Motorola Digital Signal Processors

DSP56001 Interface Techniques and Examples

by
Roman Robles
Digital Signal Processor Operation

Table of Contents

Illustrations

SECTION 1

Interfacing Motorola's DSP56001 to Pseudo Static RAM

"PSRAM combines the economies of DRAM with the straightforward interface of fully Static RAM to provide 128K bytes in a 32-pin DIP."

When the design definition of a DSP subsystem calls for a large memory space, the cost of populating this space with static RAM (SRAM) can be prohibitive. Although SRAM offers the advantages of high speed and a very simple interface, the complex structure of the SRAM storage cell results in SRAM price/density ratios which are inferior to those of dynamic RAM. Pseudo Static RAM (PSRAM) presents one possible compromise between the contradictory requirements of high density, low cost, high speed and interface simplicity.

This section presents a simple implementation of a PSRAM interface to the DSP56001. Using an array of three 128K x 8 PSRAMs, the circuit provides access to 128K 24-bit words of data space. With the DSP56001 operating from a 33MHz clock, this memory subsystem will operate with 2 wait states for non-consecutive accesses.

Pseudo-Static RAM

PSRAM combines the economies of DRAM with the straightforward interface of fully Static RAM to provide 128K bytes in a 32-pin DIP. Internally, the device contains a dynamic RAM array with on-board address multiplexing, an internal refresh row counter, and an internal refresh timer. The memory array is divided into eight sections, each consisting of a of 512 (row) x 256 (column) matrix of storage cells, forming a byte-wide memory which is 128K locations deep.

The device is pin compatible with the 128Kx8 SRAM JEDEC pinout with the exception of pin 1 (on standard SRAMs, this pin would be a no-connect; on PSRAM, it is the refresh strobe $\overline{F}$). These features enable the PSRAM to replace fully static RAM in many applications with a minimum amount of “glue”.

PSRAM has two complementary enable lines, $\overline{E1}$ and E2. During read and write operations, these enable lines must *strobe* the address into the device. This is another difference between PSRAM and fully static RAM.

Since the PSRAM is based on DRAM storage elements, it requires a precharge delay between successive accesses and a periodic refresh. PSRAM supports three different refresh modes; CE-only refresh, auto refresh and self refresh.

- CE-only refresh requires external hardware or software to provide periodic addressing of each of the 512 rows. Use of this method would add a considerable amount of interface hardware or would cause significant degradation to software performance.
- Self refresh can be entered after 8 ms in standby mode. In this mode the on-board refresh timer and refresh counter are used to provide the refresh sequencing. A delay slightly greater than one access cycle is required when leaving this mode before data read/write operations can proceed. This mode is useful for long standby periods, but is not suitable for device refresh during periods of normal DSP activity due to the unique timing requirements. To use this mode during idle periods would require mode selection logic as well as the circuitry associated with one of the other “active access” modes.

- Auto refresh occurs when the PSRAM is disabled by either of the device select inputs going false followed by the refresh pin $\overline{F}$ going active. For each transition of $\overline{F}$, one row of each section is refreshed and the refresh row counter is advanced in preparation for the next refresh cycle. The example presented in this note uses this mode because it requires the least amount of external logic and impacts the normal DSP software only when a data transfer contends with a refresh cycle.

Figure 1-1 depicts an auto refresh cycle in which two rows are refreshed in succession. Note that either $\overline{E1}$ or $\overline{E2}$ can disable the device. Refer to the data sheets specific to the PSRAMs selected for any particular application.

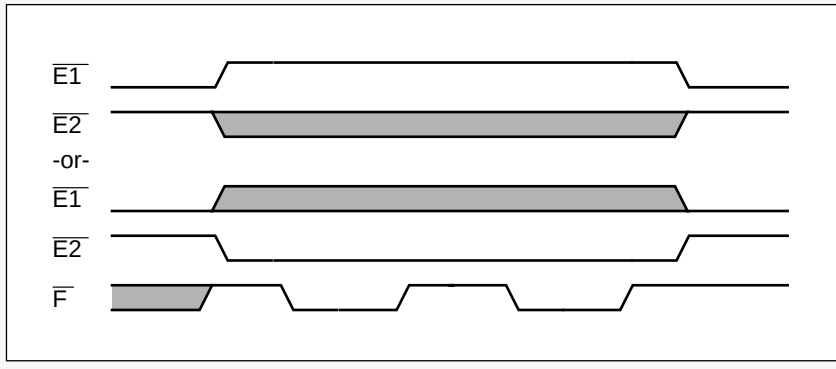


Figure 1-1 Pseudo Static RAM Auto Refresh Timing *in which 2 rows are refreshed in succession. Note that either $\overline{E1}$ or E2 can disable the device. Please refer to the data sheets specific to the PSRAMs selected for any particular application.*

1.1 DSP56001 Memory I/O Basics

Memory interface to the DSP56001 occurs over Port A of the processor. Port A consists of 24 bi-directional data lines (D0-D23), 16 address lines (A0-A15), three memory reference lines ($\overline{PS}$,$\overline{DS}$, X/$\overline{Y}$) and two data strobes ($\overline{RD}$, $\overline{WR}$). Additionally, a pair of bus access control signals, $\overline{\text{Bus Request}}$/$\overline{\text{Bus Grant}}$ ($\overline{BR}$/$\overline{BG}$), can be used to synchronize access requests between the processor and another device attempting to gain mastership of the bus. The bus access pins have alternate functions, $\overline{\text{Bus Strobe}}$ /$\overline{\text{Wait}}$ ($\overline{BS}$/$\overline{WT}$), which allow external circuitry to insert additional wait states in external bus cycles. To minimize power consumption, the address lines remain stable until the beginning of the next external access. The memory reference signals ($\overline{PS}$, $\overline{DS}$ and X/$\overline{Y}$) are deasserted during periods when the external bus is idle, but are ***not*** deasserted during successive accesses to the same external memory space.

Setting bit 7 of the processor's Operating Mode Register (OMR) causes the bus access control bits to assume the $\overline{\text{Bus Strobe}}$/$\overline{\text{Wait}}$ ($\overline{BS}$/$\overline{WT}$) mode. In this mode, the $\overline{BS}$ pin is asserted at the beginning of every external access and is released during T3 of each external cycle. Assertion of the $\overline{WT}$ pin during T2 ***while $\overline{BS}$ is asserted*** adds wait states to the bus cycle.

Wait states will continue to be inserted until two falling edges of EXTAL occur in succession with the release of $\overline{WT}$. $\overline{WT}$ should never be asserted when $\overline{BS}$ is inactive.

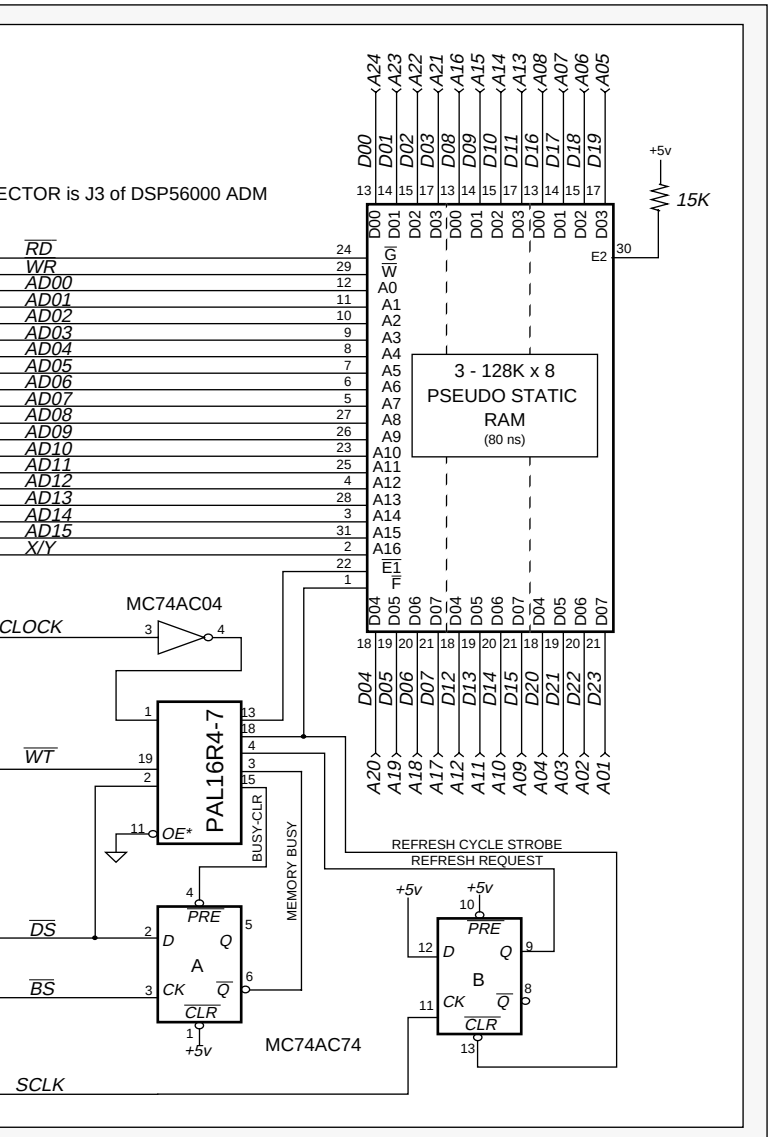
When the DSP56001 is reading data from the bus, the data must be stable for the specified setup and hold periods before and after (respectively) the rising edge of the read strobe $\overline{RD}$. During processor write operations to the external bus, the data is valid for a specified time before and after the rising edge of the write strobe $\overline{WR}$.

These relationships in are shown in the simplified PSRAM timing diagram of Figure 1-4. (For DRAM timing see Figure 2-3.) For more detailed information, refer to the **DSP56001 User's Manual** and the **DSP56001 Data Sheet.**

1.2 Memory Subsystem Overview

The circuit in Figure 1-2 is designed to serve as an extension of the Motorola DSP56000 ADS Application Development Module (ADM). The Static RAM on the ADM should be configured to reside solely within the DSP56001 program space. The PSRAMs and their interface circuitry are attached to the DSP56001's Data and Address Buses via ADM connector J3.

The PSRAM bank consists of three devices. Each device provides 128K storage cells for each of 8 data bits, forming an array of 128K 24-bit words. The DSP56001 can address 64K 24-bit words in each of its two data spaces, X:memory and Y: memory. Therefore, this PSRAM array fully populates both of the processor's data spaces.



