Motorola Digital Signal Processors

DSP96002 Interface Techniques and Examples

by R. Robles
Z. Rozenshein
O. Rubinstein
A. Vainberg

Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters can and do vary in different applications. All operating parameters, including “Typicals” must be validated for each customer application by customer’s technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part. Motorola and Ⓜ are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

Table of Contents

SECTION 1

DSP96002 Data Transfer Techniques

Table of Contents

Table of Contents

SECTION 2

Connecting the DSP96002 as an Attached Processor to the ISA Bus Turns PC/AT into a Fast and Powerful IEEE Compatible Floating Point Computer

Table of Contents

Table of Contents

SECTION 5

A Non-Intrusive Cycle Counter for the DSP96002 ADS

Illustrations

Illustrations

List of Tables

SECTION 1

DSP96002 Data Transfer Techniques

by A. Vainberg

1.1 Introduction

This section presents three high performance interconnection techniques with several DSP96002s. Transfer procedures are designed in such a manner that minimum DSP96002 CPU intervention is required. For this purpose, one of the two on-chip DMA channels is used and CPU intervention is required only in the initial phase for programming the DMA channels.

"The three transfer techniques presented in this section are Full Handshake DMA transfer..., Partial Handshake DMA transfer..., and No Handshake DMA transfer..."

Unidirectional data transfer is assumed for simplicity; however, bidirectional data transfer can be implemented in the same manner. The model is composed of two processors, one is the Master Processor and the other is the Slave Processor. The model can be easily expanded for configurations with more than two processors. The data transfer direction is from Master to Slave. The particular implementation in this section is based on data transfer from the Master Processor internal memory, to Slave Processor internal memory. One of the Master Processor's two DMA channels is used to transfer data from Master internal memory to the Slave Processor’s Host Interface. One of the Slave Processor DMA channels is used to transfer the received data from the Host Interface to internal memory.

The three transfer techniques presented in this section are:

- *Full Handshake DMA transfer:*
  one 4 byte transfer every 4 instruction cycles
- *Partial Handshake DMA transfer:*
  one 4 byte transfer every 2 instruction cycles
- *No Handshake DMA transfer:*
  one 4 byte transfer every 1 instruction cycle

1.2 Full Handshake DMA Transfer

1.2.1 Description

A full handshake DMA transfer provides data transfer from the internal memory of the Master Processor to the internal memory of the Slave Processor. For this purpose, one DMA channel is allocated to each processor. One of the Master Processor DMA channels is programmed to read data from internal memory and then to write this data to the Slave Processor Host Transmit Register. The Slave Processor DMA channel is programmed to read data from the Host Receive Register and to write this data in its internal memory.

The Slave Processor DMA channel is programmed as “Single Block, Word Transfer, Triggered by the DMA Request” where the DMA request is the Host Receive Data Full (HRDF=1) flag. After the Slave Processor DMA channel is enabled, Transmit Data Register Empty (TXDE) status asserts the $\overline{\text{HR}}$ line of the Host Interface. The Slave Processor $\overline{\text{HR}}$ line connects to the Master Processor $\overline{\text{IRQA}}$ line.

The Master Processor DMA channel is programmed as “Single Block, Word Transfer, Triggered by DMA Request”, where the DMA request is $\overline{\text{IRQA}}$. A DMA request is generated when the Master Processor DMA channel is enabled and $\overline{\text{IRQA}}$ is asserted. In response to this DMA request, the Master Processor DMA channel reads data from internal memory and starts a bus write cycle. New data is written into the Slave Host Interface Transmit Data Register with the deassertion of $\overline{\text{TS}}$. The TXDE bit is cleared, the $\overline{\text{HR}}$ line is deasserted, and a new data transfer to the HRX register is initiated.

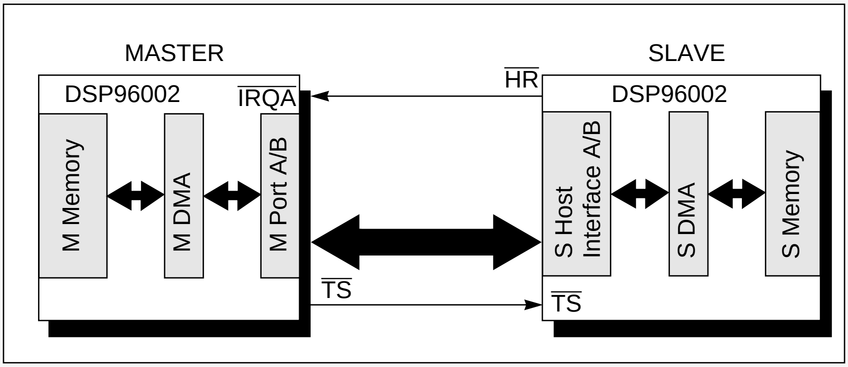


Figure 1-1 *Full Handshake DMA Transfer*

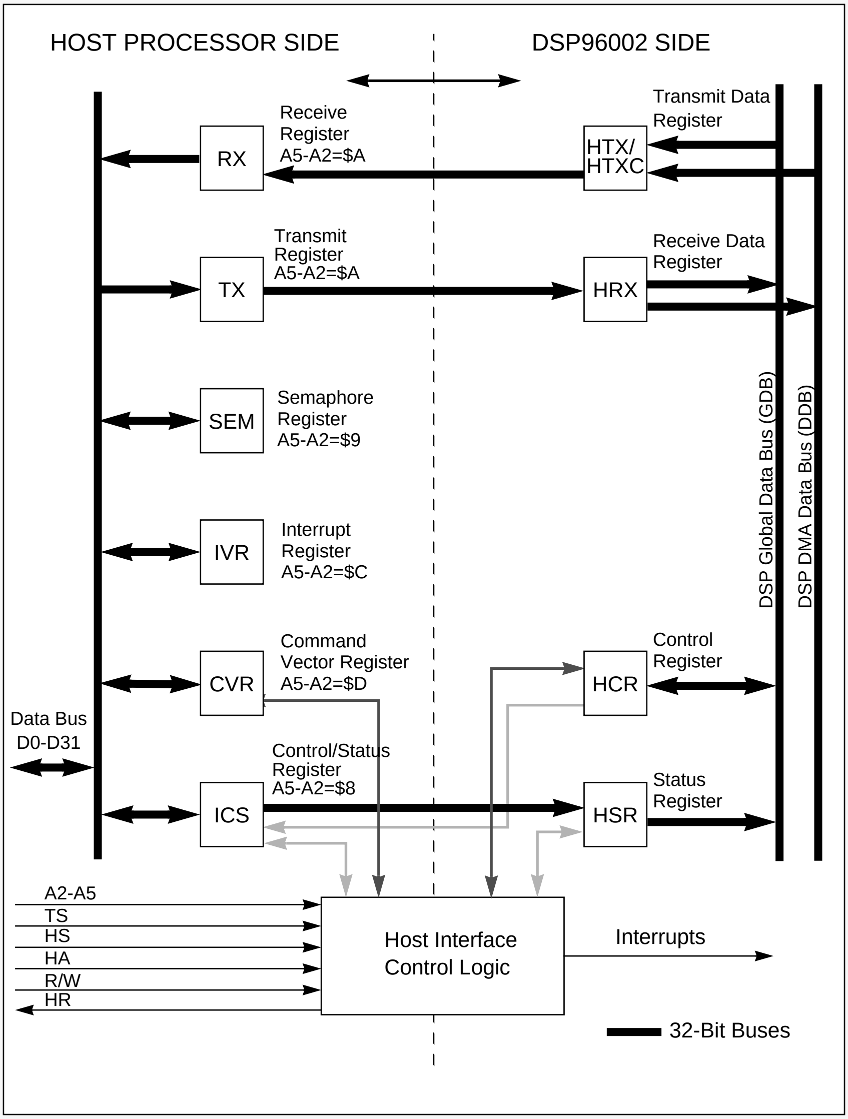


Figure 1-2 *Host Interface Block Diagram*

A DMA request is generated after the status of Host Receive Data Full (HRDF) is updated and data is transferred to HRX register. This DMA request enables the DMA channel to read the HRX register and to write the received data into internal memory. The HRDF status is then cleared and the $\overline{\text{HR}}$ line is asserted. After the $\overline{\text{HR}}$ line is asserted, a new data transfer cycle is performed. If the DMA channel interrupt enable line has been set, an interrupt is generated at the end of DMA transfer.

1.2.2 Interconnection Model

The data bus is common to both processors. The Master Processor is configured in master mode, i.e., Bus Grant ($\overline{\text{BG}}$) is connected to “0”. Also, Transfer Acknowledge ($\overline{\text{TA}}$) is connected to “0” which means that the Master Processor will always receive an automatic data acknowledge so that no wait states will be inserted.

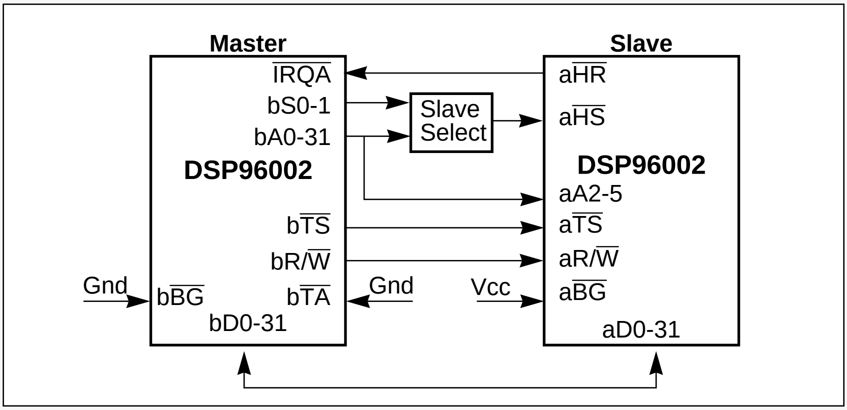


Figure 1-3 *Full Handshake DMA Interconnection Model*

The Master Processor address bus (A5-A31) and Space Select (S0-S1) lines are decoded to generate $\overline{\text{HS}}$ to select the Slave Processor Host Interface. The A2-A5 lines are used to select the Slave Processor Host Interface Register. The Master Processor R/$\overline{\text{W}}$ line is connected to the Slave Processor R/$\overline{\text{W}}$ line and is used to signal a Read or Write transfer. The Master processor Transfer Strobe ($\overline{\text{TS}}$) is connected to the Slave Processor $\overline{\text{TS}}$; $\overline{\text{TS}}$ is asserted when a bus write or a bus read is taking place. The Slave Processor Bus Grant ($\overline{\text{BG}}$) signal is connected to “1”, placing the Slave Processor in the Bus Slave mode. A common clock is ***not*** necessary in this configuration and no special timing precautions need to be considered.

1.2.3 Programming Consideration for Full Handshake DMA Transfer

1.2.3.1 Master Processor DMA Programming

One of the Master Processor’s two DMA channels is programmed to read data from internal memory and to write this data to the Slave Processor Host Interface Transmit Data Register.

The DMA Source Address Register receives the beginning address of the internal memory data block. The DMA Source Modifier Register and the DMA Source Offset Register are programmed according to the data organization.

The DMA Destination Address Register is programmed with the Slave Processor Host Interface Transmit Data Register address which is the Slave

Select Address + $28. The DMA Destination Modifier Register is programmed for linear increment, the DMA Destination Offset is programmed with “0”, and the destination pointer is not incremented. The DMA destination counter register is programmed with the data block length.

The DMA Control Status Register is programmed as follows:

- *DMA Enable is set to start the DMA transfer*
- *DMA Source Space Control (DSS) is set for transfer from internal X or Y memory*
- *DMA Destination Space Control (DDS) will be set for transfer to external X or Y memory*
- *If an interrupt is requested at the end of transfer, the DIE bit should be set and the Interrupt Priority Register must be initialized to receive Interrupt Requests from the DMA channel*
- *The DMA request source is the* $\overline{\textit{IRQA}}$ *line so the DMA Request Mask bits should be configured as M1-M6=0 and M0=1*
- *The DMA Transfer Mode must be programmed as “Single Block, Word Transfer Triggered by DMA Request”, where the DMA Request in this case is the* $\overline{\textit{IRQA}}$ *line*
- *No special caution needs to be taken regarding the DMA/Core Priority bit, DMAP*
- *Also, it is not necessary for this DMA channel to have a higher priority than the second internal DMA channel*

1.2.3.2 Master Processor Port Programming

The Port Select Register allocates the address range for each port. The port allocated address must permit selection of the Slave Processor Host Interface.

No special precautions need to be considered when programming the Bus Control Register; however, introducing wait states slows data transfer.

1.2.3.3 Slave Processor Host Interface Programming

If the Host Interface is used only for DMA, no special programming is necessary after reset. The Master Processor only needs to write data to the Host Interface TX register.

1.2.3.4 Slave Processor DMA Channel Programming

One of the Slave Processor's two DMA channels is programmed to read data from the Host Interface HRX register and to write data to internal memory.

The DMA Source Address Register is programmed with the Host Interface RX register address. The DMA Source Modifier Register and the DMA Source Offset Register are programmed so that the DMA Source Address Register remains constant, which means the DMA Source Offset Register is cleared.

The DMA Destination Address Register receives the internal memory data block beginning address. The DMA Destination Modifier Register and the

DMA Destination Offset Register is programmed according to the data organization. The DMA destination counter register is programmed with the data block length.

The DMA Control Status Register is programmed as follows:

- *DMA Enable is set to start the DMA transfer.*
- *DMA Destination Space Control (DDS) is set for transfer to internal X or Y memory.*
- *DMA Source Space Control (DDS) is set for transfer from the Host Interface HRX register. If an interrupt is requested at the end of the transfer, the DIE bit should be set and the Interrupt Priority Register must be initialized to receive Interrupt Requests from the DMA channel.*
- *The DMA request source is the HRDF status so the DMA Request Mask bits are M0-M6=$8.*
- *The DMA Transfer Mode needs to be programmed as “Single Block, Word Transfer, Triggered by DMA Request”, where DMA Request, in this case, is the HRDF status.*

1.2.4 Timing Diagram of Full Handshake DMA Transfer

We assume that both DSP96002 processors work with the same clock although this is not essential. Each DSP96002 system clock is composed of four phases, or two clock periods. If the two processors work with different clocks then, generally, the transfers are longer due to synchronization delays.

The $\overline{\text{HR}}$ line is asserted on phase T1 of the clock *after* the Slave Processor DMA is initialized and HRX is empty (therefore the Host Interface TX Register is empty). $\overline{\text{IRQA}}$, which is connected to $\overline{\text{HR}}$, is asserted at the same time.

The DMA request is sampled on T1 and a Master DMA transfer is then started. The DMA controller generates a valid DMA address on the first T0 after $\overline{\text{IRQA}}$ is asserted and new data is read from internal memory. After four more phases (i.e., on the next T0), the DMA data bus receives valid data from internal memory, and starts an external memory write cycle. The $\overline{\text{TS}}$ line is asserted on phase T1, and when $\overline{\text{TS}}$ is deasserted, the data is written to the Slave Processor Host Interface TX register on T3.

The TXDE bit is cleared on the first T0 after the Slave Processor TX register is written. This deasserts the $\overline{\text{HR}}$ signal. On phase T3 of that cycle, the internal Host Interface signal HldHRX is asserted for one phase, and data is transferred from the TX register to the Host Interface HRX register. After this transfer, the TXDE status is set and $\overline{\text{HR}}$ is asserted, initiating the next full handshake transfer. When $\overline{\text{HR}}$ is asserted, a valid DMA address is written to the DMA address bus following T0. During the next T0, the new data is read from the HRX register to the DMA bus. The HRDF Status is cleared after data is transferred to the DMA data bus from the HRX. The minimum cycle data transfer length is 8 clocks or 4 instruction cycles.

A listing of the programming model for the full handshake DMA transfer is provided in **SECTION 1.5**.

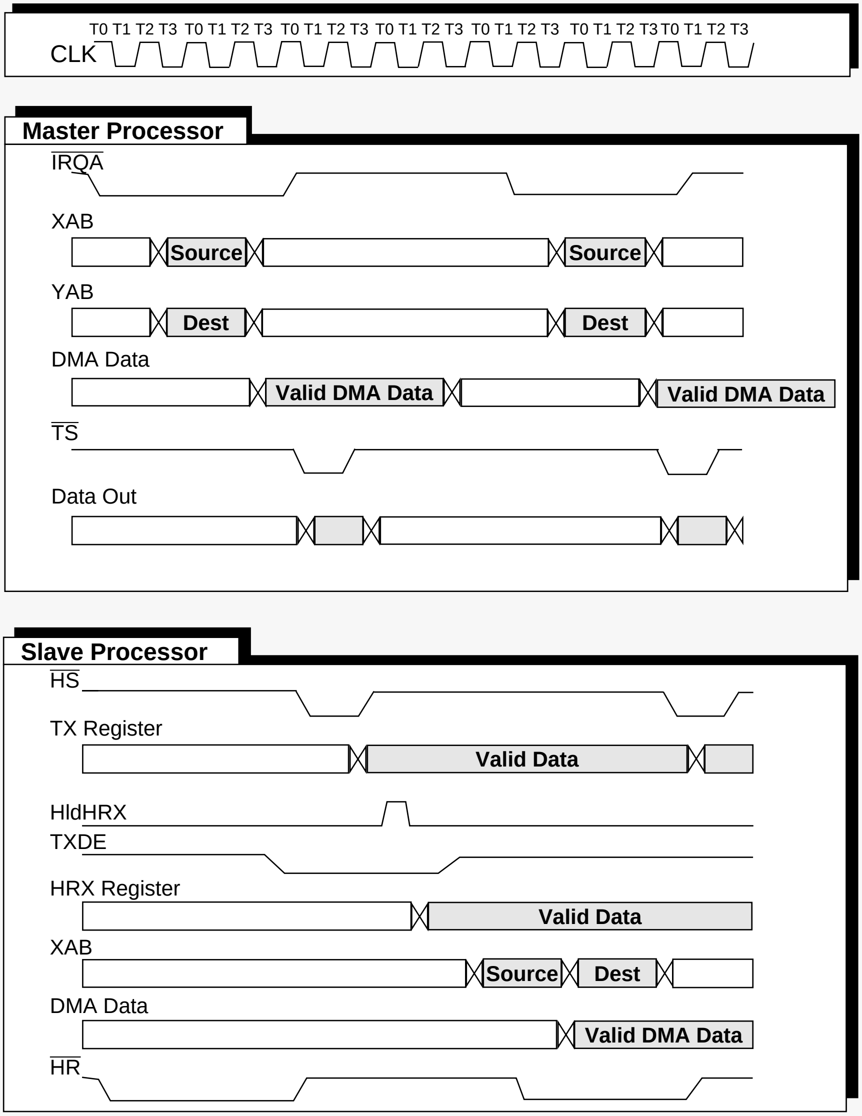


Figure 1-4 *Timing Diagram of Full Handshake DMA Transfer*

1.3 Partial Handshake DMA Transfer

1.3.1 Description

The purpose of this implementation is to provide a faster transfer between two DSP96002 processors. One of the processors is configured as the Master and the other one as the Slave, with the data transfer from the Master to the Slave. The advantage of this implementation is the high transfer rate, one 32-bit transfer every four clocks.

The substantial speed improvement is achieved through a pipeline-type transfer. The Master Processor receives a DMA request through $\overline{\text{IRQA}}$; the DMA channel is programmed as “Single Block, Word Transfer, Triggered by DMA request”. The Master Processor DMA channel initiates a data transfer from Master Processor internal memory to the Slave Processor Host Interface TX register. Data is placed in the Slave Processor TX register and from there, in the HRX register. The Slave Processor DMA channel is programmed as “Single Block, Word Transfer, Triggered by DMA request”, where the DMA request is $\overline{\text{IRQA}}$. Because the Master Processor $\overline{\text{TS}}$ line is connected to the Slave Processor $\overline{\text{IRQA}}$ line, a DMA transfer is initiated from the HRX register to internal memory. Meanwhile, the Master Processor can start a new DMA transfer cycle. Before new data is placed in the Slave TX register, the Slave DMA channel transfers the contents of the HRX register into internal memory.

