CRM68000C or the ring node both CS1 and CS2 must be asserted. Address lines A0 and A1 are used to determine whether the ring node or the CRM68000C has been selected. Address lines A2 and A3 are used to determine which of the internal registers of the CRM68000C are to be accessed. Table 2 summarises the various addressing modes.

Table 2: Addressing Summary

Device	Address				Internal Register
	A3	A2	A1	A0	
Node	1	1	X	X	
	1	0	X	X	
	0	1	X	X	
CRM68000C	0	0	0	0	Node Status
	0	0	0	1	Control
	0	0	1	0	Interrupt Vector
	0	0	1	1	Transmit Time-out

X : not applicable.

5. Data Bus Operation

The most important point to note about the data bus organisation is that the data from the node does not go through the CRM68000C but is controlled by a bi-directional buffer external to both the ring node and the CRM68000C. The direction of the buffer is controlled by the state of the Read Strobe (RD) signal of the node. The buffer is normally set in the write direction but is reversed when RD is asserted.

6. Signal Description

This section contains a brief description of the input and output signals. Included at the end is a summary describing the electrical characteristics and active states of each signal.

It is important to note that the terms *assertion* and *deassertion* will be used to indicate a signals' state. Assertion is used to indicate that a signal is driven active and deassertion is used to indicate that a signal is driven inactive. The signal summary describes the asserted states for each signal.

6.1. Signal Organisation

Some related signals can be grouped and each of these groups is discussed in the following paragraphs.

6.1.1. Ring Node Signals

Read Done (RDN) is an input to the CRM68000C and is asserted (low) by the ring node to indicate that the last receive operation by the ring node is complete and another may be initiated.

Transmit Done (TDN) is an input to the CRM68000C and is asserted (low) by the ring node to indicate that the last transmit by the node is complete and another may be initiated.

Node Read Request (RD) is asserted (low) by the CRM68000C to request a read operation by the ring node.

Node Transmit Request (WT) is asserted (low) by the CRM68000C to request a transmit operation by the ring node.

Node Acknowledge (ECHO) is asserted (low) by the ring node in response to a read or write request to indicate that the pending operation may go ahead.

Node and Mini-packet Status Signals (BPR RTP0 RTP1); BPR is the broadcast mini-packet signal. It is asserted (low) by the node to indicate that the last mini-packet received was a broadcast mini-packet. RTP0 and RTP1 are node status signals (see the Polynet Ring Node manual (1)).

Node Address Lines (NA0-NA3) are outputs from the CRM68000C and are used to select the ring node's internal registers during all data transfers.

6.1.2. Processor Signals

Address Bus Lines (A0-A3) are inputs to the CRM68000C from the host machine and are used to select either the ring node's or the CRM68000C's internal registers.

The **Data Bus Lines** (D0-D7) are used to read and write to the internal registers of the CRM68000C.

Lower Data Strobe (LDS) is asserted by the processor to indicate data is present on the lower eight bits of the data bus (D0-D7).

Address strobe (AS) is asserted (low) by the processor to indicate that a valid address is present on the address bus.

The **Interrupt Request and Acknowledge** (IRQ. IACK) signals are the interrupt control signals. IRQ is the interrupt request and is asserted (low) by the CRM68000C. For an interrupt request external hardware is required if the CRM68000C is to be interfaced with the M68000. The external hardware should handle the priority of each peripheral which is daisy chained along a priority line.**Interrupt Acknowledge** (IACK) is asserted by external hardware when the appropriate set of function codes appears on the function code control lines of the M68000. (See M68000 Advanced Information (2))

The **Data Transfer Acknowledge** (DTACK) signal is bi-directional; it indicates that a data transfer is complete.

The **Read/Write Strobe** (R/W) is an input to the CRM68000C and defines the current data transfer as a read or write cycle.

6.1.3. Direct Memory Access Channel Signals

The **DMA Request** (REQ1,REQ0) signals are asserted (low) by the CRM68000C to request a data transfer between the ring node and memory (the request generation is cycle steal for the M68440). REQ0 is a request to the DMA controller to place data on the data bus for the current transmit operation. REQ1 is a request to the DMA controller to read data from the data bus for the current receive operation.

The **DMA Acknowledge** (ACK0,ACK1) signals are asserted (low) by the DMA controller to indicate to the CRM68000C that data is being transferred in response to the previous request. ACK0 is asserted in response to a DMA transmit request and ACK1 is

The **Data Transfer Complete** (DTC) signal is asserted (low) by the DMA controller to indicate that the data transfer is complete and the data has been successfully transferred.

The **DMA Done** (DONE) signal is asserted (low) by the DMA controller to indicate that the data transferred is the last in the block. This occurs when the DMA controllers internal transfer count register is decremented to zero.

6.1.4. General

The **Two Phase Clock** (PHI1,PHI2): the first and second phases of a non-overlapping two phase clock, and generated from the M68000 processor clock. These signals must be generated externally, this allows for flexibility when testing allowing frequency and mark-space ratios to be counted.

The **Chip Reset** (RESET) is asserted (low) to reset the CRM68000C. It places all the internal logic into its initial state but does not reset the register contents.

The **Chip Selects** (CS1,CS2) when both are asserted select the CRM68000C.

Vdd is the positive supply which is five volts.(see Sect.9)

Vgnd is the ground connection pin.(see Sect.9)

Table 3: Signal Summary.

Funct. group	signal name	direction	active state	driver type	pin number
Chip Sel.	CS1	in	high		14
	CS2	in	low		15
Node	TDN	in	low		44
	WT,RD	out	low		37,38
	RDN	in	low		43
	ECHO	in	low		36
	TCLK	in	low		2
	TERR	in	low		3
	BPR	in	low		45
	RTP0	in	low		46
	RTP1	in	low		47
Processor	A0-A3	in	high		10-13
	NA0-NA3	out	high		7-4
	D0-D7	in/out	high/low	tri-state	29-22
	LDS	in	low		18
	AS	in	low		17
	IRQ	out	low		48
	IACK	in	low		16
	DTACK	in/out	low	tri-state	20
	R/W	in	high/low		19
DMA	REQ0	out	low		41
	REQ1	out	low		42
	ACK0,ACK1	in	low		32,33
	PCL0,PCL1	out	low		39,40
	DTC	in	low		34
	DONE	in	low		35
General	PHI1,PHI2	in	high/low		21,9
	RESET	in	low		30
	Vgnd,Vdd	in			8,31

7. Registers

This section contains a description of the CRM68000C's internal registers and the control bit assignments within each register. There are four eight-bit registers. In each register summary any bit not used is filled with a `0'. Following this is a table summarising various characteristics of each register.

7.1. Node Status Register

The Node Status Register contains the current status of the ring node at any given time.

bit 7	6	5	4	3	2	1	0
IRQ	Zero	ERR	RTP1	RTP0	BPR	RDN	TDN

TDN,RDN,BPR,RTP0,RTP1: These have the same definitions as in the signal description.

err: when asserted (high), indicates that there has been a transmit error.

Zero Count: when asserted (high), indicates that there is a time-out interrupt.

IRQ: *when asserted (high), indicates an interrupt request has been generated.*

Register Address: A3=0, A2=0, A1=0, A0=0.

Operations: Read only.

7.2. Control Register

The control register is used to set up the different modes of operation, DMA and interrupt.

bit	7	6	5	4	3	2	1	0
	NA3	NA2	NA1	NA0	TXEN	RXEN	DMAT	DMAR

DMAR: With the DMA Receive bit (DMAR) asserted (high) , the CRM68000C will change the control of received data transfer, from status handshake to DMA control. Data is than transferred directly into memory of the host machine via the DMA controller. On completion of the DMA transfer DMAR is automatically deasserted (low). The side effect of setting this bit is, the receive interrupts will be disabled until the DMA receive transfer is completed.