-2 DSP56001-to-PSRAM Schematic *provides two functions: it controls the refresh cycles and it generates precharge delays. This is a schematic depiction of the interface circuit.*

In order to minimize the component count, the refresh request timing is supplied by the processor's Serial Control Interface (SCI) clock, SCLK. Initialization software configures this clock to provide a pulse train with a 15 μs period. Once initialized, the generation of this signal is completely transparent to any code executing on the processor. Figure 1-6 is a listing of the initialization code and a short "pass/fail" memory test routine. The value loaded into the SCI Clock Control Register (SCCR) at X:$FFF2 will vary as a function of the system clock frequency. For a 33 MHz clock, a value of $107F will yield the desired refresh rate of 15.6 μs per row.

A second task of the initialization software is the selection of the $\overline{BS}/\overline{WT}$ mode of operation. This mode allows an external source to insert wait states into bus cycles, and is employed by the interface when precharge and refresh delays are needed.

The interface operates from the same clock which drives the processor. In systems operating from an external clock source, this should be easy to provide. In this example, the DSP56001 clock is buffered by a CMOS inverter which subsequently drives the interface circuitry. It is essential that the device used to buffer this clock has a very high input impedance. The oscillator on the DSP56001 ***cannot*** drive a TTL input load.

1.3 Circuit Description

Figure 1-2 is a schematic depiction of the interface circuit. Basically, the interface provides two functions: it controls the refresh cycles and it generates pre-charge delays.

Section “B” of the MC74AC74 generates a refresh request on the rising edge of SCLK and holds the request until the PAL16R4-7 controller executes a refresh cycle and resets the Flip-Flop. As shown in Figure 1-3, the controller defers a refresh cycle until any access currently in progress completes. If the subsequent DSP56001 instruction cycle does not access this PSRAM array, this refresh is transparent. If the subsequent cycle does access this area of memory, then wait states are inserted until the refresh completes.

Section “A” of the MC74AC74 is clocked by the rising edge of $\overline{BS}$, which occurs at the end of each external bus cycle. In the event that the bus cycle which has just ended was an active cycle for the PSRAM array, the PSRAM address decode ($\overline{DS}$ in this example) will be latched into Flip-Flop “A”. The PLD will receive MEMORY BUSY status, indicating that a pre-charge cycle is in progress. The PLD will hold off further PSRAM activity until sufficient precharge delay has elapsed. Note that no extra delay is seen by the DSP56001 if the subsequent cycle does not access this particular PSRAM. If multiple banks of PSRAM are used, bank interleaving strategies can result in most (or all) of the precharge cycles being hidden

behind activity in complementary memory banks. Similarly, if the DSP56001 is executing code out of an external SRAM in another bank, the precharge activity would be transparent.

The ABEL™[1] design file for the PAL16R4-7 is a very simple Mealy type state machine (see Figure 1-5). It controls the chip enabling of the PSRAM as well as the assertion of $\overline{WT}$, which goes to the DSP56001 to hold off bus activity. In addition, the machine provides resets for the external latches. The function of the PLD is shown in the state diagram of Figure 1-3.

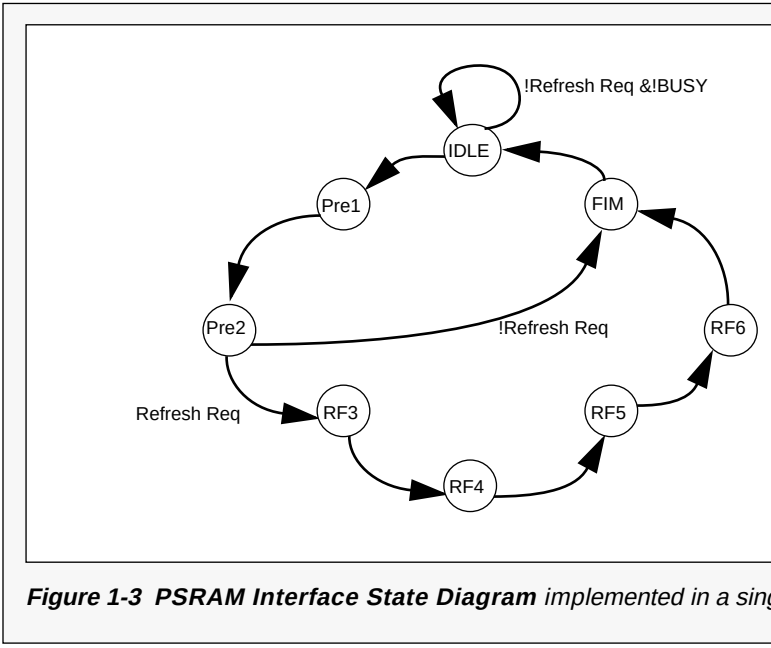


Figure 1-3 PSRAM Interface State Diagram *implemented in a sing*

1. ABEL is a trademark of the data I/O Corporation

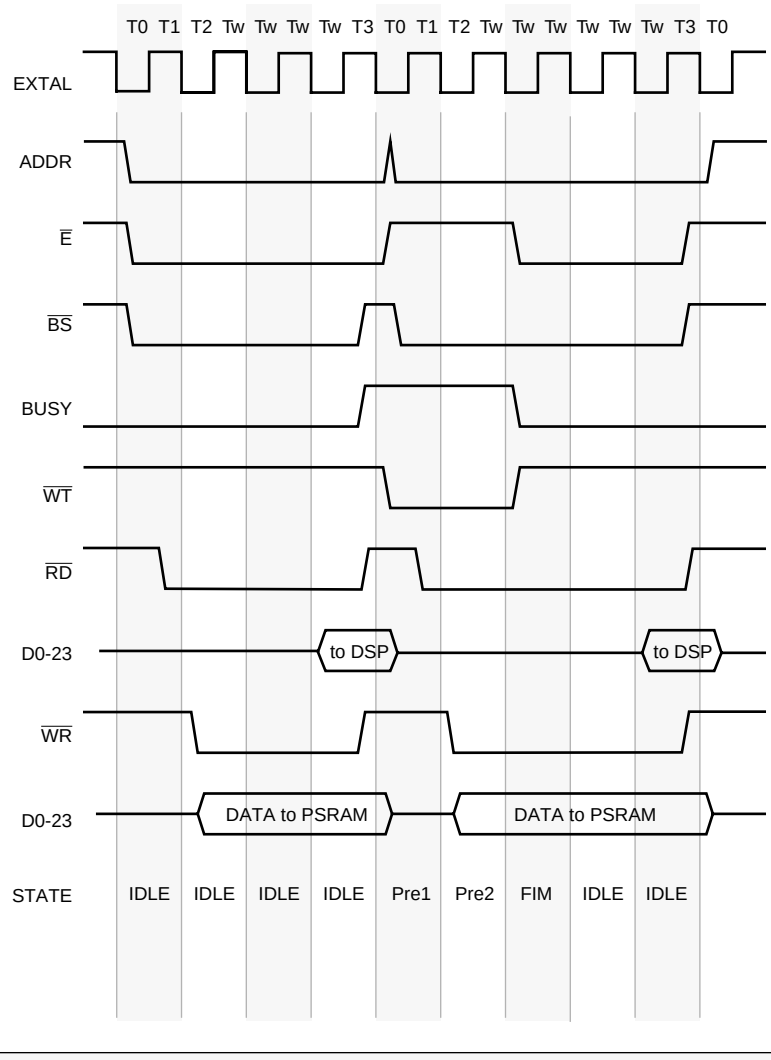


Figure 1-4 DSP56001-to-PSRAM Timing *shows the operation of the controller as it progresses through a pair of successive memory accesses.*

The timing diagram in Figure 1-4 shows the operation

```
0001 |module   pseudo
0002 |title'Pseudo-Static RAM Timing Controller Ver.1
0003 |          MOTOROLA INC.   17 July 1990'
0004 |
0005 |            U01      device'P16R4';
0006 |
0007 |"INPUTS
0008 |    CLK           pin      1;       "DSP56001 Clock "
0009 |    CSin          pin      2;       "EXT:RAM Address decode"
0010 |    Busy          pin      3;       "BUSY F/F"
0011 |    Rreq          pin      4;       "latched request for refresh c
0012 |
0013 |    OE            pin      11;          "OE*"
0014 |
0015 |"OUTPUTS"                           "----REGISTERED OUTPUTS----"
0016 |    Q0            pin      17;      "State bit 0"
0017 |    Q1            pin      16;      "State bit 1"
0018 |    Q2            pin      15;      "State bit 2 & Busy_clr"
0019 |    Q3            pin      14;      "State bit 3 (not used)"
0020 |
0021 |                                    "----COMBINATORIAL OUTPUTS----
0022 |    WTn           pin      19;      "Bus Wait*"
0023 |    Fn            pin      18;      "Clear refresh cycle request
0024 |    CSout         pin      13;      "Chip Select for EXT:RAM"
0025 |
0026 |    High,Low      = 1,0;
0027 |    H,L,C,K,X     = 1,0,.C.,.K.,.X.;
0028 |
0029 |    Qstate        = [ Q2,Q1,Q0 ];
0030 |    Idle          = [ 1,1,1 ];
0031 |    Pr1           = [ 1,1,0 ];
0032 |    Pr2           = [ 1,0,0 ];
0033 |    RF3           = [ 1,0,1 ];
0034 |    RF4           = [ 0,0,1 ];
0035 |    RF5           = [ 0,0,0 ];
0036 |    RF6           = [ 0,1,0 ];
0037 |    FIM           = [ 0,1,1 ];
0038 |
0039 |
0040 |state_diagram Qstate
0041 |    State Idle:   Fn= 1;   CSout = CSin; WTn = 1;
0042 |                  if (!Busy & !Rreq )                    THEN I
0043 |                                                         ELSE P
0044 |    State Pr1:    Fn= 1;   CSout = 1     ; WTn = CSin;      goto P
0045 |    State Pr2:    Fn= 1;   CSout = 1     ; WTn = CSin;
0046 |                  if (!Rreq)                             THEN F
0047 |                                                         ELSE R
0048 |    State RF3:    Fn= 0;   CSout = 1     ; WTn = CSin;      goto R
0049 |    State RF4:    Fn= 0;   CSout = 1     ; WTn = CSin;      goto R
0050 |    State RF5:    Fn= 1;   CSout = 1     ; WTn = CSin;      goto R
0051 |    State RF6:    Fn= 1;   CSout = 1     ; WTn = CSin;      goto F
0052 |    State FIM:    Fn= 1;   CSout = CSin  ; WTn = 1;         goto I
0053 |
0054 |    END
```

