Technical Notes on using Analog Devices' DSP components and development tools
Phone: (800) ANALOG-D, FAX: (781) 461-3010, EMAIL: dsp.support@analog.com, FTP: ftp.analog.com, WEB: www.analog.com/dsp

EPROM Booting In Host Mode With 100 Pin 218x Processors

Written By: *Greg F.*
Last Modified: *9/23/98*

This EE Note will explain how to boot an Analog Device's adsp218x DSP processor (2184/85/86/87/89) via the Byte Memory interface (BDMA) while it is configured for host memory mode. Host memory mode allows a host processor to boot or access the DSP's internal memory via the Internal Direct Memory Addressing port (IDMA). Normally, these processors can be configured to boot in *only* one of two methods; from an eprom (full memory mode), or from a host processor via the IDMA port (host memory mode).

Overview

All of the previously mentioned processors feature a 100-pin package which uses a multiplexed external bus that has the following functionality:

- **Full Memory Mode:** Provides complete use of the external address and data busses as found in the ADSP-2181. Allows BDMA operation with full external overlay memory and I/O capability. IDMA functionality is disabled
- **Host Mode:** Allows complete IDMA port operation with limited external addressing capabilities.

The functionality of the external bus is dependent upon the setting of the external ***mode C*** pin of the processor. The logic state of this pin is acknowledged by the processor during reset; mode C = 0 configures the DSP in full memory mode, mode C = 1 configures the processor in host memory mode.

Note: the state of the mode C pin ***must*** *not be changed once the DSP begins execution.* For more information on the operation and configurations of the Mode C pin, please refer to the appropriate ADSP-2185/6/7L data sheet, available on our website at the following URL; http://www.analog.com

Full Memory Mode

Full Memory Mode gives complete use of the external address and data busses as found in the ADSP-2181. In Full Memory Mode, the ADSP-2185/2186 behaves like an ADSP-2181 with the IDMA port removed. There is a 24 bit external data bus, a 14 bit address bus and 5 memory select signals. Byte memory is accessed using the middle eight bits of the data bus for data. The upper eight bits of the data bus together with the 14 address pins provide a 22 bit address for byte memory space. All of these features behave exactly the same as on the ADSP-2181. Hold Off cases (autobuffer cycle stealing, external memory accesses with wait states, etc.) are simplified because an IDMA transfer will never occur. In this mode the IDMA port is disabled as if /IS was deselected or pulled high on the ADSP-2181.

Host Mode

Host Mode gives full use of the IDMA port as found on the ADSP-2181, but there are limitations on the use of the external memory bus. In Host Mode the lower eight bits of the data bus, D[7:0] become IDMA control pins and IAD bus pins. The upper 13 bits of the address bus A[13:1] become the lower 13 bits of the IDMA address/data bus, IAD[12:0]. The Pinout Diagram and tables for the Memory Interface Pins in the ADSP-2185/6/7L Data Sheets show the alternate functions for each pin in either major I/O mode. IDMA transfers occur exactly as on the ADSP-2181.

Accessing Peripherals

The external bus in Host Mode still remains available in a limited form. The DSP's address pins A[13:1] are changed to IAD[12:0] when the Mode C pin is high. As a result, the chip cannot drive an address externally. However, internally the chip will behave as if external accesses are occurring.

The external bus will behave as an ADSP-2181 system where address bits A[13:1] and data bits D[7:0] are ignored. The upper 16 bits of the data bus can still be used for external data transfers, but only one address bit is available, A0. Writes to Data Memory or I/O Space will activate the appropriate memory select(s), /RD or /WR, place data on D[23:8], and drive a single address bit on A0.

Program memory reads and writes behave similarly but have the added consideration of the PX register. For program memory reads and writes only the upper 16 bits will be available externally. When 24 bit data is written to external program memory the upper 16 bits will be driven out on data bus pins [23:16]. The PX register will still latch the lower eight bits of the program memory word, but they will not be driven externally. If a 24 bit read of external memory occurs, no external pins will control the value of the PX register, and the PX register will be written with all ones. The missing address bits restrict using the external bus with a conventional memory device which has separated address and data buses. These external transfers might be usable with shared address/data memory chips or can be used for communication with an ASIC. The memory selects will still be active, so each memory space is effectively collapsed into two external addresses, address 0 and 1.

Byte Memory Accesses

BDMA accesses are still allowed in Host Mode. However, because address pins A[13:1], now operate as the IAD bus, construction of a complete byte address is impossible, without external circuitry.

Byte memory addresses on the ADSP-2181 were 22-bit addresses formed from external data pins D[23:16] and address pins A[13:0]. In Host Mode D[23:16] and A0 are the only address bits available externally. The values of the external data pins D[23:16], will be the values contained in the BMPAGE field of the BDMA Control Register, located at internal DM address 0x3fe3 of the processor. A0 will be 1 for odd byte addresses and 0 for even byte addresses.

