AN1751/D
Rev. 01, 05/98

DSP563xx Port A Programming

by Iantha Scheiwe

The DSP56300 expansion port, Port A, allows the DSP programmer to expand memory space accessible to the DSP core or to expand memory-mapped I/O. The interface is straightforward, and external memory is easily and quickly retrieved through the use of DMA or simple MOVE commands. This application note describes:

- The hardware and software configurations required to connect the DSP core to external SRAM and DRAM
- Examples of moves to and from external memory
- Examples of DMA accesses

Contents

Part 1 Configuration

Part 1 of this application note describes system requirements and the hardware configuration that you must establish between the DSP and off-chip memory devices prior to programming. Part 2 describes the programming required to interface with SRAM and DRAM. The techniques used in this application note to interface to external devices enable you to interface to any device that uses Port A. The code in this document was developed using the DSP56303EVM and DSP56301ADM boards. You can use other DSP56300 Family EVMs, but the memory maps for each device may vary. For complete timing information on the connections between Port A and memory, see the external memory interface (Port A) chapter in the *DSP56300 Family Manual*.

1.1 Performance

The speed of memory access through Port A depends largely on the speed of the memory used in a system. Port A provides a programmable number of wait states that correspond to the specifications of the memory used. A minimum of one wait state is required for external accesses. Any other timing issues are determined by system constraints and delays.

The DSP56300 family devices contain between 8K and 34K of on-chip memory. In many applications and systems, the core processor must access additional memory. Port A enables expansion of the on-chip memory space to 12 M x 24 bits of external memory. The memory is easily accessible to the core for processing large blocks of data. With this port, you can access a single piece of data using a common MOVE instruction or transfer a large block into on-chip memory space for faster execution using a DMA transfer. With DMA transfers, you can transfer data from external memory to internal memory and to other peripherals, such as the SCI and ESSI.

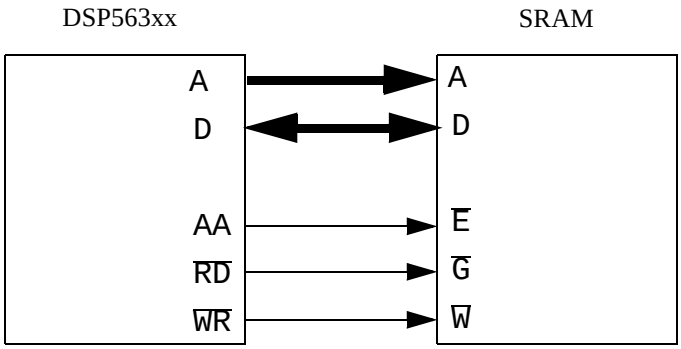
1.2 System Set-up

In order to access external memory and peripheral devices through Port A, it is important to understand pin set-ups as well as internal register configuration.

1.2.1 DSP56300 to Memory Connections

On the EVM and ADM boards, the memory connections are already made, if the proper jumpers are in place. To access SRAM on the DSP56303EVM, J9 pins 2-3 must be connected. On the DSP56301ADM, a jumper must be placed on JP1. **Figure 1-1** depicts the DSP-to-SRAM memory connection. SRAM is accessible on both EVM and ADM boards using the Address Attribute Register 0 (AAR0). However, you can also use Address Attribute Registers 1, 2, or 3 when configuring a system.

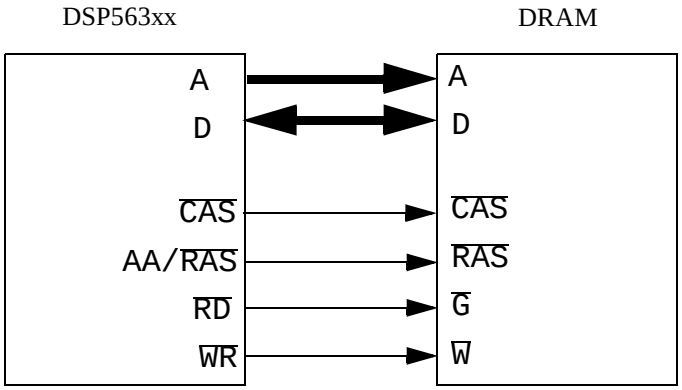
Figure 1-1. DSP-to-SRAM Connection



DRAM is available on the DSP56301ADM board, but not on the DSP56303EVM. To access DRAM on the ADM board, the jumper on JP2 must be set.

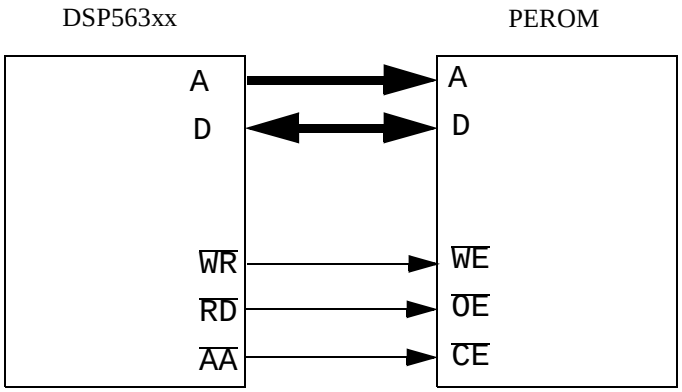
For the DSP56301ADM, DRAM is accessible to Port A using the Address Attribute 3 (AAR3) Register. **Figure 1-2** depicts the DSP-to-DRAM memory connection.

Figure 1-2. DSP-to-DRAM Connection



Both the DSP56303EVM and DSP56301ADM boards have on-board Flash PEROM that you can access using the Address Attribute 1 (AAR1) chip select. **Figure 1-3** depicts the DSP-to-Flash PEROM connection.

Figure 1-3. DSP-to-Flash PEROM Connection



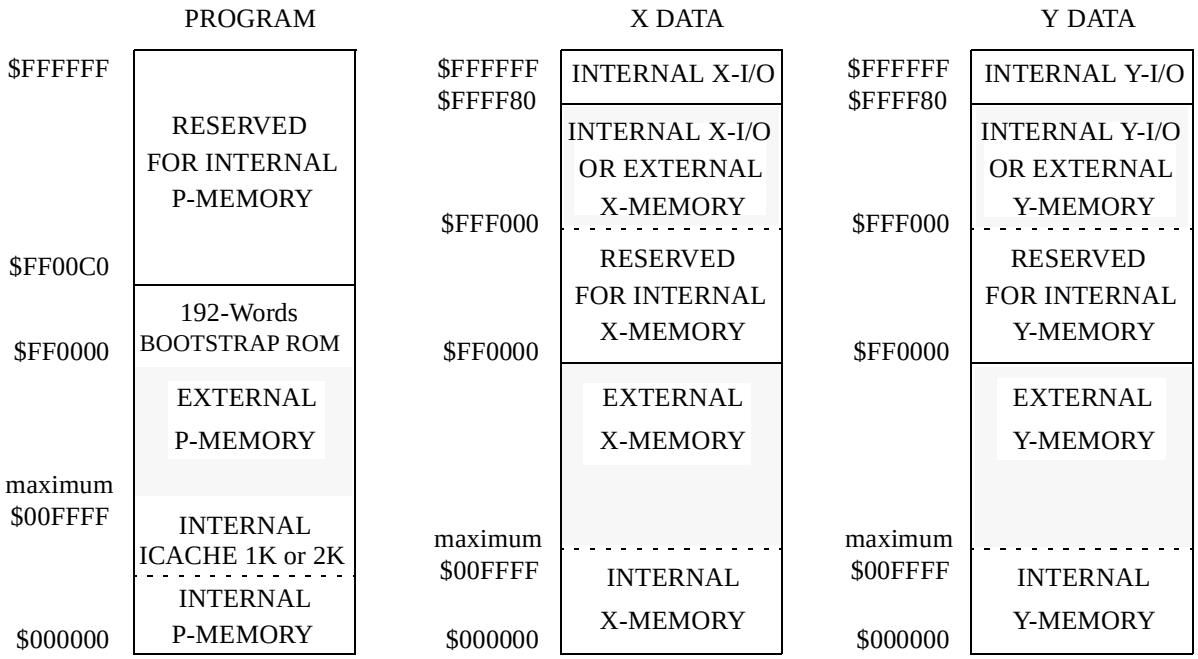
On the DSP56303EVM, the code in flash memory is run at reset. Though the following sections do not give an example of Flash PEROM usage, the program `flash.asm` is included with all DSP563xxEVMs. This program allows you to download any desired program into the flash memory so that it can be run at processor reset. Information on using `flash.asm` is given in each EVM/ADM user's manual.

1.2.2 Memory Map

To prevent overlap in the mapping of external devices or interference with internal core operations, consult the memory map before initializing access to memory.

Figure 1-4 depicts the core memory map. Devices can be mapped anywhere in the external memory spaces for X, Y, and Program (P) memory.