Figure 1-5 DSP56001-to-PSRAM PLD Definition for the ABEL™
package which implements the state diagram in Figure 1-3

```
Motorola DSP56000 Macro CrossAssemblerVersion3.0290-09-0615:06:48psram_ex.asm
                                                                    Page 1
1                                  page    255,66,3,3,5
2       ;***************************************************
3       ;     Motorola Austin DSP Operation July 17,1990
4       ;
5       ;   COPYRIGHT (C) BY MOTOROLA INC, ALL RIGHTS RESERVED
6       ;
7       ;*      ALTHOUGH THE INFORMATION CONTAINED HEREIN,    *
8       ;*      AS WELL AS ANY INFORMATION PROVIDED RELATIVE  *
9       ;*      THERETO, HAS BEEN CAREFULLY REVIEWED AND IS   *
10      ;*      BELIEVED ACCURATE, MOTOROLA ASSUMES NO        *
11      ;*      LIABILITY ARISING OUT OF ITS APPLICATION OR   *
12      ;*      USE, NEITHER DOES IT CONVEY ANY LICENSE UNDER *
13      ;*      ITS PATENT RIGHTS NOR THE RIGHTS OF OTHERS.   *
14      ;*
15      ;
16      ;psram_ex.asm pseudo-static ram exerciser
17      ;        --- quick-and-dirty test of P-SRAM prototype board ---
18      ;
19      ;This code configures the SCI SCLK output to generate the P-SRAM
20      ;refresh timing. An incrementing pattern is written to the device
21      ;at X:$1000 and Y:$1000 and then these locations are read and compared
22      ;with the expected data. If an error is detected, an error counter
23      ;is incremented. X:0000 holds the count of errors found while accessing
24      ;X: memory and Y:0000 holds the Y:memory error count.
25      ;
26      ;This quickie only tests the interface for data transfer and refresh
27      ;interference. It does NOT exercise the refresh logic functionality.
28      ;
29      ;At the end of each pass (i.e., when the 24-bit pattern rolls over to 0)
30      ;a pass counter is incremented. This counter is at Y:0001.
31      ;
32      ;The pass counter and the error logs are located in on-chip RAM in order
33      ;to allow (limited) error analysis after any type of “crash”. These
34      ;locations should be cleared before starting the test. Subsequent
35      ;restarts can continue the logging without initializing these locations.
36      ;
37    P:0100      org  P:$100
38
39    P:0100 08F4BE  movep #$2200,X:$FFFE  ;2 wait states in X:, Y:002200
40    P:0102 08F4B0  movep #$0002,X:$FFF0  ;10-bit async mode00002
41    P:0104 08F4B2  movep #$107F,X:$FFF2  ;SCI internal CLK pinconfigured:00107F
42                                         ;TCM=RCM=0, internal clock
43                                         ;SCLK output, prescale = 1:1
44                                         ;divide fosc by 4 * (127+1)
45    P:0106 08F4A1  movep #$0004,X:$FFE1  ;SCLK/PC2 selected as SCLK000004
46    P:0108 08F4A3  movep #$0004,X:$FFE3  ;SCLK pin configured as output 000004
47    P:010A 60F400  move  #>$1000,r0      ;r0 points to the two addresses 001000
48    P:010C 0AFA67  bset  #7,OMR          ;BS*/WT* selected
49    P:010D 221400  move  r0,r4           ;pointer reg. for Y: moves
50    P:010E 45F41B  clr   b#>$000001,x1   ;constant for increment
```

Figure 1-6 *PSRAM Interface Initialization Code used to initialize and run a simple functionality test.* *(sheet 1 of 2)*

```
Motorola DSP56000 Macro Cross Assembler Version 3.02 90-09-06 15:06:48 psra
                                                                        Pag
    000001
51
52  P:0110 8A0000 loop1 move a,X:(r0)a,Y:(r4);store the data in X: a
53  P:0111 C08068       add  x1,b X:(r0),x0Y:(r4),y0;retrieve data a
54                                            ;...form the next data pat
55  P:0112 200045       cmp  x0,a             ;if X: data not correct...
56  P:0113 0BF0A2       jsne X_ERR            ;...bump error count
    00011D
57  P:0115 200055       cmp  y0,a             ;now, check Y: data
58  P:0116 0BF0A2       jsne Y_ERR            ;...and log differences 00
59  P:0118 21AE00       move b1,a             ;this allows data to roll-
60  P:0119 200003       tst  a                ;check for start of new lo
61  P:011A 0BF0AA       jseq COUNT            ;.and increment count if ye
62  P:011C 0C0110       jmp  loop1
63
64
;*********************************************************************
65                X_ERR                       ;** error handler for X:me
66  P:011D 638000     move   X:(0),r3         ;get last count from stora
67  P:011E 000000     nop                     ;...can't use it yet...
68  P:011F 205B00     move   (r3)+            ;bump count...
69  P:0120 630000     move   r3,X:(0)         ;save new count
70  P:0121 00000C     rts                     ;back to the salt mine....
71
72
;*********************************************************************
73                Y_ERR                       ;** error handler for Y:me
74  P:0122 6B8000     move   Y:(0),r3
75  P:0123 000000     nop
76  P:0124 205B00     move   (r3)+
77  P:0125 6B0000     move   r3,Y:(0)
78  P:0126 00000C     rts
79
80
;*********************************************************************
81                COUNT                       ;pass counter
82  P:0127 6B8100     move   Y:(1),r3
83  P:0128 000000     nop
84  P:0129 205B00     move   (r3)+

85  P:012A 6B0100     move   r3,Y:(1)
86  P:012B 00000C     rts
87
88                      END
0   Errors
0   Warnings
```

Figure 1-6 *PSRAM Interface Initialization Code* *(shee*

SECTION 2

A Simple Dynamic RAM Interface for the DSP56001

"The high density of DRAM results from the simplicity of the storage cells; each cell consists of a single transistor and a single capacitor."

Many DSP applications, such as audio special effects, require large amounts of memory. If the system throughput can tolerate a slight reduction in memory access speed, significant cost reductions can be realized by using dynamic RAM (DRAM) in place of static RAM (SRAM). This section presents a simple implementation of a DRAM interface to the DSP56001. Using an array of six MCM514256A-P70 (256K x 4) DRAMs, the circuit provides access to 256K 24-bit words of data space. With the DSP56001 operating from a 33MHz clock, this interface can run with 2 wait states for non-consecutive accesses. For purposes of circuit simplicity, the device's fast page mode is not utilized in the following example.

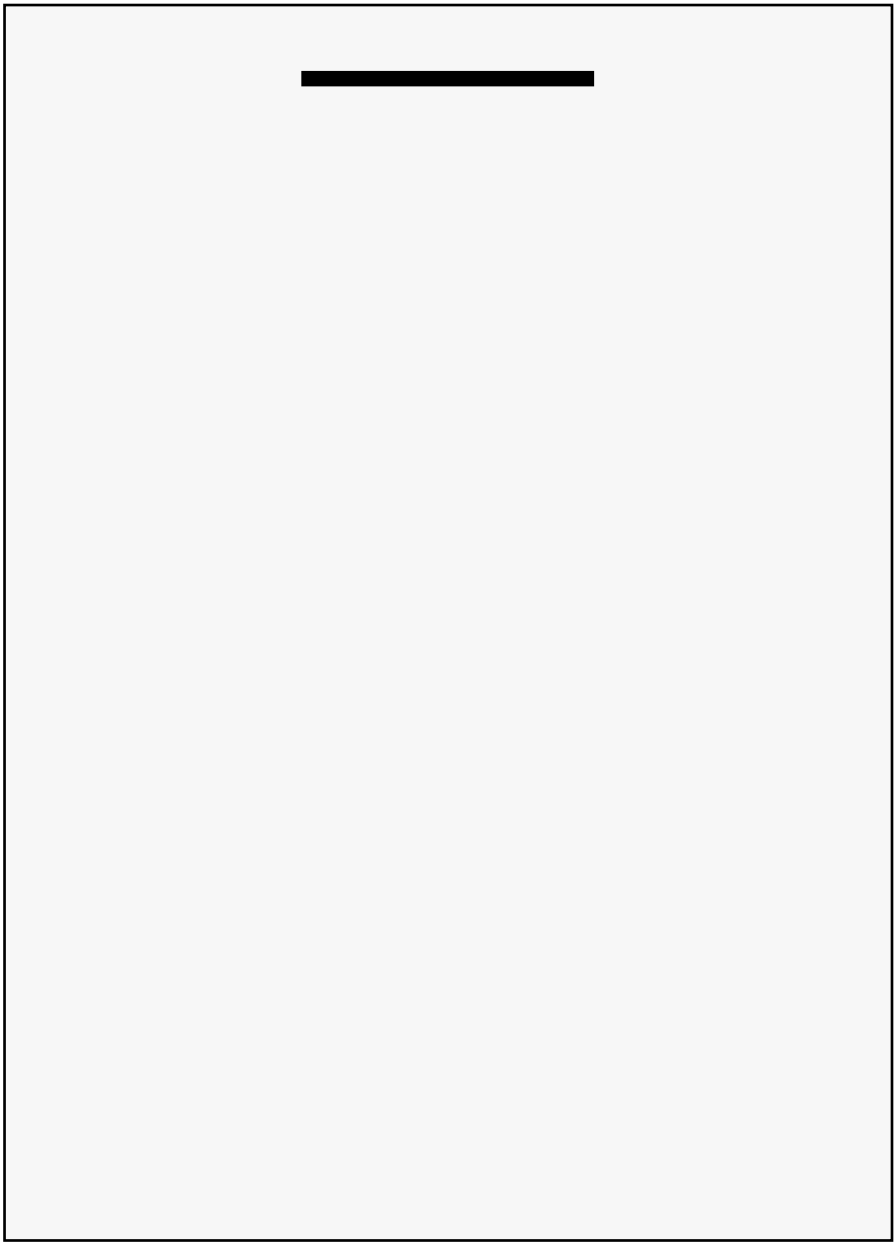
DRAM Basics

The MCM514256A DRAM is a 1 megabit part, organized as 4 sections of 256Kbits each. Each of the 4 sections is subdivided into a 512 x 512 matrix of storage cells, with each storage cell containing one bit of information. The memory cells are uniquely identified by their associated row and column numbers ("address").

In order to reduce the package size, the row addresses and the column addresses of the DRAM cells are multiplexed onto the same pins. Latches on the device are loaded with the column and row portions of the address by the signals Column Address Strobe ($\overline{CAS}$) and Row Address Strobe ($\overline{RAS}$), respectively. During a normal memory access, the cell's row number is placed on the address lines and $\overline{RAS}$ is asserted. After the specified row address hold time, the cell's column number is placed on the same address lines and $\overline{CAS}$ is asserted. This sequence is illustrated in Figure 2-1.

The high density of DRAM results from the simplicity of the storage cells; each cell consists of a single transistor and a single capacitor. During write operations, the capacitor is either charged to the "one" state or discharged to the "zero" state. The charge stored by the capacitor is quite small; typical capacitor values are on the order of 35-125 fF (fF = 1 x 10^{-15} farads). Due to leakage, the capacitor's charge must be periodically "refreshed" in order to retain the stored information. The DRAM circuitry will refresh all of the cells within a row whenever the row is addressed. Thus, by cycling through all 512 possible row address combinations, the entire array is refreshed. On the MCM514256A, no more than 8 ms is allowed to elapse between subsequent refreshes of any particular row. This can be accomplished by refreshing successive rows at 15.6 ms intervals (512 x 15.6ms = 8 ms).

