AN1177
APPLICATION NOTE

80C51XA / M88 FLASH+PSD Design Guide

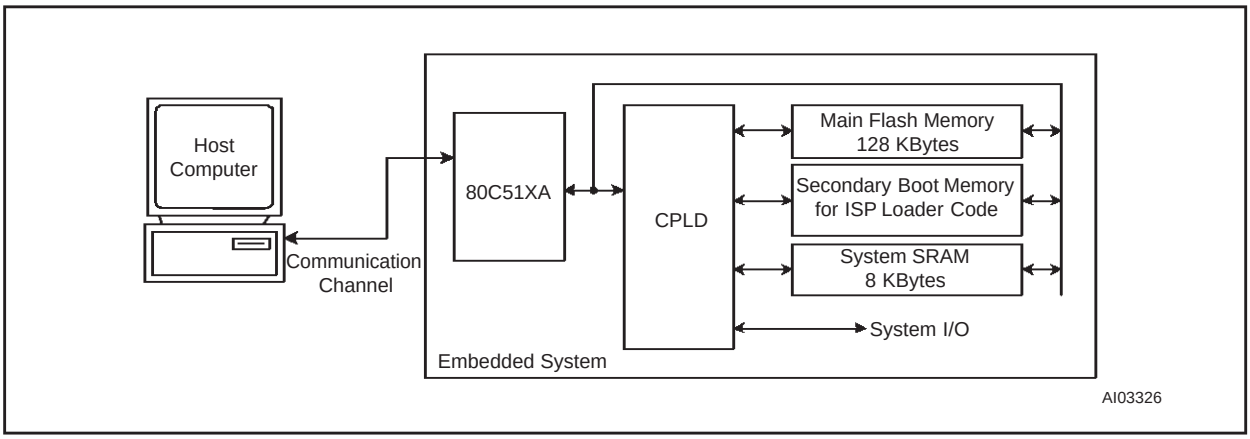
The M88x3Fxx devices are members of ST's M88 FLASH+PSD family of flash-based peripherals for use with embedded microcontrollers (MCUs). These programmable system devices (PSDs) consist of memory, logic, and I/O. When coupled with a low-cost, 80C51XA MCU, the PSD forms a complete embedded flash system that is 100% In-System Programmable (ISP). There are many features in the PSD silicon and in the PSDsoft development software that make ISP easy, regardless of how much experience you have with embedded design.

This document offers two 80C51XA-based designs using an M88 FLASH+PSD device. The first is a simple system to get up and running quickly for basic applications, or to check out your prototype 80C51XA hardware. The second design illustrates the use of enhanced features of PSD In-System Programming as applied to the 80C51XA. You can start with the first design and migrate to the second if needed.

Typically, a host computer downloads firmware into an embedded flash system through a communication channel that is controlled by the MCU. This channel is usually a UART, but any communication channel that the 80C51XA supports will do. The 80C51XA must execute the code that controls the ISP process from an independent memory array that is not being erased or programmed. Otherwise, boot code and flash programming algorithms (ISP loader code) will be unavailable to the 80C51XA. It is absolutely necessary to use a secondary boot memory array (an independent memory that is not being programmed) to store the ISP loader code.

A system designer must choose the type of secondary boot memory to store ISP loader code (SRAM, FLASH, or EEPROM); each type has advantages and disadvantages. This secondary boot memory may reside external to the MCU or on-chip. A top-level view of an embedded ISP flash system with external memory is shown in Figure 1.

Figure 1. Embedded Flash System Capable of ISP (5 devices)