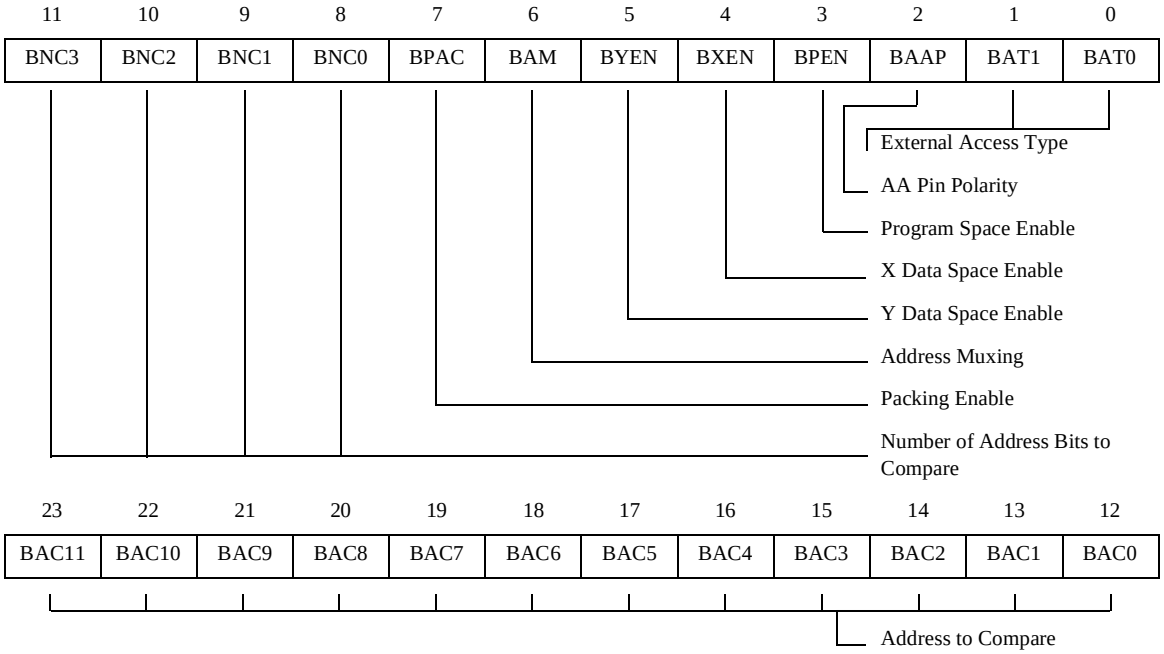
Figure 1-4. DSP56300 Core Memory Map



1.2.2.1 Address Attributes

This section describes how to map an external device to the DSP memory space. The Address Attribute Register in **Figure 1-5** sets up the memory access address and range for the external device. It also defines whether you are using SRAM or DRAM. If Address Attributes are not initialized, the DSP is not notified that a device is available in the external memory space. The Address Attribute line acts as a chip select for the external device. Any combination of X, Y, and P external memory space can be made available. For a complete explanation of each of the bits, see the *DSP56300 Family Manual*. Only the bits used in the examples in this application note are explained. **Part 2.3** explains the Address Muxing option (BAM), and **Part 2.4** explains the Packing mode option (BPAC).

Figure 1-5. Address Attributes Register (AAR0-AAR3)



Bit/Field Definitions:

- **AARx[23:12]**, Address Compare (BAC)

  AARx[11:8], Number Compare (BNC)

  These two sets of bits are related. AARx[23:12]:BAC[11:0] define the memory address at which you can access the external device. AARx[8:11]:BNC[3:0] define the range of the address that is compared for external memory accesses. For example, to map a device to the space from $100000 to $104000, set BAC[11:0] to $104 and BNC[3:0] to $A. The BNC bits act as a gatekeeper. When the address request is given, the BNC bits verify that the first bits, in this case ten, match the memory space where the device is mapped externally. If those bits pass the compare, the core is enabled to retrieve data from the external device. When setting up this address space, it is extremely important to make sure that external devices are not overlapping.

- **AARx[5]**, Y Space Enable (BYEN)

  AARx[4], X Space Enable (BXEN)

  AARx[3], Program Space Enable (BPEN)

  When these bits are activated, access to external Y data space, X data space, and program space is enabled. You can map a device to be accessible in any combination of these data spaces at a specified data range. If BYEN=0, BXEN=1, and BPEN=0, then access to $100000 in X data space is allowed (if that address range is the one specified in the BAC and BNC bits) and a move to that address in Y or P space is rejected.

NOTE: The EVM boards have a contiguous memory space, so $100000 in X and $100000 in Y space actually are the same physical memory locations. Therefore, if both X and Y space are enabled at $100000, a write to $100000 in Y space also appears in X space.

- **AARx[2]**, Address Attribute (AA) Pin Polarity (BAAP)

  This bit defines the device chip select ($\overline{AA}/\overline{RAS}$) as an active low or an active high pin. If it is cleared, the pin is active low. If it is set, the pin becomes active high.

- **AARx[1:0]**, External Access Type (BAT)

  These bits determine the type of memory or peripheral device that is accessed. If you want to access a device other than memory, set the access type to SRAM (BAT[1:0] = 01) as the default. See **Table 1-1**.

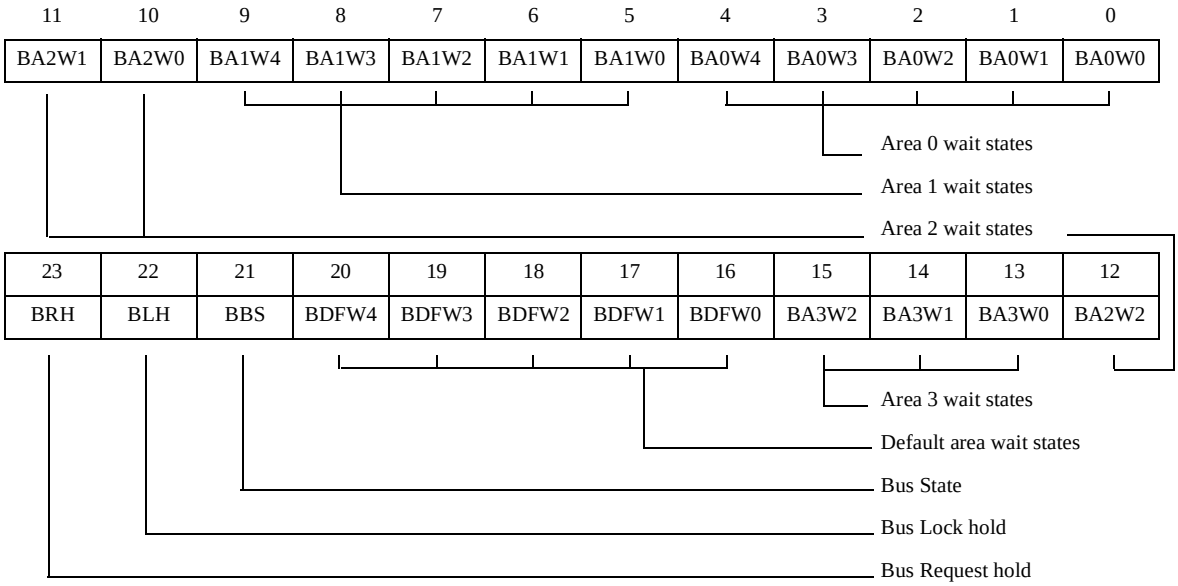
BAT1	BAT0	External Access Type
0	0	Reserved
0	1	Static RAM access
1	0	DRAM access
1	1	Reserved

Table 1-1. BAT1 and BAT0 External Access Types

1.2.2.2 Bus Control

The Bus Control Register (BCR) in **Figure 1-6** defines the number of wait states required to access each of the external devices in order to achieve proper operation. It also controls the arbitration pins.

Figure 1-6. Bus Control Register (BCR)



Bit/Field Definitions:

- **BCR[20:16]**, Default Area Wait States (BDFW)

  BCR[15:13], Area 3 Wait States (BA3W)

  BCR[12:10], Area 2 Wait States (BA2W)

  BCR[9:5], Area 1 Wait States (BA1W)