The MCM514256A supports three refresh modes: $\overline{RAS}$ only refresh, $\overline{CAS}$-before-$\overline{RAS}$ refresh and Hidden Refresh (Figure 2-2). $\overline{RAS}$ only refresh requires the processor to place successive row addresses on the address



lines, which would require either more complex interface circuitry or deterministic software action (i.e., interrupts could not be allowed to delay the refresh cycle). The Hidden Refresh mode has the disadvantage of maintaining output data on the DRAM data lines, prohibiting any bus activity during the refresh cycle. $\overline{\text{CAS}}$-before-$\overline{\text{RAS}}$ refresh utilizes an on-chip refresh row counter and three-states the device bus during the refresh cycle. The $\overline{\text{CAS}}$-before-$\overline{\text{RAS}}$ mode is employed in this example because it requires very little external circuitry and provides for bus activity concurrent with the refresh cycle.

A requirement related to refresh is “pre-charge”. During read operations, some of the charge on the cell’s capacitor is lost and the memory device must re-write the information back into the cell. The DRAM automatically performs this “write back” operation after every read, but external access must be delayed until the pre-charge is complete.

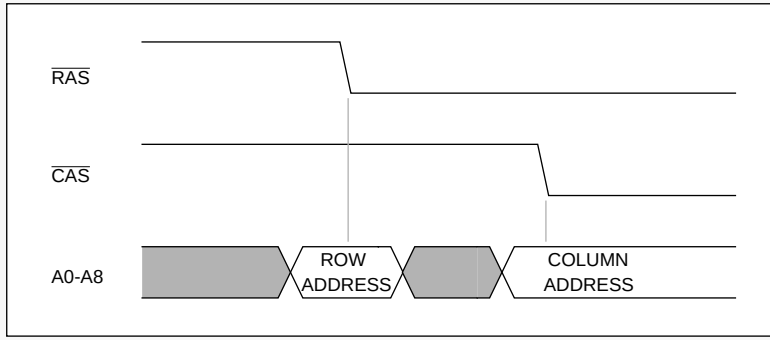


Figure 2-1 DRAM Memory Address Multiplexing *- to reduce the package size, the row addresses and the column addresses of the DRAM cells are multiplexed onto the same pins. Latches on the device are loaded with the column and row portions of the address by the signals Column* Address Strobe ($\overline{\text{CAS}}$) and Row Address Strobe ($\overline{\text{RAS}}$).

Many DRAMs available today offer special access modes which can yield improved performance in specific situations. The MCM514256A DRAM supports a fast page mode in which successive accesses to cells in the same row can be read/written much faster than in normal random access situations. Although this feature would yield improved memory bandwidth in many DSP applications, the need for external address latches and comparators would add significant complexity to the circuit. Since the design goal of this example is minimum parts count (and, therefore, minimum expense), the fast page mode of the MCM514256A is not utilized.

For more detailed information on the MCM41256A, refer to the **Motorola Memory Data Book**, DL113, Rev.5, pp.2-84 through 2-98

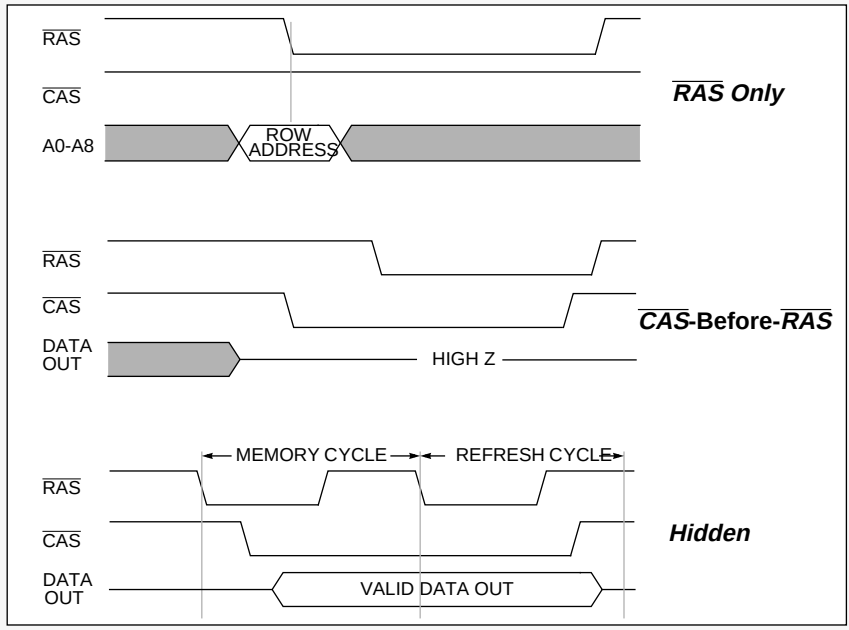


***Figure 2-2 DRAM Refresh Modes** are available but the $\overline{CAS}$ before $\overline{RAS}$ mode has clear advantages for DSP applications.*

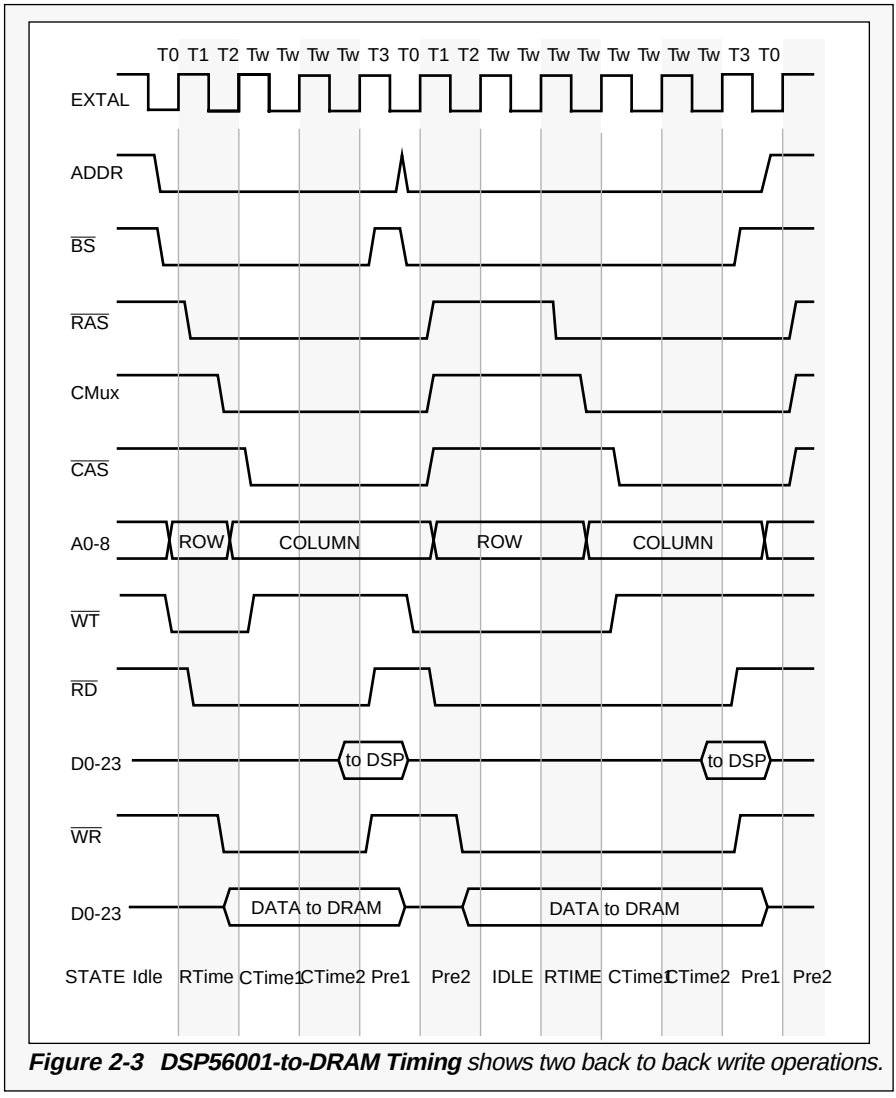


Figure 2-3 DSP56001-to-DRAM Timing *shows two back to back write operations.*

NOTE: *Figure 2-3 shows the timing relationship between the DSP56001 Port A and a DRAM module. The Port A interface is described in* ***Section 1.2 DSP56001 Memory I/0 Basics.***

2.1 Circuit Overview

The circuit in Figure 2-4 is designed to serve as an extension of the MOTOROLA DSP56000ADS Application Development Module (ADM). The SRAM on the ADM should be configured to appear only in the DSP56001 P: memory space. All data memory (X: memory and Y: memory) is provided by the DRAMs on the prototype board. The DRAMs and their interface circuitry are attached to the DSP56001's Data and Address Buses via ADM connector J3.

In order to minimize the component count, the refresh request timing is supplied by the SCI clock, SCLK. Initialization software configures this clock to provide a pulse train with a 15 μs period. Once initialized, the generation of this signal is completely transparent to any code executing on the DSP56001. Figure 2-7 is a listing of the initialization code and a short "pass/fail" memory test routine. The value loaded into the SCI Clock Control Register (SCCR) at X:$FFF2 will vary as a function of the system clock frequency. For a 33 MHz clock, a value of $107F yields the desired refresh rate of 15.6 μs per row. A second task of the initialization software is the selection of the $\overline{BS}/\overline{WT}$ mode of operation, which allows an external source to insert wait states into bus cycles. The interface uses this feature when pre-charge and refresh delays are needed.