A Common Solution

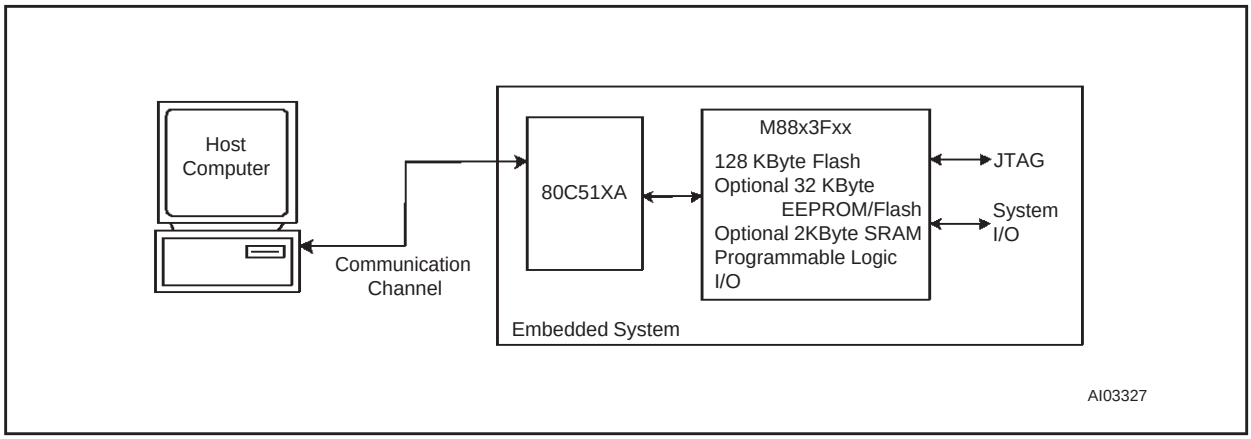
Without a PSD device, implementing ISP with the 80C51XA can be difficult and time consuming. Philips' application note *AN97019* shows three methods of connecting an 80C51XA to external flash memory. Each method has trade-offs and difficulties to overcome that add cost and design time, and no method allowed for easy updates of the ISP loader code.

A Better, Integrated Solution

Previously, ISP required MCU participation to exercise a communication channel to implement a download to the main flash memory. However, the M88x3Fxx offers an alternative ISP method that uses a built-in IEEE-1149.1 JTAG interface, which requires no MCU participation. This means that a completely blank PSD can be soldered into place, and the entire chip can be programmed in-system using ST's FlashLINK JTAG cable and PSDsoft development software, which may be purchased from ST (please see *www.st.com/flashpsd* for details of how to order).

Figure 2 shows a two-chip solution using an M88 FLASH+PSD. This system has ample main Flash memory, an optional secondary Flash memory or EEPROM, and an optional SRAM. All three of these memories can operate independently and concurrently; meaning the MCU can operate from one memory while erasing/writing the other. The system has programmable logic, expanded I/O, and design security. Since the M88x3Fxx family is 100% ISP, a blank M88x3Fxx can be connected to a ROM-less 80C51XA and initially programmed through the JTAG port. Therefore, no 80C51XA firmware needs to be written up front. Just plug in the FlashLINK cable and begin programming memory, logic, and configuration. This powerful feature of the M88 FLASH+PSD family allows immediate development of application code in your lab, smart manufacturing techniques, and easy field updates.

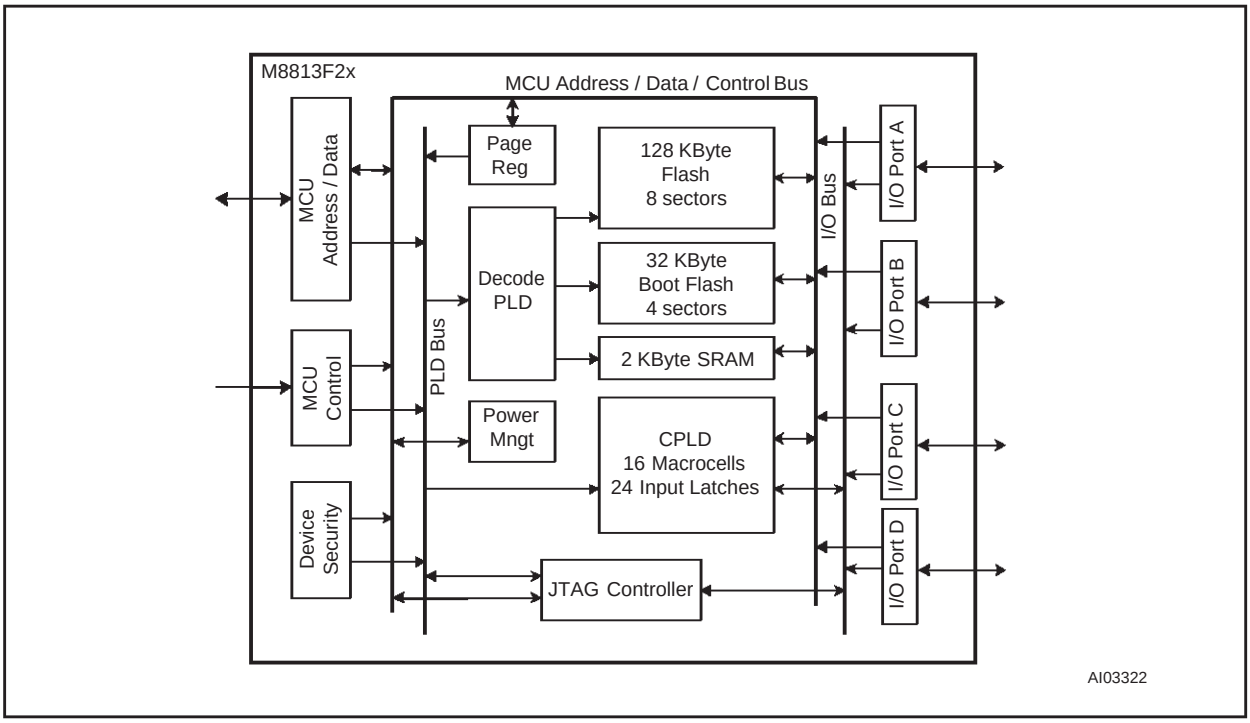
Figure 2. Embedded Flash System Capable of ISP (2 devices)



