Notes on using Analog Devices' DSP, audio, & video components from the Computer Products Division
Phone: (800) ANALOG-D or (781) 461-3881, FAX: (781) 461-3010, EMAIL: dsp.support@analog.com

Using ADSP-218x I/O Space

Last modified 11/08/96

Introduction

Digital Signal Processors (DSPs) are often chosen by designers performing arithmetic operations on binary data. DSPs perform many common signal processing algorithms, such as filtering or fast fourier transforms on input digitized data, more economically than corresponding analog circuits. The information that the DSP processes usually comes from an analog-to-digital converter, which represents the real-world value as a binary number. After processing by the DSP, the output goes through a digital-to-analog converter, which provides a continuous signal that is useful for real-world feedback. This data flow appears in Figure 1. Because real-world signal I/O can be an important part of DSP system operations, I/O between the DSP and converters is a major design issue when developing a DSP system.

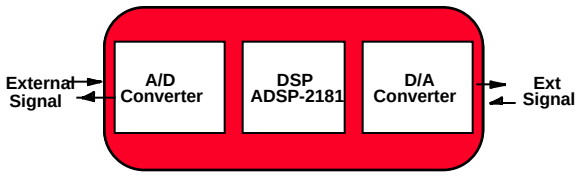

Figure 1 - DSP Data Flow

There are two methods for sending digitized information to a DSP. One method is serial communication. All Analog Devices ADSP-2100 family DSPs have synchronous serial communication ports (SPORTs). These SPORTs let you connect many types of serial-converters directly to the DSP.

The other communications method is to connect the converter as a parallel I/O source. On most ADSP-2100 family DSPs, parallel I/O is available by connecting the converter as a memory-mapped peripheral in the DSP's Program Memory or Data Memory Spaces. But, memory mapped I/O on these DSPs does require some interface hardware to manage the input from the converter.

ADSP-218x DSPs, however, differs from other processors of the ADSP-21xx family with regard to memory mapped I/O. Unlike other members of this DSP family (which required memory-mapped peripherals be connected to PM or DM spaces), ADSP-218x DSPs have their own separate I/O space. For managing I/O space access, ADSP-218x DSPs have an extra memory select line /IOMS. The ADSP-218x DSPs' I/O Space allows access to up to 2048 locations of 16-bit data. This space should be used to communicate with parallel devices such as data converters, external registers, or latches. This additional select line lets this DSP perform parallel-peripheral I/O without the additional decoding hardware that is required by other ADSP-21xx family DSPs.

Before describing how to use and ADSP-218x DSP's I/O space, it would be useful to review some I/O space features. ADSP-218x DSP I/O space has the following features:

- Provides directly-addressed locations
- Supports 16-bit transfers
- Has four waitstate ranges with 512 locations each
- Includes a dedicated I/O select line, /IOMS
- And, the ADSP-218x DSP's assembly language has syntax that supports this I/O space:

```
Dreg = IO(address);
                    !  Reads from an IO address
IO(address) = Dreg;
                      ! Writes to an IO address
```

This engineer's note explains how to set up and use parallel I/O with an ADSP-218x DSP's I/O space and port your existing ADSP-21xx code to take advantage of this I/O space. Using this space in your design lets you eliminate the external address decoding for parallel peripherals, which is required for other ADSP-21xx family DSPs, because you can use the ADSP-218x DSP's dedicated I/O select line, /IOMS, during I/O space accesses.

Porting ADSP-21xx Code To Take Advantage Of ADSP-218x I/O Space

If you are porting your ADSP- 21xx family DSP design to an ADSP-218x DSP, you must make some changes to your source code and architecture file for the I/O space device to communicate with the DSP correctly. If your current design uses an ADSP-2101, your system file includes a declaration for a data memory-mapped I/O port. For example (using Release 5.x development software tools), the system file in listing 1 shows how to define a port at address `0x0100`.

```
.SYSTEM some_example;
.ADSP2101;
.MMAP0;
.SEG/PM/RAM/ABS=0x0000/CODE/DATA int_pm[2048];
.SEG/PM/RAM/ABS=0x800/CODE/DATA ext_pm[14336];
.SEG/DM/RAM/ABS=0x0000/DATA ext_dm1[256];
.PORT/DM/ABS=0x0100  some_port;
.SEG/DM/RAM/ABS=0x0101/DATA ext_dm2[14079];
.SEG/DM/RAM/ABS=0x3800/DATA int_dm[1024];
.ENDSYS;
```