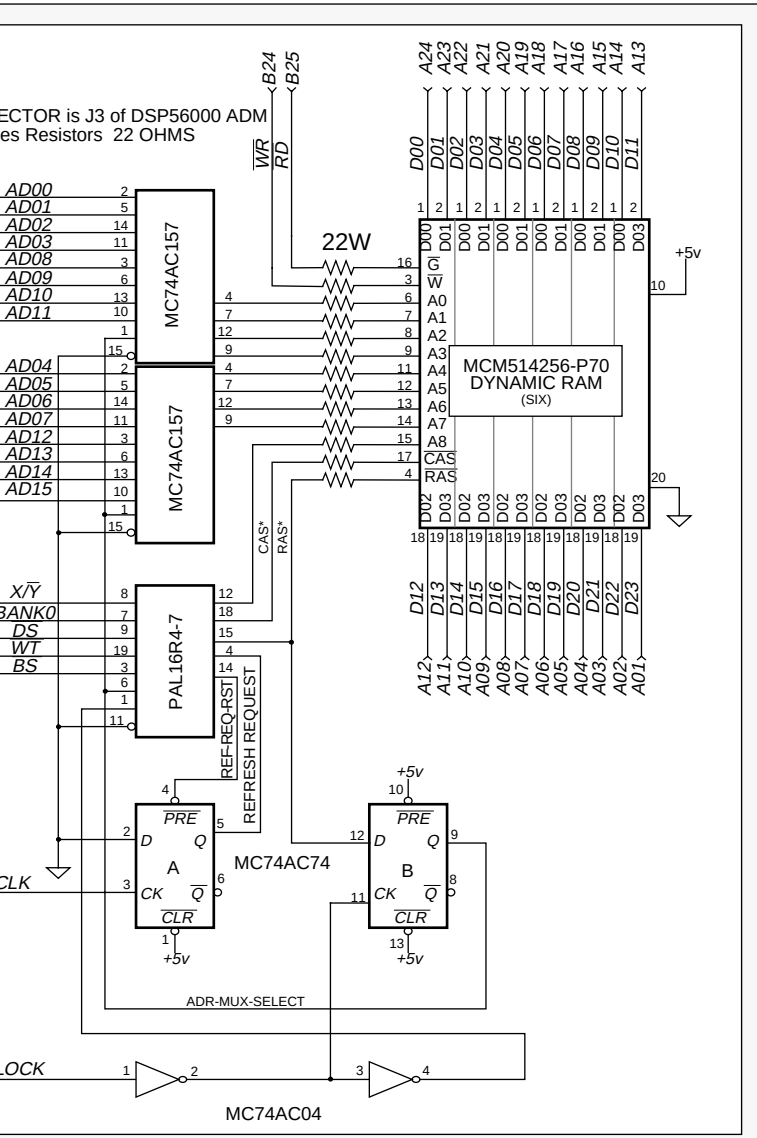
The memory bank consists of six MCM514256A devices. Since each device provides 256K storage cells for each of the 4 data bits, an array of 256K 24-bit

words is formed. The DSP56001 can address 64K 24-bit words in each of its two data spaces, X: memory and Y: memory. This DRAM array can fully populate two of these data spaces. To utilize this potential, a bit from the DSP56001's Port B is used as a bank selector. The configuration of this I/O bit is also handled by the initialization software.

Note of caution: *accesses to the DSP56001's internal peripherals and internal data RAM do not generate external memory cycles and as such, are not subject to control by the bank selection logic.*

The interface requires complementary phases of the same clock which drives the DSP56001. In systems operating from an external clock source, this should be easy to provide. In this example, the DSP56001 clock was buffered by a CMOS inverter which was subsequently used to drive the interface circuitry. It is essential that the device used to buffer this clock has a very high input impedance. The oscillator on the DSP56001 **cannot** drive a TTL input load.

Note that the Vcc and Gnd pins of the 256Kx4 DRAM do not follow the usual polarity conventions. Consult the MCM514256A data sheet for pinout information.



-4 DSP56001-to-DRAM Schematic *provides 256k words of expansion memory to the DSP56000 Application Development System.*

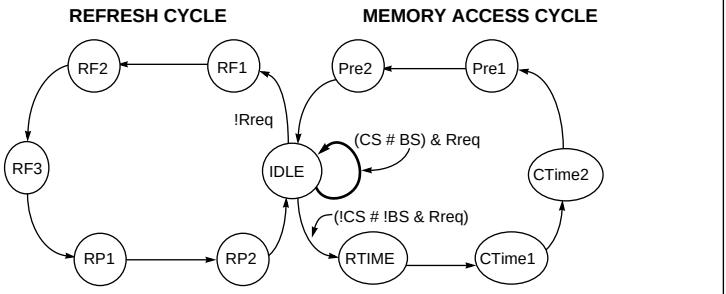
2.2 Circuit Description

The DRAM interface example provides three distinct functions:

- Memory Address Multiplexing
- Refresh Generation
- General Timing and Control

As stated earlier, DRAMs require input addresses to be subdivided into two groups — “row addresses” and “column addresses”. Referring to the schematic in Figure 2-4, two MC74AC157’s multiplex 16 bits of input address onto 8 of the DRAM’s 9 address input pins. The PAL16R4-7 multiplexes the Bank Select bit and X/$\overline{Y}$ onto the 9th DRAM address input pin. Together, these 18 bits delineate two complete banks of data memory, each containing 64K 24-bit words of X: memory and 64K 24-bit words of Y: memory. In this example, bit-0 of Port B drives the bank select signal, BANK0.

Flip-Flop “A” of the MC74AC74 generates a refresh request on the rising edge of SCLK and holds the request until the PAL16R4-7 controller executes a refresh cycle and then resets the Flip-Flop. As shown in Figure 2-5, the controller defers a refresh cycle until any access currently in progress completes. If the subsequent DSP56001 instruction cycle does not access this DRAM array, this refresh is transparent. If the subsequent cycle does access these DRAMs, then wait states are inserted until the refresh completes. The refresh cycle is very similar to a normal access cycle with the exception of $\overline{CAS}$ being asserted before $\overline{RAS}$. The states of $\overline{RD}$,$\overline{WR}$ and the address lines are irrelevant.



-5 *DRAM Interface State Diagram* *implemented in a single PAL.*

By asserting $\overline{CAS}$ before the assertion of $\overline{RAS}$, a refresh cycle is initiated. At the completion of the refresh cycle, the refresh row counter aboard the DRAMs advances in preparation for the next refresh cycle. The interface circuit described here refreshes one row every 15 µs so that all 512 rows are refreshed within the 8 ms required by the DRAMs. In order to reduce the reflected energy on the address lines, they are terminated with 22 ohm series resistors placed as close to the drivers as is practical.

Flip-Flop “B” of the MC74AC74 is clocked from the complementary phase of EXTAL and generates the multiplexer steering control signal ADR_MUX_SELECT. This signal places the “row” portion of the address on the DRAM address lines at the beginning of a memory cycle and later selects the “column” portion of the address at the appropriate point in the cycle.

The PAL16R4-7 PLD performs the timing/control tasks required by the DRAM. The ABEL™ definition of this PLD appears in Figure 2-6. The part is programmed as a Mealy-type state machine and simply advances through a sequence which selects the appropriate address portion (i.e., row or column address), generates $\overline{CAS}$ and $\overline{RAS}$ which the DRAMs require, and generates memory pre-charge delays by forcing the DSP56001 to insert wait states in any bus cycle which occurs immediately after a cycle to same DRAM array.

This pre-charge time is transparent when subsequent memory cycles do not access the same memory devices. For this reason, if more than one DRAM array is present, interleaving the arrays may yield significant improvement in the performance of the memory subsystem.

The timing diagram in Figure 2-3 shows the operation of the controller as it progresses through a pair of successive memory accesses. The diagram illustrates the case where the DRAM array was not accessed during the instruction cycle immediately preceding the start of the diagram. When operating with EXTAL at 33 MHz, the length of each T-period is 15.1 μs. The four Tw-periods in the first access cycle are the result of the DSP56001's Bus Control Register (BCR) being programmed to insert 2 wait states in cycles to this portion of its memory map. During the second cycle, the controller has inserted another 2 wait states (four Tw-periods) in order to allow the DRAM

```
0001 |module   dram6
0002 |title   'Dynamic RAM Timing Controller Ver.2
0003 |   MOTOROLA INC.   06 September 1990'
0004 |
0005 |   U01      device    'P16R4';
0006 |
0007 |   "INPUTS
0008 |   CLK      pin     1; "DSP56001 Clock "
0009 |
0010 |   BS       pin     3; "BUS Strobe from DSP56001"
0011 |   Rreq     pin     4; "latched request for refresh cycle"
0012 |
0013 |   C_Mux    pin     6;       "H = Column Mux Select"
0014 |   Bank0    pin     7;       "H = Select Bank 0"
0015 |   Xsel     pin     8;       "H = Select X:ram"
0016 |   CSin     pin     9;       "EXT:RAM Address decode"
0017 |   OE       pin    11;       "OE*"
0018 |
0019 |"OUTPUTS"      "----REGISTERED OUTPUTS----"
0020 |   Q0       pin    17;       "State bit 0"
0021 |   Q1       pin    16;       "State bit 1"
0022 |   RASn     pin    15;       "State bit 2 also RASn"
0023 |   Rrst     pin    14;       "Refresh Request Reset"
0024 |                  "----COMBINATORIAL OUTPUTS----"
0025 |   WTn      pin    19;       "Bus Wait*"
0026 |   CASn     pin    18;       "Column Address Strobe for DRAM
0027 |   A09      pin    12;       "DRAM address bit 9"
0028 |
0029 |   High,Low     = 1,0;
0030 |   H,L,C,K,X    = 1,0,.C.,.K.,.X.;
0031 |
0032 |   Qstate   = [ Rrst,RASn,Q1,Q0 ];
0033 |   Idle     = [ 1,1,0,0 ];
0034 |   Rtime    = [ 1,0,0,0 ];
0035 |   Ctime1   = [ 1,0,1,0 ];
0036 |   Ctime2   = [ 1,0,1,1 ];
0037 |   Pre1     = [ 1,1,1,1 ];
0038 |   Pre2     = [ 1,1,1,0 ];
0039 |
0040 |   RF1      = [ 0,1,0,0 ];
0041 |   RF2      = [ 0,0,0,0 ];
0042 |   RF3      = [ 0,0,1,0 ];
0043 |   RF4      = [ 0,0,1,1 ];
0044 |   RF5      = [ 0,0,0,1 ];
0045 |   RP1      = [ 0,1,0,1 ];
0046 |   RP2      = [ 1,1,0,1 ];
0047 |
0048 |   XX1      = [ 0,1,1,0 ];"just in case it wakes up lost..."
0049 |   XX2      = [ 0,1,1,1 ];
0050 |   XX3      = [ 1,0,0,1 ];
```