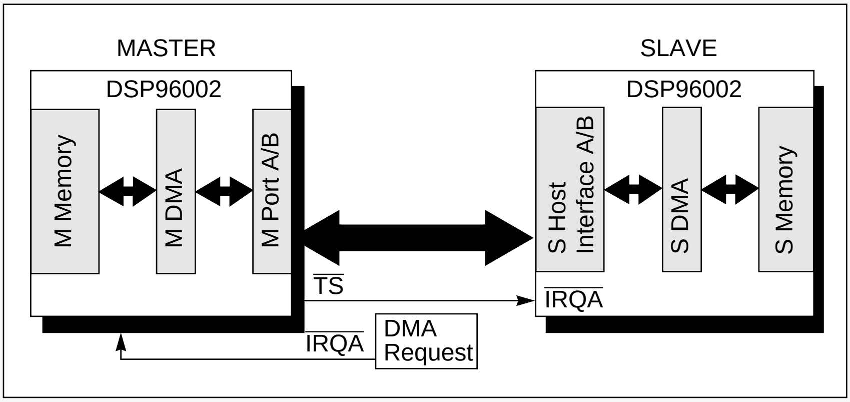


Figure 1-5 *Partial Handshake DMA Transfer*

1.3.2 Interconnection Model

The data bus is common to both processors. The Master Processor is configured in the master mode – i.e., Bus Grant ($\overline{BG}$) signal is connected to “0”. Also, Transfer Acknowledge ($\overline{TA}$) is connected to “0” which means that the Master Processor always receives automatic data acknowledge so that no wait states are inserted. The Master Processor address bus (A5-A31) and Space Select (S0-S1) lines are decoded to generate the $\overline{HS}$ signal to select the Slave Processor Host Interface. The A2-A5 lines are used to select the TX register from Slave Processor Host Interface. The Master Processor R/$\overline{W}$ line is connected to Slave Processor R/$\overline{W}$ line and is used to signal a Read or Write action.

The Master processor Transfer Strobe ($\overline{TS}$) is connected to the Slave Processor $\overline{\text{TS}}$; a bus write or a bus read takes place when $\overline{\text{TS}}$ is asserted. The Slave Processor Bus Grant ($\overline{\text{BG}}$) signal is connected to “1”, placing it in the Slave mode. The Master Processor $\overline{\text{IRQA}}$ is connected to an external DMA request source which can generate a maximum of one request every 2 instruction cycles.

The Slave Processor $\overline{\text{IRQA}}$ is connected to $\overline{\text{TS}}$ of Master Processor.

A common clock ***is*** necessary in this configuration, and no wait states are permitted.

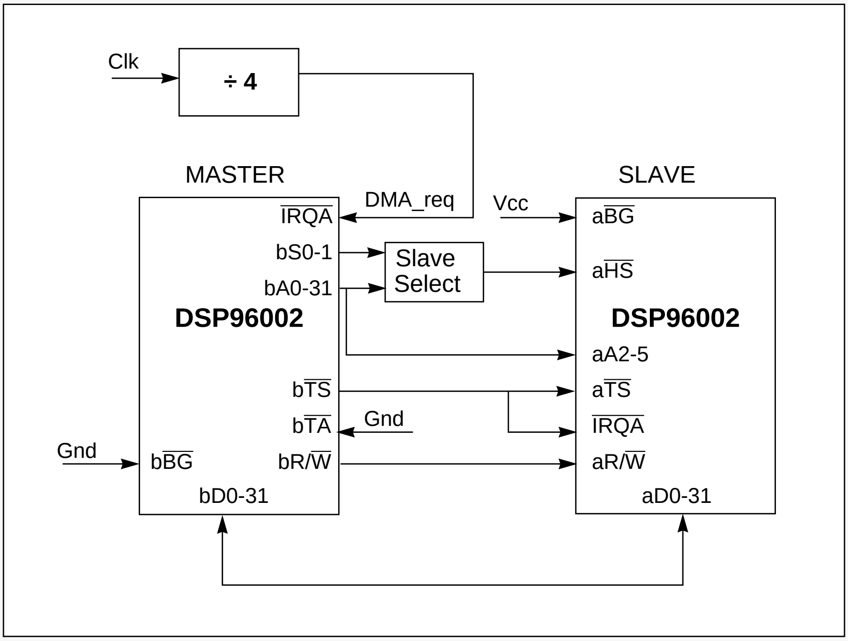


Figure 1-6 *Partial Handshake DMA Transfer Interconnection Model*

1.3.3 Programming Consideration for Partial Handshake DMA Transfer

1.3.3.1 Master Processor DMA Programming

The same rules as for the Full Handshake Programming model must be followed. It is recommended that this DMA channel be given a higher interrupt priority level than the core processor.

1.3.3.2 Master Processor Port Programming

The Port Select Register allocates the address range for each port. The port allocated address must permit selection of the Slave Processor Host Interface. No Wait States are permitted for this configuration.

1.3.3.3 Slave Processor Host Interface Programming

If the Host Interface is used only for this DMA transfer, no special programming is necessary after reset. The Master Processor only needs to write data to the Host Interface TX register.

1.3.3.4 Slave Processor DMA Channel Programming

The same rules as for the Full Handshake Programming model must be followed.

- *The DMA request source is the $\overline{IRQA}$ line so the DMA Request Mask bits are M0-M6=1.*
- *The DMA Transfer Mode must be programmed as "Single Block, Word Transfer, Triggered by DMA Request", where the DMA Request, in this case, is the $\overline{IRQA}$ line.*

1.3.4 Timing Diagram of Partial Handshake DMA Transfer

Both DSP96002s have to work with the same clock for this configuration. After both processors have been initialized, the Master Processor receives a DMA request through the $\overline{\mathrm{IRQA}}$ line. The DMA request is recognized on T1, and a new Master Processor DMA cycle starts. On the first T0 phase after the DMA request has been recognized, the DMA source address is placed on the XAB and the DMA destination address is placed on the YAB because the transfer type is from internal memory to external memory. The new data is read from internal memory, placed on the DMA data bus, and then transferred to the Slave Processor ($\overline{\mathrm{TS}}$ deasserted) on the next T3 phase. A new DMA request can be recognized and a new DMA transfer can be performed on the next T1 after $\overline{\mathrm{TS}}$ is deasserted.

The deassertion of $\overline{\mathrm{TS}}$ writes new data in the Slave Processor TX register on the Slave Processor side. The Slave Processor DMA channel is programmed to transfer data from the HRX register to the X internal memory. The DMA channel request is recognized on the first T1 after $\overline{\mathrm{IRQA}}$ deassertion. On the first T0 after the request, the DMA controller places the source address on the XAB bus, and the destination address on the following T0. Data is transferred from the TX register to the HRX register on phase T3. Data from the HRX register is then transferred to the DMA data bus on the next T0. During this T0, the Slave Processor DMA channel is also ready for a new transfer.

The minimum cycle data transfer length in this implementation is 4 clocks. A listing of the programming model for the partial handshake DMA transfer is shown in **SECTION 5.2**.

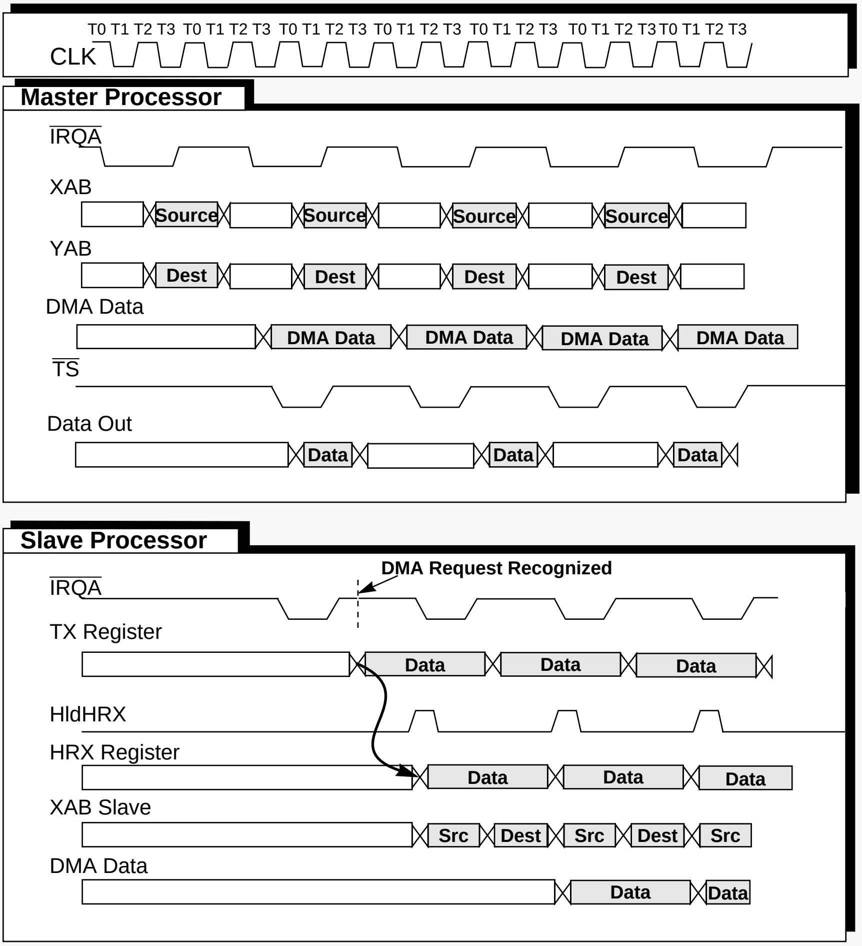


Figure 1-7 *Timing Diagram for Partial Handshake DMA Transfer*

1.4 No Handshake DMA Transfer

1.4.1 Description

This technique is a pipelined type of transfer as in the partial handshake DMA transfer; however, in this case, both processors are configured in Master Mode. One processor transmits data and the second processor receives the data.

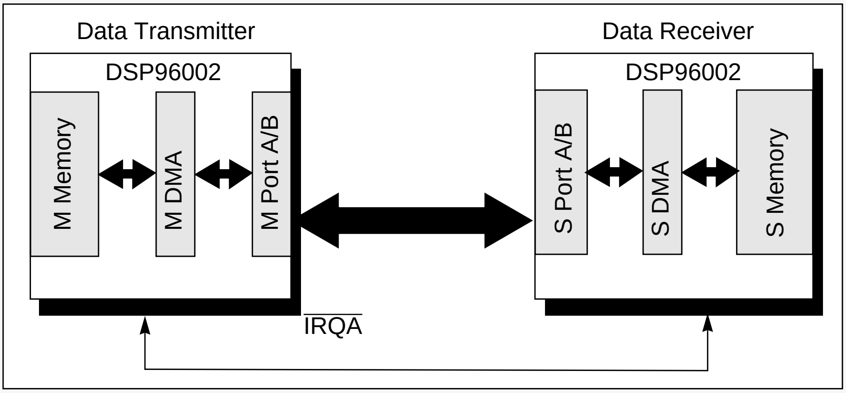


Figure 1-8 *Configuration for No Handshake DMA Transfer*

Two DMA channels are used in Figure 1-8, one on each processor. Both processors are configured as Bus Masters and the only interconnection between them is through the data bus. The Data Transmitter DMA is configured as “Single Block, Word Transfer, Triggered by DMA request”, with the transfer direction from internal X memory to external Y memory.

The Data Receiver DMA is also configured as “Single Block, Word Transfer, Triggered by DMA request”, but with the transfer direction from external Y memory to internal X memory. Both DMAs use the same trigger and both interfaces are programmed with zero wait states.

1.4.2 Interconnection Model

The data bus is common to both processors. The Master Processor is configured in master mode – i.e., the Bus Grant ($\overline{\text{BG}}$) signal is connected to “0”. Also, the Transfer Acknowledge ($\overline{\text{TA}}$) signal is connected to “0” which means that the Master Processor always receives an automatic data acknowledge so that no wait states will be inserted. The Master Processor and Slave Processor do not have the address lines, R/$\overline{\text{W}}$ lines, or $\overline{\text{TS}}$ lines connected. The Slave Processor and Master Processor Bus Grant ($\overline{\text{BG}}$) lines are connected to “0”.

The Master Processor $\overline{\text{IRQA}}$ is connected to an external DMA request source which can generate a maximum of one request every instruction cycle.The Slave Processor $\overline{\text{IRQA}}$ is connected to the same DMA request source.

It is recommended that the Slave Processor $\overline{\text{DE}}$ line not be asserted to avoid potential data bus contention.

A common clock ***is*** necessary in this configuration and no wait states are permitted.

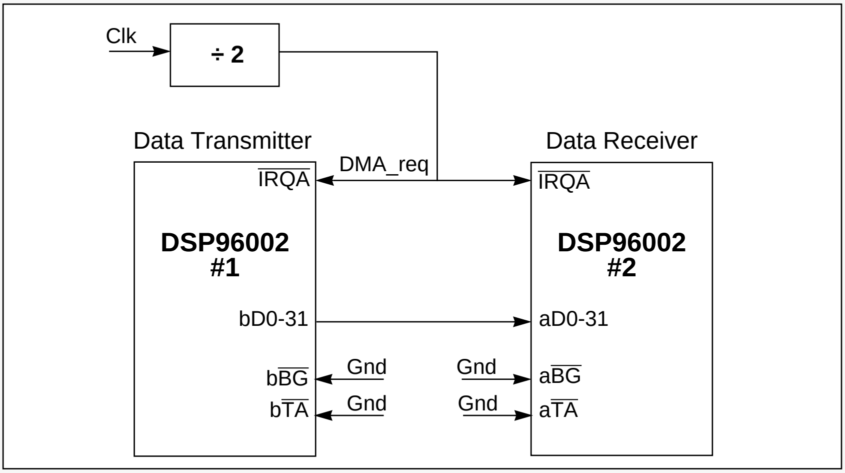


Figure 1-9 *No Handshake DMA Transfer Interconnection Model*

1.4.3 Programming Considerations for No Handshake DMA Transfer

1.4.3.1 Master Processor DMA Programming

The same rules as for the Full Handshake Programming model must be followed. It is recommended that this DMA channel be given a higher interrupt priority level than the core processor or the second on-chip DMA channel.

1.4.3.2 Master Processor Port Programming

The Port Select Register allocates the address range for each port. The port allocated address must permit selection of the Slave Processor Host Interface. No Wait States are permitted for this configuration.

1.4.3.3 Slave Processor Host Interface Programming

The Slave Processor Host Interface is not used for this type of transfer.

1.4.3.4 Slave Processor DMA Channel Programming

The same rules as for Full Handshake Programming model must be followed.

- *The Source Address must be external X or Y memory and the destination address must be Y or X memory.*
- *The DMA request source is the $\overline{IRQA}$ line so the DMA Request Mask bits are M0-M6=1.*
- *The DMA Transfer Mode has to be programmed as "Single Block, Word Transfer, Triggered by DMA Request", where DMA Request in this case is the $\overline{IRQA}$ line.*

1.4.3.5 Slave Processor Port Programming

The Port Select Register allocates the address range for each port. The port selected allows the DMA channel to read data from the connected data bus. No Wait States are permitted for this configuration.

1.4.4 Timing Diagram of No Handshake DMA Transfer

Both DSP96002s work with the same clock in this configuration. After the DMA channels of both processors have been initialized, a DMA request is applied simultaneously to all DMA channels. On the first T1 phase after $\overline{\text{IRQA}}$ is asserted, the DMA request is recognized and a DMA transfer cycle is started. On the next T0 phase after $\overline{\text{IRQA}}$ is recognized, the DMA source address is placed on the XAB internal bus and the DMA destination address is placed on the YAB internal bus. Data is read from internal memory and placed on the DMA data bus on the next T3 phase after the source and destination addresses are placed on XAB and YAB. The valid data is written on the output data pins on phase T2 after data becomes valid. A pipeline type transfer starts if a new DMA request is placed four phases after the first request. If the second processor receives the DMA request with the same timing as the first processor, the data is transferred to the second processor. The Slave processor recognizes the DMA request on the first T1. The source and destination addresses are placed on XAB and YAB on the next T0. Subsequently, the appropriate address is placed on the external address bus on the following T0. The data on the data bus is sampled at the transition from T2 to T3, and the new read data is valid on the internal DMA data bus beginning on T3.

If the DMA request signals are identical for Processor #1 and Processor #2, data written by Processor #1 is valid on the third T2 phase after the DMA request is recognized. Processor #2 reads valid data on the third T3 phase after $\overline{\text{IRQA}}$ is recognized.

This pipelined type of transfer offers the possibility of a transfer every 4 phases (i.e., every two clock periods or each instruction cycle).

A listing of the programming model for the no handshake DMA transfer is provided in **SECTION 1.5.3.**

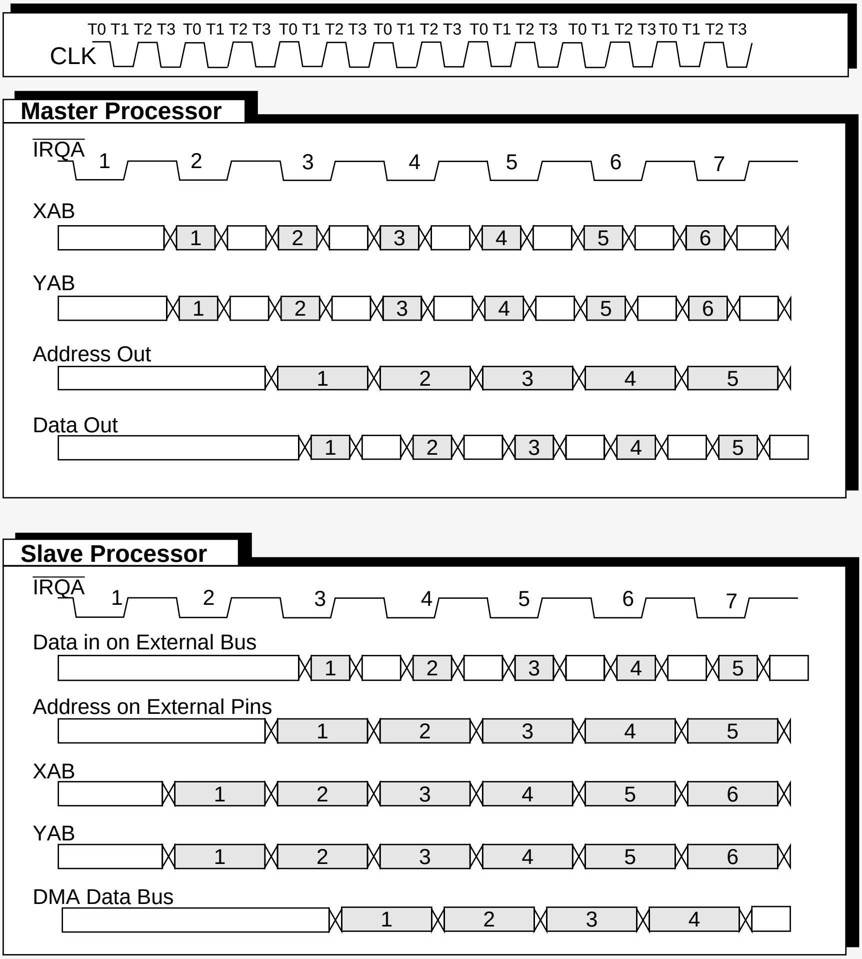


Figure 1-10 *No Handshake DMA Transfer*

1.5 Programming Examples

1.5.1 Programming Model of the Full Handshake DMA Transfer

```
IPR     equ      $ffffffff         ; Interrupt Priority Register
BCRB    equ      $fffffffd         ; Bus Control Register port B
PSR     equ      $fffffffc         ; Port Select Register
D1SMR   equ      $ffffffd7         ; DMA1 Source Modifier Register
D1SAR   equ      $ffffffd6         ; DMA1 Source Address Register
D1SOR   equ      $ffffffd5         ; DMA1 Source Offset Register
D1DMR   equ      $ffffffd3         ; DMA1 Destination Modifier Reg
D1DAR   equ      $ffffffd2         ; DMA1 Destination Address Reg
D1DOR   equ      $ffffffd1         ; DMA1 Destination Offset Reg
D1CT    equ      $ffffffd4         ; DMA1 Counter Register
D1CSR   equ      $ffffffd0         ; DMA1 Control Status Register
; Address of Slave’s Port A Host Interface Registers
Slave   equ      $40000000         ; Tx register address on the Slave
TX      equ      Slave+$28         ; DMA1 Control Status Register
; Init Procedure for Master Processor
movep   #$000C0000,X:IPR           ; Enable interrupts from DMA1
                                   ; The DMA1 channel will generate
                                   ; interrupts on level 2 for DMA
                                   ; transfer completed, if DIE=1
movep   #$00000000,x:BCRB          ; Port B Bus has no wait states
movep   #$000F0F00,x:PSR           ; All X addresses 0…$80000000
                                   ; All Y addresses 0…$80000000
                                   ; will be through port B
; Initialize the DMA Procedure for the Master Processor
; Program DMA1 source
movep   #$1ff,D1SMR                ; DMA1 source modifier is
                                   ; programmed in linear modulo
                                   ; addressing mode
movep   #$1,D1SOR                  ; DMA1 source address offset is 1
movep   #0,D1SAR                   ; DMA1 source address
                                   ; internal X memory
movep   #0,D1DOR                   ; DMA1 destination offset is 0
movep   #0,D1DMR                   ; DMA1 destination address inc
movep   #TX,D1DAR                  ; DMA1 destination address
                                   ; Slave Processor TX Register.
movep   #$100,D1CT                 ; DMA1 counter
movep   #$C400010E,D1CSR           ; Load DMA1 control status Reg
                                   ; DE=1 DMA1 enable
                                   ; DIE=1 end of transfer interrupt
                                   ; DTM1,0=10 Single Block,
                                   ; Word Transfer Triggered by IRQA
                                   ; DSS=001 from internal X Memory
                                   ; DDS=110 Destination External Y
                                   ; memory
```