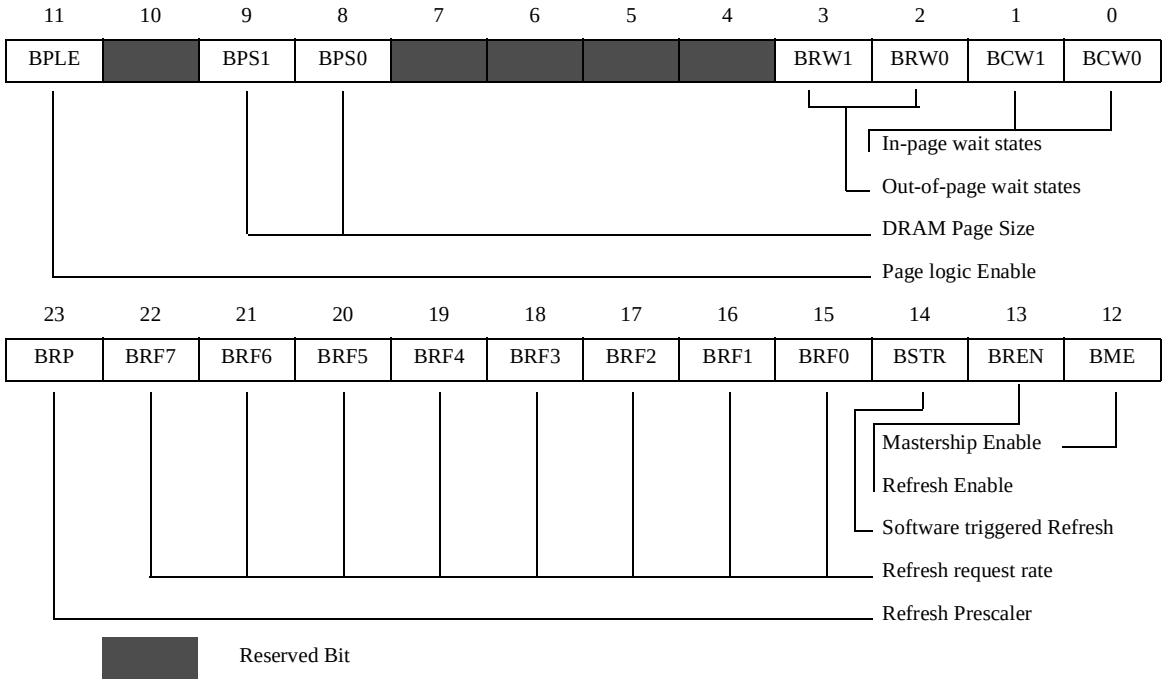
  BCR[4:0], Area 0 Wait States (BA0W)

  Each of these bit sets operates the same way. Each defines the number of wait states inserted into an external access for memory setup time. The external memory interface (Port A) chapter in the *DSP56300 Family Manual* gives details on timing for memory devices. Each set of bits defines the wait states for a different external device. The area wait state bits define the states required for accessing the devices mapped by the Address Attribute registers. The BDFW bits are inserted into each external access not defined by an attribute register. These bits should have a minimum value of one wait state, since the access type is SRAM (as stated earlier for default devices), and SRAM memory access requires at least one wait state.

1.2.2.3 DRAM Control Register

The DRAM Control Register (DCR), shown in **Figure 1-7**, defines wait states for DRAM accesses, refresh rates, and other DRAM requirements.

Figure 1-7. DRAM Control Register (DCR)



Bit/Field Definitions:

- **DCR[23]**, Refresh Prescaler (BRP)

  This read/write control bit controls a prescaler in series with the refresh clock divider. If this bit is cleared, the prescaler is bypassed. However, if it is set, a divide-by-64 prescaler is connected in series with the refresh clock divider. This bit is cleared during hardware reset.

- **DCR[22:15]**, Refresh Rate (BRF)

  These read/write bits control the refresh request rate. The rate can range from one to 256. If enabled, a refresh request is generated each time the refresh counter reaches zero.

- **DCR[14]**, Software Triggered Refresh (BSTR)

  This read/write status bit generates a software-triggered refresh request. When set, a refresh request is generated and executed to all DRAM banks. After the access is executed, the DRAM controller hardware clears the BSTR bit. The refresh cycle length depends on the BRW[1:0] bits.

- **DCR[13]**, Refresh Enable (BREN)

  When BREN is set, the refresh counter is enabled, and a refresh request is generated each time the refresh counter reaches zero. If it is cleared, the refresh counter is disabled, and software can trigger a refresh request using the BSTR bit.

- **DCR[12]**, Mastership Enable (BME)

  This bit enables/disables the interface to a local DRAM for the DSP. If it is cleared, the $\overline{RAS}$ and $\overline{CAS}$ pins are three-stated when mastership is lost. Therefore, you must connect an

external pull-up resistor to these pins. The DSP DRAM controller assumes a page fault each time mastership is lost. A DRAM refresh then requires bus mastership. If BME is set, the $\overline{RAS}$ and $\overline{CAS}$ pins are always driven from the DSP, and the DRAM refresh can be performed even if the DSP is not the bus master.

- **DCR[11]**, Page Logic Enable (BPLE)

  If the BPLE bit is set, it enables the page logic. Each in-page identification causes the DRAM controller to drive only the column address. When BPLE is cleared, the page logic is disabled, and the DRAM controller always accesses the external DRAM in out-of-page accesses. This is useful for low-power dissipation.

- **DCR[9:8]**, DRAM Page Size (BPS)

  BPS[1:0] defines the size of the external DRAM page. The internal page mechanism abides by these bits only if the page logic is enabled. **Table 1-2** shows the encoding of these bits.

BPS1	BPS0	Column Address Width	DRAM Page Size
0	0	9 bits	512 words
0	1	10 bits	1K
1	0	11 bits	2K
1	1	12 bits	4K

Table 1-2. Encoding of Bits BPS1 and BPS0

- **DCR[3:2]**, Out-of-page Wait States (BRW)

  The BRW[1:0] bits define the number of wait states that are inserted in each DRAM out-of-page access. **Table 1-3** shows the encoding of these bits.

BRW1	BRW0	DRAM External Access
0	0	4 w.s. for each out-of-page access
0	1	8 w.s. for each out-of-page access
1	0	11 w.s. for each out-of-page access
1	1	15 w.s. for each out-of-page access

Table 1-3. Encoding of Bits BRW1 and BRW0

- **DCR[1:0]**, In-page Wait States (BCW)

  These bits define the number of wait states that are inserted in each DRAM in-page access. **Table 1-4** shows the encoding of these bits.

BCW1	BCW0	DRAM External Access
0	0	1 w.s. for each in-page access
0	1	2 w.s. for each in-page access
1	0	3 w.s. for each in-page access
1	1	4 w.s. for each in-page access

Table 1-4. Encoding of Bits BCW1 and BCW0

For more information on calculating wait states for memory devices, see the *DSP56300 Family Manual*.

Part 2 Accessing External Devices

The following programming examples for accessing external devices using Port A begin with a basic access to one memory location in external SRAM. An example in which the external memory space overlaps with the internal memory space follows. Such overlaps should be avoided in practice, but we include this example as an aid to troubleshooting code. The third example is a multiplexed access using DMA. The DMA access is adjusted to show how to access an 8-bit device using Packing mode. The last example uses DMA to access DRAM and SRAM.

2.1 Simple SRAM Access Using Move Instruction

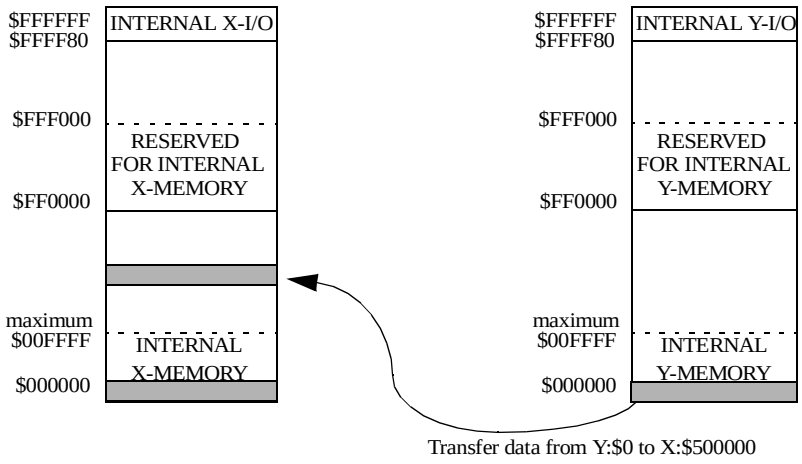
This run exercises a single, generic access to external SRAM through Port A.

The following Port A options are used for this exercise:

- Clock Frequency = 68 MHz
- Packing mode disabled
- Address muxing disabled
- X space enabled; Y and P disabled
- 64 words of data from Y:$000000 to X:$500000
- SRAM using AAR0
- Transfer method: MOVE instruction

As **Figure 2-1** shows, the code transfers 64 words of data from Y:$0 to X:$500000-$500040. The transfer is done using a MOVE instruction in a DO loop.

Figure 2-1. Data Transfer from Y to X Memory



Read the following register explanations before looking at the code.

2.1.1 Initialization

In setting up a transfer to Port A, initialize the Address Attributes and Bus Control registers. The equates for this example are set up as follows. In addition to the AAR0 and BCR registers, set the

PCTL (PLL Control Register) at the beginning of the program. If the code is running on the 66 MHz DSP56303EVM, set PCTL to $040003 (68 MHz operation).