DMAT: With the DMA Transmit bit (DMAT) asserted (high), the CRM68000C will change the control of data to be transmitted by the ring node from status handshake to DMA control. Data is then directly transferred from the memory of the host machine via the DMA controller to the node. On completion of the DMA transfer the DMAT will automatically be deasserted. As with DMAR the side effect of setting this bit is, the transmit interrupts will be disabled until the transfer is completed.

RXEN: With the Read interrupt bit (RXEN) asserted (high), the CRM68000C will interrupt the processor whenever a mini-packet arrives and thus should be read and DMAR is not asserted.

TXEN: With Transmit interrupt bit (TXEN) asserted (high), the CRM68000C will interrupt the processor whenever the node is free to service a transmission and DMAT is not asserted. Note, if there is no more data to be transmitted this bit must be deasserted.

NA0–NA3: These bits specify the node address required for a DMA transfers.

Register Address: A3=0, A2=0, A1=0, A0=1.

Operations: Read/write.

7.3. Interrupt Vector Register

The interrupt vector register controls the two error interrupts and sets the upper five bits of the interrupt vector register.

bit	7	6	5	4	3	2	1	0
	IV7	IV6	IV5	IV4	IV3	0	RTEN	TERREN

TERREN: With the Transmit Error Enable bit (TERREN) asserted (high) the CRM68000C will interrupt the processor whenever the ring node detects a transmit error.

RTEN: With the Transmit Time Out Error bit (RTEN) asserted (high) the CRM68000C will interrupt the processor whenever the number of retries for the current transmission exceeds sixteen times the number stored in the Transmit Time-out Register.

IV3-IV7: These bits specify the most significant bits of the interrupt vector Number (IV3-IV7), the lower three bits (IV0-IV2) of the interrupt vector number are specified by the type of interrupt that occurs.

Register Address: A3=1, A2=0, A1=1, A0=0.

Operations: Read/Write.

7.4. Transmit Time-out Register

The Transmit Time-out register (8 bits) sets the number of transmit retries before the CRM68000C will set a transmit time-out interrupt (one bit is equivalent to sixteen retries).

Register Address : A3=0, A2=0, A1=1, A0=1.

Read/Write.

Table 4: Register Summary.

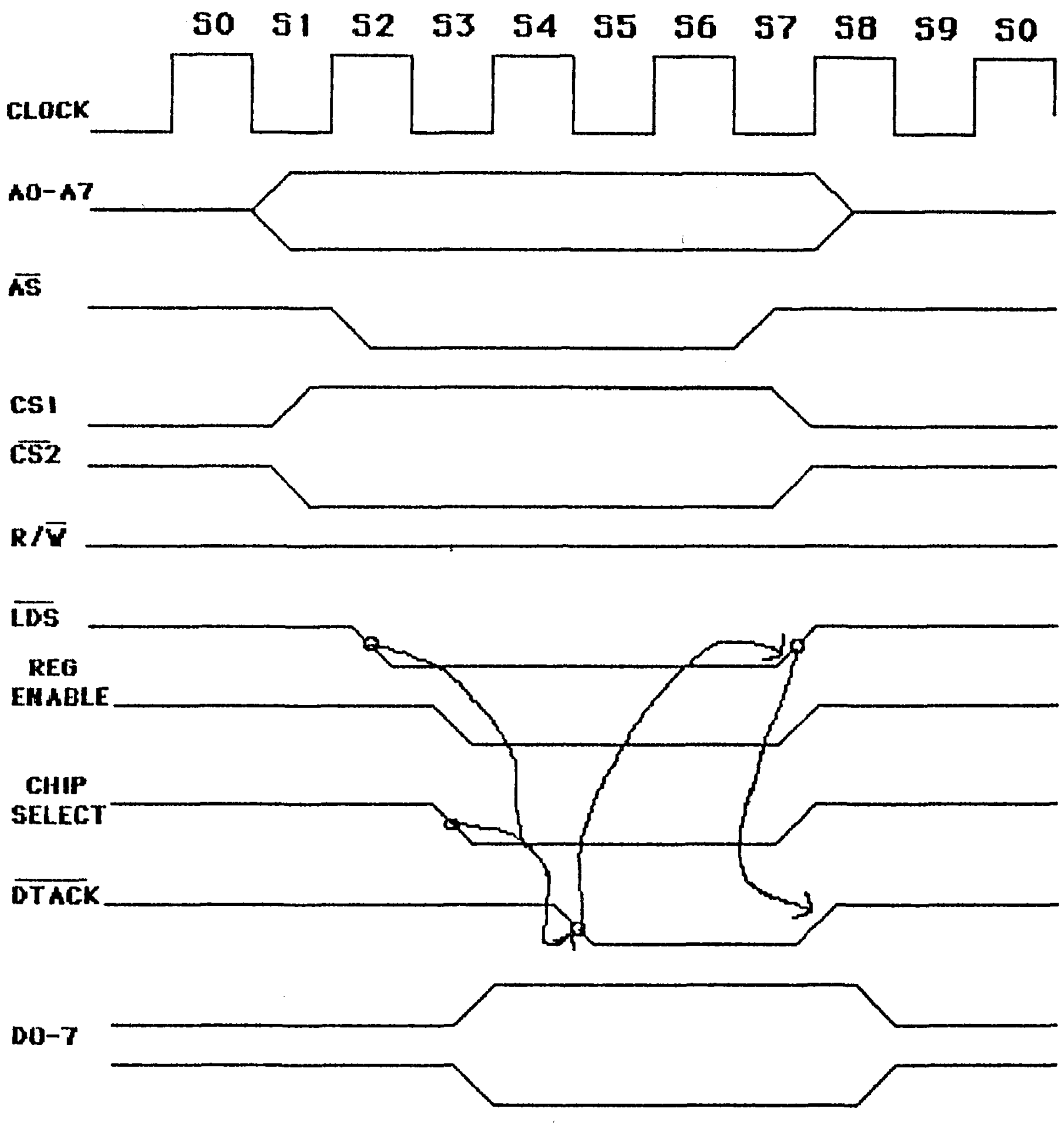
Register name	Operations	Address (A3 A2 A1 A0)
Node status	read only	0 0 0 0
Control	read/write	0 0 0 1
Interrupt Vector	read/write	0 0 1 0
Transmit Time-out	read/write	0 0 1 1

8. Data Transfer Modes: Timing Details.

This section contains the Timing diagrams for each data transfer mode. Along with these are the normal processor read and write timing diagrams which show the details for a write to a CRM68000C internal register. These timing diagrams are not precise, they are meant to show only the sequence of events that occur. More precise timing diagrams will be produced when real-time measurements can be made.