Figure 1-11 *DMA Programming Procedure for the* ***Master*** *Processor in the Full Handshake*

```
IPR      equ      $ffffffff          ; Interrupt Priority Register
BCRA     equ      $fffffffe          ; Bus Control Register port A
PSR      equ      $fffffffc          ; Port Select Register
D1SMR    equ      $ffffffd7          ; DMA1 Source Modifier Register
D1SAR    equ      $ffffffd6          ; DMA1 Source Address Register
D1SOR    equ      $ffffffd5          ; DMA1 Source Offset Register
D1DMR    equ      $ffffffd3          ; DMA1 Destination Modifier Reg
D1DAR    equ      $ffffffd2          ; DMA1 Destination Address Reg
D1DOR    equ      $ffffffd1          ; DMA1 Destination Offset Reg
D1CT     equ      $ffffffd4          ; DMA1 Counter
D1CSR    equ      $ffffffd0          ; DMA1 Control Status Register
; Initialization Procedure for Slave Processor
movep    #$000C0000,X:IPR            ; Enable interrupts from DMA1
                                     ; channel
                                     ; The channel will generate
                                     ; interrupts on level 2
                                     ; for DMA transfer completed, if
                                     ; DIE = 1
movep    #$00000000,x:BCRA           ; Port A Bus has no wait states
movep    #$00F0F000,x:PSR            ; All X addresses 0…$80000000
                                     ; All Y addresses 0…$80000000
                                     ; will be through port A
; Initialize DMA Procedure for Slave Processor
; Program DMA1 source
movep#-1,D1SMR                       ; DMA1 source modifier is
                                     ; programmed
                                     ; in linear addressing
movep    #0,D1SOR                    ; DMA1 source address offset is 0
movep    #HRX,D1SAR                  ; DMA1 source address is Host Rx
                                     ; Register
;Program DMA1 destination
movep    #$1ff,D1DMR                 ; DMA1 destination modifier is
                                     ; programmed in linear increment
                                     ; modulo
movep    #1,D1DOR                    ; DMA1 destination offset is 1
movep    #0,D1DAR                    ; DMA1 destination address
movep    #$100,DMA1CT                ; DMA1 counter
movep    #$C4000819,DMA1CSR;         ; Load DMA1 control status Reg
                                     ; DE=1 DMA1 enable
                                     ; DIE=1 end of transfer interrupt
                                     ; DTM1,0=10 Single Block,
                                     ; Word Transfer Triggered by HRDF
                                     ; DSS=011 from internal HRX Reg
                                     ; DDS=001 Destination Internal X
                                     ; memory
```

Figure 1-12 *DMA Programming Procedure for the* ***Slave*** *Processor in the Full Handshake*

1.5.2 Programming Model for Partial Handshake DMA Transfer

```
IPR      equ      $ffffffff          ; Interrupt Priority Register
BCRB     equ      $fffffffd          ; Bus Control Register port B
PSR      equ      $fffffffc          ; Port Select Register
D1SMR    equ      $ffffffd7          ; DMA1 Source Modifier Register
D1SAR    equ      $ffffffd6          ; DMA1 Source Address Register
D1SOR    equ      $ffffffd5          ; DMA1 Source Offset Register
D1DMR    equ      $ffffffd3          ; DMA1 Destination Modifier Reg
D1DAR    equ      $ffffffd2          ; DMA1 Destination Address Reg
D1DOR    equ      $ffffffd1          ; DMA1 Destination Offset Reg
D1CT     equ      $ffffffd4          ; DMA1 Counter
D1CSR    equ      $ffffffd0          ; DMA1 Control Status Register
; Address of Slave’s Port A Host Interface Registers
Slave    equ      $40000000          ; Slave Processor Host Interface
TX       equ      Slave+$28          ; Address, TX register
; Initialization Procedure for Master Processor
movep    #$000C0000,X:IPR            ; Enable interrupts from DMA1
                                     ; channel
                                     ; The channel will generate
                                     ; interrupts on level 2
                                     ; for DMA transfer completed,
                                     ; if DIE = 1
movep    #$00000000,x:BCRB           ; Port B Bus has no wait states
movep    #$000F0F00,x:PSR            ; All X addresses 0…$80000000
                                     ; All Y addresses 0…$80000000
                                     ; will be through port B
; Initialize the DMA Procedure for the Master Processor
; Program DMA1 source
movep    #$1ff,D1SMR                 ; DMA1 source modifier is
                                     ; programmed
                                     ; in linear modulo addressing
movep    #$1,D1SOR                   ; DMA1 source address offset is 1
movep    #0,D1SAR                    ; DMA1 source address
                                     ; Program DMA1 destination
movep#-1,D1DMR                       ; DMA1 destination modifier is
                                     ; programmed
                                     ; in linear increment
movep    #0,D1DOR                    ; DMA1 destination offset is 0
movep    #TX,D1DAR                   ; DMA1 destination address
movep    #$100,D1CT                  ; DMA1 counter
movep    #$C4000115,D1CSR            ; Load DMA1 control status Reg
                                     ; DE=1 DMA1 enable
                                     ; DIE=1 for end of transfer
                                     ; interrupt
                                     ; DTM1,0=10 Single Block, Word
                                     ; Transfer Triggered by DMA Req
                                     ; M=1 DMA request from IRQA
                                     ; DSS=010 for source internal Y
                                     ; DDS=101 for external X
                                     ; memory destination
```

Figure 1-13 *DMA Programming Procedure for the* ***Master*** *Processor in the Partial Handshake*

```
IPR     equ      $ffffffff          ; Interrupt Priority Register
BCRA    equu     $fffffffe          ; Bus Control Register port A
PSR     equ      $fffffffc          ; Port Select Register
D1SMR   equ      $ffffffd7          ; DMA1 Source Modifier Register
D1SAR   equ      $ffffffd6          ; DMA1 Source Address Register
D1SOR   equ      $ffffffd5          ; DMA1 Source Offset Register
D1DMR   equ      $ffffffd3          ; DMA1 Destination Modifier Reg
D1DAR   equ      $ffffffd2          ; DMA1 Destination Address Reg
D1DOR   equ      $ffffffd1          ; DMA1 Destination Offset Reg
D1CT    equ      $ffffffd4          ; DMA1 Counter
D1CSR   equ      $ffffffd0          ; DMA1 Control Status Register
; Initialization Procedure for Slave Processor
movep   #$000C0000,X:IPR            ; Enable interrupts from DMA1
                                    ; channel
                                    ; The channel will generate
                                    ; interrupts on level 2
                                    ; for DMA transfer completed,
                                    ; if DIE = 1
movep   #$00000000,x:BCRA           ; Port A Bus has no wait states
movep   #$00F0F000,x:PSR            ; All X addresses 0…$80000000
                                    ; All Y addresses 0…$80000000
                                    ; will be through port A
; Initialize the DMA Procedure for Slave Processor
; Program DMA1 source
movep   #-1,D1SMR                   ; DMA1 source modifier is
                                    ; programmed in linear addressing
movep   #0,D1SOR                    ; DMA1 source address offset is 0
movep   #HRX,D1SAR                  ; DMA1 source address is Host Rx
;Program DMA1 destination
movep   #$1ff,D1DMR                 ; DMA1 destination modifier is
                                    ; programmed in linear modulo
                                    ; increment
movep   #1,D1DOR                    ; DMA1 destination offset is 1
movep   #0,D1DAR                    ; DMA1 destination address
movep   #$100,D1CT                  ; DMA1 counter
movep   #$C4000119,D1CSR            ; Load DMA1 control status Reg
                                    ; DE=1 DMA1 enable
                                    ; DIE=1 for end of transfer
                                    ; interrupt
                                    ; DTM1,0=10 Single Block, Word
                                    ; Transfer Triggered by DMA Req
                                    ; by DMA request
                                    ; M=1 DMA request from IRQA
                                    ; DSS=010 for internal HRX Reg
                                    ; DDS=001 Destination Internal X
                                    ; memory
```

Figure 1-14 *DMA Programming Procedure for the **Slave** Processor in the Partial Handshake*

1.5.3 Programming Model for No Handshake DMA Transfer

```
IPR     equ      $ffffffff          ; Interrupt Priority Register
BCRB    equ      $fffffffd          ; Bus Control Register port B
PSR     equ      $fffffffc          ; Port Select Register
D1SMR   equ      $ffffffd7          ; DMA1 Source Modifier Register
D1SAR   equ      $ffffffd6          ; DMA1 Source Address Register
D1SOR   equ      $ffffffd5          ; DMA1 Source Offset Register
D1DMR   equ      $ffffffd3          ; DMA1 Destination Modifier Reg
D1DAR   equ      $ffffffd2          ; DMA1 Destination Address Reg
D1DOR   equ      $ffffffd1          ; DMA1 Destination Offset Reg
D1CT    equ      $ffffffd4          ; DMA1 Counter
D1CSR   equ      $ffffffd0          ; DMA1 Control Status Register
; Address of Slave’s Port A Host Interface Registers
Slave   equ      $40000000          ; Slave Processor
; Initialization Procedure for Master Processor
movep   #$000C0000,X:IPR            ; Enable interrupts from DMA1
                                    ; channel
                                    ; The channel will generate
                                    ; interrupts on level 2
                                    ; for DMA transfer completed,
                                    ; if DIE = 1
movep   #$00000000,x:BCRB           ; Port B Bus has no wait states
movep   #$000F0F00,x:PSR            ; All X addresses 0…$80000000
                                    ; All Y addresses 0…$80000000
                                    ; will be through port B
; Initialize the DMA Procedure for the Master Processor
                                    ; Program DMA1 source
movep   #$1ff,D1SMR                 ; DMA1 source modifier is
                                    ; programmed
                                    ; in linear modulo addressing
movep   #$1,D1SOR                   ; DMA1 source address offset is 1
movep   #0,D1SAR                    ; DMA1 source address
;Program DMA1 destination
movep   #-1,D1DMR                   ; DMA1 destination modifier is
                                    ; programmed
                                    ; in linear increment
movep   #0,D1DOR                    ; DMA1 destination offset is 0
movep   #Slave,D1DAR                ; DMA1 destination address
movep   #$100,D1CT                  ; DMA1 counter
movep   #$C4000115,D1CSR;           ; Load DMA1 control status Reg
                                    ; DE=1 DMA1 enable
                                    ; DIE=1 for end of transfer
                                    ; interrupt
                                    ; DTM1,0=10 Single Block, Word
                                    ; Transfer Triggered by DMA Req
                                    ; M=1 DMA request from IRQA
                                    ; DSS=010 for source internal Y
                                    ; DDS=101 to external X
                                    ; memory destination
```

Figure 1-15 *DMA Programming Procedure for the **Master** Processor in the No Handshake Transfer*

```
IPR    equu     $ffffffff          ; Interrupt Priority Register
```

```
IPR    equ      $ffffffff          ; Interrupt Priority Register
BCRA   equ      $fffffffe          ; Bus Control Register port A
PSR    equ      $fffffffc          ; Port Select Register
D1SMR  equ      $ffffffd7          ; DMA1 Source Modifier Register
D1SAR  equ      $ffffffd6          ; DMA1 Source Address Register
D1SOR  equ      $ffffffd5          ; DMA1 Source Offset Register
D1DMR  equ      $ffffffd3          ; DMA1 Destination Modifier Reg
D1DAR  equ      $ffffffd2          ; DMA1 Destination Address Reg
D1DOR  equ      $ffffffd1          ; DMA1 Destination Offset Reg
D1CT   equ      $ffffffd4          ; DMA1 Counter
D1CSR  equ      $ffffffd0          ; DMA1 Control Status Register
; Initialization Procedure for Slave Processor
movep  #$000C0000,X:IPR            ; Enable interrupts from DMA1
                                   ; channel
                                   ; The channel will generate
                                   ; interrupts on level 2
                                   ; for DMA transfer completed,
                                   ; if DIE = 1
movep  #$00000000,x:BCRA           ; Port A Bus has no wait states
movep  #$00F0F000,x:PSR            ; All X addresses 0…$80000000
                                   ; All Y addresses 0…$80000000
                                   ; will be through port A
; Initialize the DMA Procedure for Slave Processor
; Program DMA1 source
movep  #-1,D1SMR                   ; DMA1 source modifier is
                                   ; programmed in linear addressing
movep  #0,D1SOR                    ; DMA1 source address offset is 0
movep  #Master,D1SAR               ; DMA1 source address is external
                                   ; Master Processor.
                                   ; Data is read from the Master
                                   ; Processor as from an external
                                   ; memory.
;Program DMA1 destination
movep  #$1ff,D1DMR                 ; DMA1 destination modifier is
                                   ; programmed in linear modulo
                                   ; increment
movep  #1,D1DOR                    ; DMA1 destination offset is 1
movep  #0,D1DAR                    ; DMA1 destination address
movep  #$100,D1CT                  ; DMA1 counter
movep  #$C4000131,D1CSR            ; Load DMA1 control status Reg
                                   ; DE=1 DMA1 enable
                                   ; DIE=1 for end of transfer
                                   ; interrupt
                                   ; DTM1,0=10 Single Block, Word
                                   ; Transfer Triggered by DMA Req
                                   ; by DMA request
                                   ; M=1 DMA request from IRQA
                                   ; DSS=110 for external Y memory
                                   ; DDS=001 Destination Internal X
                                   ; memory
```

Figure 1-16 *DMA Programming Procedure for the **Slave** Processor in the No Handshake Transfer*

SECTION 2

Connecting the DSP96002 as an Attached Processor to the ISA Bus Turns PC/AT into a Fast and Powerful IEEE Compatible Floating Point Computer

By Z. Rozenshein

2.1 Introduction

"Compared to the 80386 + 80387 (20 MHz) PC, the DSP96002 (40.0 MHz) runs 48 times faster."

This section describes how to attach the Motorola DSP96002 to the IBM PC/AT bus (ISA BUS) so that the PC can use the DSP96002 as an IEEE floating-point numeric accelerator. A newly designed adapter board equipped with the DSP96002 chip and additional hardware achieves this goal. Numeric tasks that need massive IEEE floating point calculations can be transferred to the DSP96002 and calculated there much faster than with a standard co-processor. This board was assembled and several benchmarks were run to evaluate the performance enhancement factor achieved by using the DSP96002 as a numeric accelerator.

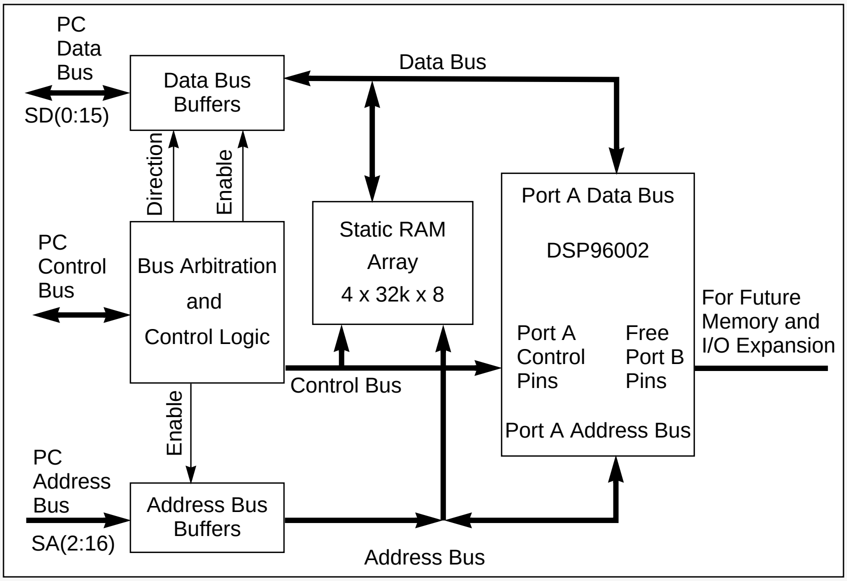


Figure 2-1 *Block Diagram*

2.2 ISA (Industry Standard Architecture) Bus Details

The ISA Bus has eight I/O slots with the following I/O support functions:

- *I/O address space for user defined hardware*
- *24-bit memory addresses (16MB)*
- *Selection of data access size (bytes or words)*
- *I/O and memory wait-state generation*

Adapter boards plug into one of the slots and receive the I/O channel signals.The application board uses the following I/O channel signals:

BALE *This is the ‘Buffered Address Latch Enable’ signal that indicates a valid microprocessor or DMA address.*

SA(0:19) *Address lines 0…19 are used to address memory and I/O within the system. These signals are latched in the PC’s mother-board on the falling edge of BALE.*

LA(17:23) *Address lines 17…23 provide the system with the ability to address up to 16M bytes of memory. These signals are not latched on the PC’s mother-board and are valid when BALE is high.*

RESET_DRV *This signal is used to reset or initialize the system at power-up.*

SD(0:15) *These signals compose the PC’s 16-bit system data bus.*

I/O_CH_RDY *This signal should be pulled low (not ready) by a slow memory or I/O device in order to lengthen I/O or memory cycles.*

$\overline{\textbf{\textit{IOR}}}$ *This is the ‘I/O Read’ signal that indicates that the ISA’s CPU is performing an I/O read cycle.*

$\overline{\textbf{\textit{IOW}}}$ *This is the ‘I/O Write’ signal that indicates that the ISA’s CPU is performing an I/O write cycle.*

$\overline{\textbf{\textit{SMEMR}}}$ *This is the ‘system memory read’ signal that indicates that the ISA’s CPU is performing a memory read cycle from only the low 1M of memory space.*

$\overline{\text{SMEMW}}$ *This is the ‘system memory write’ signal that indicates that the ISA’s CPU is performing a memory write cycle to only the low 1M of memory space.*

AEN *This ‘Address Enable’ signal is active (high) when the system’s DMA controller has control of the channel’s address bus, data bus, read command lines, and write command lines.*

$\overline{\text{SBHE}}$ *The ‘System Bus High Enable’ signal indicates that the ISA’s CPU is executing a data transfer on the upper byte of the data bus, SD(8:15).*

$\overline{\text{MEMCS16}}$ *This ‘memory 16-bit chip select’ input signal indicates that the current cycle is a 16-bit memory cycle.*

$\overline{\text{IOCS16}}$ *This ‘I/O 16-bit chip select’ input signal indicates that the current cycle is a 16-bit I/O cycle.*

For a more complete description of the I/O channel’s signals, see IBM’s ‘Technical Reference for the Personal Computer AT’.