Since the SRAM on the DSP56303EVM connects to the AAR0 enable line, the code uses the AAR0 Register to access the SRAM. The DSP has four Address Attribute registers.

```
AAR0V1          EQU   $500111
                      ; SRAM access
                      ; Address Attribute Register 0
                      ; [23:12] = BAC[11:0] = $500 (Address to compare)
                      ; [11:8] = BNC[3:0] = $C (Num of add to compare)
                      ; [7] = BPAC = 0 (Packing disabled)
                      ; [6] = BAM = 0 (Address muxing disabled)
                      ; [5] = BYEN = 0 (Y memory disabled)
                      ; [4] = BXEN = 1 (X memory enabled)
                      ; [3] = BPEN = 0 (P memory disabled)
                      ; [2] = BAAP = 0 (Active low enabled)
                      ; [1:0] = BAT[1:0] = 01 (SRAM access)
```

Note that BAC[11:0] = $500, which corresponds to the address where we want to store the data in external memory. We are using only a small section of data (64 words), so we use all 12 bits on the compare. Therefore, the BNC[1:0] value equals $C. If the first 12 bits match the address given, the memory access can occur. DMA isn't used, so Packing mode isn't invoked. Also, Address Muxing is not needed. The code accesses only external X memory, so we enable that space and disable Y and P memories. The SRAM is enabled using an active low signal.

```
BCRV            EQU   $012422
                      ; Bus Control Register
                      ; [23] = BRH = 0 bus request hold off
                      ; [22] = BLH = 0 bus lock hold off
                      ; [21] = BBS = 0 bus state
                      ; [20:16] = BDFW[4:0] = 1 default area wait states
                      ; [15:13] = BA3W[2:0] = 1 area 3 wait states
                      ; [12:10] = BA2W[2:0] = 1 area 2 wait states
                      ; [9:5] = BA1W[4:0] = 1 area 1 wait states
                      ; [4:0] = BA0W[4:0] = 2 area 0 wait states
```

In the Bus Control Register, this access is given two wait states. All SRAM accesses require at least one wait state. The other areas are set to default values of one wait state each. With this code, arbitration is not a concern. The most significant three bits, BRH, BLH, and BBS, remain at their default value of zero.

2.1.2 Program

The code that actually invokes the transfer of data from internal to external memory has three parts. In the first part, the main program defines data values. In the second part, it initializes the registers. Finally, it moves data from internal memory to external memory.

```
         include 'portaequ.asm'        ; this file holds equates for BCRV and AAR0
                 org     y:$0
; Assembly-Time Equates
START            EQU   $000400         ; start address for code
LINEAR           EQU   $FFFFFF         ; set up linear addressing mode
; Address Equates
DATA_START       EQU   PATT
MEM_START        EQU   $500000
LOOP2            EQU   $40

        org p:START
        nop
MAIN_INIT
        movep    #PCTL,x:M_PCTL        ; set PLL
        move     #LINEAR,m0            ; linear addressing
SRAM_INIT
        movep   #AAR0V1,x:M_AAR0       ; set up AAR0 as desired
        movep   #BCRV,x:M_BCR          ; set up Bus Control Register as desired
        move    #LOOP2,n7              ; data is 64 words long
        move    #MEM_START,r1          ; r1 points to address in external memory
        move    #DATA_START,r0         ; r0 points to data start address
SRAM_ACCESS
        nop
        do      n7,end_in
        move    y:(r0)+,x0             ; xfer from Y to x0
        move    x0,x:(r1)+             ; place x0 in external X memory
        nop
end_in
        nop
        jsr      *
        end
```

The `portaequ.asm` file holds the AAR0 and BCR equates described earlier. Our data is 64 words long, so LOOP2 = $40.

The main program starts by setting the PLL. It then sets the addressing mode to linear addressing. Now the program is ready to start the SRAM access. It initializes the AAR0 and BCR Registers. It also initializes loop counters and data pointers. Once initialization is complete, the program transfers the data from Y:$0 to X:$MEM_START (defined to be $500000). It copies 64 data values to external X memory.

This example is the simplest of transfers. It uses a MOVE instruction with only one external memory device mapped to X memory. The next example is a variation on this example.

2.2 Overlapping External Memory Space with Internal Core Space

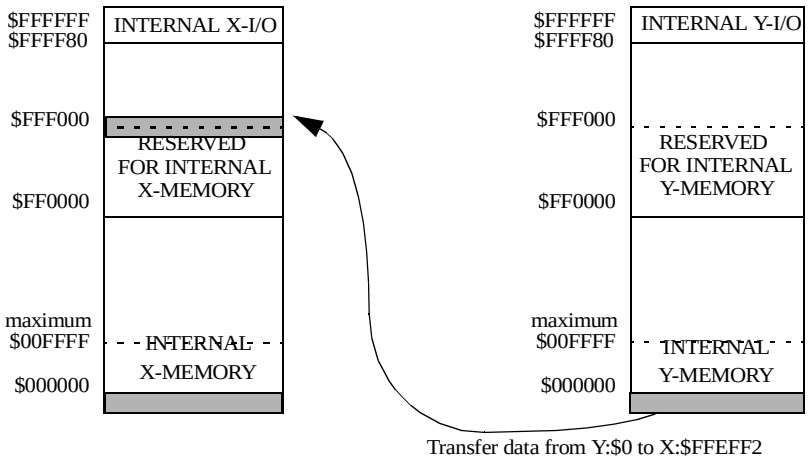
This run exercises a seemingly generic memory access, but it accidentally accesses memory in internal space, resulting in undesired operation.

The Port A options for this exercise are as follows:

- Clock Frequency = 68 MHz
- Packing mode disabled
- Address Muxing disabled
- X space enabled; Y and P disabled
- 64 words of data from Y:$0 to X:$FFEFF2
- SRAM using AAR0
- Transfer method: move instruction

As **Figure 2-2** shows, the code transfers 64 words of data from Y:$0 to X:$FFEFF2-$FFF032. The transfer is done using the MOVE instruction in a DO loop. The data is then moved back to internal memory at X:$0.

Figure 2-2. Data Transfer from Y to X Memory



2.2.1 Initialization

The data transfer begins somewhere in the $FFE000 range (the exact address given in the main program) using an SRAM access to X memory. The Bus Control Register is initialized the same way as in the first exercise. See **Part 2.1.1**.

```
AAR0V1          EQU   $FFE311
                      ; SRAM access
                      ; Address Attribute Register 0
                      ; [23:12] = BAC[11:0] = $FFE (Address to compare)
                      ; [11:8] = BNC[3:0] = $B (Num of add to compare)
                      ; [7] = BPAC = 0 (Packing disabled)
                      ; [6] = BAM = 0 (Address muxing disabled)
                      ; [5] = BYEN = 0 (Y memory disabled)
                      ; [4] = BXEN = 1 (X memory enabled)
                      ; [3] = BPEN = 0 (P memory disabled)
                      ; [2] = BAAP = 0 (Active low enabled)
                      ; [1:0] = BAT[1:0] = 01 (SRAM access)
```

2.2.2 Program

This example is identical to the previous example, except for the Address Attribute initialization and the starting memory address for the external data. Therefore, we provide only the code for the memory start address in X memory and the loop count.

```
MEM_START       EQU   $FFEFF2
LOOP2           EQU   $40
```

When these values are used in the code shown previously, the data does not transfer correctly. The program is written to transfer 64 words of data from Y:$0 to X:$FFEFF2 -> X:$FFF032. However, on the memory map external X memory does not begin until X:$FFF000. Therefore, the program attempts to write 14 words of data to read-only internal X memory instead of to the external device mapped to X:$FFE000. The processor does not halt execution to show an attempted write to the wrong memory space; the data simply does not arrive there. If the data is read back to internal memory for processing later on, or if it is sent to another device, the data is not complete. Therefore, when setting up the external memory space, verify that the space is available and not overlapping other memory device space.

2.3 Multiplexed Access Using DMA

Multiplexed access is supported on both the DSP56301 and DSP56305. The code presented in this section exercises multiplexed access to SRAM. The parameters for this exercise are as follows:

- Clock Frequency = 68 MHz
- Packing mode disabled
- Address Muxing enabled
- P space enabled; Y and X disabled
- 64 words of data from Y:$0 to P:$200000
- SRAM using AAR0
- Transfer method: DMA

Address Muxing means that only the upper eight address bits are used, and the lower eight bits are allowed to sit idle for the transfer. Transfers work the same way as before, and you can use Address Muxing if only eight address lines are required for addressing the data space.

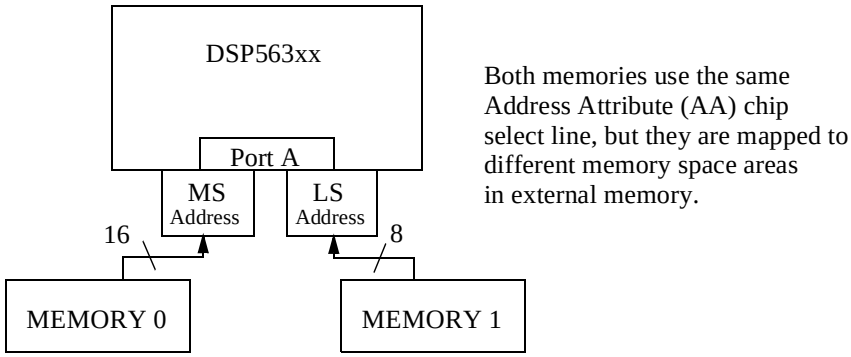
2.3.1 Address Attribute Register Bit: BAM - Address Muxing

If the AARx[6], Address Muxing Bit (BAM) is set, the eight least significant bits of the address appear on the A23-A16 pins (most significant portion of the external address bus). If the bit is cleared, the address appears on the entire external address bus (A23-A0). This bit allows you to connect an external

peripheral to the MS bits of the address and thus decrease the load on the least significant pins of the external address. This enables more efficient interface to external memories. This bit is ignored during DRAM accesses.