Taking a quick look inside the M8813F2x, as shown in Figure 3, you can see the three independent memory arrays, which are selected on a segment basis when the proper MCU address is decoded in the Decode PLD. The page register participates in memory decoding, which greatly simplifies paging. The MCU address, data, and control signals are routed throughout the chip and can be used within the general-purpose CPLD. The CPLD has 16 macrocells and 24 special latches for input signals. All 16 macrocells and 24 input latches can be accessed directly by the MCU in silicon. No extra design effort is needed. This MCU access feature is great for loadable shift registers, counters, mailboxes, and state machines. There are 27 I/O pins that can be used for PLD I/O, latched address output, MCU I/O, Peripheral I/O, and ISP. A power management scheme can selectively shut down parts of the chip and tailor special power saving mechanisms on-the-fly. The security feature can block access to all areas of

the chip from a device programmer/reader. Lastly, the self-contained JTAG controller allows ISP of all areas of the chip, even on a completely blank device that is soldered onto a circuit board.

Figure 3. Top Level Block Diagram of M8813F2x



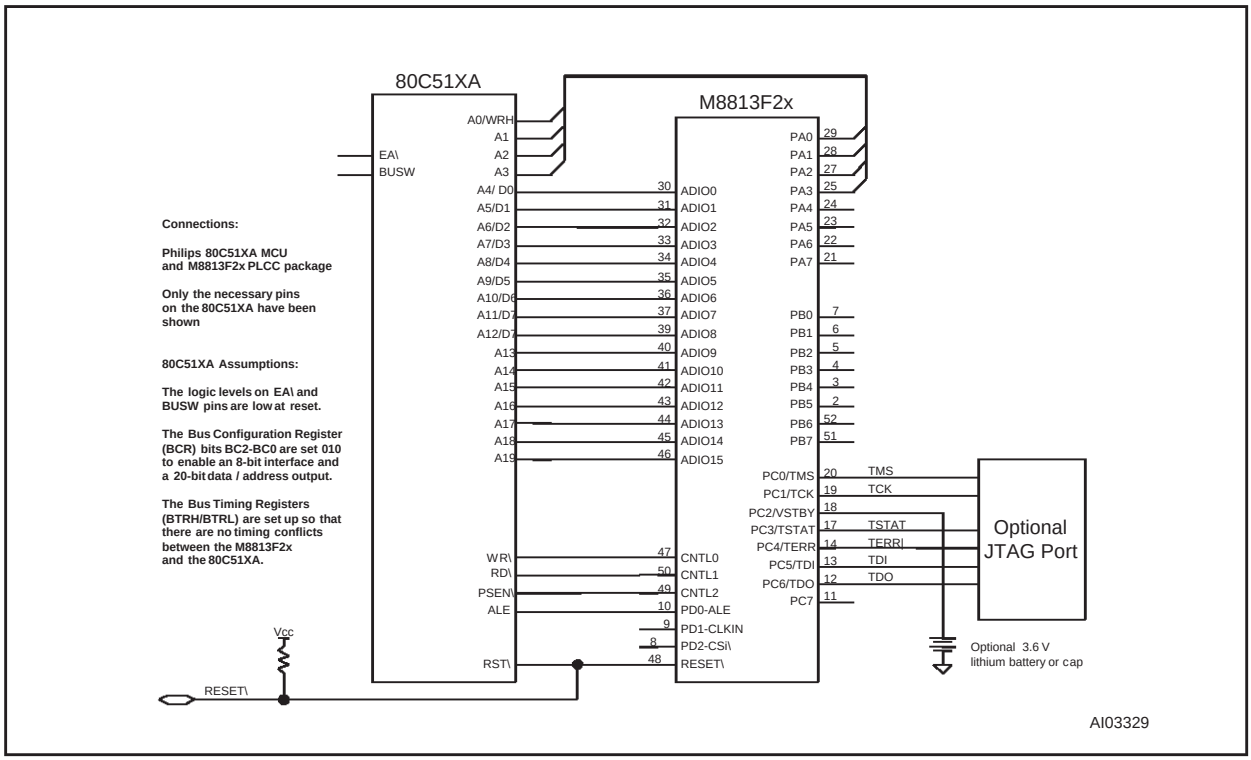
SIMPLE DESIGN EXAMPLE

The first design example outlines the steps required to get an 80C51XA-based system up and running quickly. A connection diagram, memory map, and the necessary design file for the PSDsoft software development environment are provided. An M8813F2x was used for this example. However, other members of the M88 FLASH+PSD family may be used instead, with minor changes to the sample design file. Please see the *M88 FLASH+PSD Family Data Sheet* for a comparison of family members.

Physical Connections

Connect your 80C51XA to the M8813F2x as shown in Figure 4. Similar connections can be used for other members of the M88 FLASH+PSD family. The JTAG programming channel and SRAM with battery backup connections are optional.

Figure 4. Physical Connections, 80C51XA and M8813F2x



Memory Map

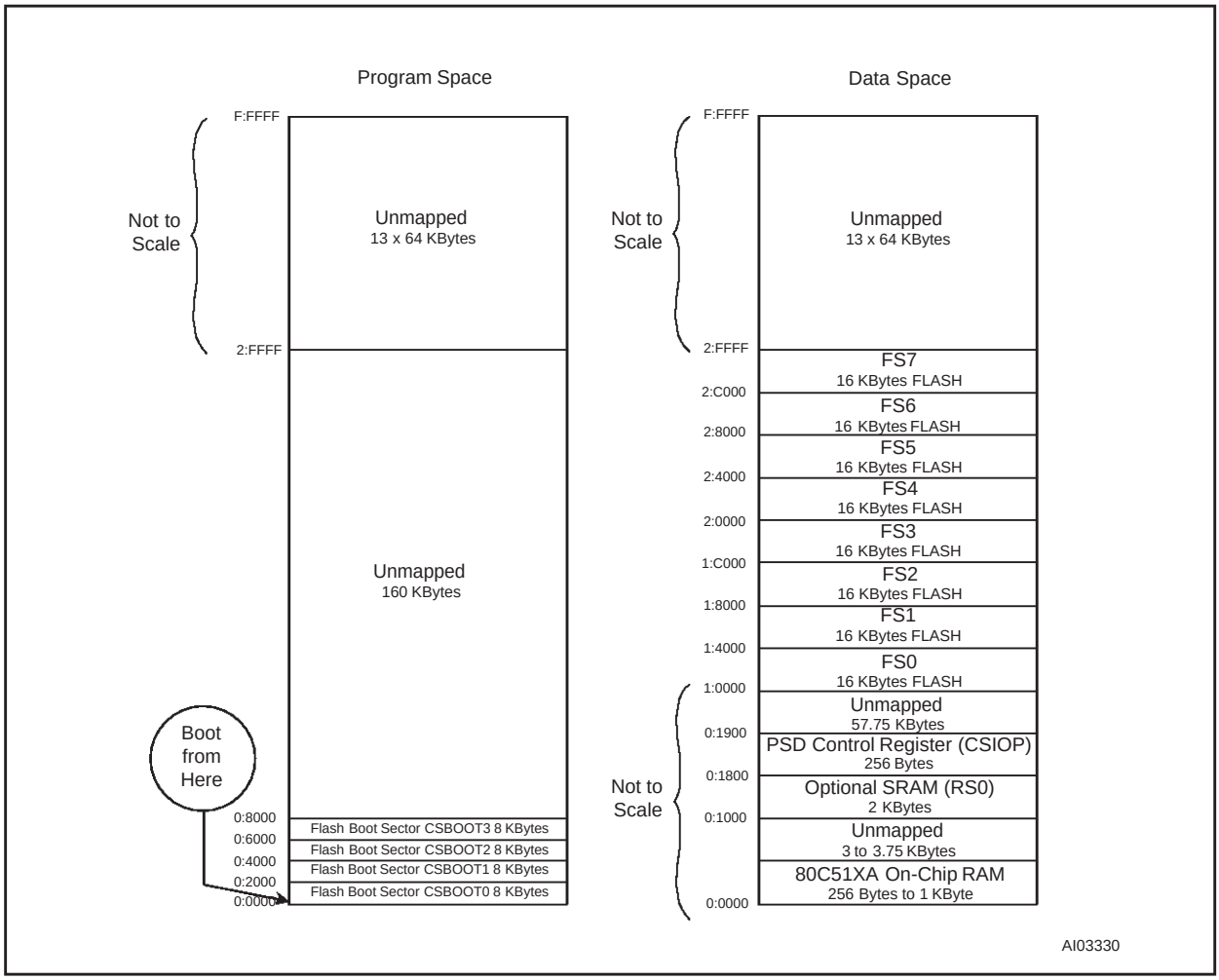
For this simple design, we used an M8813F2x with the following memories:

- 128 KBytes main flash memory, broken into eight 16 KByte segments denoted $fs_{i\ (i = 1\text{-}8)}$
- 32 KBytes boot Flash memory, broken into four 8 KByte segments denoted $csboot_{j\ (j = 1\text{-}4)}$. The M8813F1x has a boot EEPROM instead of Flash memory. Therefore, $ees_{j\ (j = 1\text{-}4)}$ would be used in place of $csboot_j$.
- 2 KByte SRAM (rs0)
- 256 Byte configuration register (csiop).

The PSD memory segments are defined in the PSDabel file in PSDsoft. We use the boot memory to hold the ISP boot loader code, 80C51XA interrupt vectors, and common firmware functions. For this example,

we execute from boot flash only and leave the main Flash memory in Data Space. A sample memory map is shown in Figure 5.

Figure 5. Memory Map, Simple 80C51XA / M8813F2x Design



Note the following about the sample memory map shown in Figure 5:

- It is broken up into sixteen 64 KByte segments.
- It shows both Program Space and Data Space.
- The 32 KBytes of the boot memory is mapped to Program Space.
- The main flash memory is mapped to Data Space so that the contents can be programmed.
- The PSD Control Register and SRAM are in the bottom 64 KByte segment of Data Space.

Note that placing the main flash and boot memory into Program Space or Data Space is accomplished with the PSD VM Register. PSDsoft is used to define the initial value of the VM Register when the system powers up or is reset. This initial value is stored in the fuse-map that gets programmed into the PSD. At run-time, the VM register can be changed by writing to it with the MCU. This is illustrated in the enhanced design (starting on page 17).

The boot memory holds the following information:

- 80C51XA reset vector and initialization routines

- 80C51XA interrupt vectors and service routines
- I/O management
- SRAM variables and SRAM stack.

Since Figure 5 is a sample memory map, you may wish to change it. To do so, simply change the Hardware Description Language (HDL) equations for the Decode PLD for the desired segments. Equations are written and compiled in the PSDsoft software development environment, using the standard Hardware Description Language, ABEL (called “PSDabel” as packaged in PSDsoft). For example, if you have an M88 FLASH+PSD part that does not contain the optional boot memory, you will want to have the main flash located at the bottom of Program Space (from 0:0000h to 1:FFFFh). See Appendix A for a sample memory map for parts with no secondary boot memory.