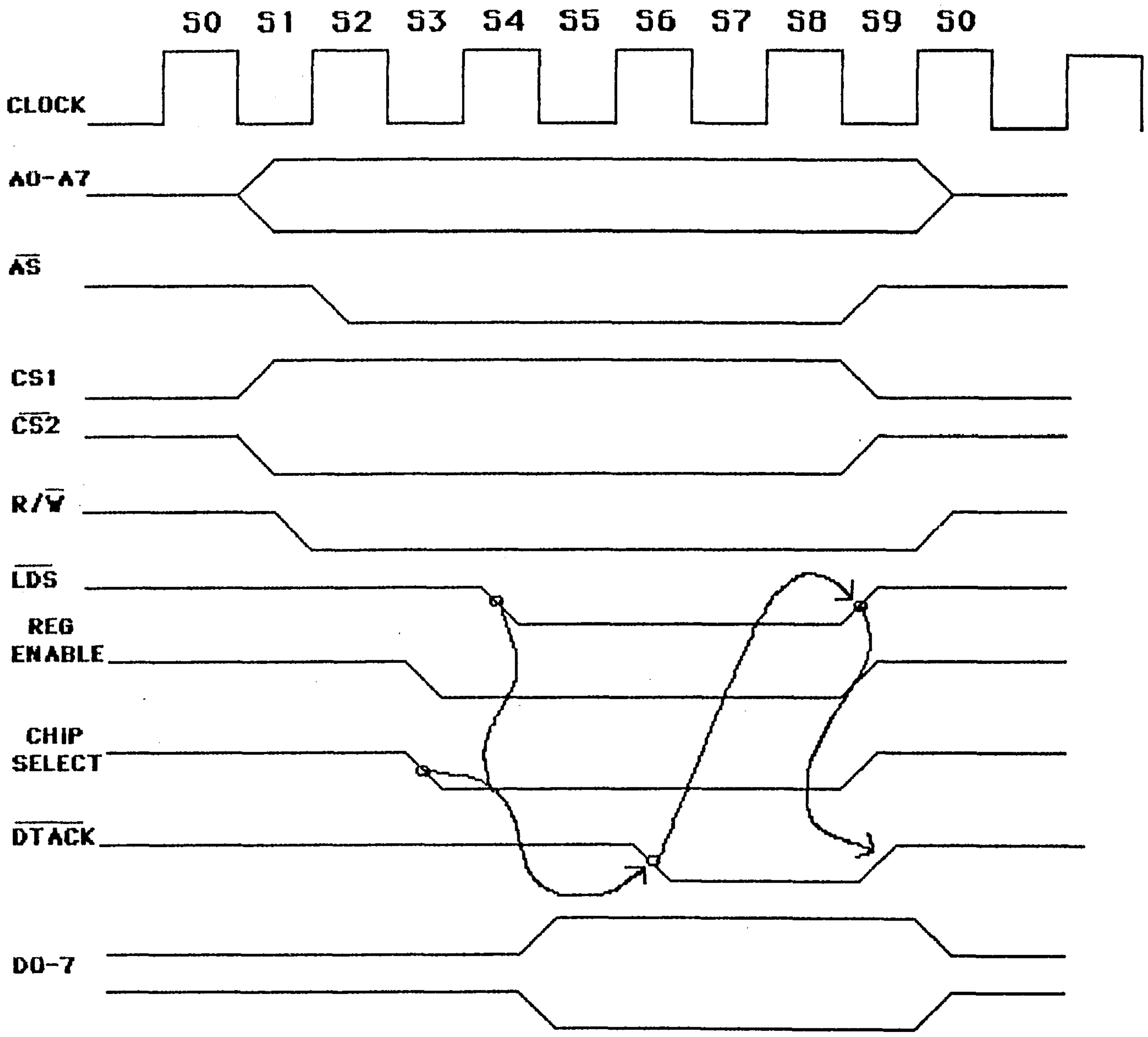
8.1. Processor Read Timing Diagram (odd byte read from the CRM68000C)

This timing diagram illustrates the sequence of events when an internal register is read.



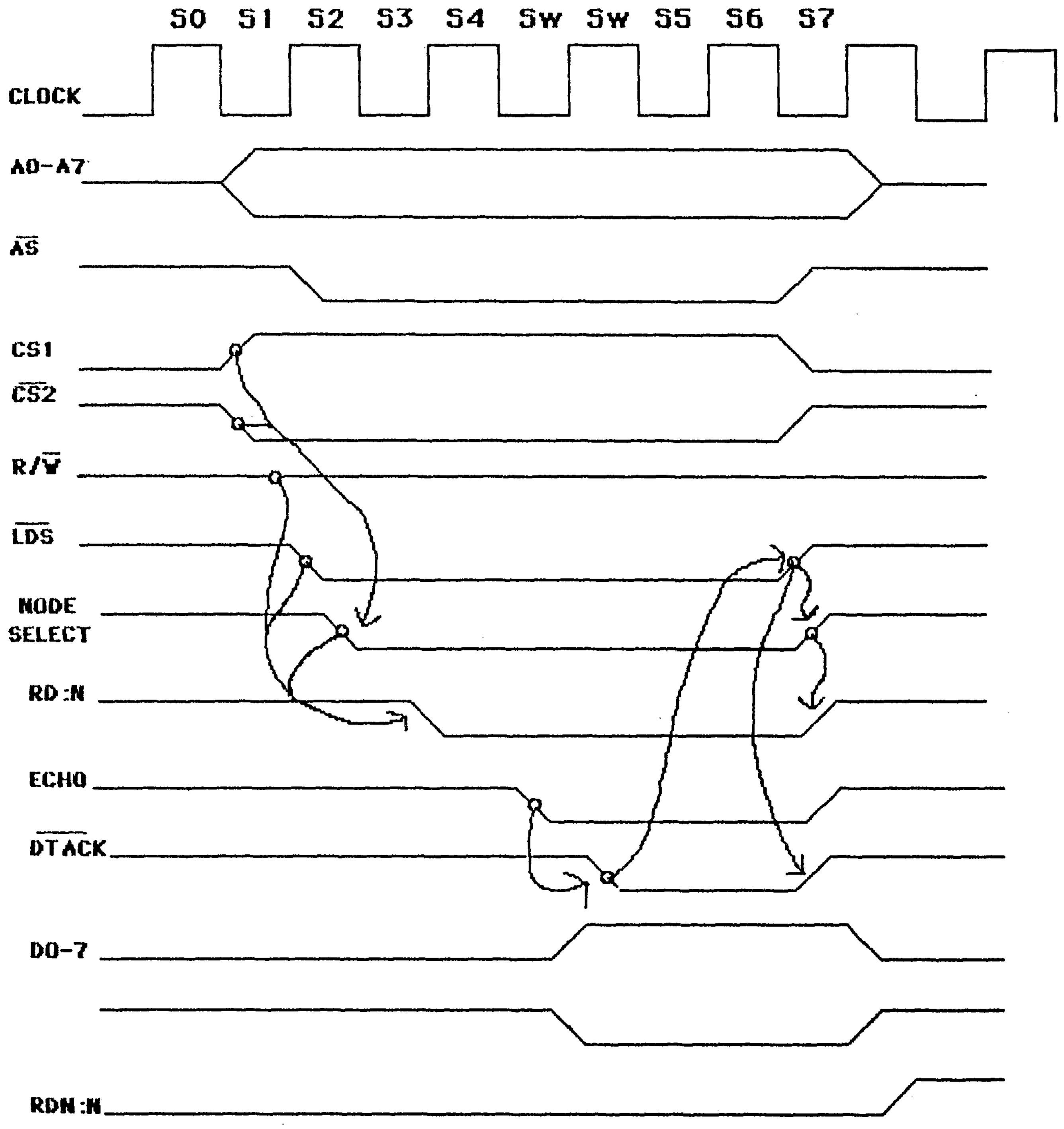
8.2. Processor Write Timing Diagram (odd byte write to the CRM68000C)

This timing diagram illustrates the sequence of events when an internal register is written to.