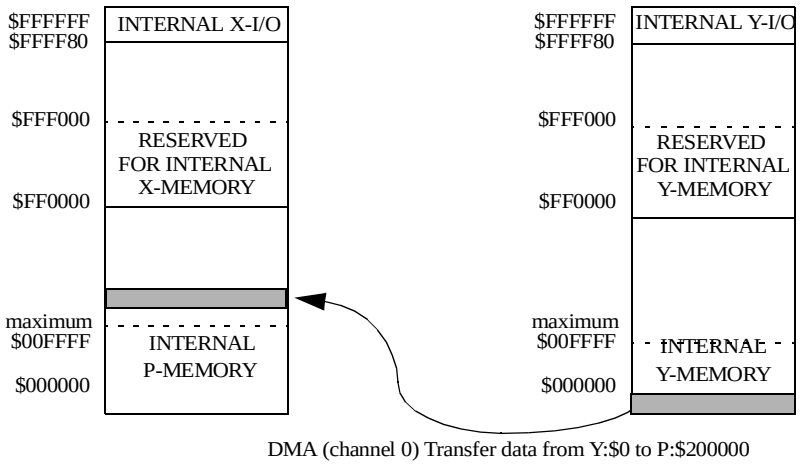
Figure 2-3 shows how to connect two memory devices to the DSP using the same Address Attribute chip select.The eight LS bits appear on the A23-A16 pins when Address Muxing is set in the Address Attribute Register.

Figure 2-3. Connections with Same Address Attribute Chip Select



The EVM boards are not set up for Address Muxing (i.e. with different memories connected to the same address line as diagrammed in **Figure 2-3**), so Address Muxing is simulated in this example. We use the same memory setup as in previous examples, but when the external access occurs, only eight of the address bits are used. As **Figure 2-4** shows, this example sends data from Y:$0 to P:$200000 using DMA.

Figure 2-4. Transfer from Y to X Memory



DMA (channel 0) Transfer data from Y:$0 to P:$200000

2.3.2 Initialization

The code presented in this section uses DMA to access P memory in SRAM at $200000. Address Muxing is enabled.

```
AAR0V1          EQU   $200749
                      ; SRAM access
                      ; Address Attribute Register 0
                      ; [23:12] = BAC[11:0] = $200 (Address to compare)
                      ; [11:8] = BNC[3:0] = $7 (Num of add to compare)
                      ; [7] = BPAC = 0 (Packing disabled)
                      ; [6] = BAM = 1 (Address muxing enabled)
                      ; [5] = BYEN = 0 (Y memory disabled)
                      ; [4] = BXEN = 0 (X memory disabled)
                      ; [3] = BPEN = 1 (P memory enabled)
                      ; [2] = BAAP = 0 (Active low enabled)
                      ; [1:0] = BAT[1:0] = 01 (SRAM access)
```

DMA initialization follows. This application note does not cover the various modes of DMA access. Instead, it covers only the bit settings required for the following examples. For more information on DMA access modes, see the DMA Controller chapter in the *DSP56300 Family Manual*.

The program initiates one DMA transfer to transfer data to P memory. The DMA transfer is from $000000 in Y memory to $200000 in P memory using DMA channel 0. The transfer is a priority level 1 transfer and the DMA interrupt is enabled. The interrupt tells the program when the transfer is complete. DCO0V1 designates that the transfer sends 64 words to external memory. The DAM[5:0] bits in the DCR0 Register indicate that the source and destination addresses increment by one after each word transfer. The data also transfers continuously.

```
DSR0V1          EQU   $000000
                      ; DMA Source Address Register for channel 0
DDR0V1          EQU   $200000
                      ; DMA Destination Address Register for channel 0
DCO0V1          EQU   $000040
                      ; DMA Counter for channel 0
DOR0V1          EQU   $000000
                      ; DMA Offset Register for channel 0
DCR0V1          EQU   $5B02D9
                      ; DMA Control Register for channel 0
                      ; [23] = DE = 0 DMA Operation disabled
                      ; [22] = DIE = 1 DMA Interrupt enabled
                      ; [21:19] = DTM[2:0] = 011 triggered by DE, DE=0@end
                      ; [18:17] = DPR[1:0] = 01 priority level 1
                      ; [16] = DCON = 1 Continuous mode enabled
                      ; [15:11] = DRS[4:0] = 00100 Transfer done from chan 0
                      ; [10] = D3D = 0 non 3-d mode
                      ; [9:4] = DAM[5:0] = 101101 post inc. s/d by 1
                      ; [3:2] = DDS[1:0] = 10 P memory destination
                      ; [1:0] = DSS[1:0] = 01 Y memory source
```

2.3.3 Program

```
       org p:START
       nop
MAIN_INIT
        movep    #PCTL,x:M_PCTL          ; set PLL
        move     #LINEAR,m0              ; linear addressing
DMA_INIT
        nop
        movep   #AAR0V1,x:M_AAR0         ; set up AAR0 as desired
        movep   #BCRV,x:M_BCR            ; set up Bus Control Register

        movep   #DSR0V1,x:M_DSR0         ; setup DMA src add
        movep   #DDR0V1,x:M_DDR0         ; setup DMA dest add
        movep   #DCO0V1,x:M_DCO0         ; set DMA counter
        movep   #DCR0V1,x:M_DCR0         ; initialize DMA control reg
DMA_XFER
        nop
        bset    #23,x:<<M_DCR0           ; start DMA transfer
        bra     *
        end
```

Since the DMA registers include address initialization, the MEM_START and DATA_START addresses are not required in the main program. Instead of initiating an SRAM_INIT and SRAM_ACCESS, this program executes DMA accesses.

The DMA_INIT routine initializes the bus and Address Attribute registers, followed by initialization of the DMA registers. The DMA_XFER routine turns on the DMA access and waits for the transfer to complete.

The code in this example gives a basic introduction to executing DMA transfers through Port A. The only unusual setting is the Address Muxing Bit; the Address Muxing function is transparent during execution.

2.4 Access to an 8-bit Peripheral Using DMA and Packing Mode

This run exercises both a conventional DMA access to SRAM and an access using Packing mode. Packing mode is generally used to access a byte-wide peripheral. The DMA access, if enabled, accesses three byte-wide memory locations and places them in a single 24-bit location in internal memory. Be careful to set the DMA specifications appropriately to handle the transfer.

The Port A options for this exercise are as follows:

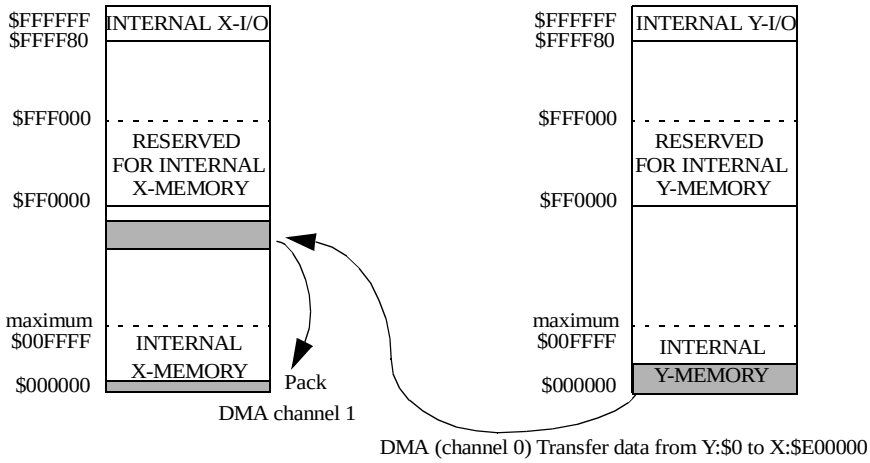
- Clock Frequency = 68 MHz
- Packing mode enabled
- Address Muxing disabled
- X space enabled; Y and P disabled
- 153 words of data from Y:$0 to X:$E00000
- 51 words of packed data from X:$E00000 to X:$0
- SRAM using AAR0
- Transfer method: DMA

2.4.1 Address Attribute Register: BPAC - Packing Mode

The BPAC bit pertains only to DMA accesses into Port A from an external device. Suppose you want to place an 8-bit peripheral on a DSP board and need to receive data from that peripheral: Access can easily be set up using Port A without wasting data space in the on-chip memory. Turning on Packing mode causes three external DMA accesses to the peripheral to be "packed" into one 24-bit data word in the on-chip memory. Packing is done automatically by hardware once you initialize it. The external memory should reside in the eight least significant bits of the external data bus. You must set up DMA with proper offsets and updates for this to operate correctly.