2.3 DSP96002 Features

The DSP96002 is the first member of Motorola’s family of dual-port IEEE floating point programmable CMOS processors. The DSP96002’s main features include the support of IEEE 754 Single Precision and Single Extended Precision Floating-Point with 32-bit signed and unsigned fixed point arithmetic, and two identical external memory expansion ports.

DSP96002 features are:

- *IEEE 754 Standard SP and SEP Arithmetic*
- *20.0 Million Instructions per Second (MIPS) with a 40.0 MHz clock*
- *60 Million Floating Point Instructions per Second (MFLOPS) peak with a 40.0 MHz clock*
- *Single-Cycle 32 x 32-bit Parallel Multiplier*
- *Highly Parallel Instruction Set with Unique DSP Addressing Modes*
- *Nested Hardware Do Loops*
- *Fast Auto-Return Interrupts*
- *2 Independent On-Chip 512 x 32-bit Data RAMs*
- *2 Independent On-Chip 1024 x 32-bit Data ROMs*
- *Off-Chip Expansion to 2 x* 2^{32} *32-bit Words of Data Memory*
- *On-Chip 1024 x 32-bit Program RAM*
- *On-Chip 64 x 32-bit Bootstrap ROM*
- *Off-Chip Expansion to* 2^{32} *32-Bit Words of Program Memory*
- *Two Identical External Memory Expansion Ports*
- *Two 32-Bit Parallel Host MPU/DMA Slave Interfaces*
- *On-Chip Two-Channel DMA controller*
- *On-Chip Emulator (OnCE*^{TM}*)* [1]

The application board uses the following DSP96002 signals:

$\overline{\textbf{\textit{RESET}}}$ *Assertion of this signal places the DSP96002 in the reset state.*

MODA/$\overline{\textbf{\textit{IRQA}}}$ *Mode Select A/External Interrupt Request A. This signal selects the initial DSP96002 operating mode during hardware reset and becomes a maskable interrupt request input during normal instruction processing.*

[1] **OnCE** is a trademark of Motorola, Inc.

MODB/$\overline{\textit{IRQB}}$ *Mode Select B/External Interrupt Request B. This signal selects the initial DSP96002 operating mode during hardware reset and becomes a maskable interrupt request input during normal instruction processing.*

MODC/$\overline{\textit{IRQC}}$ *Mode Select C/External Interrupt Request C. This signal selects the initial DSP96002 operating mode during hardware reset and becomes a maskable interrupt request input during normal instruction processing.*

$\overline{\textit{DR}}$ *Debug Request. This input provides a means of entering the debug mode of operation from the external command converter.*

DSCK/OS1 *Debug Serial Clock/Chip Status 1. When this pin is configured as an input, it provides serial clock to the OnCE$^{TM.}$ When an output, this pin provides chip status information.*

DSI/OS0 *Debug Serial Input/Chip Status 0. This pin can be configured as an input, providing serial data or commands to the OnCE^{TM}. When an output, this pin provides information about the chip status.*

DSO *Debug Serial Output. This pin provides the data contained in the OnCE^{TM} registers to the external command converter*

aA(0:31) *These signals are the 32 Port A address lines.*

aD(0:31) *These signals are the 32 Port A data lines.*

aR/$\overline{\textit{W}}$ *Read/$\overline{\textit{Write}}$ input for Port A. This signal is high for the read cycle and low for the write cycle.*

$a\overline{TS}$ *Transfer Strobe input for Port A. This signal is asserted to indicate that the address and Port A control lines are stable and that a bus read or write is taking place.*

$a\overline{HS}$ *Host Select input for Port A. This signal is asserted to enable selection of the Host Interface functions.*

$a\overline{HA}$ *Host Acknowledge input for Port A. This signal is used to acknowledge a request to the Host Interface.*

$a\overline{BG}$ *Bus Grant input for Port A. This signal is asserted by an external bus arbiter when the DSP96002 may become the next bus master.*

$a\overline{BA}$ *Bus Acknowledge output for Port A. This signal is asserted when the DSP96002 has taken the bus and is the bus master.*

For a more complete description of the DSP96002 and its signals see Motorola's **DSP96002 IEEE Floating-Point Dual-Port Processor User's Manual.**

2.4 Application Board Detailed Description

The application board's main functions are:

- *128K Bytes of RAM, shared by the PC and the DSP96002*
- *DSP96002 is a host processor to the ISA's CPU*
- *Unused DSP port for future I/O and memory expansion (Port B)*

The card achieves these functions when it contains the following sub-blocks:

2.4.1 CPU

This sub-block contains the DSP96002 using only its Port A pins, leaving Port B pins in their non-active states and available for future expansion. The block also contains the clock generator that provides a clock signal running at 40.0 MHz to the DSP96002.

Finally, extra logic in this sub-block generates the $\overline{\text{RESET}}$ signal to the DSP96002 during the PCs power-up and upon OnCE$^{\text{TM}}$ or the PC’s software requests. This extra logic also generates the mode signals to the DSP96002 in order to program the DSP’s power-up mode and to provide it with external interrupt request capability.

2.4.2 Memory

This sub-block contains 4 RAM chips of 32K bytes each forming a memory array of 32K, 32-bit words. This sub-block also contains control logic providing the memory array with read and write strobes and distinguishing between PC and DSP96002 accesses.

2.4.3 Data Bus and Address Bus Buffers

These sub-blocks contain bus buffers to form internal common data buses and address buses which are shared between the DSP96002 and PC. The buffers are activated when the PC accesses the memory array or the DSP96002’s host port. The direction of the data bus buffers is determined by whether the access is a read or a write. When the DSP96002 is the master of this internal bus, the bus buffers are three-stated.

2.4.4 Bus Arbitration Logic

This sub-block contains control logic to generate all control signals required to maintain a bus-sharing mechanism, allowing either the DSP96002 or the PC access to internal common bus resources (memory and host port). This logic generates the DSP96002's a$\overline{\text{BG}}$ by telling the DSP96002 when it can be the bus master, and the PC's IO_CH_RDY by telling the PC to lengthen its bus cycle until the DSP96002 releases the bus. The following paragraphs describe the operation of these sub-blocks.

2.5 CPU Sub-Block Detailed Description

The DSP96002 is driven by a 40.0 MHz clock produced by a clock generator.

The OnCE^{TM} port connector allows an external command converter to be connected to the DSP96002 for purposes of debugging the DSP96002 software and on-board hardware. This connector provides access to DSI, DSO, DSCK, and $\overline{\text{DR}}$ which control the OnCE^{TM} port, and $\overline{\text{RESOUT}}$ which can assert $\overline{\text{RESET}}$ on the DSP96002.

The $\overline{\text{RESET}}$ signal drives the DSP96002's $\overline{\text{RESET}}$ pin and can be asserted by one of three sources:

- *The* RESET_DRV *signal indicates a power-up sequence in the PC*
- *The* $\overline{\text{RESOUT}}$ *signal is generated by the external command converter*
- *A write operation from the PC to the PLD, serving as an output port for the PC*

PLD1 collects the three sources and drives the $\overline{\text{RESET}}$ signal accordingly.

The following equations in the PLD generate $\overline{\text{RESET}}$:

RSTF = RESET_DRV;

!RES := WRITE_STROBE • SD0 + !WRITE_STROBE • !RES;

$\overline{\text{RESET}}$ = !RES • $\overline{\text{RESOUT}}$;

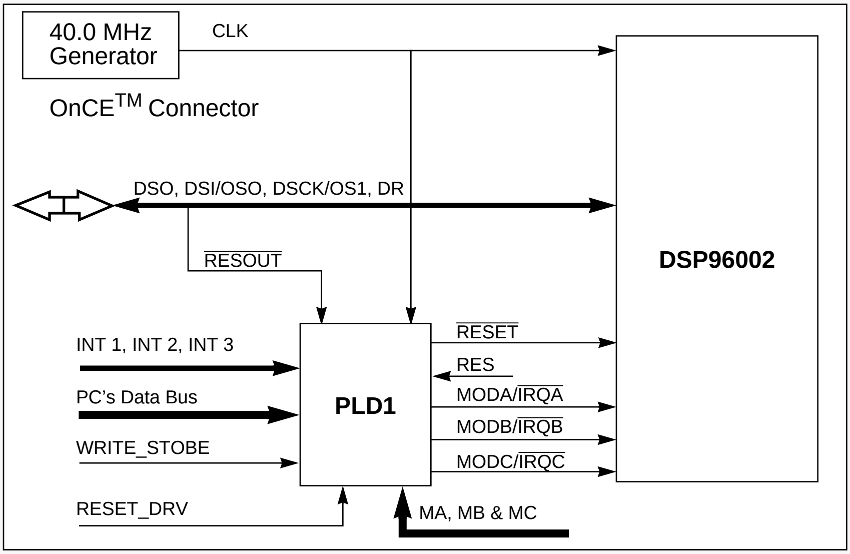

40.0 MHz Generator
CLK
OnCE^{TM} Connector
DSO, DSI/OSO, DSCK/OS1, DR
$\overline{\text{RESOUT}}$
DSP96002
INT 1, INT 2, INT 3
PC's Data Bus
WRITE_STOBE
RESET_DRV
PLD1
$\overline{\text{RESET}}$
RES
MODA/$\overline{\text{IRQA}}$
MODB/$\overline{\text{IRQB}}$
MODC/$\overline{\text{IRQC}}$
MA, MB & MC

Figure 2-2 *CPU Block Diagram*

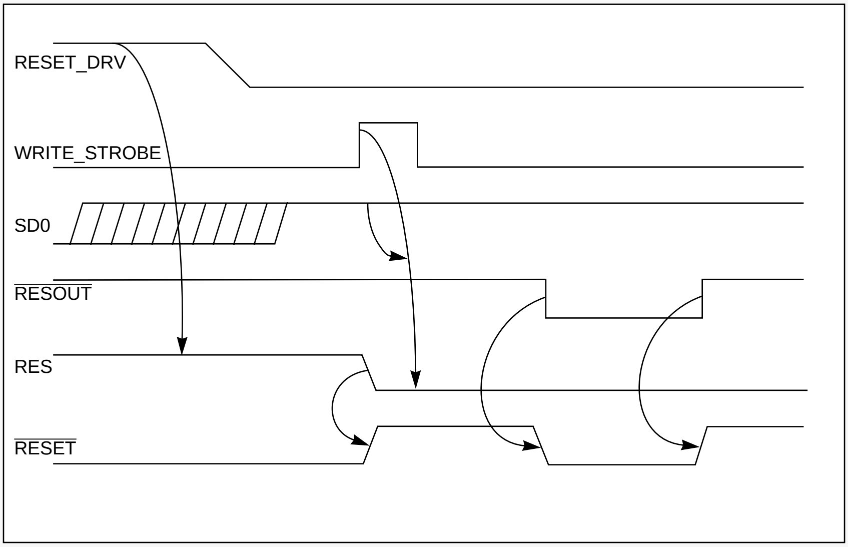


Figure 2-3 *RES Generation Timing Diagram*

The RES signal is an internal signal generated by the PLD which causes the PLD to appear to the ISA's CPU as a write-only latch. The WRITE_STROBE signal is generated by the control logic on the board and is a result of decoding the PC address and control buses. It is generated when the PC writes to this one-bit output port.

When the PC writes to the port, bit SD0 of the PC's data-bus is written to the PLD and changes the RES signal accordingly. The RES signal is cleared by the RESET_DRV signal which is the signal indicating a power-up sequence in the PC.

Finally, the $\overline{\text{RESET}}$ signal is the logical AND between RES (which indicates a reset request from the PC), and $\overline{\text{RESOUT}}$ (which indicates a reset request from the external command converter).

The DSP96002 has three interrupt inputs which have two functions:

- *When $\overline{\text{RESET}}$ is asserted, the interrupt inputs behave as mode programing inputs which determine which operation mode the DSP96002 will enter after $\overline{\text{RESET}}$ is deasserted.*
- *After the $\overline{\text{RESET}}$ signal is deasserted, these pins behave as interrupt request inputs.*

The PC programs the DSP96002's operation mode by writing to a second section of PLD1 which appears to be a three-bit output port in the PC's address space. The following additional equations in PLD1 generate MODA/$\overline{\text{IRQA}}$, MODB/$\overline{\text{IRQB}}$, and MODC/$\overline{\text{IRQC}}$:

MA := WRITE_STROBE • SD2 + !WRITE_STROBE • MA;

MB := WRITE_STROBE • SD3 + !WRITE_STROBE • MB;

MC := WRITE_STROBE • SD4 + !WRITE_STROBE • MC;

MODA/$\overline{\text{IRQA}}$ = MA • $\overline{\text{!RESET}}$ + INT1 • $\overline{\text{RESET}}$;

MODB/$\overline{\text{IRQB}}$ = MB • $\overline{\text{!RESET}}$ + INT2 • $\overline{\text{RESET}}$;

MODC/$\overline{\text{IRQC}}$ = MC • $\overline{\text{!RESET}}$ + INT3 • $\overline{\text{RESET}}$;

The signals MA, MB, and MC appear as a three-bit output latch to the PC and determine the mode of operation that the DSP96002 will enter when $\overline{\text{RESET}}$ is deasserted.

The PC's software can change the state of MA, MB, and MC by writing to this output port and asserting the WRITE_STROBE signal.

The last three equations describe a simple multiplexer that routes to the MODA/$\overline{\text{IRQA}}$, MODB/$\overline{\text{IRQB}}$, and MODC/$\overline{\text{IRQC}}$ input pins either the MA, MB, and MC signals (when $\overline{\text{RESET}}$ is asserted to program the DSP96002's mode), or the INT1, INT2, and INT3 external signals that serve as external interrupt requests.

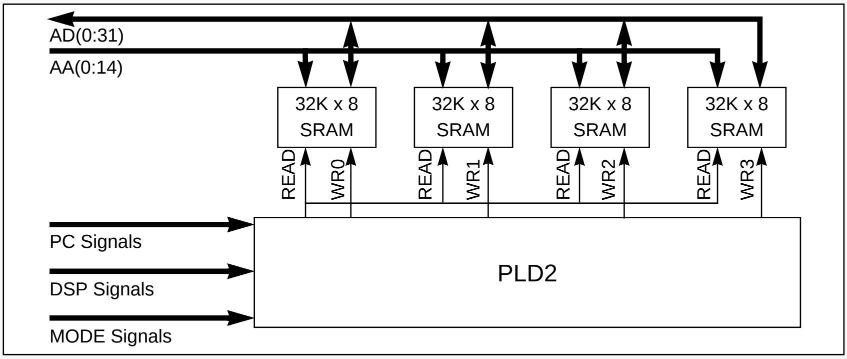


Figure 2-4 *Memory Detailed Block Diagram*

2.6 Memory Sub-Block Detailed Description

Two CPUs can read or write to/from the memory array — one is the ISA's CPU, and the other is the DSP96002. The DSP96002 accesses 32-bit words while the ISA's CPU accesses 8-bit bytes or 16-bit words.

Shared RAM between the PC and the DSP96002 is achieved by using four byte-wide RAM chips, each holding 32K bytes.Thus, a memory array is created that is accessible as 32K words by the DSP96002, while the same memory array is accessed as 128K bytes by the PC.

Another PLD is responsible for generating the $\overline{\text{READ}}$ strobe that controls all RAM chips, and the four write strobes ($\overline{\text{WR0}}$, $\overline{\text{WR1}}$, $\overline{\text{WR2}}$, and $\overline{\text{WR3}}$) that control the write operation to each of the RAM chips.

The bus labeled ‘PC signals’ in Figure 2-4 contains the following control signals:

MEMORY *an internal on-board signal that is a result of decoding the PC’s address bus and latching it with the PC’s BALE signal*

LAEN *an internal on-board signal that is the PC’s AEN signal latched with the PC’s BALE signal. Valid PC accesses are those when LAEN is c[illegible] ISA bus signals.*

SBHE

SA0

SA1

$\overline{\text{SMEMR}}$

$\overline{\text{SMEMW}}$

The bus ‘DSP signals’ contains the following DSP96002 control signals: aA31, a$\overline{\text{TS}}$, aR/$\overline{\text{W}}$, and a$\overline{\text{BA}}$.

The bus labeled ‘MODE signals’ in Figure 2-4 contains the following on-board internal signals:

OFF_RAM *a signal that disables the PC from accessing the RAM array*

$\overline{ADD\_EN}$ *a signal that disables the DSP96002 from accessing the RAM array*

The following equations in PLD2 generates the PLD’s outputs:

$$\overline{READ} = \overline{ADD\_EN} \bullet a\overline{BA} \bullet aR/\overline{W} \bullet a\overline{TS}$$
$$+!\overline{ADD\_EN} \bullet a\overline{BA} \bullet !OFF\_RAM \bullet MEMORY \bullet !LAEN \bullet !\overline{SMEMR}\ ;$$

$$\overline{WR0} = \overline{ADD\_EN} \bullet a\overline{BA} \bullet aR/\overline{W} \bullet a\overline{TS}$$
$$+\ !\overline{ADD\_EN} \bullet a\overline{BA} \bullet !OFF\_RAM \bullet MEMORY \bullet !LAEN \bullet !\overline{SMEMW} \bullet$$
$$!SA1 \bullet !SA0;$$

$$\overline{WR1} = \overline{ADD\_EN} \bullet a\overline{BA} \bullet aR/\overline{W} \bullet a\overline{TS}$$
$$+\ !\overline{ADD\_EN} \bullet a\overline{BA} \bullet !OFF\_RAM \bullet MEMORY \bullet !LAEN \bullet !\overline{SMEMW} \bullet$$
$$!SA1 \bullet !SBHE;$$

$$\overline{WR2} = \overline{ADD\_EN} \bullet a\overline{BA} \bullet aR/\overline{W} \bullet a\overline{TS}$$
$$+\ !\overline{ADD\_EN} \bullet a\overline{BA} \bullet !OFF\_RAM \bullet MEMORY \bullet !LAEN \bullet !\overline{SMEMW} \bullet$$
$$SA1 \bullet !SA0;$$

$$\overline{WR3} = \overline{ADD\_EN} \bullet a\overline{BA} \bullet aR/\overline{W} \bullet a\overline{TS}$$
$$+\ !\overline{ADD\_EN} \bullet a\overline{BA} \bullet !OFF\_RAM \bullet MEMORY \bullet !LAEN \bullet !\overline{SMEMW} \bullet$$
$$SA1 \bullet !SBHE;$$

The above equations reveal two states:

- *DSP96002 access, when $\overline{ADD\_EN}$ is deasserted and a$\overline{BA}$ is asserted*
- *PC access, when $\overline{ADD\_EN}$ is asserted and a$\overline{BA}$*

is deasserted

Table 2-1 summarizes the conditions that generate the read and write signals.

Table 2-1 *Generation of Read and Write Signals*

DSP96002 Access	PC Access

READ when:
$a\overline{TS} = 0$ & $aR/\overline{W} = 1$;
	READ when:
	MEMORY = 1 & LAEN = 0 &
	OFF_RAM = 0 & $\overline{SMEMW} = 0$;

WRITE when:
$a\overline{TS} = 0$ & $aR/\overline{W} = 0$;
	WRITE when:
	MEMORY = 1 & LAEN = 0 &
	OFF_RAM = 0 & $\overline{SMEMW} = 0$

All chips receive the same write strobe.	and to the 1st chip when:
	SA1 = 0 & SA0 = 0 ;
	to the 2nd chip when:
	SA1 = 0 & SBHE = 0 ;
	to the 3rd chip when:
	SA1 = 1 & SBHE = 0 ;
	to the 4th chip when:
	SA1 = 1 & SBHE = 0 ;

While the DSP96002 accesses 32-bit words, the PC can only access 8-bit bytes or 16-bit words (see Table 2-1). When the PC reads data from memory, all memory chips receive a read signal although the PC accesses only one or two of the memory chips. However, the Bus ARBITRATION logic enables only the appropriate data-bus buffers, putting 8- or 16-bit data on the bus. When the PC writes data to the memory, only the memory chips that should be written receive the write strobe. This prevents data from being written to the wrong memory cells and allows the PC to pack more than one byte into a DSP96002 word.