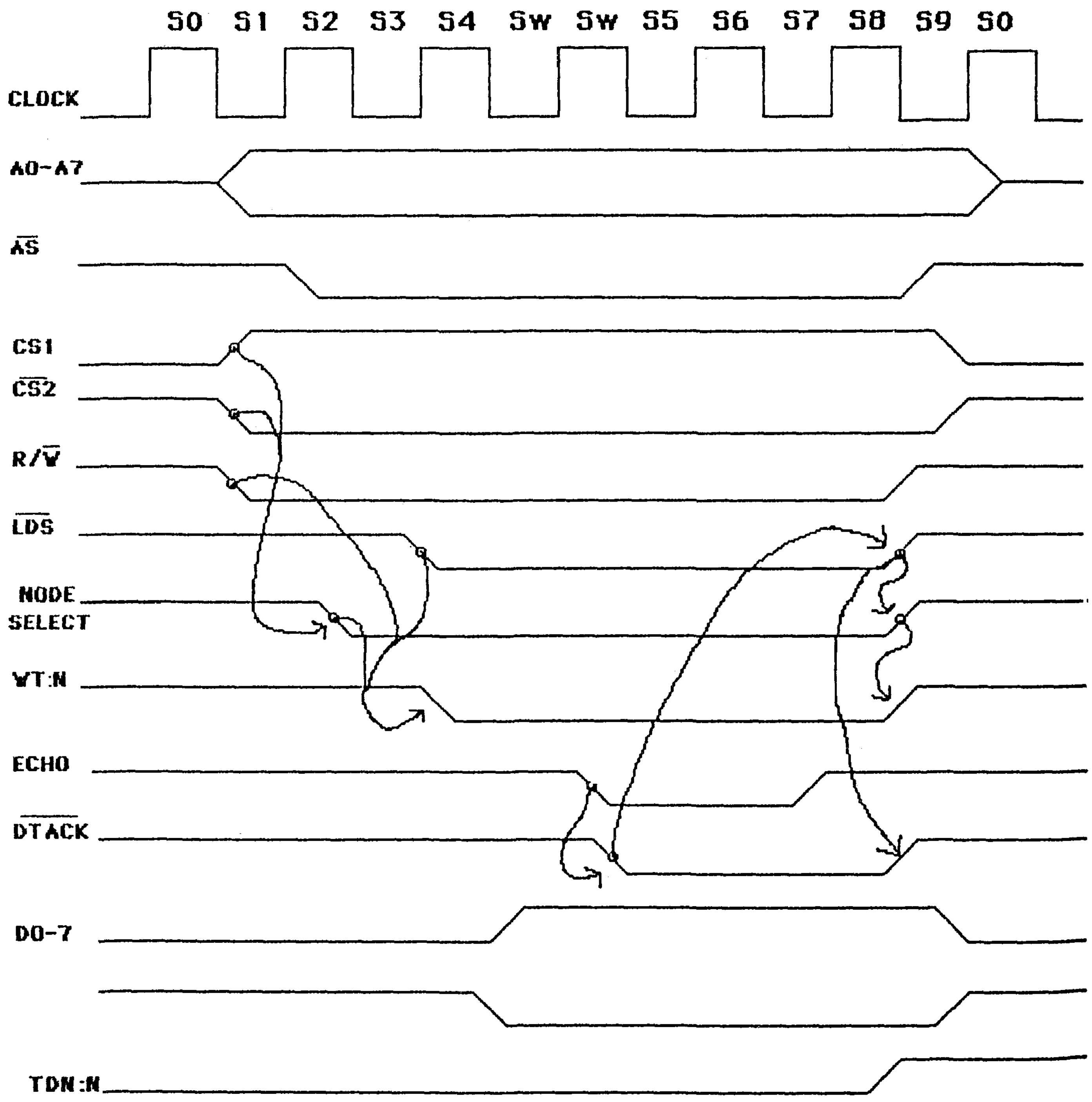
8.3. *Processor Read Timing Diagram* (odd byte read from the ring node)

This timing diagram illustrates the sequence of events when the ring node is addressed via CRM68000C for a receive operation.



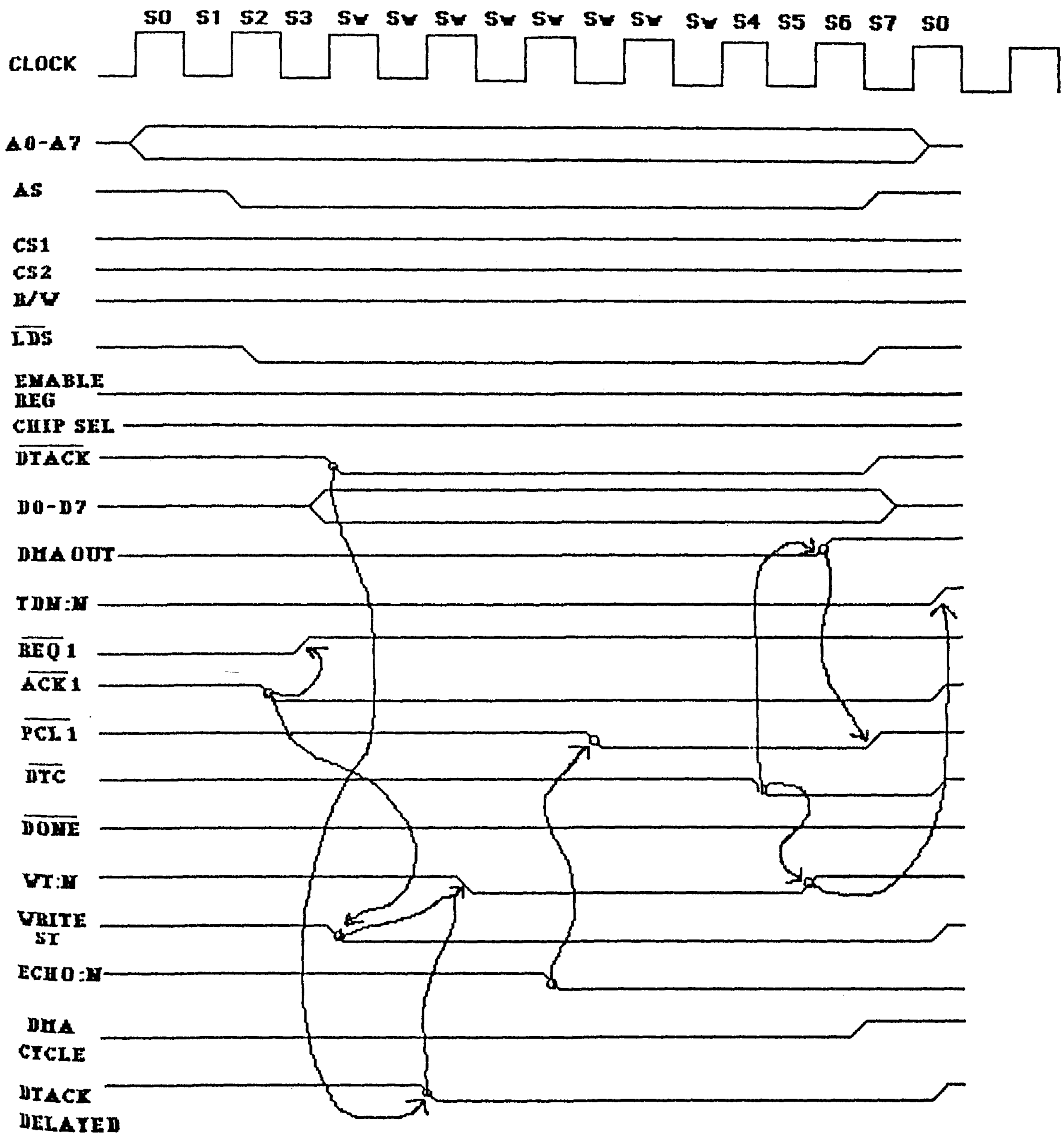
8.4. Processor Write Timing Diagram (odd byte write to the ring node)

This timing diagram illustrates the sequence of events when the ring node is addressed via CRM68000C for a transmit operation.