The exercise invokes two DMA transfers. The transfer on DMA channel 0 is a standard DMA access. As **Figure 2-5** shows, it takes 153 words from internal Y memory and places it in external X memory. When the first transfer is complete, it triggers an interrupt that in turn starts the second DMA transfer. The second transfer (now on DMA channel 1) uses Packing mode and brings 51 words of data (153 bytes) back to internal memory and places it starting at X:$0. When the code finishes, compare the data at Y:$0 to the data at X:$0 to understand how Packing mode affects the data.

Figure 2-5. DMA Transfers



2.4.2 Initialization

Data is transferred to/from SRAM X memory at $E00000 using Packing mode.

```
AAR0V1          EQU   $E00C91
                      ; SRAM access
                      ; Address Attribute Register 0
                      ; [23:12] = BAC[11:0] = $E00 (Address to compare)
                      ; [11:8] = BNC[3:0] = $C (Num of add to compare)
                      ; [7] = BPAC = 1 (Packing enabled)
                      ; [6] = BAM = 0 (Address muxing disabled)
                      ; [5] = BYEN = 0 (Y memory disabled)
                      ; [4] = BXEN = 1 (X memory enabled)
                      ; [3] = BPEN = 0 (P memory disabled)
                      ; [2] = BAAP = 0 (Active low enabled)
                      ; [1:0] = BAT[1:0] = 01 (SRAM access)
```

DMA initialization for the channel 0 transfer is almost identical to the initialization in the previous Address Muxing example; the only differences are the addresses for data source and destination and the length of the transfer.

```
DSR0V1          EQU   $000000
                      ; DMA Source Address Register for channel 0
DDR0V1          EQU   $E00000
                      ; DMA Destination Address Register for channel 0
DCO0V1          EQU   $000099
                      ; DMA Counter for channel 0
DCR0V1          EQU   $5B02D1
                      ; DMA Control Register for channel 0
                      ; [23] = DE = 0 DMA Operation disabled
                      ; [22] = DIE = 1 DMA Interrupt enabled
                      ; [21:19] = DTM[2:0] = 011 triggered by DE, DE=0@end
                      ; [18:17] = DPR[1:0] = 01 priority level 1
                      ; [16] = DCON = 1 Continuous mode enabled
                      ; [15:11] = DRS[4:0] = 00000 Transfer done from chan 0
                      ; [10] = D3D = 0 non 3-d mode
                      ; [9:4] = DAM[5:0] = 101101 post inc. s/d by 1
                      ; [3:2] = DDS[1:0] = 00 X memory destination
                      ; [1:0] = DSS[1:0] = 01 Y memory source
```

Initialization for channel 1 must account for the use of Packing mode, making the setup more complicated. The source and destination registers are initialized in the same way. However, since Packing mode is used, a two-dimensional DMA access is required.[1] The first dimension of the access refers to the amount of data transferred (51 words), while the second dimension uses the offset register to update the pointer. We need to update the pointer by three each time, since we are using a 24-bit peripheral to simulate an 8-bit peripheral, and three memory locations are accessed for each transfer. When used in this way, the DCO Register is turned into a dual counter. Since 153 bytes are transferred, but three are transferred at a time using Packing mode, we set the first dimensional counter to 51 or $33. This value must be in the first 12 bits of the counter, designated as the DCOH value. The data values are automatically grouped in threes by packing hardware, so we set the second counter, DCOL, to zero. The offset register, DOR, must be set to $000003 since the values are taken in groups of three, and the address pointer must be updated accordingly. This register automatically updates the DMA pointer registers.

We must also initialize the DAM[5:0] bits in the DCR Register differently than for the channel 0 setup. Since a 2-D transfer is executed, the control register must be notified of which offset register to use as its address pointer update value. The source register must be updated by three each transfer, but the destination should receive the data in contiguous memory locations. Thus, DAM[5:3] = 001 sets the address mode to 2-D, and selects DOR1 as the offset select. DAM[2:0] = 101, which means that the destination is updated by one for each transfer. DAM[2:0] is the same mode as used in the channel 0 transfer.

1. Refer to the application note entitled *Using the DSP56300 Direct Memory Access Controller* for more information on DMA setup.

```
DSR1V1          EQU   $E00000
                      ; DMA Source Address Register for channel 1
DDR1V1          EQU   $000000
                      ; DMA Destination Address Register for channel 1
DCO1V1          EQU   $033000
                      ; DMA Counter for channel 1
DOR0V1          EQU   $000003
DCR1V1          EQU   $5B0280
                      ; DMA Control Register for channel 1
                      ; [23] = DE = 0 DMA Operation disabled
                      ; [22] = DIE = 1 DMA Interrupt enabled
                      ; [21:19] = DTM[2:0] = 011 triggered by DE, DE=0@end
                      ; [18:17] = DPR[1:0] = 01 priority level 1
                      ; [16] = DCON = 1 Continuous mode enabled
                      ; [15:11] = DRS[4:0] = 00000 Transfer done from chan 0
                      ; [10] = D3D = 0 non 3-d mode
                      ; [9:4] = DAM[5:0] = 101000 post inc. d by 1, 2D s
                      ; [3:2] = DDS[1:0] = 00 X memory destination
                      ; [1:0] = DSS[1:0] = 00 X memory source
```

2.4.3 Program

```
        org     p:
DMA_INIT0
        nop
        bset    #8,sr
        bclr    #9,sr                   ; set interrupt masks
        movep   #$00A000,x:<<M_IPRC     ; set DMA interrupt priorities levels
        andi    #$1C,eom                ; set core-DMA priority to 00
        bclr    #23,sr                  ; set core priority level

        movep   #AAR0V1,x:M_AAR0        ; set up AAR0 as desired
        movep   #BCRV,x:M_BCR           ; set up Bus Control Register

        movep   #DSR0V1,x:M_DSR0        ; setup DMA src add
        movep   #DDR0V1,x:M_DDR0        ; setup DMA dest add
        movep   #DCO0V1,x:M_DCO0        ; set DMA counter
        movep   #DCR0V1,x:M_DCR0        ; initialize DMA control reg
DMA_XFER
        nop
        bset    #23,x:<<M_DCR0          ; start channel 0 DMA xfer
        bra     *
end_dma1
        nop
DMA_PACK
        nop
        bset    #8,sr
        bclr    #9,sr                   ; set interrupt masks
        movep   #$00A000,x:<<M_IPRC     ; set DMA interrupt priority levels
        andi    #$1C,eom                ; set core-DMA priority to 00
        bclr    #23,sr                  ; set core priority level
DMA_INIT1
        movep   #DSR1V1,x:M_DSR1        ; setup DMA src add
        movep   #DDR1V1,x:M_DDR1        ; setup DMA dest add
        movep   #DCO1V1,x:M_DCO1        ; set DMA counter
        movep   #DOR0V1,x:M_DOR0        ; set DMA Offset Register
        movep   #DCR1V1,x:M_DCR1        ; initialize DMA control reg
start_dma2
        nop
        bset    #23,x:<<M_DCR1          ; start DMA pack transfer
        bra     *
end_dma2
        nop
INTR_ROUT                               ; interrupt routine
        clr      a
        inc      a
        jsr      DMA_PACK
END
        jsr      *
        end
```

The DMA subroutines are similar to the Address Muxing DMA routines, since they too are performing DMA accesses. This example shows that once Packing mode is initialized for the registers, the program runs similarly to most DMA transfers—hardware does all the work. The interrupt jump table is shown next.

```
   xref  START
   xref  INTR_ROUT
   xref  END
;
        ORG       P:0
;
vectors  JMP       START            ; Hardware RESET
;
        jmp       *
.
.
.
;
        jsr      INTR_ROUT         ;- DMA Channel 0
;
        jmp      END               ;- DMA Channel 1
        NOP
;
        jmp       *
        NOP                        ;- DMA Channel 2
;
.
.
.
```

Only the section of the table relevant to this example is shown. Notice that when the DMA channel 0 interrupt occurs, the code jumps to INTR_ROUT, which is an interrupt service routine that initiates the Packing mode transfer routine, DMA_PACK. DMA_PACK transfers data using DMA channel 1. When that transfer is complete, the interrupt directs program flow to the end of the program.

The comparison in **Table 2-1** shows how the DMA transfer back to internal X memory takes only the low eight bits of each word, treating the memory space as an 8-bit peripheral. It takes three consecutive bytes and packs them into one word of data starting at X:$0.