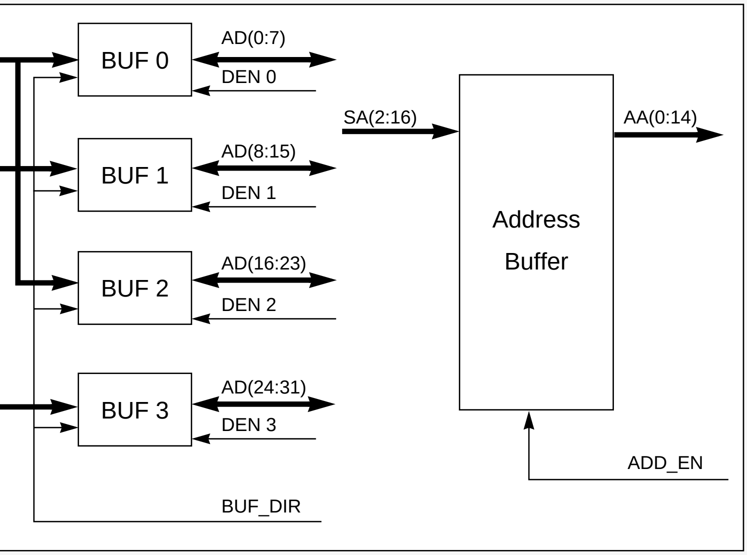
The OFF_RAM signal is used primarily during PC power-up to disable the PC from accessing memory. The OFF_RAM signal is set by circuitry on the card during PC power-up, thus preventing the PC's operating system software from recognizing this memory as part of the system's free memory.

The LAEN signal, which also disables the PC from accessing the DSP96002 memory, indicates that the PC is performing DMA transfers so that the DSP96002 memory cannot be accessed by the PC using its own DMA.

2.7 Data and Address Bus Buffers Detailed Description

This sub-block is responsible for connecting the PC data and address buses to the internal common bus, allowing data to be transferred between these two memory spaces. Since the PC's data bus is 16 bits wide and the DSP96002's internal bus is 32 bits wide, the PC can transfer data onto either the low portion (bits 0-15) or the high portion (bits 16-31) of the DSP96002's internal data bus in any one access. If the PC transfers data on the byte wide bus, it can put the low byte (bits SD0-SD7) or the high byte (bits SD8-SD15) of its 16-bit word on the data bus. Therefore, the data bus buffer is organized as four parts, each handles one byte of the internal 32-bit data bus. Each part of the buffer is controlled by a separate enable signal i.e., $\overline{\text{DEN0}}$, $\overline{\text{DEN1}}$, $\overline{\text{DEN2}}$, and $\overline{\text{DEN3}}$. The direction of the buffer is controlled by the BUF_DIR signal depending on the kind of transfer – read or write.

The address bus buffer drives the DSP96002's address bus, creating an internal address bus that is shared between the PC and the DSP96002. The buffer is controlled by the $\overline{\text{ADD\_EN}}$ signal which is asserted when the PC wants to access the internal bus resources and the DSP96002 is not the bus master. Note that PC addresses SA(2:16) drive DSP's addresses AA(0:14) while PC addresses SA(0:1) are used in the control logic to detect the nature of the PC access.



-5 *Data and Address Bus Buffers Detailed Block Diagram*

2.7.1 Bus Arbitration and Control Logic Detailed Description

This sub-block has 4 functions:

Host Interface *Between the PC and the DSP96002's host port.*

Address Decoder *Decodes the PC's address lines to identify PC accesses.*

Buffers Control *Controls the buffers that connect the PC and DSP buses.*

Bus Arbiter *Arbitrates between the PC and the DSP96002 buses.*

2.7.2 Host Interface Detailed Description

PLD3 generates some control signals for the DSP96002 when the PC is accessing the DSP host port. PLD3 also generates the WRITE_STROBE signal when the PC accesses some output ports e.g., the 3-bit port used to program the mode bits MA, MB, and MC. The CLK signal is the same clock signal that the DSP96002 uses.

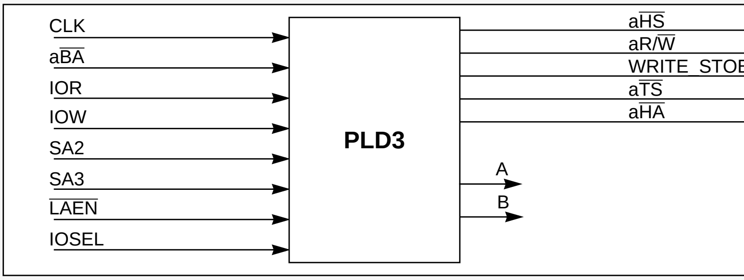


Figure 2-6 *Host Interface Detailed Block Diagram*

The following are the equations of PLD3:

aR/$\overline{W}$.TRST	=	a$\overline{BA}$;
a$\overline{TS}$.TRST	=	a$\overline{BA}$;
a$\overline{HA}$.TRST	=	a$\overline{BA}$;
WRITE_STROBE	=	!LAEN ● IOSEL ● !$\overline{SIOW}$ ● SA3 ● !SA2 ;
!a$\overline{HS}$	=	a$\overline{BA}$ ● (!$\overline{SIOR}$ + !$\overline{SIOW}$) ● !SA3 ● !SA2 ● !LAEN ● IOSEL ;
aR/$\overline{W}$	=	!$\overline{SIOR}$ ● !LAEN ● IOSEL ;
!a$\overline{HA}$	=	a$\overline{BA}$ ● (!$\overline{SIOR}$ + !$\overline{SIOW}$) ● !SA3 ● SA2 ● IOSEL ● !LAEN ;
A	:=	a$\overline{BA}$ ● (!$\overline{SIOR}$ + !$\overline{SIOW}$) ● !SA3 ● !SA2 ● IOSEL ● !LAEN ;
B	:=	A ;
!a$\overline{TS}$	=	a$\overline{BA}$ ● (!$\overline{SIOR}$ + !$\overline{SIOW}$) ● !SA3 ● !SA2 ● IOSEL ● !LAEN ;

The first three equations indicate that aR/$\overline{W}$, a$\overline{TS}$, and a$\overline{\text{HA}}$ are three-stated when the a$\overline{\text{BA}}$ is asserted. This happens if the DSP96002 is the Bus Master and generates those signals.

The IOSEL signal, which is an input to the PLD generated in another section of the control logic, is a product of decoding the PC’s address bus. It is generated when the PC is accessing addresses in the range $F100 - $F1EF.

In this range of addresses, the PC has the following I/O ports on board:

SA3=1 & SA2=0 *generating WRITE_STROBE for some output ports in the card*

SA3=0 & SA2=0 *generating a$\overline{HS}$ and a$\overline{TS}$ when selecting the DSP96002’s host port*

SA3=0 & SA2=1 *generating a$\overline{HA}$ when selecting the DSP96002’s host port*

The signals A and B are used to delay the generation of a$\overline{\text{TS}}$ to satisfy the DSP96002 timing requirements.

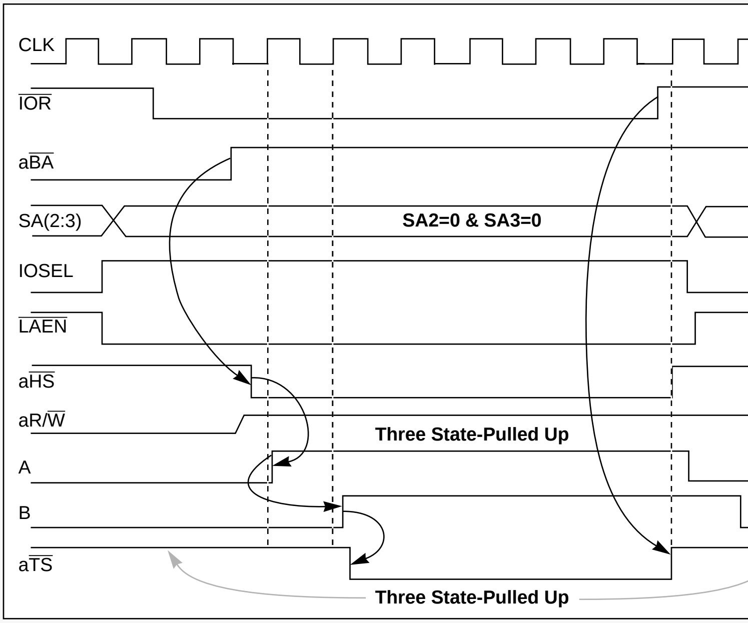


Figure 2-7 *Host Select Timing Diagram*

2.7.3 Address Decoder Detailed Description

PLD4 in Figure 2-8 decodes the addresses coming from the PC for some of the common bus resources e.g. one of the memory locations or the DSP96002’s host port.

The PLD generates two signals:
MEMSEL, which corresponds to memory selection, and **IOSEL**, which corresponds to I/O selection.

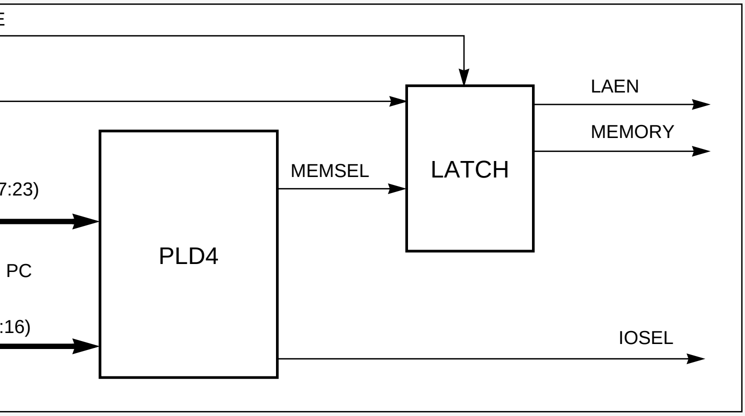
The following equations describe PLD4:

```
IOSEL       =    (SA(15:8) == $F1) • !SA7
                + (SA(15:8) == $F1) • !SA6
                + (SA(15:8) == $F1) • !SA5
                + (SA(15:8) == $F1) • !SA4;
MEMSEL      =   LA(23:17) == $04;
```

As mentioned in previous sections, the PC can access I/O addresses in the range of $F100 - $F1EF.

The permitted memory access range is $80000 - $9FFFF which is a total of 128K bytes.

The signals MEMSEL and AEN are latched by BALE and enable generation of the MEMORY signal used by the static RAM array.



8 *Address Decoder Detailed Block Diagram*

2.7.4 Buffer Controller Detailed Description

PLD5 supplies control signals to the data bus buffers. The signals $\overline{\text{DEN0}}$, $\overline{\text{DEN1}}$, $\overline{\text{DEN2}}$, and $\overline{\text{DEN3}}$ enable the data bus buffers when the PC accesses the internal common bus and the DSP96002 is no longer the bus master.

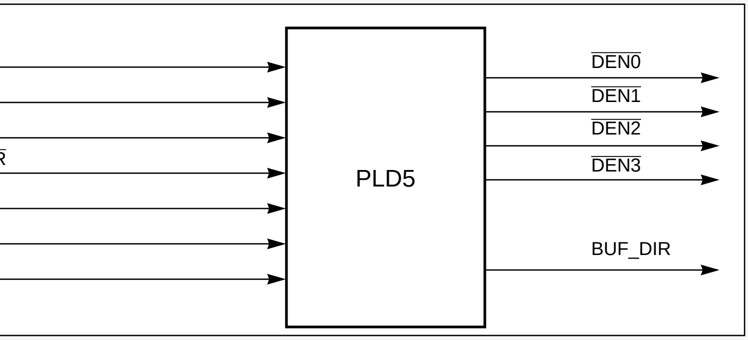
The BUF_DIR signal controls the data bus buffer direction and relies upon the nature of the PC access i.e., read or write.

The following equations describe PLD5:

$$
\begin{aligned}
!\overline{\text{DEN0}} &= \text{!LAEN} \bullet \text{ACC} \bullet \text{!SA1} \bullet \text{!SA0;}\\
!\overline{\text{DEN1}} &= \text{!LAEN} \bullet \text{ACC} \bullet \text{!SA1} \bullet \text{!SBHE;}\\
!\overline{\text{DEN2}} &= \text{!LAEN} \bullet \text{ACC} \bullet \text{SA1} \bullet \text{!SA0;}\\
!\overline{\text{DEN3}} &= \text{!LAEN} \bullet \text{ACC} \bullet \text{SA1} \bullet \text{!SBHE;}\\
\text{BUF\_DIR} &= \overline{\text{SMEMR}} \bullet \overline{\text{SIOR}}\ ;
\end{aligned}
$$

The ACC signal is generated by another section of the control logic and indicates that the PC is in the middle of an access while the DSP96002 is no longer the bus master.

The data bus buffer direction is controlled by BUF_DIR which is asserted when both $\overline{\text{SMEMR}}$ and $\overline{\text{SIOR}}$ are deasserted i.e., the PC executes a write cycle. In this case, the data bus buffers are directed to route data from the PC to the DSP96002.



-9 *Buffer Controller Detailed Block Diagram*

2.8 Bus Arbiter Detailed Description

This sub-block has six functions:

- *To assert the ACC signal when the PC executes an access to the common bus while the DSP96002 is not the bus master*
- *To assert the $\overline{\text{ADD\_EN}}$ signal when the PC accesses the common bus and the address bus can be driven by the PC*
- *To pull the $\overline{\text{MEMCS16}}$ signal low, telling the PC that it accessed a 16-bit wide memory port*
- *To pull the $\overline{\text{IOCS16}}$ signal low, telling the PC that it accessed a 16-bit wide I/O port*
- *To pull the I/O_CH_RDY signal low when the PC accesses the common bus while the DSP96002 is still the bus master*
- *To assert the a$\overline{\text{BG}}$ signal when the PC does not access the common bus, or deassert a$\overline{\text{BG}}$ when the PC tries to access the common bus*

The ISA bus signals $\overline{\text{MEMCS16}}$, $\overline{\text{IOCS16}}$, and I/O_CH_RDY are driven by three-state buffers in order to let other ISA bus boards drive these signals if needed.

The PLD clock is the inverted DSP96002 clock which ensures both the setup and hold-time requirements for a$\overline{\text{BG}}$.

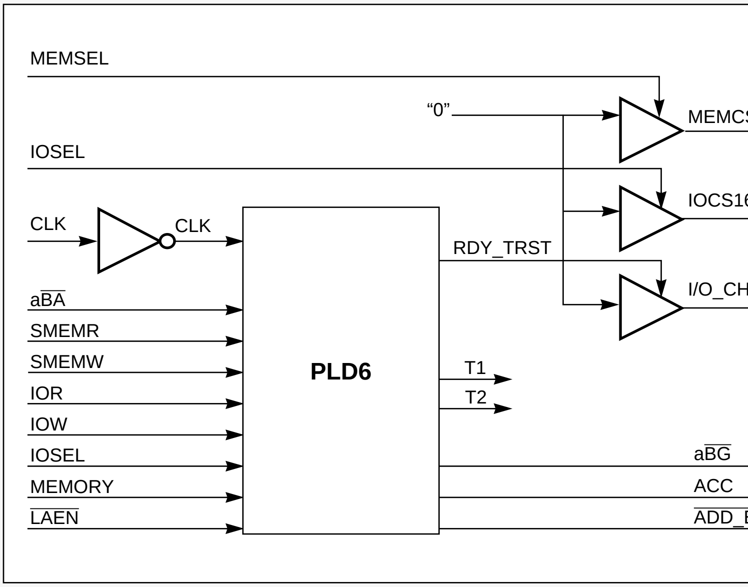


Figure 2-10 *Bus Arbiter Detailed Block Diagram*

The Bus Arbiter PLD equations are:

= !LAEN • MEMORY + !LAEN • IOSEL • (!$\overline{\text{SIOR}}$ + !$\overline{\text{SIOW}}$)

= !(a$\overline{\text{BA}}$ • a$\overline{\text{BG}}$ • PC_ACC)

T = PC_ACC • $\overline{\text{ADD\_EN}}$

:= PC_ACC • (!T1 • !T2 • !a$\overline{\text{BG}}$) + PC_ACC • (T1 • !T2 • !a$\overline{\text{BG}}$)
+ PC_ACC • (T1 • T2 • !a$\overline{\text{BG}}$) + PC_ACC • (T1 • T2 • a$\overline{\text{BG}}$)

:= T1

:= T2

= PC_ACC • !$\overline{\text{ADD\_EN}}$

The first equation is a macro (not an output definition) that defines the state when the PC accesses the internal common bus resources. When LAEN is low, a valid address bus value driven by the ISA’s CPU is indicated. When memory is high, the PC is accessing the on-board memory, or when ‘IOSEL * ($\overline{\text{SIOR}}$ + $\overline{\text{SIOW}}$) is high the PC is accessing the on-board I/O space.

The $\overline{\text{ADD\_EN}}$ signal is used by the address bus buffers and is asserted when a$\overline{\text{BA}}$, a$\overline{\text{BG}}$, and PC_ACC are high. This state indicates that the DSP96002 is no longer the bus master and the PC is in the middle of a transfer cycle to the on-board common resources.

The RDY_TRST signal, when asserted, pulls the ISA bus I/O_CH_RDY pin low, thus indicating to the PC that the present cycle should be lengthened by extra CPU cycles until the DSP96002 releases the bus. The signal is high as long as PC_ACC and $\overline{\text{ADD\_EN}}$ are both high. During a PC access, when the DSP96002 releases the bus, the $\overline{\text{ADD\_EN}}$ signal is asserted causing RDY_TRST to be deasserted and causing the PC to end its transfer cycle.

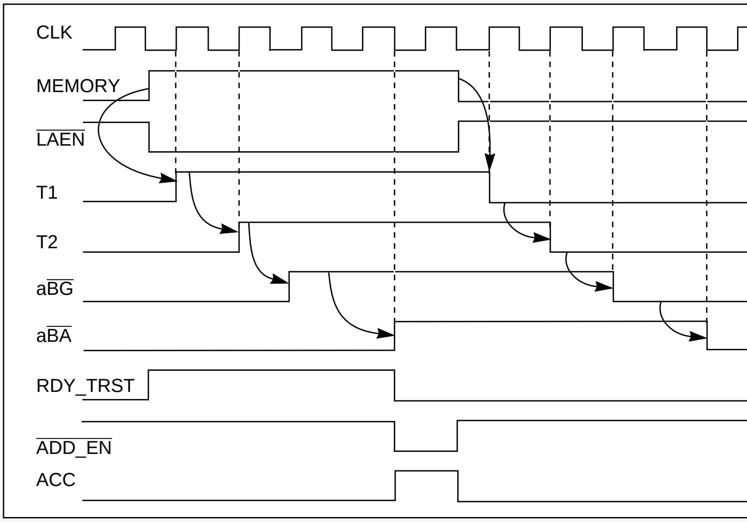


Figure 2-11 *Bus Arbitration Timing Diagram*

Signals T1 and T2 are used to synchronize generation of a$\overline{\text{BG}}$ which is a DSP96002 input signal that must be synchronized with the DSP96002 clock input.

The purpose of this three-stage shift register (T1, T2, a$\overline{\text{BG}}$) is to deassert a$\overline{\text{BG}}$ when the PC tries to access the common bus while the DSP96002 is the bus master. First, the T1 signal goes high if and only if a$\overline{\text{BG}}$, T1, and T2 are low. If one of these signals is still high from a previous PC transfer, then T1 is asserted only when all three stages are cleared.

After 2 clocks, a$\overline{\text{BG}}$ is deasserted, causing the DSP96002 to release the bus, thus allowing the PC to gain bus control and accomplish its transfer.

Finally, the ACC signal is generated by this PLD when PC_ACC and $\overline{\text{ADD\_EN}}$ are asserted to indicate to other parts of the board’s control logic that the PC is in the middle of a board access and that it is the bus master.