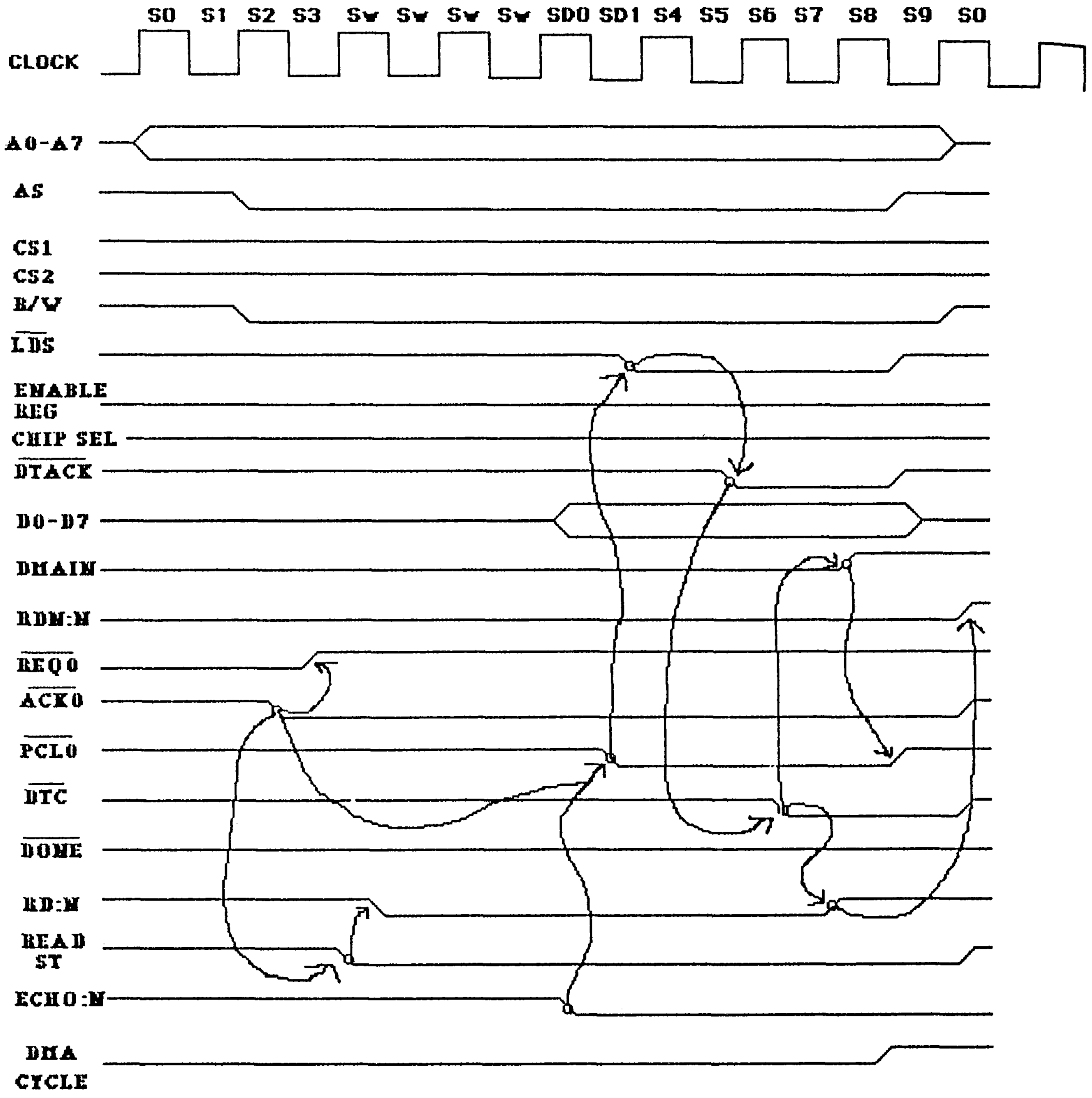
8.5. DMA Receive Timing Diagram

The DMA Receive Timing diagram shows the sequence of events when the CRM68000C is controlling a DMA receive operation.



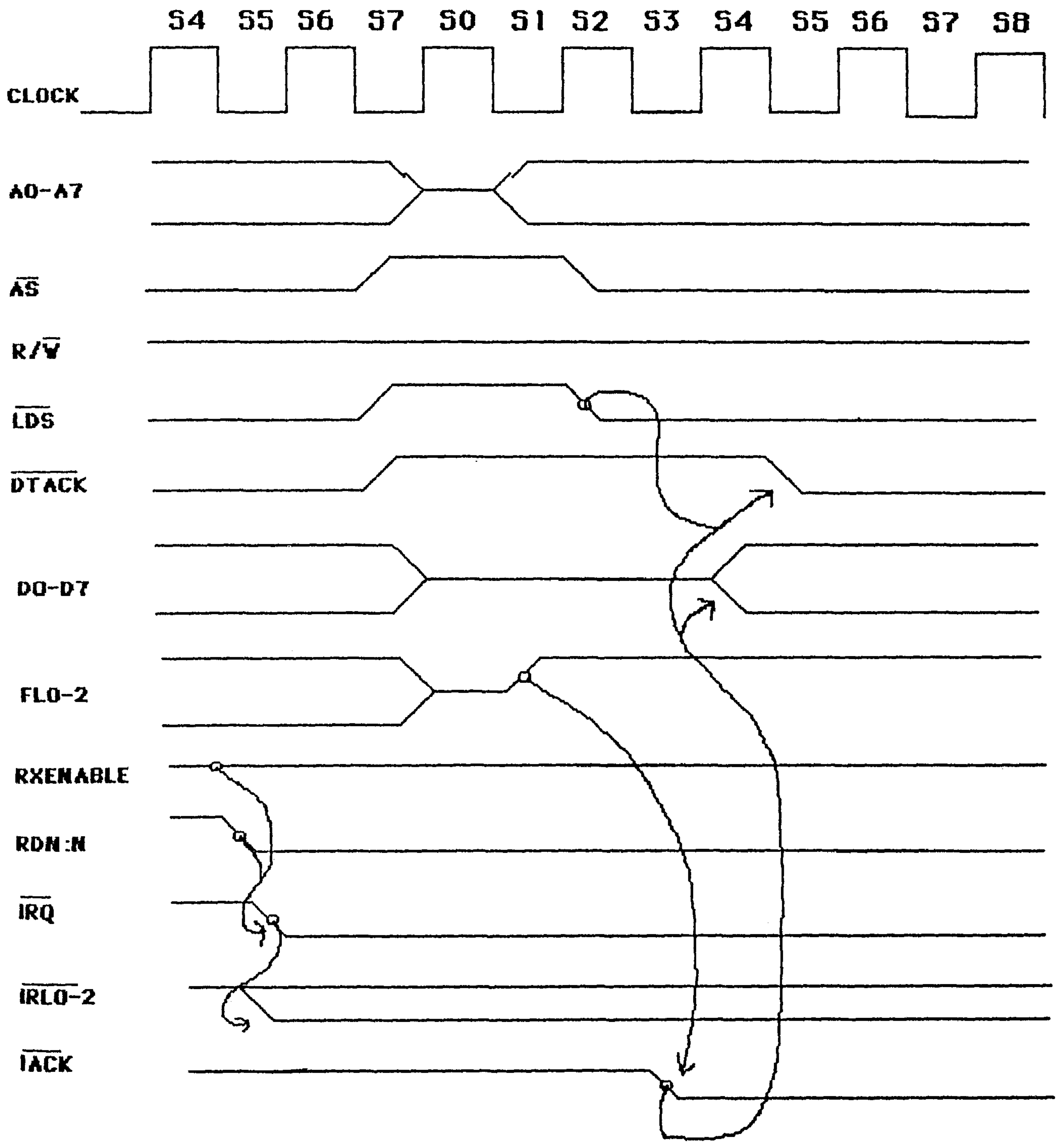
8.6. DMA Transmit Timing Diagram

The DMA Transmit Timing diagram shows the sequence of events when the CRM68000C is controlling a DMA transmit operation.



8.7. Interrupt Acknowledge Timing Diagram

This timing diagram shows the CRM68000C's response to an IACK from the processor via the external hardware.