Listing 1 - ADSP-2101 System File

From this file, you create your architecture file created using the command (in DOS):

```
bld21 my_2101.sys
```

The output of this command results in Listing 2.

```
$SOME_EXAMPLE
$ADSP2101
$MMAP0
$0000 07FF paxINT_PM t
$0800 3FFF paxEXT_PM t
$0000 00FF dadEXT_DM1 t
$0100 0100 dapSOME_PORT t
$0101 37FF dadEXT_DM2 t
$3800 3BFF dadINT_DM t
$
```

Listing 2 - ADSP-2101 Architecture File

In the main module of your ADSP-2101 source program, your code accesses a port using the assembly language directives and instructions in Listing 3.

```
.MODULE/SEG=int_pm/RAM/ABS = 0 test;
.PORT some_port;
JUMP start; RTI;NOP;NOP;
RTI;NOP;NOP;NOP;
RTI;NOP;NOP;NOP;
RTI;NOP;NOP;NOP;
RTI;NOP;NOP;NOP;
RTI;NOP;NOP;NOP;

start:  ar = 0x0100;
dm(some_port) = ar;
nop;
.ENDMOD;
```

Listing 3 - ADSP-2101 Memory Mapped I/O Code

To port Listings 2 and 3 so that they can work on an ADSP-218x, you must modify your source files. Use the following steps:

1. Remove all instances of the .PORT directive from your system builder file
2. Use the IO command in your source programs to write to or read from this port address

So, your new system file, `new_218x.sys`, looks like Listing 4.

```
.SYSTEM some_example;
.ADSP218x;
.MMAP0;
.SEG/PM/RAM/ABS=0x0000/CODE/DATA
int_pm[16384];
.SEG/DM/RAM/ABS=0x0000/DATA int_dm[16384];
.ENDSYS;
```

Listing 4 - ADSP-218x System File

And, your architecture file looks like Listing 5.

```
$SOME_EXAMPLE
$ADSP218x
$MMAP0
$0000 3FFF paxINT_PM t
$0000 3FFF dadINT_DM t
$
```

Listing 5 - ADSP-218x Architecture File

Corresponding code for I/O space accesses in an ADSP-218x program appears in Listing 6.

```
.MODULE/SEG=int_pm/RAM/ABS = 0 test;
.CONST some_port =0x0100;
JUMP start; RTI;NOP;NOP;
RTI;NOP;NOP;NOP;
RTI;NOP;NOP;NOP;
RTI;NOP;NOP;NOP;
RTI;NOP;NOP;NOP;
RTI;NOP;NOP;NOP;
RTI;NOP;NOP;NOP;
RTI;NOP;NOP;NOP;
RTI;NOP;NOP;NOP;
RTI;NOP;NOP;NOP;
RTI;NOP;NOP;NOP;


start:  i0 = 0x0100;
m0 = 0;
l0 = 0;
dm(i0,m0) = 0x0100;
ar = dm(i0,m0);
io(some_port) = ar;
nop;
.ENDMOD;
```

Listing 6 - ADSP-218x Assembly Code For I/O Space Access

Accessing I/O space in C

Using the G21 compiler in the Release 5.x development tools, you must use in-line assembly language to access an I/O port from within a C program. Listing 7 shows example C code for this operation.

```
asm(“#define some_port  0x0100”);
asm(“IO(some_port) = ar;”);
```

Listing 7 - ADSP-218x C Code For I/O Space Access

The example code in Listing 7 writes to an I/O port at address `0x0100` from the `ar` data register. You can similarly read from an I/O port into any data register.

It is important to note that you have to use a data register (`dreg`) to either read from or write to the I/O port. For a list of data registers, refer the *ADSP-2100 Family User's Manual.*

Piping a C variable to an I/O Port

If you want to send the value of a C variable to an I/O port, use in-line assembly as shown in Listing 8.

```
int myvar;
             /* variable declared globally  */
void main(void     )
{
   myvar=256 ;
   asm(".external  myvar_;");
   asm("ax0=dm(myvar_);");
   asm("IO(0x0100)=ax0;");
}
```

Listing 8 - ADSP-218x C Code For Variable I/O Space Access

The code in Listing 8 pipes the value of a C variable myvar to I/O port at address 0x0100. Note that the variable is declared globally to make it accessible to the assembly code. It is also necessary to declare the variable as an external before trying to access it. Finally, note the choice of the ax0 register to store the value. This register is regarded as a scratch register by the C compiler and is hence safe to use.