Memory Address	Original Data				Packed Data			
X:$000000	0000**00**	FFFF**FF**	AAAA**AA**	5555**55**	AAFF00	000055	000000	000000
X:$000004	8000**00**	4000**00**	2000**00**	1000**00**	000000	000000	800000	102040
X:$000008	0800**00**	0400**00**	0200**00**	0100**00**	020408	FFFF01	FFFFFF	FFFFFF
X:$00000C	0080**00**	0040**00**	0020**00**	0010**00**	FFFFFF	FFFFFF	7FFFFF	EFDFBF
X:$000010	0008**00**	0004**00**	0002**00**	0001**00**	FDFBF7	56BAFE	BAA945	A94556
X:$000014	0000**80**	0000**40**	0000**20**	0000**10**	4556BA	5555A9	555555	555555
X:$000018	0000**08**	0000**04**	0000**02**	0000**01**	555555	555555	555555	...
...	7FFF**FF**	BFFF**FF**	DFFF**FF**	EFFF**FF**				
	F7FF**FF**	FBFF**FF**	FDFF**FF**	FEFF**FF**				
	FF7F**FF**	FFBF**FF**	FFDF**FF**	FFEF**FF**				
	FFF7**FF**	FFFB**FF**	FFFD**FF**	FFFE**FF**				
	FFFF**7F**	FFFF**BF**	FFFF**DF**	FFFF**EF**				
	FFFF**F7**	FFFF**FB**	FFFF**FD**	FFFF**FE**				
	FEDC**BA**	1234**56**	0123**45**	EDCB**A9**				
	FEDC**BA**	1234**56**	0123**45**	EDCB**A9**				
	FEDC**BA**	1234**56**	0123**45**	EDCB**A9**				
	00AA**55**	00AA**55**	00AA**55**	...				

Table 2-1. Comparison of Original and Packed Data

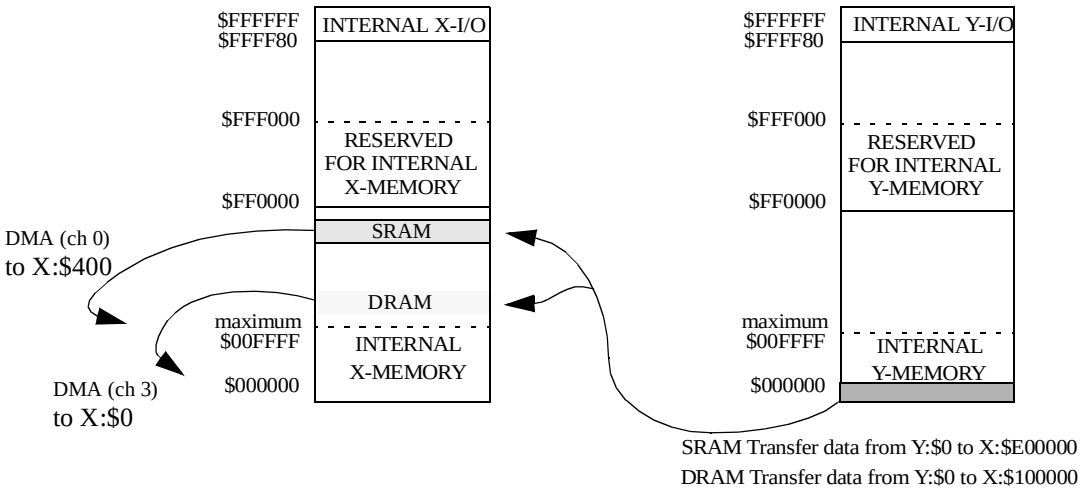
2.5 Accessing DRAM and SRAM Using DMA Through Port A

The ADM boards have resident DRAM, so the example presented in this section can run on the ADM boards. Also, you can remove the DRAM portion and then run the code on the DSP563xxEVM boards if the PLL value is adjusted properly. The parameters for this exercise are as follows:

- Clock Frequency = 66 MHz
- Packing mode disabled
- Address Muxing disabled
- X space enabled; Y and P disabled
- 256 words of data from Y:$0 to X:$100000 DRAM
- 256 words of data from Y:$0 to X:$E00000 SRAM
- SRAM using AAR0
- DRAM using AAR3
- Transfer method: MOVE instruction and DMA

As **Figure 2-6** shows, this example maps two external memory devices, SRAM and DRAM, to the external memory space accessible by Port A. You can use multiple devices if the space accessible by each device does not overlap. Since the DSP56301ADM DRAM connects to the AA3 chip select pin on the DSP, Address Attribute Register AA3 initializes DRAM.

Figure 2-6. SRAM and DRAM Mapped to External Memory



2.5.1 Initialization

DRAM is mapped to X memory at $100000. Packing mode and Address Muxing are disabled. SRAM is mapped to X memory at $E00000. While both memory devices are in the X memory space, they do not overlap at any point. Also, neither device is mapped into internal memory space. AAR0 and AAR3 are initialized to a range of 2 M words. This range is much larger than the space used in this example, but is acceptable since no other devices require use of the memory space.

```
AAR0V1          EQU   $E00311
                      ; SRAM access Y:$0-> X:$E00000
                      ; Address Attribute Register 0
                      ; [23:12] = BAC[11:0] = $E00 (Address to compare)
                      ; [11:8] = BNC[3:0] = $3 (Number of add to compare)
                      ; [7] = BPAC = 0 (Packing disabled)
                      ; [6] = BAM = 0 (Address muxing disabled)
                      ; [5] = BYEN = 0 (Y memory disabled)
                      ; [4] = BXEN = 1 (X memory enabled)
                      ; [3] = BPEN = 0 (P memory disabled)
                      ; [2] = BAAP = 0 (Active low enabled)
                      ; [1:0] = BAT[1:0] = 01 (SRAM access)
AAR3V1          EQU   $100312
                      ; DRAM access Y:$0 -> X:$100000
                      ; Address Attribute Register 3
                      ; [23:12] = BAC[11:0] = $100 (Address to compare)
                      ; [11:8] = BNC[3:0] = $3 (Number of add to compare)
                      ; [7] = BPAC = 0 (Packing disabled)
                      ; [6] = BAM = 0 (Address muxing disabled)
                      ; [5] = BYEN = 0 (Y memory disabled)
                      ; [4] = BXEN = 1 (X memory enabled)
                      ; [3] = BPEN = 0 (P memory disabled)
                      ; [2] = BAAP = 0 (Active low enabled)
                      ; [1:0] = BAT[1:0] = 10 (DRAM access)
```

The DRAM Control Register sets up wait states for DRAM accesses. The values in this register are defined mostly by the choice of DRAM. The Bus Control Register value here is similar to that in previous examples.

```
BCRV            EQU   $004042
                      ; Bus Control Register
                      ; [23] = BRH = 0 bus request hold off
                      ; [22] = BLH = 0 bus lock hold off
                      ; [21] = BBS = 0 bus state
                      ; [20:16] = BDFW[4:0] = 00000 default area wait states
                      ; [15:13] = BA3W[2:0] = 010 area 3 wait states
                      ; [12:10] = BA2W[2:0] = 000 area 2 wait states
                      ; [9:5] = BA1W[4:0] = 00010 area 1 wait states
                      ; [4:0] = BA0W[4:0] = 00010 area 0 wait states
DCRV            EQU   $873A0A
                      ; DRAM Control Register
                      ; [23] = BRP = 1 refresh prescaler on
                      ; [22:15] = BRF[7:0] = 00001110 refresh request rate
                      ; [14] = BSTR = 0 software triggered refresh off
                      ; [13] = BREN = 1 refresh enable
                      ; [12] = BME = 1 mastership enable
                      ; [11] = BPLE = 1 page logic enable
                      ; [10] reserved
                      ; [9:8] = BPS[1:0] = 10 DRAM page size (.5,1,*2K*,4)
                      ; [7:4] reserved
                      ; [3:2] = BRW[1:0] = 10 Out of page w.s. (4,8,*11*,15)
                      ; [1:0] = BCW[1:0] = 10 In page w.s. (1-4 ws; 3 ws)
```

The transfers to SRAM and DRAM are done using the MOVE instruction. However, DMA accesses transfer the data back to internal memory. The transfers are standard DMA transfers, so the setup is similar to the setup described in **Part 2.3**.

```
; SRAM Access DMA channel 0
DSR0V1            EQU   $E00000
                        ; DMA Source Address Register for channel 0
DDR0V1            EQU   $000400
                        ; DMA Destination Address Register for channel 0
DCO0V1            EQU   $000100
                        ; DMA Counter for channel 0
DOR0V1            EQU   $000000
DCR0V1            EQU   $5C02D0
                        ; DMA Control Register for channel 0
                        ; [23] = DE = 0 DMA Operation disabled
                        ; [22] = DIE = 1 DMA Interrupt enabled
                        ; [21:19] = DTM[2:0] = 011 triggered by DE, DE=0@end
                        ; [18:17] = DPR[1:0] = 10 priority level 2
                        ; [16] = DCON = 0 Continuous mode disabled
                        ; [15:11] = DRS[4:0] = 00000 Transfer done from chan 0
                        ; [10] = D3D = 0 non 3-d mode
                        ; [9:4] = DAM[5:0] = 101101 post inc. s/d by 1
                        ; [3:2] = DDS[1:0] = 00 X memory destination
                        ; [1:0] = DSS[1:0] = 00 X memory source
; DRAM Access DMA channel 3
DSR3V1            EQU   $100000
                        ; DMA Source Address Register for channel 3
DDR3V1            EQU   $000000
                        ; DMA Destination Address Register for channel 3
DCO3V1            EQU   $000100
                        ; DMA Counter for channel 3
DOR3V1            EQU   $000000
DCR3V1            EQU   $5C02D0
                        ; DMA Control Register for channel 3
                        ; [23] = DE = 0 DMA Operation disabled
                        ; [22] = DIE = 1 DMA Interrupt enabled
                        ; [21:19] = DTM[2:0] = 011 triggered by DE, DE=0@end
                        ; [18:17] = DPR[1:0] = 10 priority level 2
                        ; [16] = DCON = 0 Continuous mode disabled
                        ; [15:11] = DRS[4:0] = 00000 Transfer done from chan 3
                        ; [10] = D3D = 0 non 3-d mode
                        ; [9:4] = DAM[5:0] = 101101 post inc. s/d by 1
                        ; [3:2] = DDS[1:0] = 00 X memory destination
                        ; [1:0] = DSS[1:0] = 00 X memory source
```