9. Hardware Details

9.1. Power Supply

Vdd : +5 volts.
Vgnd: ground

9.2. Pin out

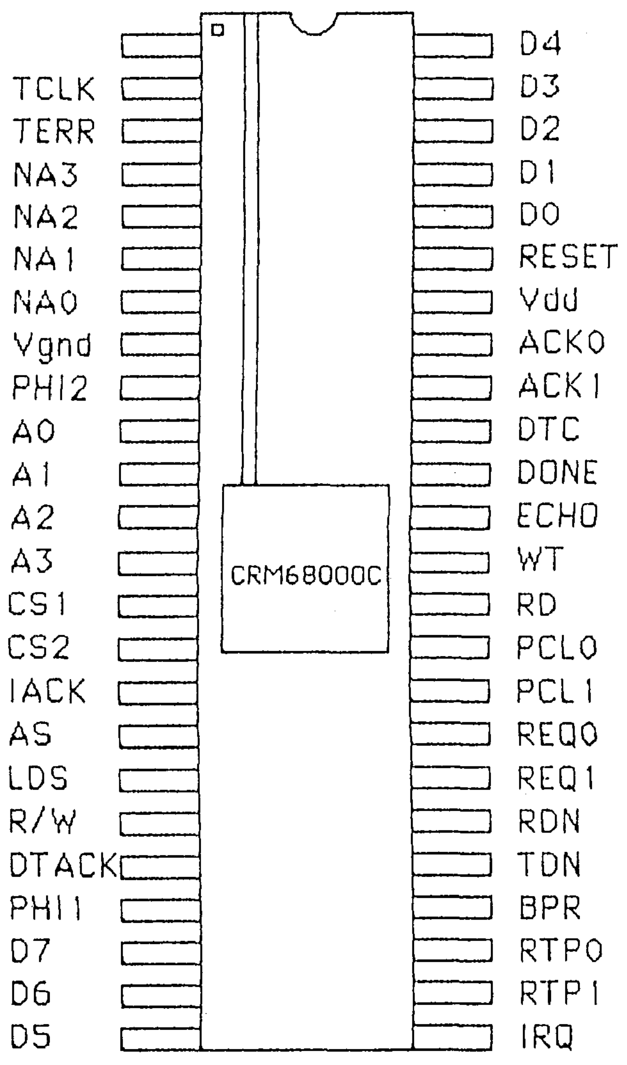


Figure 1. Typical system configuration.

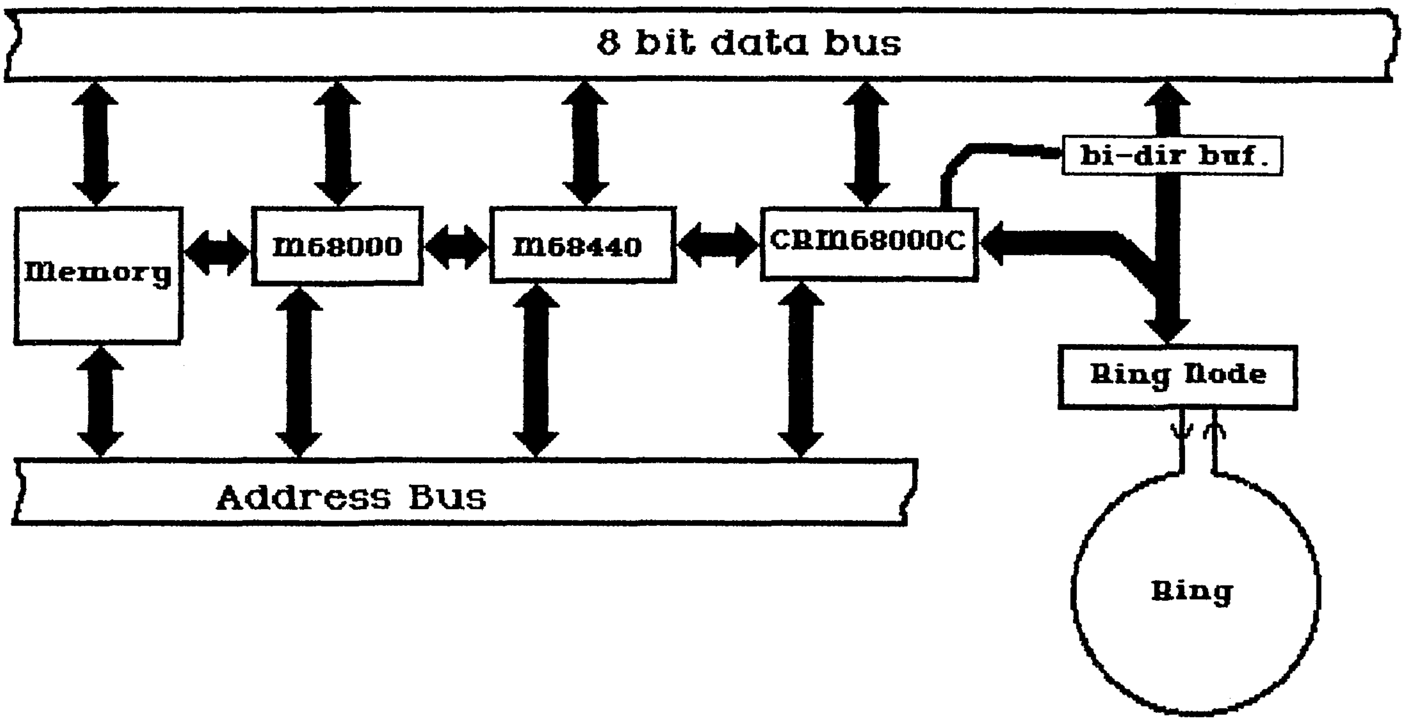


Figure 2. Sequence of events during a receive operation, using status handshake control.

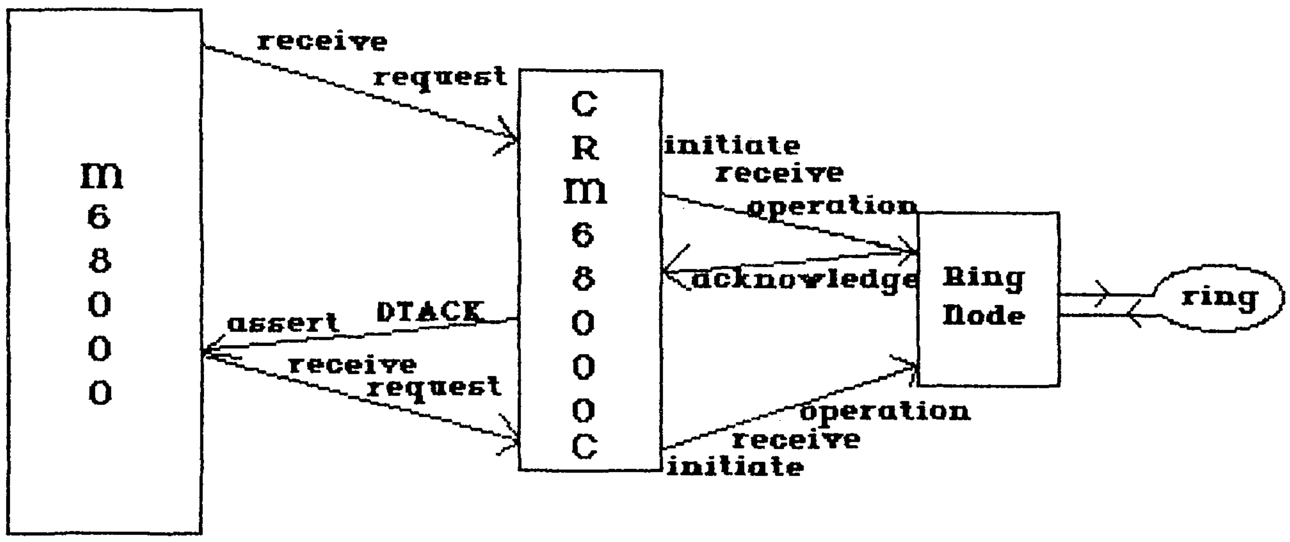


Figure 3. Sequence of events during a receive interrupt.