PSDsoft Design Entry

Highlights of design entry will be given here. Please refer to Application Note *AN1154* for a thorough coverage of all the features of PSDsoft. This section is meant to show you just the essentials to get you going. Here are the steps:

Invoke PSDsoft and Open a New Project

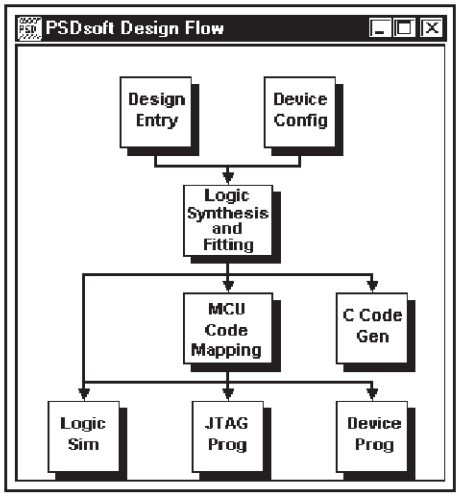
- Start PSDsoft.
- Open a new project.
- Enter your project name and directory (in this example, it is named “yourfile” in the directory PSDSOFT\YOUR_PROJECT).
- Select an M8813F2x
- Select the 80C51XA template
- Click OK.

Now you have your project established, based on an M8813F2x and a 80C51XA.

Design Flow Window

The design flow window shows all of the major steps of the design process, as shown in Figure 6. Clicking on a box within the design flow window invokes the associated process. You should see this:

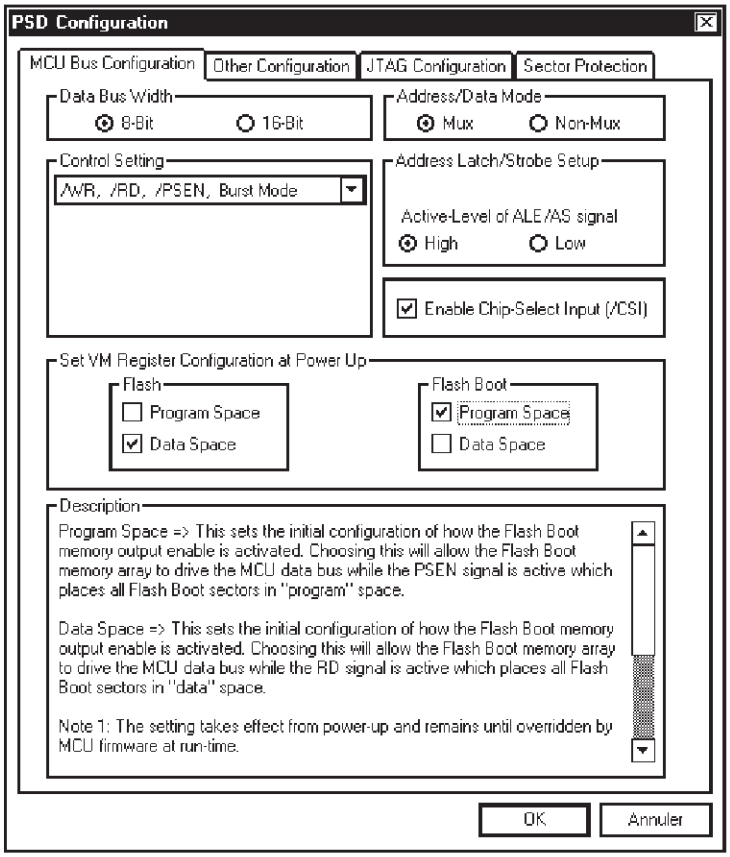
Figure 6. Design Flow Window



Device Configuration

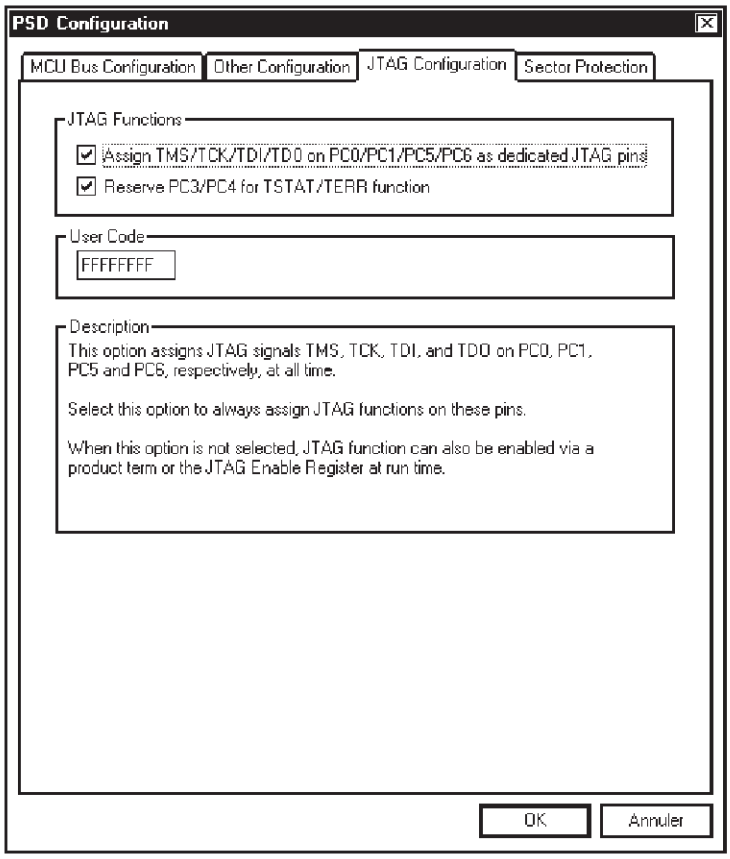
Click on the Device Configuration box in the design flow window to configure the PSD for connection to a 80C51XA. You should make selections with your mouse to match Figure 7.

Figure 7. Device Configuration Window



Now go to the JTAG Configuration tab and click the boxes as shown if you want to dedicate six pins of Port C to be the JTAG channel for In-System-Programming, as shown in Figure 8. See Application Note *AN1153* for options regarding JTAG.

Figure 8. Device Configuration Window – JTAG Configuration Tab



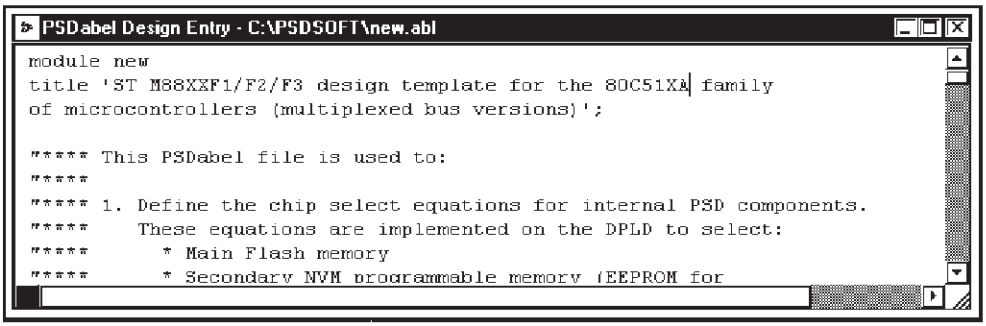
Click OK to save this configuration and get back to the design flow window.

By default, the device is set to allow JTAG downloads, but the default setting of PSDSOFT is to disable this function. If you do not configure the device in the way described (in Figure 8) the JTAG download cell will be disabled for all subsequent programming sequences, and the device will disallow JTAG downloads. From this state, the only way to restore the device is to use PSDpro programmer (M8PSDPRO). You are strongly advised to use only the configuration described above.

PSDabel Design Entry

Click on the “Design Entry” box of the design flow window so you can enter your HDL equations using PSDabel. Since you selected an 80C51XA template previously, the basis of the design will already be there; you just have to edit the PSDabel template, as shown in Figure 9.