You can similarly use in-line assembly to direct a value from an I/O port into a C variable.

I/O simulation

One of the main features of the I/O ports in the ADSP-218x simulator is transferring input data to a simulated I/O port from a data file and redirecting output data to a data file. To simulate port I/O with file I/O, choose the following within the simulator:

```
Memory
   Port Display
      Open
```

Use these menu options, as shown in Figure 2, to open the simulator's Port Open menu. You must select the *IO* option inside the Port Open Menu, selecting `PM` or `DM` causes an error.

Once you have set up the data files names and clicked `Yes` in the `Accept Port Configuration` field, you can execute or step through the dummy instructions in the Program Memory Window, also shown in Figure 2. It is important to note that the value that is displayed at the IO memory address location 0x0100 in the I/O memory window maybe inaccurate. It is possible, however, to verify the correctness of the data that is actually written to the I/O port by exiting/quitting the simulator and looking at the out.dat data file (shown in Listing 9).

```
C:\PROJECTS>type out.dat
0100


C:\PROJECTS>
```

Listing 9 - Viewing The OUT.DAT File (In DOS)

Notes on using Analog Devices' DSP, audio, & video components from the Computer Products Division
Phone: (800) ANALOG-D or (781) 461-3881, FAX: (781) 461-3010, EMAIL: dsp.support@analog.com

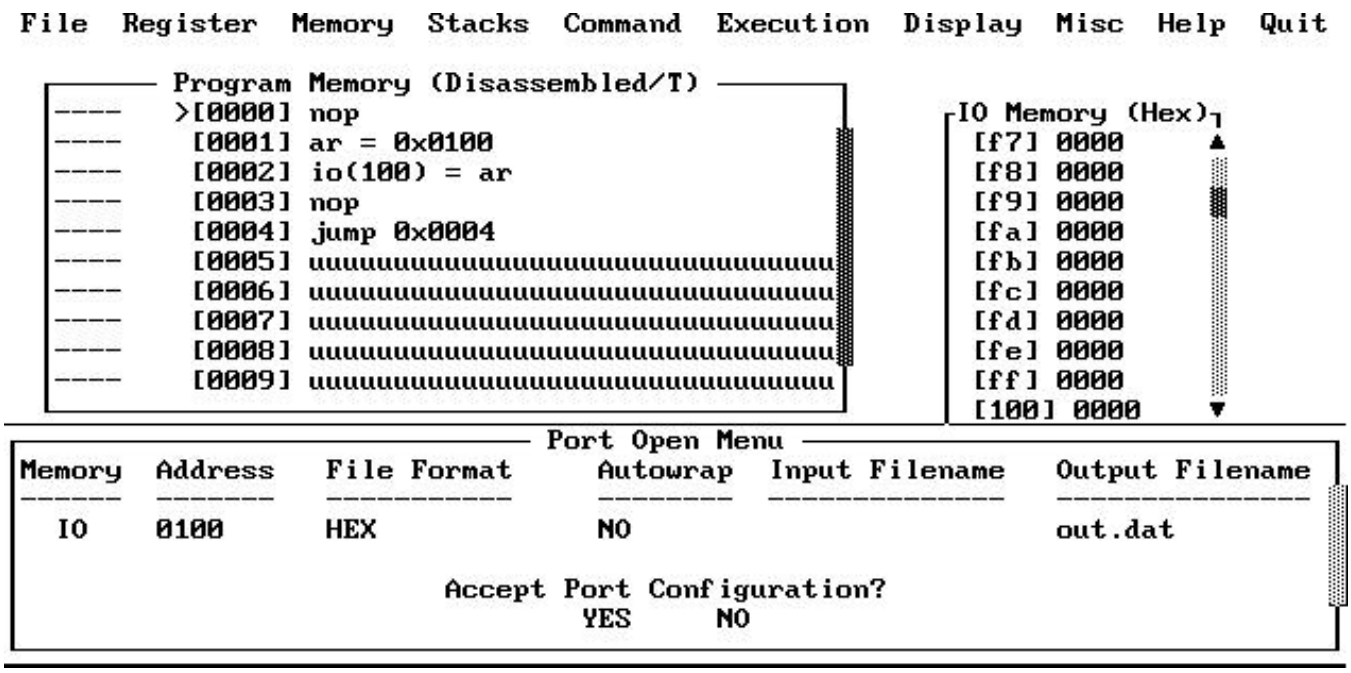

Figure 2 - ADSP-218x Simulator Program Memory Window, IO Memory Window, & Port Open Menu