Figure 2-6 PLD Design File *-generated by ABEL™ for the DRAM Int*
(sheet 1 of 2)

```
tate_diagram Qstate
tate Idle:  CASn = 1;WTn = !(!CSin & !BS);
                      if ((CSin # BS) & Rreq)   THEN Idle
                      if (!CSin & !BS & Rreq)   THEN Rtime
                      if ( !Rreq)               THEN RF1;
tate Rtime:   CASn = 1;WTn = !(!CSin & !BS);goto Ctime1;
tate Ctime1:  CASn = 0;WTn = 1;             goto Ctime2;
tate Ctime2:  CASn = 0;WTn = 1;             goto Pre1;
tate Pre1:    CASn = 1;WTn = !(!CSin & !BS);goto Pre2;
tate Pre2:    CASn = 1;WTn = !(!CSin & !BS);goto Idle;

         " --- Refresh States --- "
tate RF1:     CASn = 0; WTn = !(!CSin & !BS);   goto RF2;
tate RF2:     CASn = 0; WTn = !(!CSin & !BS);   goto RF3;
tate RF3:     CASn = 0; WTn = !(!CSin & !BS);   goto RF4;
tate RF4:     CASn = 1; WTn = !(!CSin & !BS);   goto RF5;
tate RF5:     CASn = 1; WTn = !(!CSin & !BS);   goto RP1;
tate RP1:     CASn = 1; WTn = !(!CSin & !BS);   goto RP2;
tate RP2:     CASn = 1; WTn = !(!CSin & !BS);   goto Idle;

tate XX1:     goto Idle;          "if lost, go home PAL..."
tate XX2:     goto Idle;
tate XX3:     goto Idle;

quations
 A09   = (Bank0 & C_Mux) # (Xsel & !C_Mux);

ND
```

***-6 PLD Design File** -generated by ABEL™ for the DRAM Interface (sheet 2 of 2)*

```
Motorola DSP56000 MacroCrossAssemblerVersion3.0290-09-0610:54:50dra

1                                    page    255,66,3,3,5
2   ;*************************************************
3   ;     Motorola Austin DSP Operation  August 15,1990
4   ;
5   ;   COPYRIGHT (C) BY MOTOROLA INC, ALL RIGHTS RESERVED
6   ;
7   ;*  ALTHOUGH THE INFORMATION CONTAINED HEREIN,      *
8   ;*  AS WELL AS ANY INFORMATION PROVIDED RELATIVE    *
9   ;*  THERETO, HAS BEEN CAREFULLY REVIEWED AND IS     *
10  ;*  BELIEVED ACCURATE, MOTOROLA ASSUMES NO          *
11  ;*  LIABILITY ARISING OUT OF ITS APPLICATION OR     *
12  ;*  USE, NEITHER DOES IT CONVEY ANY LICENSE UNDER   *
13  ;*  ITS PATENT RIGHTS NOR THE RIGHTS OF OTHERS.     *
14  ;*
15  ;
16  ;   dram_ex.asm dynamic ram exerciser
17  ;   ---- quick-and-dirty test of DRAM prototype board ----
18  ;
19  ;This code configures the SCI SCLK output to generate the P-S
20  ;refresh timing.  An incrementing pattern is written to the d
21  ;at X:$1000 and Y:$1000 and then these locations are read and
22  ;compared with the expected data.  If an error is detected, a
23  ;error counter is incremented.  X:0000 holds the count of err
24  ;found while accessing X: memory and Y:0000 holds the Y:memor
25  ;error count.
26  ;This quickie only tests the interface for data transfer and
27  ;refresh interference. It does NOT exercise the refresh logic
28  ;functionality. Bit 0 of PORT B is used to select between two
29  ;banks of 64k x 24 X 2,but this is not used in this exercise.
30  ;At the end of each pass (i.e., when the 24-bit pattern rolls
31  ;to 0) a pass counter is incremented.  This counter is at Y:0
32  ;The pass counter and the error logs are located in on-chip R
33  ;in order to allow (limited) error analysis after any type of
34  ;"crash".  These locations should be cleared before starting
35  ;test.  Subsequent restarts can continue the logging without
36  ;initializing these locations.
37  ;
38  P:0100   org P:$100
39  P:0100 08F4BE  movep #$2200,X:$FFFE  ;2 wait states in X:, Y:0
40  P:0102 08F4A2  movep #1,X:$FFE2      ;Port B, Bit 0 is output0
41  P:0104 08F4A4  movep #0,X:$FFE4      ;Port B data is all 0's00
42  P:0106 08F4A0  movep #0,X:$FFE0      ;Port is G.P I/O   00000
43  P:0108 08F4B0  movep #$0002,X:$FFF   ;10-bit async mode 00000
44  P:010A 08F4B2  movep #$107F,X:$FFF2;SCI internal CLK pin configure
45                ;TCM=RCM=0, internal clock
```

Figure 2-7 DRAM Interface Initialization Code *provides both the initialization of the DRAM interface and a simple test of the (Sheet 1 of 2)*

```
          ;TCM=RCM=0, internal clock
          ;SCLK output, prescale = 1:1
          ;divide fosc by 4x(127+1)
10C 08F4A1 movep #$0004,X:$FFE1 ;SCLK/PC2 selected as SCLK000004
10E 60F400 move  #>$1000,r0     ;r0 points to the two addresses 001000
110 0AFA67 bset  #7,OMR         ;BS*/WT* selected
111 221400 mover 0,r4           ;pointer reg. for Y: moves
112 45F41B clrb  #>$000001,x1   ;constant for increment000001

114 8A0000 loop1 move  a,X:(r0)a,Y:(r4)   ;store the data in X: & Y:
115 C08068 add   x1,bX:(r0),x0Y:(r4),y0   ;retrieve data and
                                ;...form the next data pattern
116 200045 cmp   x0,a           ;if X: data not correct...
117 0BF0A2 jsne  X_ERR          ;...bump error count000121
119 200055 cmp   y0,a           ;now, check Y: data
11A 0BF0A2 jsne  Y_ERR          ;...and log differences000126
11C 21AE00 move     b1,a        ;this allows data to roll-over
11D 200003 tst   a              ;check for start of new loop
11E 0BF0AA jseq  COUNT          ;...and increment count if yes00012B
120 0C0114 jmp   loop1

                                ;*******************************
X_ERR                           ;** error handler for X:memory **
121 638000 move  X:(0),r3       ;get last count from storage
122 000000 nop                  ;...can't use it yet...
123 205B00 move  (r3)+          ;bump count...
124 630000 move  r3,X:(0)       ;save new count
125 00000C rts                  ;back to the salt mine....

                                ;*******************************
Y_ERR                           ;** error handler for Y:memory **
126 6B8000 move  Y:(0),r3
127 000000 nop
128 205B00 move  (r3)+          ; refer to X-ERR for comments
129 6B0000 move  r3,Y:(0)
12A 00000C rts

                                ;*******************************
COUNT                           ;pass counter
12B 6B8100 move  Y:(1),r3
12C 000000 nop
12D 205B00 move  (r3)+          ; refer to X-ERR for comments
12E 6B0100 move  r3,Y:(1)
12F 00000C rts

    END
ors
```

-7 DRAM Interface Initialization Code *(Sheet 2 of 2)*

SECTION 3

A Simple ISA Bus Interface for the DSP56001

"The 8 registers which comprise the DSP56001 Host Interface are mapped into the ISA bus I/O space. . .

Communications with the DSP56001, including program bootstrapping, are accomplished via I/O reads and I/O writes to the appropriate register."

The Host Port of the DSP56001 provides much of the logic necessary for interfacing this device to another processor. With very little external logic, this port can be used to interconnect the DSP56001 and an ISA Bus host processor (i.e., a PC-Clone). This brief note describes one implementation of such an interface using only two external parts.

3.1 Interface Circuit Overview

The interface consists of a single PAL22V10 and one MC74ACT245 octal data transceiver. The PLD generates the control signals required by the Host Interface of the DSP56001 ($\overline{HEN}$, HR/$\overline{W}$) as well as the boot mode selection during reset. The schematic of the interface appears in Figure 3-1. The PLD definition is shown in Figure 3-2.

The MC74ACT245 buffers the data lines between the Host Interface and the ISA Bus. The Host Interface address lines are not buffered in this example because the DSP56001 load to these lines is equivalent to that of a typical CMOS buffer. In some cases, adding a buffer to these lines might be desirable.

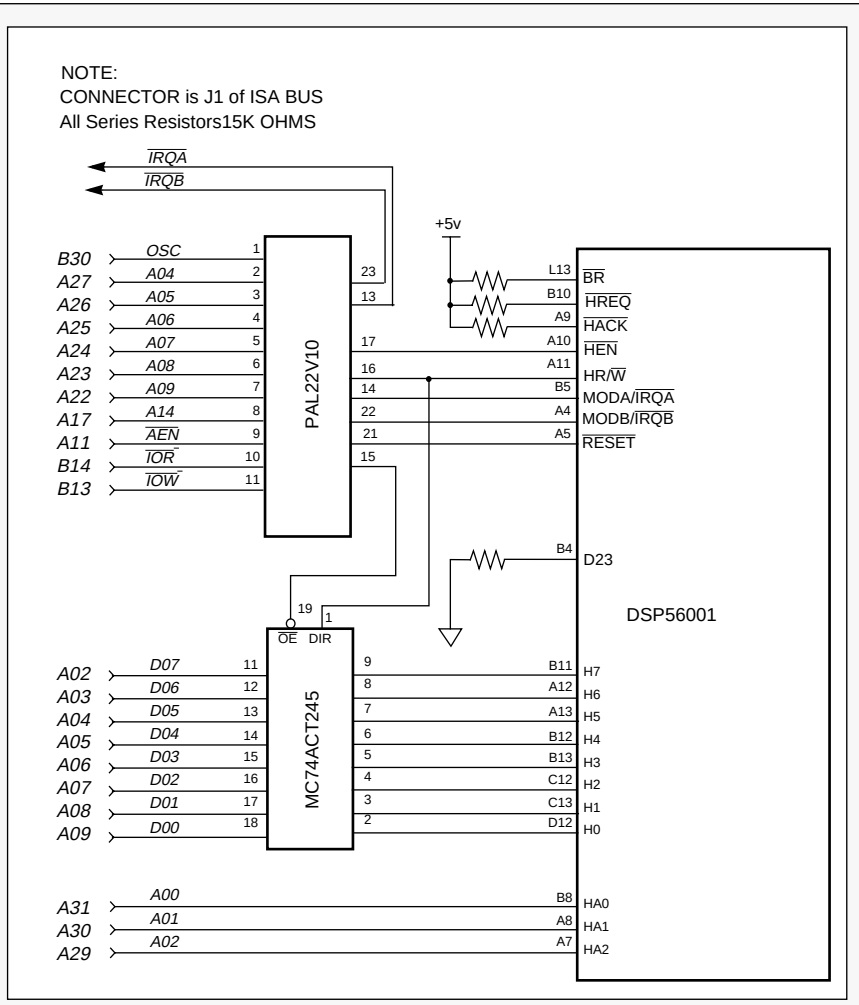


Figure 3-1 DSP56001-to-ISA Bus Interface Schematic *illustrates how simple the circuitry is to connect to the ISA bus.*

3.2 Detailed Circuit Description

The ISA bus delineates two types of bus accesses — memory and I/O. The distinction is made by the use of separate read and write strobes for each type of access. The interface in this example is mapped into the ISA Bus processor's I/O space in the address range $340-$34F.