2.9 Sample Software Applications

Two very simple software applications are presented here to show the simple yet powerful interface achieved by the application board.

The first example is a data download program where the PC transfers data to the DSP96002’s internal data RAM by using the Host Interface.

The second example is also a data download program where the PC transfers data to the DSP96002’s internal data RAM, but by using some of the shared memory cells as semaphore and data registers.

2.9.1 Sample Application: Download Through the Host Interface

```
#define  HA_ICS   0x00F180 /*port A Host ICS register */
#define  HA_RTX   0x00F1A0 /*port A Host RX/TX register */
main()
{
int i;

for (i=0;i<0x200;i++)
            /*this loop will transfer 0x200 16-bit words to the DSP9600
              through it's Host port.*/
            {
            /*first the PC waits until the transmit data register is emp
              by checking the TXDE bit in ICS. It will be empty when the
              DSP96002 reads it's receive data register (HRX),
              thus asserting TXDE. */
       while ( (inport(HA_ICS) & 0x0002) == 0 );

            /*now the PC will transfer the data to the transmit data reg
            outport(HA_RTX,i);
            }
```

Figure 2-12 *'C' Language Program Listing*

```
       opt     mu,cre,cex,mex
       page    132,66,0,3,1

HCRA     equ      $FFFFFFEC                ;HCRA register
HSRA     equ      $FFFFFFED                ;HSRA register
HRXA     equ      $FFFFFFEF                ;HRXA register

         org      P:0

start    movep    #$00000000,x:$ffffffe   ;bcra programming for 0 wait

         move     #$0000ffff,d2.l          ;set mask
         move     #0,r0                    ;init pointer
         bclr     #5,x:HCRA                ;clear HRES bit in HCRA
         do       #$100,endloop            ;read 0x100 words from the H
loop1    jclr     #0,x:HSRA,loop1          ;wait until HRDF bit is set
         movep    x:HRXA,d0.l              ;read Receive Data Register
loop2    jclr     #0,x:HSRA,loop2          ;wait until HRDF bit is set
         movep    x:HRXA,d1.l              ;read Receive Data Register
         asl      #16,d1                   ;the 2nd word holds the 16 M
         and      d2,d0                    ;get only 16 LSBs
         or       d0,d1                    ;get the complete word
         move     d1.l,x:(r0)+             ;store in memory
endloop  nop                                             ;end of transfe
         nop
         nop
endp     jmp      endp
```

Figure 2-13 *DSP96002 Assembly Language Listing*

2.9.2 Sample Application: Download Through Common Memory

```
        READY   0x80000800      /*status handshake word      */
        DATA    0x80000820      /*data handshake word        */




 far *i,*j;


Y ;
;

;n<0x200;n++)
oop will transfer 0x200 16-bit words to the DSP96002’s internal memory
the board’s common memory.*/

the PC waits until the DSP96002 is ready to receive a new word by checking
Y word in the common memory.*/
 while (*i != 0);

 /*now the PC will transfer the data to common memory */
 *j = n;

 /*now the PC signals the 96002 that the transfer is READY*/
 *i = 0xFFFFFFFF;
```

-14 *‘C’ Language Program Listing*

```
 opt       mu,cre,cex,mex
 page      132,66,0,3,1

 equ       $200                     ;READY handshake word
 equ       $208                     ;DATA handshake word

 org       P:0

 movep     #$00000000,x:$fffffff    ;bcra programming for 0 wait states

 move      #$0000ffff,d2.l          ;set mask
 move      #0,r0                    ;init pointer
 move      r0,r7
 move      #READY,r3                ;init pointer
 do        #$100,endloop            ;read 0x100 words from the Host
 move      r7,x:(r3)                ;clear READY flag
 jclr      #0,x:(r3),loop1          ;wait until READY is set
 move      x:DATA,d0.l              ;read Data handshake word
 move      r7,x:(r3)                ;clear READY flag
 jclr      #0,x:(r3),loop2          ;wait until READY is set
DATA,d1.l;read Data handshake word
```

-15 *DSP96002 Assembly Language Listing*

2.10 Benchmarks

A FRACTAL program, based on one of Dr. Benoit Mandelbrot’s functions, was written to calculate the acceleration factor by using the DSP96002 as an attached processor.

First, the program was written in ‘C’ language and was run on the IBM PC/AT (8 MHz). The PC needed approximately *4 hours* to finish calculating and drawing the picture.

The same program was then run on a PC (80386 + 80387) running at 20 MHz. The PC finished the job in about *4.8 minutes*.

The program was then re-written in DSP96002 assembler code and was run on the application board (DSP96002 running at 40.0 MHz), transferring data to the IBM PC’s video RAM. ***The DSP96002 completed the job in about 6 seconds!***

2.11 Acceleration Factor

Compared to the 80286 (8 MHz) AT, the DSP96002 (40.0 MHz) runs **2400** times faster.

Compared to the 80386 + 80387 (20 MHz) PC, the DSP96002 (40.0 MHz) runs **48** times faster. ■

SECTION 3

Connecting the DSP96002 to the VMEbus

By O. Rubinstein

3.1 Introduction

This section provides design guidelines for interfacing the DSP96002 to the VMEbus. The design is complete for the specific case of the ADS96002 acting as a VMEbus slave, including such considerations as bus arbitration for the DSP96002, VMEbus protocol, and timing.

“The VMEbus has the ability to issue host commands to the DSP96002, to initiate data exchange in both directions, or to be asynchronously interrupted by the DSP96002.”

3.2 The VMEbus

VMEbus, also known as IEC 821 BUS or IEEE P1014/D1.2, is an asynchronous bus using the Eurocard format. References to the VMEbus are made throughout the design description, based on the assumption that the user is familiar with the bus operation. A description of VMEbus protocol can be found in the **IEEE Standard for a Versatile Backplane Bus: VMEbus**, ANSI/IEEE Std 1014-1987.

3.3 The DSP96002

The DSP96002 is a dual-port IEEE floating point processor capable of acting both as an independent/ main processor and as a slave processor through its Host MPU/DMA Interfaces (one on each port).

The VMEbus, described in the following section, communicates with the DSP96002 through the Port B Host Interface. The VMEbus has the ability to issue host commands to the DSP96002, to initiate data exchange in both directions, or to be asynchronously interrupted by the DSP96002. Support for local DSP96002 Port B memory is also provided.

References to the DSP96002 operation and timing are made throughout the design description, and user familiarity with the DSP96002 is assumed. A description of the DSP96002 can be found in the **DSP96002 User's Manual** and timing information can be found in the **DSP96002 Data Sheet**.

3.4 Design Description

This interface is connected to the ADS96002 (see Figure 3-1) on one side (referenced in the design as the "J" connector) and to the VMEbus on the other side (referenced as the "P" connector). The DSP96002 is mapped in the VMEbus memory space starting at hex address 80000000 and occupies sixteen 32-bit words. VMEbus address bits A5-A2 are mapped to the corresponding DSP96002 bits (bA5-bA2). The DSP96002 responds to data accesses in the address range AM5-AM0 = 001x01.

The DSP96002 can interrupt the VMEbus master on $\overline{\text{IRQ6}}$ if a “1” is written to the RREQ/TREQ bit in the ICS register (which enables assertion of $\overline{\text{HR}}$ when the RXDF/TXDE bit is asserted) and if the internal DSP96002 DMA is programmed to transfer data automatically (on HTDE/HRDF asserted). Unattended data transfer is then allowed through the on-chip DMA.

This design provides local memory on DSP96002 Port B. Whenever the VMEbus master is not accessing the DSP96002, the bus arbitration portion of the interface assigns the local bus to the DSP96002. However, the VMEbus master has higher priority than the DSP96002, allowing for fast response.

NOTE: *Two different symbols for inversion are used in this application report. An overbar is used for signals that are low true (e.g. $\overline{\text{IRQ6}}$). An exclamation is used for an inversion that occurs in a PLD (e.g.!A11).*

3.5 Signals Between the Interface and the VMEbus

D31-D0	=	data bus
A31-A1	=	address bus
AM5-AM0	=	address modifier
$\overline{\text{AS}}$, $\overline{\text{DS0}}$, $\overline{\text{DS1}}$	=	address & data strobes
$\overline{\text{LWORD}}$	=	32-bit transfer indication
$\overline{\text{WRITE}}$	=	transfer direction
$\overline{\text{DTACK}}$	=	transfer acknowledge
$\overline{\text{IACK}}$, $\overline{\text{IACKIN}}$, $\overline{\text{IACKOUT}}$, $\overline{\text{IRQ6}}$	=	interrupt handling
$\overline{\text{SYSRESET}}$	=	reset

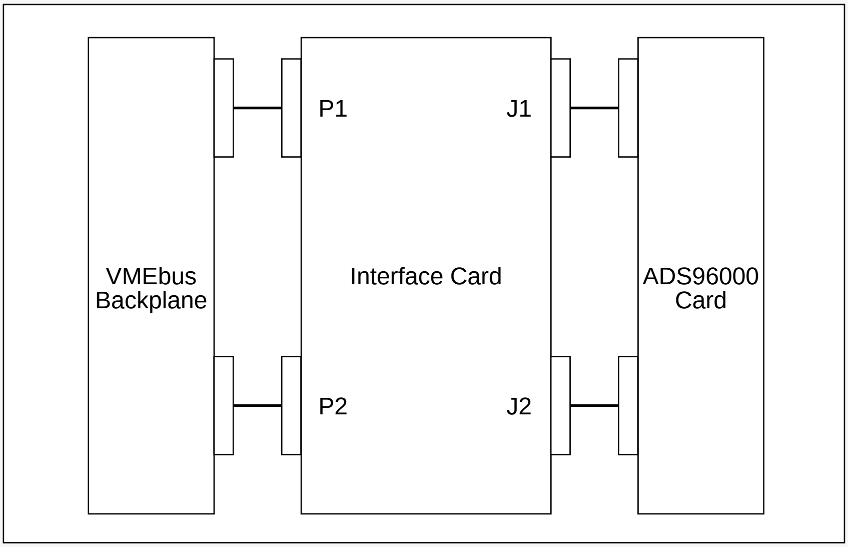


Figure 3-1 *Connection Block Diagram*

3.6 Signals Between the Interface and the ADS96002

bD31-bD0	=	data bus
bA5-bA2	=	address
b$\overline{TS}$, b$\overline{HS}$, b$\overline{HA}$, b$\overline{RW}$	=	selection signals
b$\overline{BG}$, b$\overline{BB}$, b$\overline{BA}$	=	arbitration signals
CLK_IO	=	DSP96002 clock

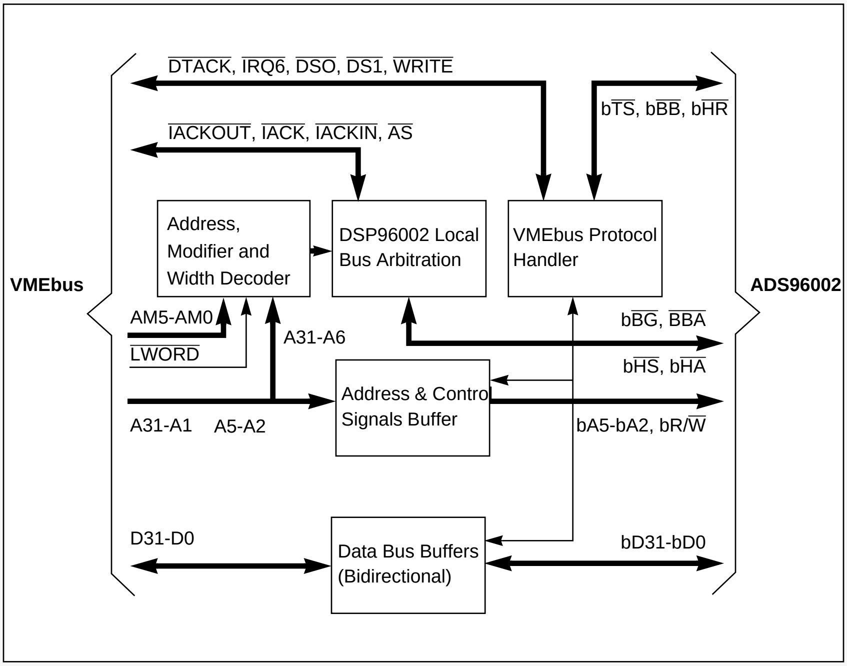

DTACK, IRQ6, DSO, DS1, WRITE
bTS, bBB, bHR
IACKOUT, IACK, IACKIN, AS
Address, Modifier and Width Decoder
DSP96002 Local Bus Arbitration
VMEbus Protocol Handler
VMEbus
ADS96002
AM5-AM0
LWORD
A31-A6
bBG, BBA
bHS, bHA
Address & Control Signals Buffer
A31-A1
A5-A2
bA5-bA2, bR/W
D31-D0
Data Bus Buffers (Bidirectional)
bD31-bD0

Figure 3-2 *Interface Card Block Diagram*

3.7 Address and Modifier Decoding

The decoder signals the following conditions:

address = $80000000 - $8000003C;

address modifier = 001x01; A1 = 0;

$\overline{\text{LWORD}}$ = 0.

3.7.1 Address Decoder PLD Equations

$\overline{\text{O1}}$ = ([A31..A12]==^H80000);
A = !$\overline{\text{B1}}$ & !A11 & !A10 & !A9 & !A8 & !A7 & !A6 & !AM5 & !AM4 & !AM3 & !AM1 & AM0;
I = !A1 &!$\overline{\text{LWORD}}$;

3.8 Description— Bus Arbiter

The DSP96002 is parked on the local bus (idle state) unless the VMEbus master accesses the DSP96002 or the DSP96002 local memory (see Figure 3-3).

If the VMEbus master is servicing an interrupt other than from the DSP96002 (i.e., the $\overline{\text{IR}}$ signal is deasserted because the DSP96002 did not issue an interrupt) or the master is servicing an interrupt request other than $\overline{\text{IRQ6}}$, the interface acts as a participating interrupter and passes $\overline{\text{IACKIN}}$ to $\overline{\text{IACKOUT}}$.

If the VMEbus master is accessing the DSP96002, either in a regular data transfer bus (DTB) access or in a STATUS/ID read cycle, then the bus ownership is taken from the DSP96002 (b$\overline{\text{BG}}$ synchronously deasserted).

A regular DTB access, indicating a data read/write cycle, is identified by $\overline{\text{IACK}}$ = 1 with the correct address, address modifier, and $\overline{\text{AS}}$ = 0. A STATUS/ID read cycle indicating the start of interrupt service is identified by $\overline{\text{IACKIN}}$ = 0, A3 = 1, A2 = 1, A1 = 0, $\overline{\text{LWORD}}$ = 0, $\overline{\text{IR}}$ = 0, and $\overline{\text{AS}}$ = 0.

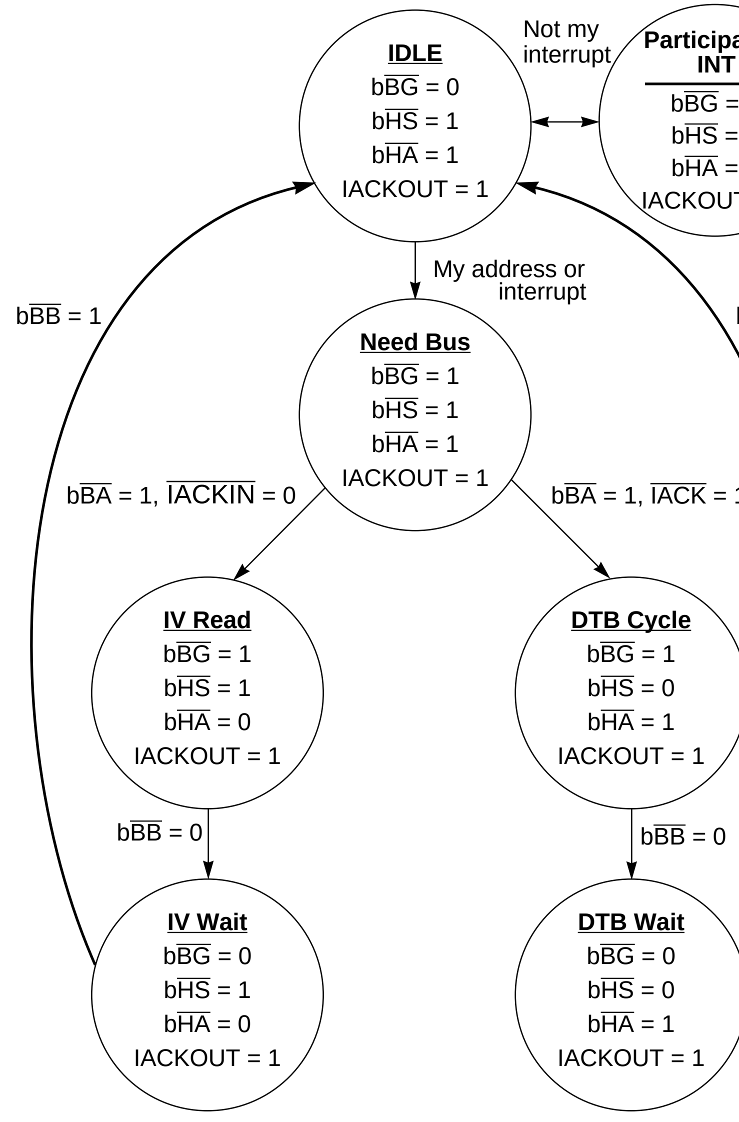


Figure 3-3 *DSP96002 Bus Arbitration*

As soon as the DSP96002 acknowledges the access (b$\overline{\text{BA}}$ deasserted), the interface asserts either the b$\overline{\text{HS}}$ signal (for a regular DTB cycle) or the b$\overline{\text{HA}}$ signal (for STATUS/ID read).

As soon as the VMEbus protocol handler receives the bus (signal b$\overline{\text{BB}}$ asserted), the interface asserts the b$\overline{\text{BG}}$ signal. When the VMEbus access is done and the protocol handler deasserts the b$\overline{\text{BB}}$ signal, the DSP96002 is enabled to take bus ownership immediately and the bus arbiter consequently returns to the idle state.

The $\overline{\text{INIT}}$ signal, which is asserted either if $\overline{\text{SYSRESET}}$ is asserted or on interface power up, always resets the arbiter to the idle state.

3.8.1 Bus Arbiter PLD Equations

$$\overline{\text{OUT}} \quad = \quad !\overline{\text{INIT}} \;\#\; !(!\overline{\text{BG}} \;\&\; \overline{\text{HS}} \;\&\; \overline{\text{HA}} \;\&\; !\overline{\text{IACKIN}} \;\&\; !\overline{\text{AS}} \;\&$$

$$!(\text{A3} \;\&\; \text{A2} \;\&\; \text{I} \;\&\; !\overline{\text{IR}})) \;\&\; (\overline{\text{IACKIN}} \;\#\; \text{I}\overline{\text{ACKOUT}});$$

$$\overline{\text{BG}} \quad := \quad \overline{\text{INIT}} \;\&\; \overline{\text{BB}} \;\&\; (\overline{\text{BG}} \;\#\; \overline{\text{IACK}} \;\&\; \text{A} \;\&\; !\overline{\text{AS}} \;\&\; \overline{\text{DTCK}} \;\&\; \overline{\text{BB}} \;\#$$

$$!\overline{\text{IACKIN}} \;\&\; \text{A3} \;\&\; \text{A2} \;\&\; \text{I} \;\&\; !\overline{\text{IR}} \;\&\; !\overline{\text{AS}} \;\&\; \overline{\text{DTCK}} \;\&\; \overline{\text{BB}});$$

$$\overline{\text{HS}} \quad = \quad !\overline{\text{INIT}} \;\#\; !((\overline{\text{BG}} \;\#\; !\overline{\text{BB}}) \;\&\; \overline{\text{BA}} \;\&\; \overline{\text{IACK}}) \;\&\; (\overline{\text{HS}} \;\#\; \overline{\text{BB}});$$

$$\overline{\text{HA}} \quad = \quad !\overline{\text{INIT}} \;\#\; !((\overline{\text{BG}} \;\#\; !\overline{\text{BB}}) \;\&\; \overline{\text{BA}} \;\&\; !\overline{\text{IACKIN}}) \;\&\; (\overline{\text{HA}} \;\#\; \overline{\text{BB}});$$

Protocol Handler

When the VMEbus master is not accessing the DSP96002, the VMEbus protocol handler state machine is in the idle state (see Figure 3-4). Only in this state is the b$\overline{\text{RW}}$ signal changed according to the $\overline{\text{WRITE}}$ signal and the $\overline{\text{IR}}$ signal is changed according to the b$\overline{\text{HR}}$ signal if interrupts are not disabled.