BDMA and IDMA timing and cycle stealing are the same as on the ADSP-2181. BDMA with limited address bits available still provides a flexible interface to the DSP. Without full address bits addressing memory will be more difficult but host or microcontroller communication is possible because the order of the byte sequence is known. For information on byte sequencing, refer to *Byte Memory Word Formats* in Ch. 11 of the ADSP-2100 Family User's Manual.

So What Does This All Mean?

Since the DSP is configured in host mode, the external address bus is limited to one pin, A0. What this means to us is that we now need to use address generators to drive the external address pins of the EPROM to facilitate booting of the DSP. (Please refer to the included data sheet for more information on the operation of the address generators, Philips part number 74HC4040.)

The one caveat of building this hardware configuration is that the address generators output *sequential* address values, while the prom splitter for the 218x family does not. So the executable file *must* be massaged to allow the prom splitter to output sequential data for the eprom. Normally, the prom splitter (when used in conjunction with the -2181 and -loader switches), prepends a boot loader kernel to the beginning of the eprom file. This is done because the development tools for the 218x family use a different boot paging scheme than the rest of the 21xx family DSP processors to initialize program memory segments, program data segments, and data memory segments.

The Real Solution

Here are the following steps that are needed to complete our design:

1. Create a system file (*.sys) for a 2101 processor that declares on chip memory space for a 218x system, but also includes boot page segments.
2. Declare a boot page segment in your source code's .module directive.
3. Replace the @BO symbol in your *.exe file with an @PA symbol. (The @BO symbol is located on the second line of the *.exe file; you can edit this file with any text editor, since the file is written in an ASCII file format.)

The following example is the system file (Boothost.sys) used to build this project:

```
.system BootHost;
.adsp2101;
.mmap0;
.seg/pm/ram/code/data/abs=0          int_pm[0x4000];
.seg/dm/ram/data/abs=0               int_dm[0x3c00];
.seg/rom/boot=0  boot0[2048];
.endsys;
```

From this example system file, you'll notice that we've declared an adsp2101 system with roughly 16k words of PM and DM respectively!! This step is required to "trick" the tools into using boot segments, (".seg/rom/boot=0 boot0[2048];"), with a 2181 memory model.

Now here is our code example. Here you'll notice again that a boot segment is used in this source file (located at line 1), which is normally illegal for a 218x system. But, since we've tricked the tools into using our *modified* 2101 system file, everything will work accordingly.

```
/* Filename : Boothost.DSP */
.module/ram/boot=0    booty;
 #include "vectab.h"
#define IRQ2ON

IRQ2INT:
   toggle fl0;
   none = pass sr0;
   if eq call resetpf;       {reset PF data}
   ar = sr0 OR 0x80;
   dm(0x3fde)  = ar;         {set data for BDMA wr}
   ax0 = 0x7fdf;
   dm(0x3fe0)  = ax0;        {set IDMAA to PF DATA reg}
   ay0 = 0x0;
   dm(0x3fe2)  = ay0;        {set BEAD}
   ay0 = 0x3fde;
   dm(0x3fe1)  = ay0;        {set BIAD}
   ay0 = 0x7;
   dm(0x3fe3)  = ay0;        {set BDMA Control}
   ay0 = 0x1;
   dm(0x3fe4)  = ay0;        {set BWCOUNT, start BDMA}
   nop; nop; nop; nop;
   sr = lshift sr0 by -1 (LO);  {shift reg holding PF data}
   ay1 = dm(0x3fdf);
   dm(0x3fe5)  = ay1;
   rti;

RESETPF:
   sr0       = 0x40;
   rts;

START:
   ax0 = 0x0400;
   dm(0x3fff)  = ax0;
   ax0 = 0x0000;      {set IOWAIT to 0}
   dm(0x3ffe)  = ax0;
   imask = 0x200;       {enable IRQ2}
   ax0 = 0x1f7f;
   dm(0x3fe6)  = ax0;        {make PFs output exc PF7}
   ax0 = 0x80;
   dm(0x3fe5)  = ax0;        {make all PFs exc PF7 lo}
   sr0       = 0x40;       {set data for first PF}
   reset fl0;
   cntr = 0x6;
   do FLASHER until ce;
      cntr = 0x1000;
      do FLASHER1 until ce;
         cntr = 0x2000;
         do FLASHER2 until ce;
FLASHER2:      nop;                  {wait for IRQ2}
FLASHER1:   nop;
FLASHER:
      toggle fl0;
NOWHERE: idle;
   jump NOWHERE;

#include " trailer.h"
```

```
.endmod;                /* end of boothost.dsp program */
```

From this point, we're able to build our executable file. We'll use the following command line(s) at the DOS prompt:

```
bld21 boothost

asm21 boothost.dsp -o boothost

ld21 boothost -a boothost.ach -e boothost -x -g
```

The first line creates our architecture file from the file boothost.sys. The second line assembles the file boothost.dsp and creates an object file boothost.obj. The third line creates our executable file boothost.exe using the boothost.obj and boothost.ach files.