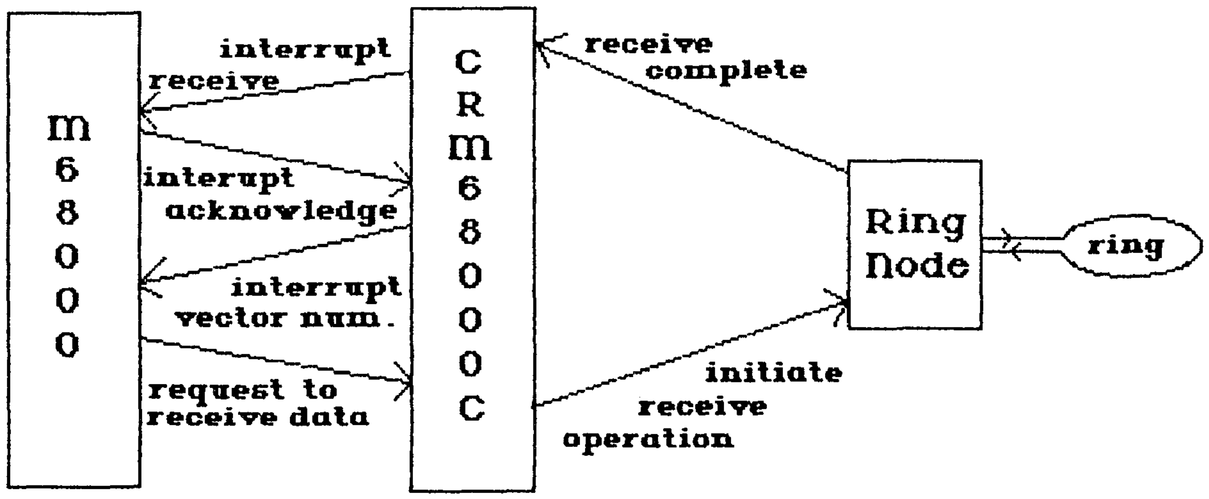
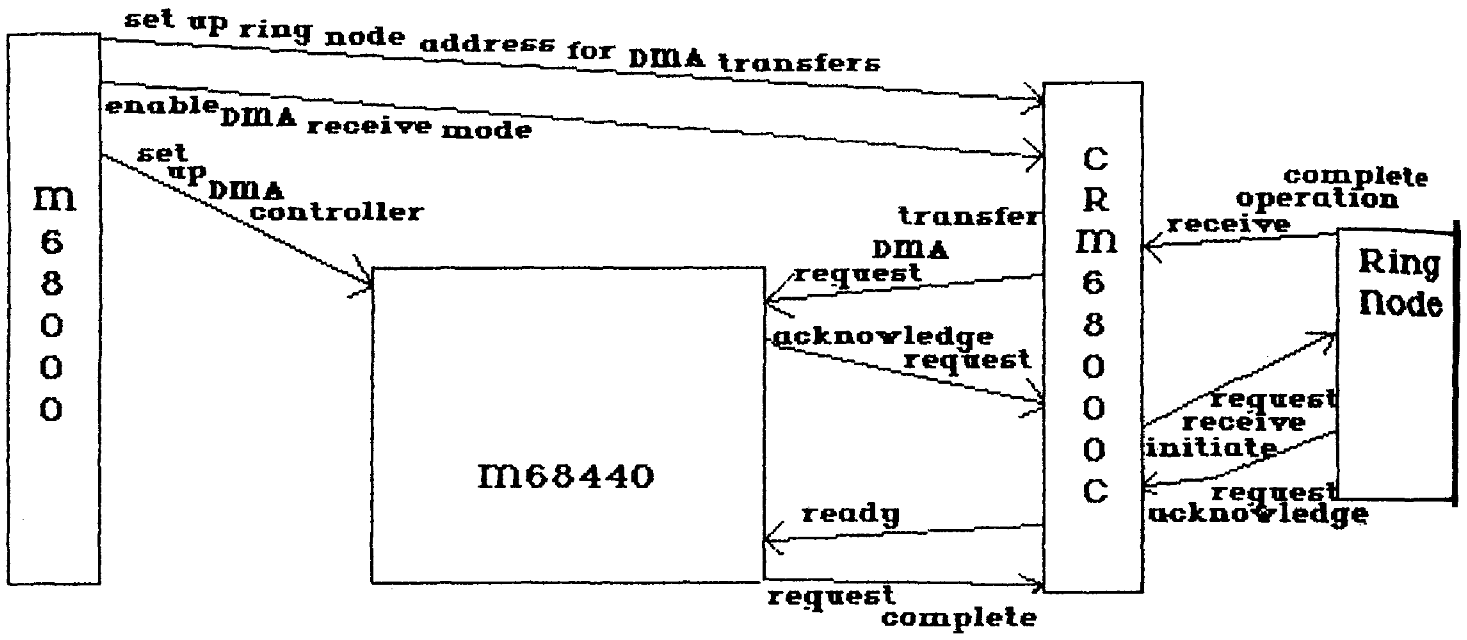


Figure 4.Sequence of events during a DMA receive operation.



Appendix 4

Software manual pages.

NAME
alt – Activate the alternate port on the VC4404 terminals

SYNOPSIS
alt

DESCRIPTION
Alt reads a file from standard input, turns on the alternate port on the VC4404 terminal, writes the file to standard output and then turns off the alternate port.

SEE ALSO
plotcif (1) belle(1), celle(1), plagen(1), getsymbol(1)

AUTHOR
Paul C Bunn
University of Wollongong

NAME

belle – integrated circuit mask layout language

SYNOPSIS

belle filename.bel

DESCRIPTION

Belle will generate an intergrated circuit mask layout from the BELLE specification in the input file and place it in *filename*.cif.

BELLE is the procedural integrated circuit mask layout language distributed by the CSIRO Division of Computing Research VLSI program as an aid to designing circuits for AUSMPC 5/82. It is embedded in the Pascal programming language. Embedding the language allows the power of a high level programming language to be used to aid the description of integrated circuit mask layouts, without the need to develop an entirely new programming language. BELLE is similar to the Simula package LAP developed at Caltech. Because of the differences between the base languages, the syntax of user statements is rather different.

BELLE is composed of a set of Pascal procedures which can be used to describe the various structures in a layout. BELLE generates its output in CIF (Caltech Intermediate Form) [Hon and Sequin, 1980], which is a low level description of the circuit. CIF is the standard data format chosen for communication of designs during the MPC.

BELLE is a leaf-cell tool. That is, it is intended to describe the geometries of relatively small subcircuit elements. It may also be used to some extent as a composition tool to place several cells (i.e., compose them) to generate a complete layout.

One of the major features of BELLE is that it provides a very convenient means for parameterising circuit components. A circuit used in one design can be used in another by simply modifying the parameters for size, number of inputs or outputs, etc. As BELLE modules are created by designers for various circuits it may be possible for the design community to build up a library of BELLE procedures for commonly used circuit components such as memory cells, decoders, comparators, etc.

The built-in functions of BELLE are:

SETSYMNO	– Set the symbol number of the next symbol to be defined.
DEFINE	– Define a symbol.
ENDDEF	– Delimits symbol definition.
DRAW	– Draw an instance of a symbol.
MX	– Mirror a symbol in the X-axis.
MY	– Mirror a symbol in the Y-axis.
ROT	– Rotate a symbol.
LAYER	– Set the current layer.
BOX	– Draw a box (rectangle).
FLASH	– Draw a circle.
WIRE	– Starts the definition of a wire.
X	– Add a horizontal segment to the wire.

Y – Add a Vertical segment to the wire.

XY – Add a 45 degree segment to the wire.

DX – Add a relative horizontal wire segment.

DY – Add a relative vertical wire segment.

DXY – Add a relative 45 degree wire segment.

NODELABEL
– Label a node.

COMMENT
– Insert a Comment into the CIF.

EXISTING
– See if a symbol has been defined.

IMPORTSYMBOLS
– Read a header file containing a list of symbol names, numbers and bounding boxes of externally defined symbols that are used by the BELLE module.

SETNOEND
– Suppress generation of the End statement in the CIF output file.

SET45
– Allows 45 degree wires.

BOUNDINGBOX
– Calculate the bounding box of a previously defined symbol.

ABORT
– Stop execution and output an error message.

POLYCUT
– Generates a Metal to Polysilicon contact cut.

DIFFCUT
– Generates a Metal to Diffusion contact cut.

BUTTCONTACT
– Generates a Poly to Diffusion contact via a Butting Contact.

SEE ALSO
BELLE users tutorial.
plotcif (1) belle(1), celle(1), plagen(1), getsymbol(1), simpl(1), simul(1), plalog(1).
The CSIRO VLSI programme.

BUGS
The BUTTCONTACT fuctions will sometimes fail to draw the Butting Contact in the orientation specfied.