When the VMEbus master accesses the DSP96002 (either b$\overline{\text{HS}}$ or b$\overline{\text{HA}}$ is asserted by the bus arbiter), the protocol handler asserts b$\overline{\text{BB}}$ which opens the address and b$\overline{\text{RW}}$ buffer.

As soon as both data strobes $\overline{\text{DS0}}$ and $\overline{\text{DS1}}$ are asserted, the protocol handler opens the data bus buffer in the correct direction (either $\overline{\text{BUFR}}$ or $\overline{\text{BUFW}}$) and asserts the b$\overline{\text{TS}}$ signal, thus initiating the data transfer with the DSP96002 (see Figure 3-5 and Figure 3-5).

The b$\overline{\text{TS}}$ signal is then deasserted after one clock cycle, which also latches the data in the buffer. This is necessary when transferring data from the DSP96002 to the VMEbus master (read cycles) because the hold time of the DSP96002 is 2 ns, insufficient for the VMEbus which is typically much slower than the DSP96002. Although it is not required, data is also latched during write cycles for reasons of symmetry and simplicity.

The $\overline{\text{DTACK}}$ signal is asserted after one more clock which signals to the VMEbus master that the access has been completed. If the access is a write cycle, the buffers are also closed at this point.

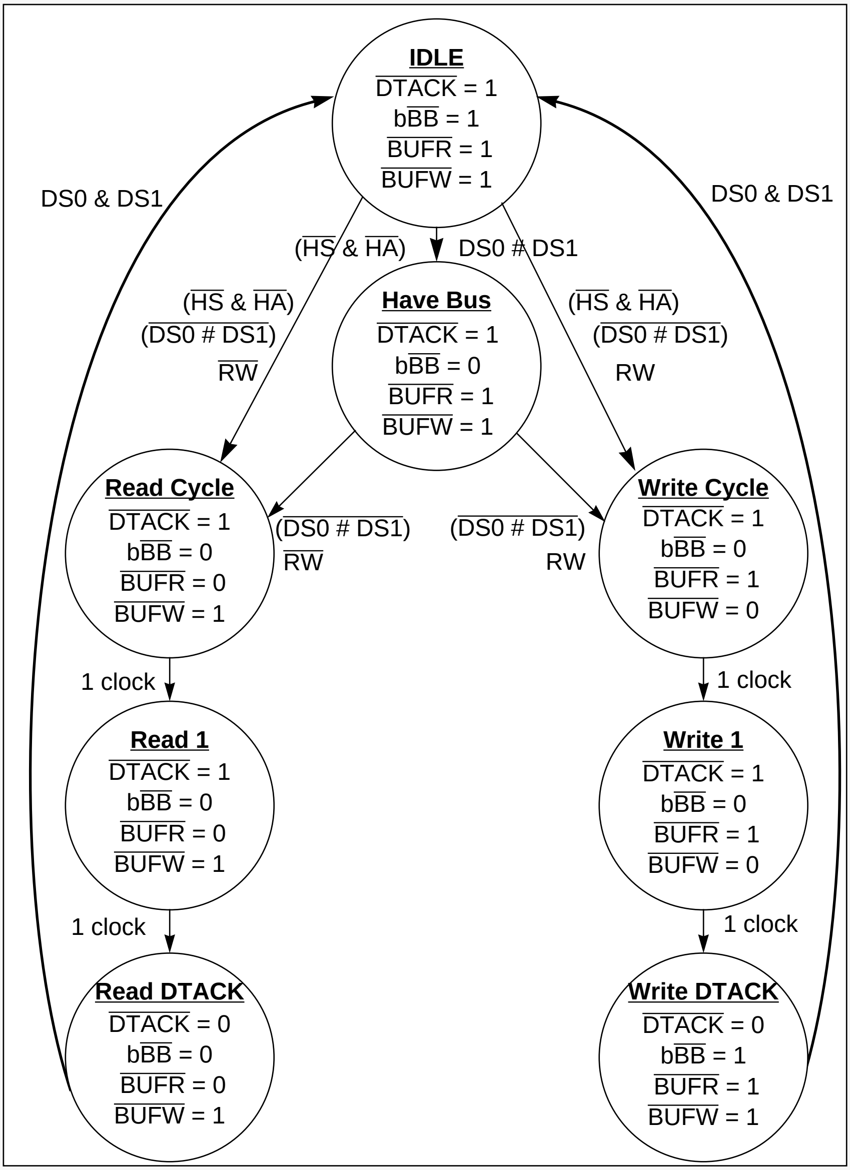


Figure 3-4 *VMEbus Protocol*

```
  disable   =  !INIT + ((reg==RD_DTCK) & !HA) + (disable & !HR) ;
  IR        =  !((reg==IDLE) & !disable & !HR) & IR + (reg==IDLE) & (disable + HR) ;
  RW        =  !((reg==IDLE) & !WRITE) & RW + (reg==IDLE) & WRITE ;

  reg       =  [DTCK,BB,BUFR,BUFW,TS] ;

"The following is a list of the state machine states. The TS output is
"deasserted on output in order to have an IDLE state of all ones.
  IDLE     = ^B11111 ;
  HAVE_BUS = ^B10111 ;
  RD_CYC   = ^B10010 ;
  WR_CYC   = ^B10100 ;
  RD_1     = ^B10011 ;
  WR_1     = ^B10101 ;
  RD_DTCK  = ^B00011 ;
  WR_DTCK  = ^B01111 ;
State  IDLE:      if       !(HS & HA) & (DS0 # DS1) then HAVE_BUS
                  else if  !(HS & HA) & !(DS0 # DS1) & !RW then WR_CYC
                  else if  !(HS & HA) & !(DS0 # DS1) & RW then RD_CYC
                  else     IDLE ;

State HAVE_BUS:   if       !(DS0 # DS1) & !RW then WR_CYC
                  else if  !(DS0 # DS1) & RW then RD_CYC
                  else if  !INIT then IDLE
                  else     HAVE_BUS ;

State WR_CYC:     if       !INIT then IDLE else WR_1 ;

State RD_CYC:     if       !INIT then IDLE else RD_1 ;

State WR_1:       if       !INIT then IDLE else WR_DTCK ;

State RD_1:       if       !INIT then IDLE else RD_DTCK ;

State WR_DTCK:    if       DS0 & DS1 then IDLE
                  else if  !INIT then IDLE
                  else     WR_DTCK ;

State RD_DTCK:    if       DS0 & DS1 then IDLE
                  else if  !INIT then IDLE
                  else     RD_DTCK ;
```

Figure 3-5 *VMEbus Protocol handler PLD Equations*

After another clock (if the data strobes $\overline{\text{DS0}}$ and $\overline{\text{DS1}}$ are deasserted), the state machine returns to the idle state. At this point, the buffers are closed if it is a read cycle.

The interrupts are locally disabled if the cycle is a STATUS/ID read (interrupt service). From this moment, the interface will not interrupt the VMEbus master until the interrupts are re-enabled. Re-enabling of the interrupts occurs when the b$\overline{\text{HR}}$ signal is deasserted. This is necessary because the DSP96002 deasserts the b$\overline{\text{HR}}$ signal only after the VMEbus master reads/writes to the Host receive/ transmit register and the VMEbus specifications require that the $\overline{\text{IRQ6}}$ signal be deasserted within 500 ns of the STATUS/ID read cycle.

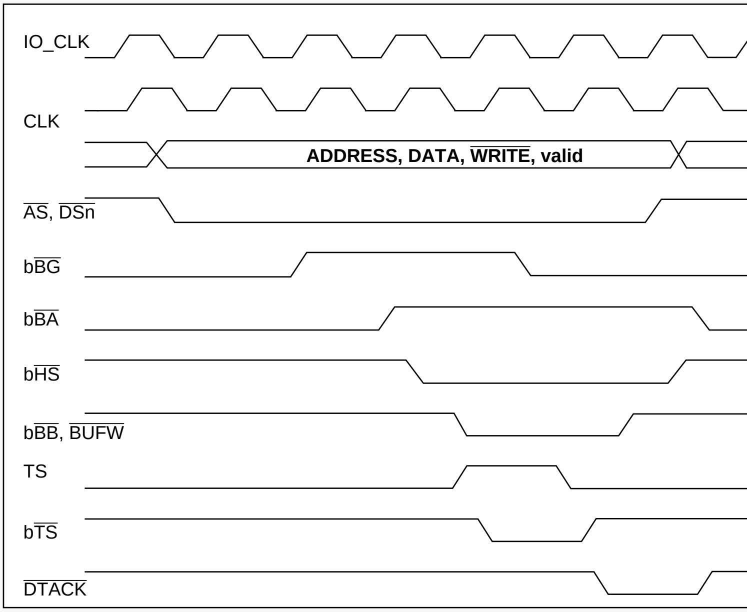
3.10 Timing Considerations

Signal b$\overline{\text{BG}}$ must be synchronous to the DSP96002 clock (IO_CLK).

The maximum delay for the 74F08 is 5.6 ns. Using a GAL20V8A12 (12 ns delay) leaves 6.4 ns setup time for 40 MHz operation which is more than the setup time required for the DSP96002. Not meeting the setup time requirement results in an extra clock period before b$\overline{\text{BA}}$ deassertion. This extra delay may occur anyway if b$\overline{\text{BG}}$ is deasserted before the first clock of an external DSP96002 Port B data access.

The setup and hold times for the 74F373 latches are critical. Latching should occur on the trailing edge of TS. The maximum delay for the leading edge of b$\overline{\text{TS}}$ is 4.7 ns for the 74F240. Since the 74F373 requires a setup time of 2 ns, this leaves 18.3 ns for a 25 ns clock (40 MHz operation) which is more than the delay from b$\overline{\text{TS}}$ assertion to data valid.

The minimum delay from the trailing edge of TS to the trailing edge of b$\overline{\text{TS}}$ is 1.5 ns which, together with the 2 ns DSP96002 hold time, results in a minimum of 3.5 ns total hold time. This hold time is greaterthanthe3.0nsholdtimerequiredforthe74F373.



IO_CLK
CLK
ADDRESS, DATA, $\overline{\text{WRITE}}$, valid
$\overline{\text{AS}}$, $\overline{\text{DSn}}$
b$\overline{\text{BG}}$
b$\overline{\text{BA}}$
b$\overline{\text{HS}}$
b$\overline{\text{BB}}$, $\overline{\text{BUFW}}$
TS
b$\overline{\text{TS}}$
$\overline{\text{DTACK}}$

Figure 3-5 *Write Timing (Master-to-Slave)*

NOTE: *The timing data reference is from the*

“MOTOROLA FAST AND LS TTL DATA” catalog.

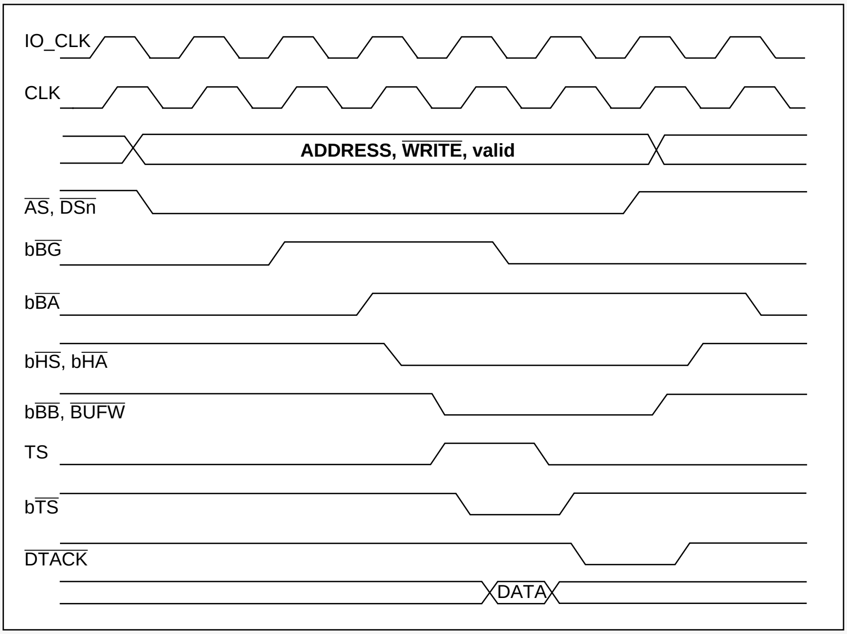


Figure 3-6 *Read Timing (Slave-to-Master)*

SECTION 4

Interfacing the DSP96002 Media Engine™ Processor to 56ADC16 Sigma-Delta A/D Converters

by R. Robles

4.1 Introduction

"Double buffering allows the DSP96002 to read the data anytime during the transmission of the subsequent data word."

The DSP96002 is a powerful DSP engine with application potential in many varied areas. A number of these applications require the digitization of analog signals. The following example demonstrates a simple method for connecting the DSP96002 to a pair of high performance Analog-to-Digital Converters with serial I/O ports.

The circuit described enables the user to interface a pair of DSP56ADC16 16-bit Sigma-Delta Analog-to-Digital Converters to the external Port A of the DSP96002, thereby providing a stereo input to the processor. In addition to the converters themselves, the interface circuitry consists of only five other devices — four MC74HC595A serial-to-parallel latches and one 22V10-10 PLD

4.2 The DSP56ADC16 Analog-to-Digital Converter

This device is a high performance Analog-to-Digital Converter based on sigma-delta conversion technology. With the internal FIR filter enabled, the DSP56ADC16 achieves 16-bit accuracy at output data rates up to 100 kHz with 96 dB of dynamic range and a 90 dB signal-to-noise ratio. 12-bit accuracy is delivered for output rates as high as 400 kHz by taking the output of the first (comb) filter stage.

The converter requires an input clock frequency 128 times the output sample rate. For a 48 kHz output sample rate, the input clock should be 6.144 MHz. A 44.1 kHz output rate requires a 5.6448 MHz clock.

Data samples are transmitted serially over a synchronous interface to the target processor. This very simple interface scheme requires only three signal lines: a clock, frame sync, and a data stream. For more detailed information on this A/D Converter, please refer to the **Motorola Data Sheet DSP56ADC16/D**.

4.3 The DSP96002 Media Engine^{TM} Processor

The DSP96002 is the first member of Motorola's family of single-chip, dual port, IEEE-754 compliant Digital Signal Processors. The DSP96002 delivers 60 MFLOPs (million floating point operations per second) and 200 MOPS (million operations per second) when operating from a 40 MHz clock. The architecture of the DSP96002 permits a number of concurrent operations during each instruction cycle. The data ALU, the Address Generation Unit, and the program controller operate in parallel within the CPU which permits each instruction cycle to accomplish:

- *an instruction prefetch*
- *up to three floating point operations (a multiply, an add and a subtract)*
- *three data moves*
- *four address pointer updates*

The speed, the parallelism of the architecture, and the mathematical precision inherent in the use of the IEEE-754 floating point standard combine to form a processor which is especially well suited to the mixture of tasks typical with multi-media applications. Please refer to the **DSP96002 User's Manual** and the **DSP96002 Data Sheet** for detailed information on this processor.

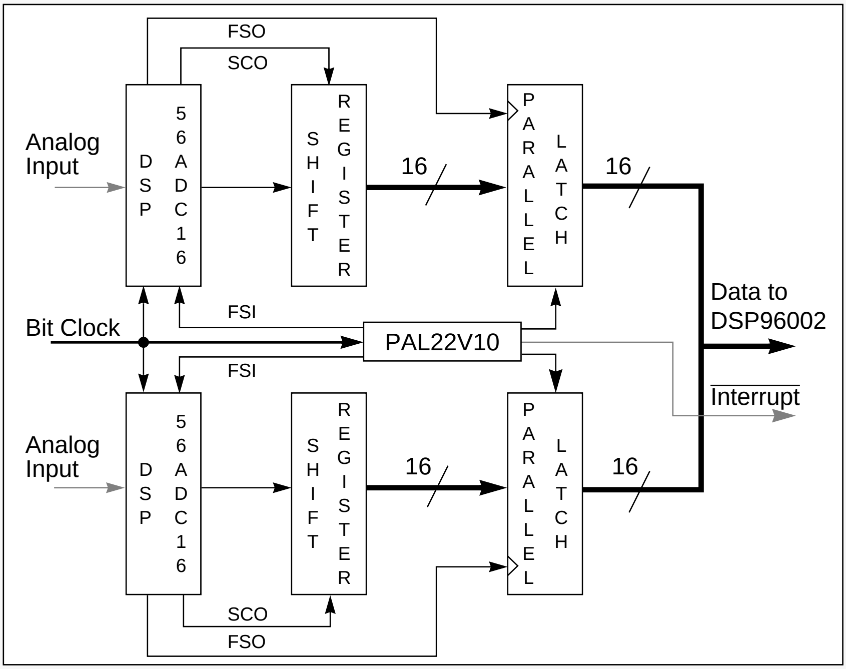


***Figure 4-1** Serial Interface Block Diagram*

4.4 Interface Hardware Description

Figure 4-1 depicts the system in block diagram form. Briefly, the two DSP56ADC16 A/D Converters run synchronously to each other, transmitting separate serial bit streams to their associated serial shift registers where the data is converted to parallel. When the least significant bit of a word arrives, the parallel word is latched into a three-state buffer.

“Double buffering” allows the DSP96002 to read the data anytime during the transmission of the subsequent data word. A 44.1 kHz word rate allows 22.7 µs for the DSP96002 to retrieve the data in the buffer before the next word arrives or 453 instruction cycles (40 MHz clock).

The PLD (PAL22V10) used in this interface provides three functions:

- *a 7-bit synchronous counter for generating Frame Sync Input to the two converters*
- *address decoding which enables the DSP96002 to read the three-state buffers*
- *interrupt Request (three-state) to the processor*

The counter generates a common Frame Sync Input (FSI) to the two DSP56ADC16s from its most significant bit (msb). This technique maintains synchronism between the two converter data streams and initiates subsequent transmissions at the earliest possible point in the system timing. The converters start transmitting serial data on the eighth clock after the rising edge of FSI. This subject will be covered in better detail in the timing section.

The address decoder in this example utilizes only the five most significant address bits (aA31-27) to map the serial data buffers into the DSP96002 memory space. When reading from the address range which is reserved for these buffers, the least significant address bit (aA00) determines whether the left or the right channel buffer will be placed onto the bus. This design assigns a unique address to the two buffers, placing the msb of each converter (the sign bit) onto the msb of the data bus.

Left/Right channel is distinguished by the read address. The PLD equations shown in Figure 4-4 map the two buffers into the following areas of DSP96002 memory:

- *Y:$F8xx xxx0 — Right Channel Data*
- *Y:$F8xx xxx1 — Left Channel Data*

Variations of this theme can be easily implemented in the form of changes to the PLD equations and/or the hardwiring of the address/data lines. For example, an alternate technique would be to map both buffers at the same address, placing one channel on the upper 16-bits of the bus and the second channel buffer onto the lower 16-bits of the bus. This method was not chosen due to the increased complexity of parsing the 32-bit data word, but some applications may find this approach advantageous.

The Frame Sync Output (FSO) from the converters strobes the serial shift register data into the parallel buffers. FSO occurs 8 clock cycles after the counter has rolled over to zero. The PLD asserts the $\overline{\text{IRQ}}$ line during the 10th clock cycle after FSO, assuring that adequate set-up time has been provided to the latches. $\overline{\text{IRQ}}$ is **not** three-stated. The interrupt request is conditioned with the processor $\overline{\text{RESET}}$ line in order to guarantee that the line is high during processor reset, regardless of the activity of the converters. Should the user’s design call fort MODA to be low during the reset sequence, a simple change to the PLD equations can satisfy this requirement.