At this step in the build we need to perform one more task; we need to manually edit the executable file to generate a program memory executable file, not a ROM executable file. This can be done by performing the following steps;

1. Edit the file boothost.exe.
2. Change the first line in the file from @BO to @PA. (This changes the file from a ROM bootable file to a program memory executable file.)
3. Save the file as boothost.exe.

At this point, we are able to perform the final step in our build, by typing the following command line at the DOS prompt; spl21 boothost boothost -loader -2181. This will create the file boothost.bnm, that we can use to burn into the EPROM for our system.

References And Appendices

Please refer to chapters two and five of the 2100 Family Assembler Tools and Simulator Manual, (System Builder and PROM Splitter, respectively), as well as the latest version of the development tools release notes for more information on the usage and functionality of these tools. (All of these documents can be downloaded from our website at the following URL; http://www.analog.com/support/product_documentation/dsp_prdoc.html.) Also included at the end of this application note is a schematic that shows the interface used for this design. For more information please contact Analog Devices at 1-800-ANALOGD, or http://www.analog.com/dsp.

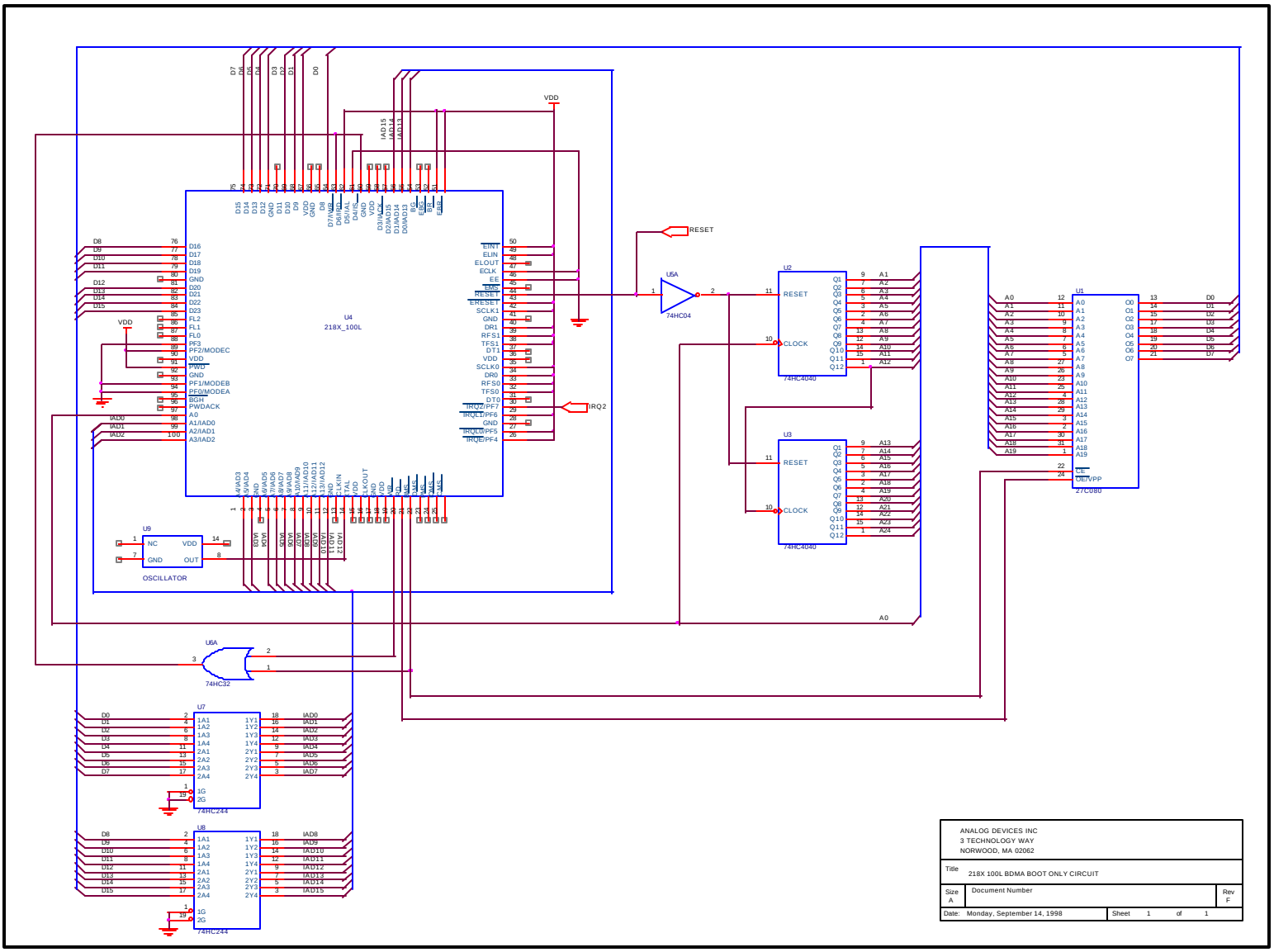

Figure 1: Example schematic for Host Mode EPROM booting system