In order to provide a facility for bootstrap initiation, the $\overline{\text{RESET}}$ pin of the DSP56001 is driven by a latch which is mapped into the host I/O space. Writes to any I/O address in the range $348-$34F will assert $\overline{\text{RESET}}$ to the DSP56001. A write to any address within the range $340-$347 will deassert the $\overline{\text{RESET}}$ latch.

The 8 registers which comprise the DSP56001 Host Interface are mapped into the ISA bus I/O space between address $340-$347 (inclusive). Communications with the DSP56001, including program bootstrapping, are accomplished via I/O reads and I/O writes to the appropriate register. Please refer to the **DSP56001 User's Manual**, especially chapter 10, for a detailed description of the Host Interface and its usage.

The bootstrap mode on the DSP56001 is selected via the processor's MODA and MODB inputs. The PLD provides the proper logic levels on these lines during reset. After reset, the MODA and

MODB inputs reflect the state of the interrupt request inputs $\overline{\text{IRQA}}$ and $\overline{\text{IRQB}}$, thus permitting the normal use of the external interrupt structure of the DSP56001 without forcing constraints on the behavior of the interrupt lines during reset.

During a transfer cycle to/from the Host Interface Registers, the PLD functions as a simple state machine which sequences the control signals for the bus transceiver and the data strobe. Because the ISA bus I/O cycles have relatively long periods, slower PLD's often prove adequate. When attaching this interface to 33 MHz machines, a 15 ns PLD is recommended.

3.3 Timing

Figure 3-3 depicts the timing relationships present during ISA Bus I/O Read cycles and I/O Write cycles. The duty cycle of the processor clock has a 2:1 ratio of low period to high period and a frequency of one-third that of the master oscillator. During an I/O cycle either $\overline{\text{IOW}}$ or $\overline{\text{IOR}}$ will be asserted.

The most critical aspect of this interface is the relationship between Host Enable ($\overline{\text{HEN}}$) and the other interface signals. The Host address lines, HA0-2 and Host Read/$\overline{\text{Write}}$ (HR/$\overline{\text{W}}$) must remain stable during the period in which $\overline{\text{HEN}}$ is asserted. The propagation delays associated with the MC74ACT245 transceiver have been considered

and the complications typical of asynchronous interfaces have been avoided by running the PLD clock from the system oscillator which operates at 3x the processor clock. During successive cycles of the oscillator, the transceiver's direction is established, the transceiver is enabled, and $\overline{\text{HEN}}$ is strobed. The ISA bus indicates the completion of the data transfer by releasing IOR or IOW (as appropriate to the direction of transfer) and the PLD deasserts $\overline{\text{HEN}}$ on the following oscillator cycle. The MC74ACT245 is disabled on the device enable. Successive oscillator cycles

Please refer to the ***DSP56001 User's Manual***, Motorola Document DSP56001/D, for a complete description of the operation of the DSP56001. ■

```
0001e|      module           pcio2
0002e|      title            'ISA (IBM-PC) Interface  Ver.2
0003e|      MOTOROLA INC.    14 February 1991'
0004e|
0005e|      U01              device   'P22V10';
0006e|
0007e|"INPUTS"
0008e|      CLK              pin      1;       "ISA-Bus Clock "
0009e|      AEN              pin      9;       "Address Enable -- NOT DMA cycle"
0010e|      A14,A9,A8        pin      8,7,6;   "ADDRESS Bits14, 09-08"
0011e|      A7,A6,A5,A4      pin      5,4,3,2; "ADDRESS Bits 07-04"
0012e|      IOR,IOW          pin      10,11;   "I/O Read*,I/O Write*"
0013e|      IRQA,IRQB        pin      13,23;
0014e|
0015e|"OUTPUTS"
0016e|      MODA,MODB        pin      14,22;
0017e|      RESET            pin      21;      "Reset* latched"
0018e|      Q2,Q1,Q0         pin      20,19,18;
0019e|      HEN              pin      17;      "HOST ENABLE*"
0020e|      HRw              pin      16;      "HOST R/W*"
0021e|      Ben              pin      15;      "Buffer Enable for 74AC245"
0022e|
0023e|      RESET            ISTYPE            'reg_D,Buffer';
0024e|      Q2,Q1,Q0         ISTYPE            'reg_D,Buffer';
0025e|      HRw              ISTYPE            'reg_D,Buffer';
0026e|
0027e|      High,Low         = 1,0;
0028e|      H,L,C,K,X        = 1,0,.C.,.K.,.X.;
0029e|
0030e|      StateReset       = [RESET];
0031e|      Normal           = [ 1 ];
0032e|      SetReset         = [ 0 ];
0033e|
0034e|      StateDir         = [ HRw ];"Host Read/WRITE*, buffer direction"
0035e|      ReadDir          = [ 1 ];
0036e|      WritDir          = [ 0 ];
0037e|
0038e|      StateNo          = [ Q2..Q0 ];
0039e|      Idle             = [ 0,0,0 ];
0040e|      S1               = [ 0,0,1 ];
0041e|      S2               = [ 0,1,0 ];
0042e|      S3               = [ 0,1,1 ];
0043e|      S4               = [ 1,0,0 ];
0044e|      S5               = [ 1,0,1 ];
0045e|      S6               = [ 1,1,0 ];
0046e|      S7               = [ 1,1,1 ];
0047e|
0048e|      Rcyc             = !AEN & A9 & A8 & !A7 & A6 & !A5 & !A4 & !IOR;
0049e|      Wcyc             = !AEN & A9 & A8 & !A7 & A6 & !A5 & !A4 & !IOW;
0050e|      Addr             = [AEN,A14,A9,A8,A7,A6,A5,A4];
0051e|
0052e|state_diagram StateReset
0053e|      state Normal:    if (Wcyc &  A14) THEN SetReset
0054e|                       ELSE     Normal;
```

Figure 3-2 PLD Definition for the ISA Bus Interface *for the PAL22V10 shown in Figure 3-1* *(Sheet 1 of 2)*

```
0055e|     state SetReset: if (Wcyc & !A14) THEN Normal
0056e|                       ELSE              SetReset;
0057e|
0058e|state_diagram StateDir
0059e|     state ReadDir:  if (!IOW) THEN WritDir
0060e|                       ELSE     ReadDir;
0061e|     state WritDir:  if (!IOR) THEN ReadDir
0062e|                       ELSE     WritDir;
0063e|
0064e|state_diagram StateNo
0065e|    State Idle: HEN = 1; Ben = 1;
0066e|             if (!Rcyc & !Wcyc) THEN Idle “stay put if not for me...”
0067e|                                  ELSE S1;
0068e|State S1: HEN = 1; Ben = 1; goto S2;“allow time to select 74AC245 direction”
0069e|    State S2:  HEN = 1; Ben = 0;goto S3; “Now, enable the 74AC245 output”
0070e|    State S3: HEN = 0;Ben = 0;          “assert HEN*”
0071e|      if (Rcyc # Wcyc)   THEN S3    “...adn loop until the end of the I/O cycle”
0072e|                       ELSE S4;
0073e|    State S4: HEN = 1; Ben = 0;  goto Idle; “deassert HEN*, and quit”
0074e|    State S5: goto Idle;               “these are dummies, just in case..”
0075e|    State S6: goto Idle;
0076e|    State S7: goto Idle;
0077e|equations
0078e|    [Q2,Q1,Q0].ck = CLK;
0079e|    RESET.ck      = CLK;
0080e|    HRw.ck        = CLK;
0081e|
0082e|    MODA   =    !(RESET.Q & !IRQA);
0083e|    MODB   =      RESET.Q &  IRQB;
0084e|Test_vectors
0085e|    ([CLK,Addr,IOW,IOR,IRQA,IRQB] -> [HRw,RESET,MODA,MODB])
0086e|    [ C,^h34,0,1,0,1] -> [0,1,0,1];“write to port, sets write dir.”
0087e|    [ C,^h74,0,1,0,1] -> [0,0,1,0];“write to  reset  address, asserts reset”
0088e|    [ C,^h34,1,0,0,1] -> [1,0,1,0];“read from normal address, reset ”
0089e|    [ C,^h34,0,1,0,1] -> [0,1,0,1];“write to  normal address, deasserts reset”
0090e|    [ C,^h34,1,1,0,1] -> [0,1,0,1];
0091e|    [ C,^h34,1,1,1,1] -> [0,1,1,1];
0092e|Test_vectors
0093e|   ([CLK,Addr,IOW,IOR] -> [HEN,Ben,HRw])
0094e|    [C,^h24,0,1] -> [1,1,0];      “this is NOT for me...., wrong addr”
0095e|    [C,^h34,1,1] -> [1,1,0];      “cycle starts, addresses valid...  “
0096e|    [C,^h34,1,1] -> [1,1,0];
0097e|    [C,^h34,1,0] -> [1,1,1];      “read cycle identified by IOR*     “
0098e|    [C,^h34,1,0] -> [1,0,1];
0099e|    [C,^h34,1,0] -> [0,0,1];
0100e|    [C,^h34,1,0] -> [0,0,1];
0101e|    [C,^h34,1,0] -> [0,0,1];
0102e|    [C,^h34,1,1] -> [1,0,1];
0103e|    [C,^h34,1,1] -> [1,1,1];
0104e|    [C,^h34,0,1] -> [1,1,0];
0105e|
0106e|END pcio2
```

Figure 3-2 PLD Definition for the ISA Bus Interface *(Sheet 2 of 2)*

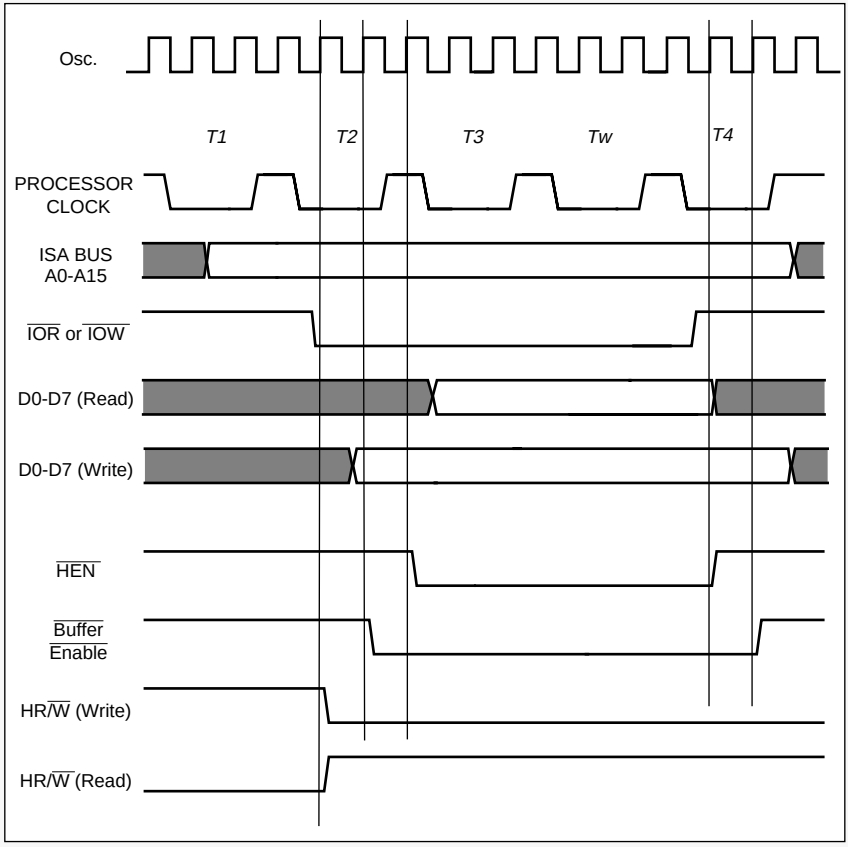


Figure 3-3 *DSP56001-to-ISA Bus Interface Timing shows the timing for both a read and a write operation.*

SECTION 4

Communicate with the DSP56000 Host Interface Using C Language

4.1 Introduction

"The C language source code and the source for the PLD used in the hardware interface are both available on Motorola's Dr. BuB BBS."