Both MOVE and DMA transfers generate a DMA interrupt. In the Interrupt Priority Register C (IPRC), the DMA channel 0 (SRAM) and DMA channel 3 (DRAM) interrupts are set to priority level 2. This priority level setting is arbitrary, since no other devices should be causing interrupts. If this exercise were included in a larger program, we would set the interrupt priority as determined by application requirements.

```
IPRCV1          EQU   $082000
                      ; Interrupt Priority Register C (includes DMA int)
                      ; [23:22] = D5L[1:0] = 00 DMA5 Int Priority Level
                      ; [21:20] = D4L[1:0] = 00 DMA4 IPL
                      ; [19:18] = D3L[1:0] = 10 DMA3 IPL
                      ; [17:16] = D2L[1:0] = 00 DMA2 IPL
                      ; [15:14] = D1L[1:0] = 00 DMA1 IPL
                      ; [13:12] = D0L[1:0] = 10 DMA0 IPL
                      ; [11] = IDL2 = 0 IRQD mode
                      ; [10:9] = IDL[1:0] = 00 IRQD IPL
                      ; [8] = ICL2 = 0 IRQC mode
                      ; [7:6] = ICL[1:0] = 00 IRQC IPL
                      ; [5] = IBL2 = 0 IRQB mode
                      ; [4:3] = IBL[1:0] = 00 IRQB IPL
                      ; [2] = IAL2 = 0 IRQA mode
                      ; [1:0] = IAL[1:0] = 00 IRQA IPL
```

The DSP56301ADM uses a faster external clock, so we set the PLL to a smaller value for 66 MHz parts. In previous examples, the PLL value was set to $400003, which multiplied the external clock by four. For the DSP56301ADM, we multiply the clock by two.

2.5.2 Program

```
; Assembly-Time Equates
START           EQU   $000400   ; start address for code
LINEAR          EQU   $FFFFFF   ; set up linear addressing mode
; Address Equates
DATA_START      EQU   PATT
MEM_STARTD      EQU   $100000   ; DRAM start address
MEM_STARTS      EQU   $E00000   ; SRAM start address
LOOP_NUM        EQU   $4
LOOP_PATT       EQU   $40

       org p:START
       nop
MAIN_INIT
        movep    #PCTL,x:M_PCTL      ; set PLL
        movep    #DCRV,x:M_DCR       ; set DRAM Control Register
        move     #LINEAR,m0          ; linear addressing
DRAM_INIT
        nop
        movep   #AAR3V1,x:M_AAR3     ; set up AAR3 as desired
        move    #LOOP_PATT,n7        ; data pattern is 64 words long
        move    #LOOP_NUM,n4         ; repeat data pattern
        move    #MEM_STARTD,r1       ; r1 points to address in memory
DRAM_ACCESS
        nop
        do      n4,end_dmem          ; go through memory and fill with pattern
        nop
        move    #DATA_START,r0       ; r0 points to data start address
        do      n7,end_din
        move    y:(r0)+,x0           ; xfer from y mem to temporary register
        move    x0,x:(r1)+           ; xfer from temp reg to external X memory
end_din
        nop
end_dmem
        nop
SRAM_INIT
        nop
        movep   #AAR0V1,x:M_AAR0     ; set up AAR0 as desired
        movep   #BCRV,x:M_BCR        ; set up Bus Control Register as desired
        move    #LOOP_NUM,n4         ; repeat data pattern
        move    #LOOP_PATT,n7        ; data pattern is 64 words long
        move    #MEM_STARTS,r1       ; r1 points to address in memory
SRAM_ACCESS
        nop
        do      n4,end_smem         ; go through memory and fill with pattern
        nop
        move    #DATA_START,r0      ; r0 points to data start address
        do      n7,end_sin
        move    y:(r0)+,x0          ; xfer from y mem to temporary register
        move    x0,x:(r1)+          ; xfer from temp reg to external X memory
end_sin
        nop
end_smem
        nop

DMA_INIT3
        nop
        bset    #8,sr
        bclr    #9,sr                ; unmask IPLs
        movep   #IPRCV1,x:<<M_IPRC
        andi    #$1C,eom
        bclr    #23,sr               ; core priority 1
```

```
        movep   #DSR3V1,x:M_DSR3     ; setup DMA src add
        movep   #DDR3V1,x:M_DDR3     ; setup DMA dest add
        movep   #DOR3V1,x:M_DOR3     ; set up offset register
        movep   #DCO3V1,x:M_DCO3     ; set DMA counter
        movep   #DCR3V1,x:M_DCR3     ; initialize DMA control reg
DMA_INIT0
        nop
        movep   #DCR0V1,x:M_DCR0     ; initialize DMA control reg
        movep   #DSR0V1,x:M_DSR0     ; setup DMA src add
        movep   #DDR0V1,x:M_DDR0     ; setup DMA dest add
        movep   #DOR0V1,x:M_DOR0     ; setup DMA Offset Register
        movep   #DCO0V1,x:M_DCO0     ; set DMA counter
        rts
DMA_XFER
        nop
        bset    #23,x:<<M_DCR3       ; start DRAM DMA xfer
        bset    #23,x:<<M_DCR0       ; start SRAM DMA xfer
        bra     *

END
        nop
        jsr     *

        include 'intr_rout.asm'
        end
```

In this code, the DRAM access is first. The registers are initialized and then the access begins, followed by the SRAM access. Finally, the DMA registers are initialized for DRAM and SRAM access, and the two transfers start almost simultaneously.

Each access is similar to those previous examples, but here they are brought together in one program. Notice that the DMA transfer for each device starts within one cycle of the previous transfer. Since each channel has the same priority level as the other, the hardware initiates a "round-robin" transfer. In this mode, each DMA channel can transfer one piece of data, at which point the bus access is given to the other DMA channel. This alternation continues until each channel complets its transfer. In this example, both channels transfer the same amount of data and finish their transfers within a cycle of each other. However, if the DRAM were transferring twice as much data as SRAM, the two memory devices would work in the round-robin fashion until SRAM completed its transfer, at which point DRAM would gain full control of the bus.

```
DMA_ENDSRAM
      move   #$000001,y0
      move   y0,y:$400
      jmp    END
DMA_ENDDRAM
      move   #$000002,y0
      move   y0,y:$401
      bra    *
```

The interrupt routines are short. The DRAM access initiates first and completes first. Thus, when the interrupt is generated, a value is placed in Y memory for debugging purposes, and the core waits for the SRAM transfer to complete. When the SRAM transfer is complete, it places a value in Y memory for debugging purposes.

2.6 Troubleshooting

If a program using Port A does not work properly, use the following list as a guideline for debugging.

- Check the address space where data is transferred externally. Does the address match the one given in the AARx:BAC[11:0] and AARx:BNC[3:0] bits? Is the correct data space (X,Y,P) enabled?
- Is the external access type (AARx:BAT[1:0]) set for either DRAM or SRAM? Is the pin polarity for the device (AARx:BAAP) set correctly?
- Make sure that the number of wait states matches what is necessary for the system. Increase wait states in the Bus Control Register or DRAM Control Register, if necessary.
- If using Packing mode, check DMA offset and counter registers. Make sure that two-dimensional mode is being used and the counter is set correctly.
- Check the PLL value to make sure it is within operating range for the device.

If these are all correct, the problem probably lies in an area of code other than the Port A setup. Check the program flow to ensure that registers are used properly.

2.7 Conclusions

As the examples in this application note show, external memory access using Port A on Motorola's DSP56300 family of devices is fast and straightforward. Port A is a necessary peripheral for systems that require immediate access to off-chip memory. With six DMA channels available in the core, you can execute transfers to and from external memory on demand. Using Port A in combination with the DSP core and other peripherals provides a powerful system solution for wireless applications.

2.8 References

1. *DSP56300 Family Manual*, Motorola, 1995. Order this document by document order number DSP56300FM/AD. Or download it from the Freescale Web site at http://www.freescale.com/DSP/documentation/DSP56300.html
2. *DSP56301ADM User's Manual*, Motorola, 1995. Order this document by document order number DSP56301ADMUM/AD. Or download it from the Freescale Web site at http://www.freescale.com/pub/DSP/LIBRARY/TOOLSDOC/ADS
3. *DSP56303EVM User's Manual*, Motorola, 1996. Order this document by document order number DSP56303EVMUM/AD. Or download it from the Freescale Web site at http://www.freescale.com/DSP/documentation/DSP56300.html

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com