4.4.1 Timing

The DSP56ADC16 supports two serial timing structures (see Figure 4-2 and Figure 4-3). The example circuit utilizes mode 0 which is selected by placing a logic zero on the Format Select (FSEL) pin of the device. In this mode, Frame Sync Output (FSO) is low for the entire period during which the 16 data bits are present on the SDO (serial data output) pin. This mode offers a rising edge on the Serial Clock Output (SCO) pin during the middle of each bit's cell time.

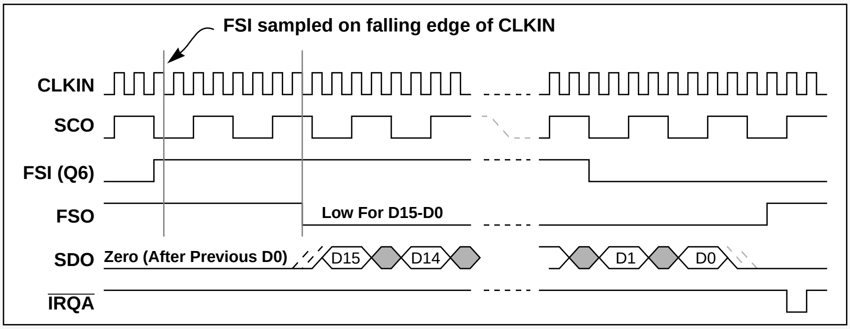


Figure 4-2 *Serial Interface Timing*

The circuit is clocked from a free running oscillator which operates at 128 times the desired word rate from the DSP56ADC16. In this example, a 5.6448 MHz clock is used to generate data at 44.1 kHz, the data rate used in common CD players. As the counter passes from a count of 63 to a count of 64, the msb of the counter, Q6, is asserted. Q6 is connected to the Frame Sync Input (FSI) pin of both DSP56ADC16 Converters. This rising edge on FSI initiates a serial word transfer out of both converters. After 8 clock cycles elapse, the msb of each data stream appears on the SDO pin and remains present for 4 clock cycles, or 1 SCO cycle.

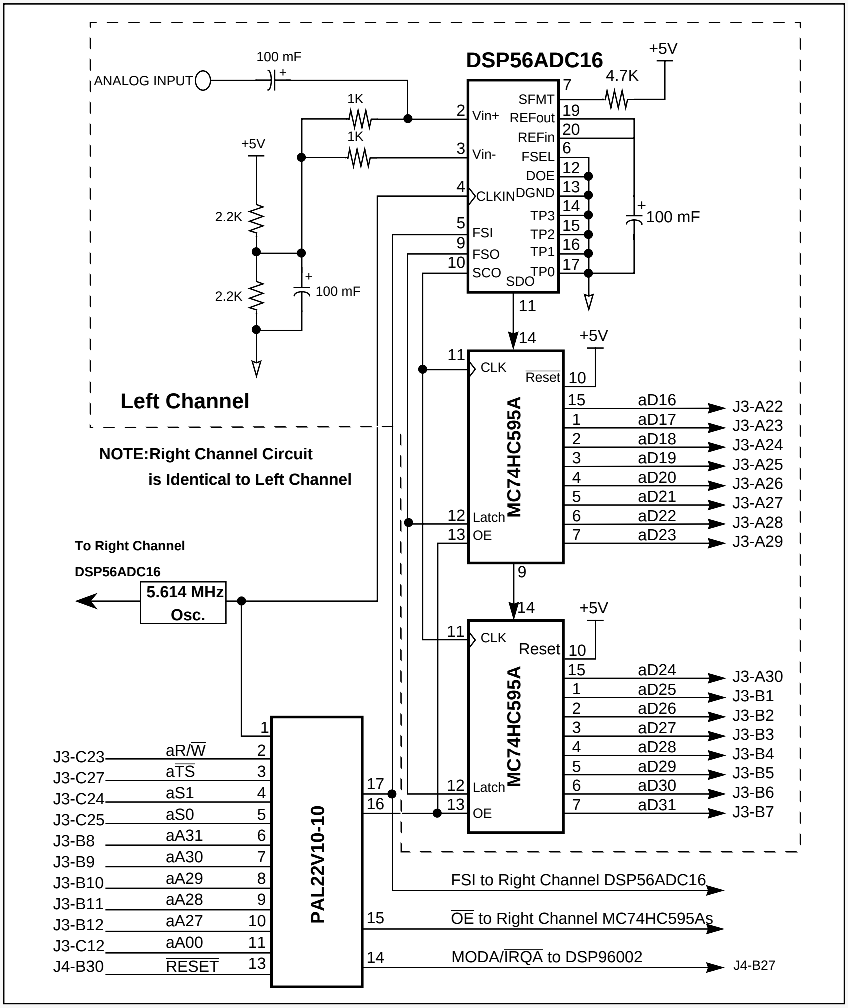


Figure 4-3 DSP96002 Serial Interface Schematic

SCO from the converter presents a rising edge during the center of each bit-wide time cell. This clock is used by the MC74HC595A's to advance the serial data through their shift registers. Subsequent data bits progress out of the converter until, finally, the 16th bit, bit 0, is on the SDO pin.

After bit 0 has been presented to the serial bus for 1 SCO cycle, SDO is driven to zero and the converter signals the end of the data stream by bringing FSO high. FSO is connected to the MC74HC595A's parallel latch strobe. This rising edge causes the data in the shift registers to be copied into the parallel buffers of the MC74HC595A's where it remains until the next rising edge of FSO. Effectively, this double buffering permits the processor to retrieve the data anytime during the next word's transmission period without losing data.

When FSO latches the data into the parallel buffers, the PLD drives $\overline{\text{IRQ}}$ low for one input clock period, assuming that the DSP96002 is configured for edge-sensitive interrupt sensing. This interrupt informs the processor that there is new data present in both the left channel buffer and the right channel buffer. Since the circuit provides unique addressing for each channel, there is no ambiguity regarding the origin of the data, simplifying the task of the software.

After the final bit is transmitted, another 64 clock cycles are required before another FSI can be sent to the converters; the PLD counter generates the successive FSI rising edge after these 64 clocks elapse.

```
module       ssi96c
title        'DSP96002 SSI-type Interface  Ver.4
AUTHOR:      Roman Robles
COMPANY:     MOTOROLA INC.
DATE:        30 January 1991'

          SSI96C     device    'P22V10';

"INPUTS
    CLK                    pin 1;        "Bit-Rate Clock"
    RWn                    pin 2;        "Read/Write*"
    TSn                    pin 3;        "Transfer Strobe"
    S1,S0                  pin 4,5;      "Address Selectors"
  A31,A30,A29,A28,A27,A00  pin 6,7,8,9,10,11;
    RST                    pin 13;       "Reset* input"


 Q6,Q5,Q4,Q3,Q2,Q1,Q0      pin 17,18,19,20,21,22,23;
 Q6,Q5,Q4,Q3,Q2,Q1,Q0                    ISTYPE 'invert';
 SSI_Lf_Rd                 pin  16       ISTYPE 'com,invert';
SSI_Rt_Rd                  pin  15       ISTYPE 'com,invert';
 IRQn                      pin  14;      "SSI Interrupt Request - 3 state"

 High,Low                  = 1,0;
 H,L,C,K,X,Z               = 1,0,.C.,.K.,.X.,.Z.;

    Address                = [RWn,TSn,S1,S0,A31,A30,A29,A28,A27,A00];
    BitCount               = [ Q6,Q5,Q4,Q3,Q2,Q1,Q0 ];

    count = ^h0B;
    inc    macro (a) {@const ?a=?a+1;};

equations
  " the machine is a simple, 7-bit counter (synchronous)"
  " it is clocked by Pin 1, CLK - the same clock used by the ADC16's"

    Q6.OE = 1; Q5.OE = 1; Q4.OE = 1;
    Q3.OE = 1; Q2.OE = 1; Q1.OE = 1; Q0.OE = 1;

    BitCount   := (BitCount + 1);
    BitCount.C  = CLK;

  " ----- ADDRESS DECODER -----"
    SSI_Rt_Rd   = !(Address == ^h26E);  "Y:$F8xx xxx0 READ Rt SSI data"
    SSI_Lf_Rd   = !(Address == ^h26F);  "Y:$F8xx xxx1 READ Lt SSI data"

  " Generate the interrupt request at count = 10 "
  " on the negative-going edge of the bit-clock"

    IRQn.OE     = 1;
    IRQn        = !((BitCount == 10) & !CLK & RST);


```

Figure 4-4 *DSP96002 Serial Interface — PLD Definition (sheet 1 of 2)*

```
”---------------------- TEST VECTORS for SSI96C ----------------”
Test_vectors                           “check the address decode”
   ([Address]  -> [SSI_Rt_Rd,SSI_Lf_Rd])
    [^h26E]    -> [0,1];
    [^h26F]    -> [1,0];
    [^h06F]    -> [1,1];

Test_vectors                           “check the counter & IRQ”
   ([CLK,RST] -> [BitCount,IRQn])
    [C,  1]   -> [^h00,1];
    [C,  1]   -> [^h01,1];
    [C,  1]   -> [^h02,1];
    [C,  1]   -> [^h03,1];
    [C,  1]   -> [^h04,1];
    [C,  1]   -> [^h05,1];
    [C,  1]   -> [^h06,1];
    [C,  1]   -> [^h07,1];
    [C,  1]   -> [^h08,1];
    [C,  1]   -> [^h09,0];
    [C,  1]   -> [^h0A,1];

 @repeat  116 {
    [ C,1]  -> [count,1];
          inc(count);}

Test_vectors
   ([CLK,RST]   -> [BitCount,IRQn])
    [C,  1]     -> [    ^h7F,   1];
    [C,  1]     -> [    ^h00,   1];
    [C,  1]     -> [    ^h01,   1];
    [C,  1]     -> [    ^h02,   1];

END  ssi96c

```

Figure 4-4 *DSP96002 Serial Interface — PLD Definition* *(sheet 2 of 2)*

SECTION 5

A Non-Intrusive Cycle Counter for the DSP96002 ADS

by R. Robles

5.1 Introduction

A common question which arises during algorithm development concerns the number of machine cycles that the algorithm will require. Iterative algorithms and lengthy code sections with multiple branch paths can complicate the task of determining the execution speed of a section of code. One simple solution to this problem is a hardware cycle counter. Figure 5-1 depicts a simple counter which can be connected to the DSP96002 Application Development System (ADS).

"The counter for this example is located on the processor's Port A. Simple changes to the connector pin-out can move the counter to Port B if required."

5.2 Circuit description

This circuit consists of four 8-bit counters constructed from 22V10-10 PLDs. The ABEL^{TM1} source file for these counters is shown in Figure 5-2. These counters are simple synchronous counters with DSP96002 address decoding and three-state outputs. In this example, the counters respond to any address in the DSP96002's Y: address space between Y:$FC00 0000 and Y$FFFF FFFF. Writing any value to this address range resets the

[1] ABEL is a trademark of the DATA I/O Corporation

counter. Reading any address in this range yields the number of clock cycles which have elapsed since the last reset. The counter for this example is located on the processor's Port A. Simple changes to the connector pin-out can move the counter to Port B if required. Figure 5-3 shows a sample program which tests the circuit. ■

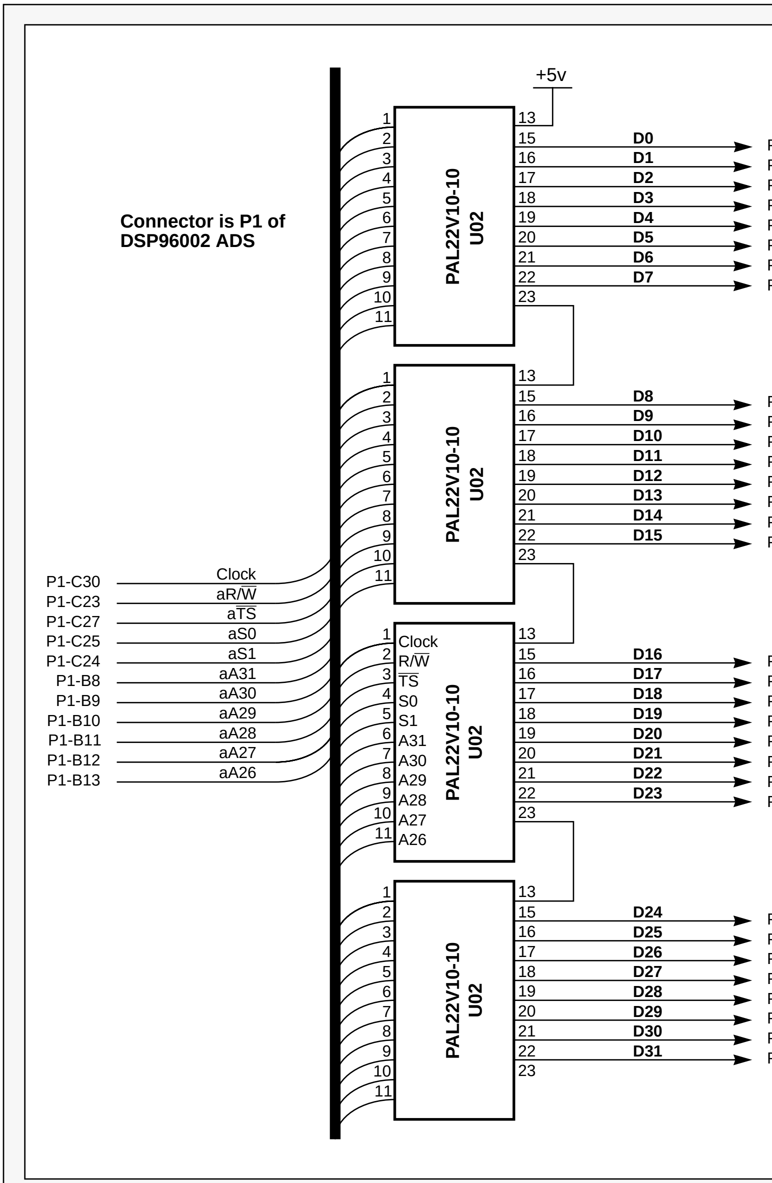


Figure 5-1 *DSP96002ADS Cycle Counter Schematic*

```
module adm96cnt
title ‘ADS96K Cycle Counter PLD U01 Ver.1
 MOTOROLA INC. 14 February 1991’

 ADM96CNT device ‘P22V10’;

”INPUTS
 CLK pin 1; “DSP96002 Clock “
 RWn,TSn,S1,S0 pin 2,3,4,5; “Read/Write*”
 A31,A30,A29,A28 pin 6,7,8,9; “Address 31-28”
 A27,A26 pin 10,11; “Address 27,26”
 CarryIn pin 13; “Look-Ahead Carry input”

”OUTPUTS”
 CarryOut pin 23 ISTYPE ‘buffer’;
 Q7,Q6,Q5,Q4 pin 22,21,20,19 ISTYPE ‘reg_d,buffer’;
 Q3,Q2,Q1,Q0 pin 18,17,16,15 ISTYPE ‘reg_d,buffer’;

 High,Low,Z = 1,0,.Z.;
 H,L,C,K,X = 1,0,.C.,.K.,.X.;

 BitCount = [ Q7..Q0 ];
 Address = [RWn,TSn,S1,S0,A31..A26]; “RT SSAA AAAA”

 count = 4;
 inc macro (a) {@const ?a=?a+1;};

equations
 “ the state machine is a simple, 8-bit counter (synchronous)”
 “ it is clocked by Pin 1, CLK and it is reset whenever the”
 “ Host READS from the Reset address.”

 CarryOut = (Q7.fb & Q6.fb & Q5.fb & Q4.fb & Q3.fb & Q2.fb & Q1.fb & Q0.fb & CarryIn);
 CarryOut.oe = 1;
 BitCount.clk = CLK;
 BitCount.ar = A31 & A30 & A29 & A28 & A27 & A26 & !RWn & !TSn & !S1 & !S0;
 BitCount.oe = A31 & A30 & A29 & A28 & A27 & A26 & RWn & !TSn & !S1 & !S0;
 WHEN (CarryIn == 1) THEN BitCount.d := BitCount.fb + 1;
 ELSE BitCount.d := BitCount.fb;

Test_vectors
 ([CLK,Address,CarryIn] -> [BitCount,CarryOut])
 [ C,^h023F,0] -> [0,0]; “CarryIn = 0, hold count”
 [ C,^h023F,0] -> [0,0];
 [ C,^h023F,1] -> [1,0];
```

5-2 *PLD Source for the DSP96002 ADS Cycle Counter (sheet 1 of 2)*

```
 [ C,^h023F,1] -> [2,0];
 [ C,^h023F,1] -> [3,0];
 [ C,^h023F,1] -> [4,0];
 [ C,^h003F,1] -> [Z,0]; “clear the counter”
 [ C,^h023F,0] -> [0,0]; “CarryIn = 0, hold count”
 [ C,^h023F,1] -> [1,0];
 [ C,^h023F,1] -> [2,0];
 [ C,^h023F,0] -> [2,0]; “hold it, again”
 [ C,^h023F,1] -> [3,0];
         “I’m NOT typing another 250 lines!”
 @repeat 250 {
 [ C,^h023F,1] -> [count,0];
 inc(count);}

 [ C,^h023F,1] -> [254,0];
 [ C,^h023F,1] -> [255,1]; “set CarryOut”
 [ C,^h023F,1] -> [0,0];“count rolls over”
 [ C,^h023F,1] -> [1,0];

END adm96cnt
```

Figure 5-3 *PLD Source for the DSP96002 ADS Cycle Counter (sheet 2 of 2)*

```
Motorola DSP96000 Assembler Version 1.1.2 91-03-26 13:49:29 96cnt.asm Page 1

1 page 132,66,3,3
2;-------------------------------------------------------------------
3; 96cnt.asm - quickie for initializing and testing the ADM96K Cycle
4; counter H/W
5;-------------------------------------------------------------------
6 FFFFFFFE    aBCR   equu    $FFFFFFFE          ;Port A Bus Control Register
7 FFFFFFFC    PSR    equ    $FFFFFFFC          ;Port Select Register
8 FFFFFFF0    CYC    equ    $FFFFFFF0          ;address of cycle counter(s)
9
10   P:00000000            org  p:$0
11   P:00000000   389D3088 clr  d0.l #$12345678,d1.l;load repeat count in D1.L
                  12345678
12   P:00000002   007101FE  movep    d0.l,x:aBCR   ;zero wait states on Port A
13   P:00000003   007101FC  movep    d0.l,x:PSR    ;locate all memory on Port A
14   P:00000004   007103F0  movep    d0.l,y:CYC    ;reset timer/counter
15 loop
16   P:00000005   01E00089 rep      d1.l          ;verify the counter
17   P:00000006   00000000 nop
18   P:00000007   007102F0 movep    y:CYC,d0.l    ;read timer (s/b $2468ACF8)
19   P:00000008   007103F0 movep    d0.l,y:CYC    ;reset timer
20   P:00000009   007202F0 movep    y:CYC,d0.m    ;read timer (s/b $00000004)
21   P:0000000A   03803F85 jmp  loop
22
23 END
```

Figure 5-4 *Sample Program to Test the DSP96002ADS Cycle Counter*

REFERENCES

1. Motorola DSP96002 IEEE Floating-Point Dual-Port Processor User’s Manual, DSP96002UM/AD

2. Motorola DSP96002 56-Bit General Purpose IEEE Floating-Point Dual Port Processor, Advance Information, DSP96002/D, Rev. 1

3. Motorola DSP56ADC16 16-Bit Sigma-Delta Analog-to-Digital Converter, Advance Information, DSP56ADC16/D

4. Motorola High-Speed CMOS Logic Data, DL129, Rev.4

5. PAL® Device Handbook, Advanced Micro Devices / Monolithic Memories Inc., 1988

6. PAL®DevicesDatabook,AdvancedMicroDevices,1990 ■