Interfacing a DSP56000/1/2 target system to an ISA-bus is only partially complete when the hardware is in place. Download software is the other element required before debugging begins. The specific target hardware determines the types of tasks relegated to the download software. We assume that the user has a target system similar to the DSP-to-ISA interface described in Figure 3-1**.**

4.2 Example Program

The following example uses the DSP56002. The download task can be subdivided into four steps:

- reset the DSP
- verify that the DSP is present (at the expected location)
- transfer code into the DSP's internal P:RAM terminate the boot
- execute the loaded application

Assume that the DSP56002's target host interface (HI) registers have been mapped by the interface

hardware into the ISA-bus I/O address space from 0x340 through 0x347. Refer to the ***DSP56002 User's Manual*, SECTION 5** for a description of the HI register set visible to the host processor. Additionally, the target hardware has a latch attached to the DSP56002 RESET* which can be set (RESET* asserted) by ISA-bus writes to 0x350 (the data is ignored) and cleared by ISA-bus reads from the same address.

The C language program which appears in Figure 4-1 performs all of the download tasks listed above. After some initialization of the screen, the code jumps to the download routine "RESET56." This routine resets the DSP and waits for a short delay to assure that the device receives a reset pulse of adequate duration. Following this, the reset is released and another delay is invoked simply to provide time for the target system to recognize the release of reset and to start executing the routine in its internal bootstrap ROM. This routine will:

- sample the mode selection lines (MODA/$\overline{\text{IRQA}}$, MODB/$\overline{\text{IRQB}}$, MODC/$\overline{\text{NMI}}$) to determine the type of boot desired (from a target processor via the HI in this case)
- branch to the ROM code which will initialize the DSP56002 as required (for booting from the HI)
- start receiving code from the selected bootstrap peripheral (the HI)
- jump to the start of the newly downloaded code upon completion of the boot.

After the reset sequence, the download routine which is running on the ISA-bus host will check for the pres-

ence of a functional DSP56002 host interface by reading four of the HI registers and by checking for presence of the default values. This is a good check of the DSP56002 reset sequence (if it does not reset properly and detect the desired boot mode, the HI will not be enabled) and of the interface hardware's ability to read the HI. If the proper default values are not sensed, the program exits and returns to the command line prompt.

If a DSP56002 is found at the expected address, the ISA download program proceeds to load the initial target code into the DSP's internal P:RAM at addresses P:$0000-01FF. Recall that this process occurs while the DSP56002 on-chip PLL is set to multiply the external oscillator frequency by one, so it can be a (relatively) slow process if the DSP is being run from a slow external clock. To save time when loading DSP routines which do not require the entire on-chip P:RAM space, the boot can be terminated early by setting HOST FLAG bit 0 (as is shown in the listing in Figure 4-1). For brevity, the actual code to be downloaded is present in the example as a statically declared buffer. The user may prefer to write a function to place.LOD or.CLD formatted disk data into a buffer which is passed to the download function.

This example should serve as a beginning for a host download capability through the DSP56000/1/2 Host Interface. The C language source code and the source for the PLD used in the hardware interface are both available on Motorola's Dr. BuB BBS. ■

```
/* hostio.c - host I/F test                    */
/*  compiled with Tubro-C version 2.01         */
/*      20 May 1992                            */

#include <stdio.h>
#include <dos.h>
#include <process.h>

#define   RSTADDR        0x0350   /* ISA-bus address of RESET latch   */
#define   BASE           0x0340   /* ISA-bus address of Host Interface*/
#define   ICR            BASE
#define   CVR            BASE+1
#define   ISR            BASE+2
#define   IVR            BASE+3
#define   RXH            BASE+5
#define   RXM            BASE+6
#define   RXL            BASE+7
#define   TXH            BASE+5
#define   TXM            BASE+6
#define   TXL            BASE+7

#define   DELAY          10000
#define   BOOTSIZE1  7
#define   BOOTSIZE2  9

int  reset56 (int, unsigned char *);

/*********************************************************************/
/* simple code to send an incrementing pattern of bytes to host      */
/*********************************************************************/
                                        /*   org     p:$0            */
                                        /*   begin                   */
          unsigned char  BOOT1[] =
                  {       0x08,0xF4,0xA0,   /*   movep#$0001,x:$FFE0    */
                          0x00,0x00,0x01,   /*                          */
                          0x08,0xC8,0x2B,   /*   movep   A0,x:$FFEB     */
                          0x00,0x00,0x08,   /*   inc     A              */
                          0x0A,0xA9,0x81,   /*   jclr    #1,x:$FFE9,*   */
                          0x00,0x00,0x04,   /*                          */
                          0x0C,0x00,0x00};  /*   jmp    <begin          */

 void main()
{
          unsigned        char     i,j;
          int                      k;

          system("cls");

/*********************************************************************/
/* boot the 56002 with the patterns-to-host routine                   */
/*********************************************************************/
    printf("\n BOOTING 56002-Incrementing Patterns to Host");
    if (reset56(BOOTSIZE1,BOOT1) == -1)
       exit(-1);
```

Figure 4-1 *Example Program of DSP56000 Host Interface Using C Language (sheet 1 of 3)*

```
 printf("\n TESTING DATA READ (56002-to-HOST)\n");
  k = 0xFFFF;
  outportb (ICR,0);                 /* assure proper mode, DMA off,etc  */
  while ((inportb(ISR) & 0x01)==0); /* wait for RXDF == 1               */
  j = (inportb(RXL) & 0xFF);        /* get first pattern                */
 do {

 while ((inportb(ISR) & 0x01) == 0);            /* wait for RXDF == 1   */
       i = (inportb(RXL) & 0xFF);               /* get next pattern     */
       if(i != ((j+1) & 0xFF))                  /* check and advise     */
           printf("\n ERROR!   Rcvd: %2X   Expected: %2X",i,j);
      j = i;                                    /* re-sync pattern      */
      } while(k—);
   printf("\n READ TEST COMPLETE\n\n");
}

/*********************************************************************/
/* routine to reset and boot the DSP56001/2                          */
/*********************************************************************/
int reset56( int codesize, unsigned char *codeptr)
  {
   unsigned char   icr_rd,cvr_rd,isr_rd,ivr_rd;
   int             i,j,k;

/* first, assert reset. Leave reset asserted for a while before       */
/* releasing it...give the DSP time to exit reset, sample the         */
/* MODA,MODB,MODC pins and start the bootstrap code...                */

  outp ((int)RSTADDR,0);                        /* reset the DSP */
  for (k=0; k<DELAY; k++);                      /* wait          */
  inportb (RSTADDR);                            /* clear reset   */
  for (k=0; k<DELAY; k++);                      /* wait again    */
  printf ("\nRESET CYCLED\n");                  /* eye candy     */

/* then, verify that DSP56002 is present: read the icr,cvr,isr,ivr    */
/* ...and look for default values in these registers.   Refer to the  */
/* DSP56000/1/2 Family User's Manual for a complete description of    */
/* Host Interface (Port-B) and its register set.                      */

  icr_rd = inportb(ICR);  /* Interrupt Control Register s/b 0x00       */
  cvr_rd = inportb(CVR);  /* Command Vector Register s/b 0x12          */
  isr_rd = inportb(ISR);  /* Interrupt Status Register s/b 0x06        */
  ivr_rd = inportb(IVR);  /* Interrupt Vector Register s/b 0x0F        */

   printf
   ("\n HOST I/F RETURNED:  ICR: %2X  IVR: %2X  ISR: %2X  IVR: %2X\a",
                    icr_rd,cvr_rd,isr_rd,ivr_rd);
   if ((icr_rd != 0x00) || (cvr_rd != 0x12)
           || (isr_rd != 0x06) || (ivr_rd != 0x0F))
      {                                 /* if default values        */
      printf("\n\a RESET FAILED! ");    /* NOT found, advise        */
      return(-1);                       /* and return error         */
```

Figure 4-1 *Example Program of DSP56000 Host Interface Using C Language (sheet 2 of 3)*

```
    }
 else
    {                                          /* BOOT THE DSP         */
    for (i=0, j=0; i<codesize; i++)            /* NOTE: always send    */
        {                                      /* the lsbyte last      */
        while ( (inportb(ISR) & 0x02) != 2); /* wait for TXDE == 1   */
        outportb(TXH,codeptr[j++]);            /* send upper  byte     */
        outportb(TXM,codeptr[j++]);            /* send middle byte     */
        outportb(TXL,codeptr[j++]);            /* send low    byte     */
        outportb(ICR,0x08;                     /* terminate Host Boot  */
    return(0);
    }
}
```

Figure 4-1 *Example Program of DSP56000 Host Interface Using C Language (sheet 3 of 3)*

INDEX

REFERENCES

1. Eggebrecht, Lewis, Interfacing to the IBM Personal Computer, Howard W. Sams & Company, 1988

2. *DRAM Refresh Modes*, Motorola Application Note AN987

3. Motorola's DSP56000/DSP56001 Digital Signal Processor User's Manual, DSP56000UM/AD, Rev.2

4. Motorola's DSP56001 56-Bit General Purpose Digital Signal Processor, Advance Information DSP56001/D, Rev.1

5. Motorola FACT Data, DL138, Rev.1

6. Motorola's MCM514256A Data sheet, Motorola Memory Data, DL113, Rev.5, pp.84-98

7. *Page, Nibble, and Static Column Modes...*, Motorola Application Note AN986

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

9. PAL® Devices Databook, Advanced Micro