NAME

celle – Integrated circuit mask layout language

SYNOPSIS

celle [option]filename.cel

DESCRIPTION

Celle will generate an intergrated circuit mask layout from the **CELLE** specification in the input file and place it in *filename*.cif.

CELLE is the procedural integrated circuit mask layout language based on BELLE. The only difference between CELLE and BELLE is CELLE is embedded in C not Pascal. For a description of CELLE see belle(1) and belle(5). The advantage of celle over belle is that celle is more that six times faster, especially for large layouts.
The options that *celle* interprets are:

–r link in the router.

SEE ALSO

BELLE users tutorial.
belle(1), plotcif (1) celle(1), celle(1), plagen(1), getsymbol(1), simpl(1), simul(1), plalog(1).
The CSIRO VLSI programme.

BUGS

The BUTTCONTACT functions will sometimes fail to draw the Butting Contact in the orientation specified.

AUTHOR

Paul C Bunn University of Wollongong

NAME

extract – Extract symbol definitions from a ciffile.

SYNOPSIS

extract [option...] ciffile outputfile.

DESCRIPTION

Extract will extract symbol definitions from a **CIF 2.0** ciffile.

The following options are interpreted by *extract*.

–s Print a summary of all the symbol definition names and their symbol numbers on standard output that are in the ciffile.

–c Insert a call to the extracted symbols.

–l Specify lambda for the extracted symbols.

–n Do not place the **End** symbol on the end of the ciffile

SEE ALSO

plotcif (1) belle(1), celle(1), plagen(1), getsymbol(1), simpl(1), simul(1), plalog(1).
The CSIRO VLSI programme.

AUTHOR

Paul C Bunn
University of Wollongong

NAME

getsymbol – Getsymbol generates a header file from a ciffile.

SYNOPSIS

getsymbol ciffile.cif

DESCRIPTION

Getsymbol is an auxiliary programme to **BELLE**. It is used in conjunction with the **IMPORTSYMBOLS** function within BELLE. It is used to generate a header file of the type required by IMPORTSYMBOLS. *Getsymbol* operates on a ciffile and produces a file containing the names of all symbols, their symbol numbers and their bounding box. An example output from *getsymbol* looks like:

```
Lambda 250
901 DiffCut -500,-500,500,500
902 PolyCut -500,-500,500,500
903 ButtCont -750,-500,750,500
```

The first record specifies lambda for this library.
The second and subsequent records have the format
    Sym.No. Sym.Name Bounding box (lower left, upper right corners)
All Coordinates are given in hundredths of microns.

SEE ALSO

plotcif (1) belle(1), celle(1), plagen(1), extract(1), simpl(1), simul(1), plalog(1).
The CSIRO VLSI programme.

AUTHOR

Paul C Bunn
University of Wollongong

NAME

plagen - PLA generator

SYNOPSIS

plagen plafile.pla

DESCRIPTION

Plagen is a program which will generate a pla in CIF code from a truth table description. The program is largely interactive, from the point of view of specifying options in the pla generation. These user supplied options will be described shortly.

The general operation of *plagen* is that it reads from a file *plafile.pla* and creates the pla in a file *plafile.cif.* The input file is of the form of a truth table, preceded by a line which contains the number of inputs, product terms and outputs. For example, consider a pla with 4 inputs, 5 product terms and 3 outputs. The input to plagen might then be:

```
4 5 3
10x1 010
xx11 00x
1100 1x1
100x x01
001x 000
```

A single space is required to separate the inputs from the outputs for each product term and also to separate the three fields on the first line. Generation of the pla is then under the control of the following requested information:

symbol start number: the main symbol number of the pla is given this value and all subsymbols are given numbers relative to this. This is needed for Belle to avoid conflicting symbol numbers.

subsymbol generation: suppression of cif code for the subsymbols may be achieved for cases where more than one pla is being used, each using the same subsymbols.

pla programmed: placement of programming cells may be ommitted for cases where only the overall size is required.

inputs/outputs: a number of obvious selections regarding clocked or unclocked inputs/outputs and whether a finite state machine is being generated.

outputs position: the "traditional" pla has the outputs and inputs on the same side (necessary for finite state machine generation). This flag allows generation of a pla with ouputs on the opposite side of the pla to the inputs.

labels: unless specifically ommitted all inputs and outputs are labelled. For a pla with symbol number N specified, inputs are labelled as plaNinx where x ranges from 0 to (inputs-1), with '0' referring to the left-most input (furthest from the OR plane). Outputs are labelled as plaNoutx (e.g. pla100out3) with x=0 being the output closest to the AND plane. All labelling is done using the **94** extension of **CIF**.

The cells which plagen uses to build up the pla are similar but not the same as the standard Hon & Sequin cells distributed. The input and output drivers are the same but programming and other cells are not. For this reason the symbol names have been changed to avoid confusion with the other pla cells. CIF code for all these cells is contained in the program.

The main symbol for the pla (which is always generated) has a symbol name of plaN, where N is the symbol number specified by the user.

SEE ALSO

plotcif (1) belle(1), celle(1), plagen(1), extract(1), simpl(1), simul(1), plalog(1).
The CSIRO VLSI programme.

AUTHOR

Paul C Bunn
University of Wollongong

NAME

plotcif – Plot a cif file

SYNOPSIS

plotcif [*option*...] *filename.cif*

DESCRIPTION

Plotcif will produce input for a Servogor 281 plotter, Tektronix 4010 style terminals, Hewlett Packard 7574a plotter, Calcomp drum plotter and VC404, VC4404 terminals, from a standard **CIF 2.0** file. *The file name must end in `.cif'*. *Plotcif* produces GAP output and passes it via a pipe to a nominated gap interpreter, which produces the specific device input (default being the Tektronix 4010 style) on standard output.

The Following options are interpreted by *plotcif*.

–tek Produce graphics output specific to a Tektronix 4010 style graphics terminal.

–tty Produce output suitable for the VC404, VC4404 terminals and 300 series Ballistic printers.

–hpg Produce output suitable for the Hewlett Packard 7574a plotter.

–ser Produce output suitable for the Servogor 281 flat bed plotter.

–cal Produce output for the Calcomp drum plotters.

–gap Produce GAP output.

–b Report the bounding box size of the CIF layout.

–h *header*.
Specify a header. The string *header* is drawn at the base of the plot along with the current date and time.

–r *angle,x,y*.
Rotate the plot *angle* degrees around the centre *x,y*.

–w *xl,yl,xu,yu*.
Select a window from *xl,yl* to *xu,yu*.

The *plotcif* CIF parser will report any inconsistencies within the ciffile.

FILES

/usr/gap/tty	standard terminal GAP interpreter.
/usr/gap/tek	Tektronix GAP interpreter.
/usr/gap/ser	Servogor 281 GAP interpreter.
/usr/gap/hpg	Hewlett Packard 7574a GAP interpreter.
/usr/gap/cal	Calcomp drum plotter GAP interpreter.

SEE ALSO

belle(1), celle(1), extract(1), plagen(1), getsymbol(1), simpl(1), simul(1), plalog(1), gap(1), gap(5).
GAP, Graphics Assistance Package, R.S.Nealon, University of Wollongong, preprint 80–6.
The CSIRO VLSI programme.

DIAGNOSTICS

The diagnostics produced by *plotcif* itself are intended to be self–explanatory.

AUTHOR

Paul C Bunn
University of Wollongong

BUGS

Plotcif requires the whole layout to be in memory, due to the way in which it builds the parse tree. Therefore, generation of plots for extremely large layouts may be slow.

NAME

renumber – Renumber the symbol definitions within a ciffile.

SYNOPSIS

renumber [option...]

DESCRIPTION

Renumber will read a standard **CIF 2.0** ciffile from standard input renumber all the symbol definitions and symbol calls and with them to standard output.

The following options are interpreted by *renumber*.

–s *number*
: will start the renumbering from *number*, the default starting value is 0.

SEE ALSO

plotcif (1) belle(1), celle(1), plagen(1), getsymbol(1)

AUTHOR

Paul C Bunn
University of Wollongong