Figure 9. PSDabel Design Entry Window



Edit the template

The default 80C51XA template that you see on the screen has numerous elements inside to illustrate many features of the M88 FLASH+PSD family. To simplify things, edit the template file that you see to look like the simple PSDabel file shown here. You can cut and replace the following text from this document by using Adobe's "Text Selection Tool". Be sure to change the module name in the very first line from "new" to whatever you named your project when you first created it.

```
module new
title 'Simple 80C51XA embedded flash design';

////////////////////////////////////////////////////////////////////////////
// The following declarations are 80C51XA bus input signals to the PSD PLDs.
//
wr    pin;"Active-low write strobe input
rd    pin;"Active-low read strobe input
psen   pin;"Active-low Program Space enable input
ale   pin;"Address latch enable input
reset  pin;"Active-low reset input
a19..a0 pin;"Address input (demultiplexed)
address = [a19..a0];"Group the address inputs
//
////////////////////////////////////////////////////////////////////////////
//
// Port pin assignments. See Application Notes AN1171 & AN1154 for details on
// assigning port pins. No I/O pins are used in this simple 80C51XA example.
//
// Be aware that you have clicked the boxes in 'Device Config' to dedicate
// 6 of the 8 Port C pins for JTAG use. See Application Note AN1153 for details
// on JTAG and Port C
////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////
//
//            Internal Node Declarations
//
// Main flash memory segments: all M88x3Fxx devices have a 256 KByte main
// flash, which is broken up into eight 16 KByte segments (fs0-fs7).
//
fs7..fs0node;
//
// Optional Boot flash: the M8813F2x devices have a 32 KByte boot flash.
// This boot flash is broken up into four 8 KByte segments (csboot0-csboot3).
//
csboot3..csboot0node;
//
// Optional SRAM: the M88x3Fxx devices have a 2 KByte SRAM. The
// SRAM is one segment.
//
rs0 node;
//
// PSD Control Register: all M88x3Fxx devices have a control register that
// is 256 bytes.
//
csiopnode;
////////////////////////////////////////////////////////////////////////////
//
//              Equations Section
//
EQUATIONS
//
// Write chip select equations to implement the memory map
//
fs0 = (address >= ^h10000) & (address <= ^h13FFF);
fs1 = (address >= ^h14000) & (address <= ^h17FFF);
fs2 = (address >= ^h18000) & (address <= ^h1BFFF);
fs3 = (address >= ^h1C000) & (address <= ^h1FFFF);
fs4 = (address >= ^h20000) & (address <= ^h23FFF);
fs5 = (address >= ^h24000) & (address <= ^h27FFF);
fs6 = (address >= ^h28000) & (address <= ^h2BFFF);
fs7 = (address >= ^h2C000) & (address <= ^h2FFFF);

csboot0 = (address >= ^h00000) & (address <= ^h01FFF);
csboot1 = (address >= ^h02000) & (address <= ^h03FFF);
csboot2 = (address >= ^h04000) & (address <= ^h05FFF);
csboot3 = (address >= ^h06000) & (address <= ^h07FFF);
```

```
rs0 = (address >= ^h01000) & (address <= ^h017FF);

csiop = (address >= ^h01800) & (address <= ^h018FF);

//
// Write equations for general logic and external chip select equations
// here. See Application Notes AN1171 and AN1154 for details.
//

end
```

Notice the following about this simple PSDabel file (in order):

- The pin declarations for connection to the 80C51XA occur at the beginning of the file.
- No PSD general-purpose I/O pins are declared in this example for simplicity. These features can be easily implemented. (See Application Notes *AN1171* and *AN1154* for examples using I/O pins.)
- Internal PSD nodes are declared for memory selection and PSD control.
- The HDL equations appear for the selection of internal PSD memory segments and the PSD control register. Each memory segment is assigned to an address range according to the memory map of Figure 5.
- No equations for the CPLD were shown. For examples, see Application Notes *AN1171* and *AN1154*.

Logic Synthesis and Fitting

After you have edited the PSDabel template file to look like the above file, go to the design flow window and click on the “Logic Synthesis and Fitting” box. Now PSDsoft will compile the PSDabel file and then synthesize the PSDabel statements into reduced logic that fits the M8813F2x silicon. When this process is complete, a report will pop up that shows the resulting pin assignments and reduced equations. This is the fitter report, which you can use to document your design. Notice the pin assignments for the JTAG channel in the fitter report.

C Code Generation

You can take advantage of the provided low-level C code drivers for accessing memory elements within the PSD by clicking on the “C Code Gen” box in the design flow window. To get the C functions and headers, specify the folder in which you want the ANSI C files to be written. ANSI C code functions and headers are generated for you to paste into your 80C51XA C compiler environment in the folder you specify. Simply tailor the code to meet your system needs. See Application Note *AN1154* for details on the C code generation feature.

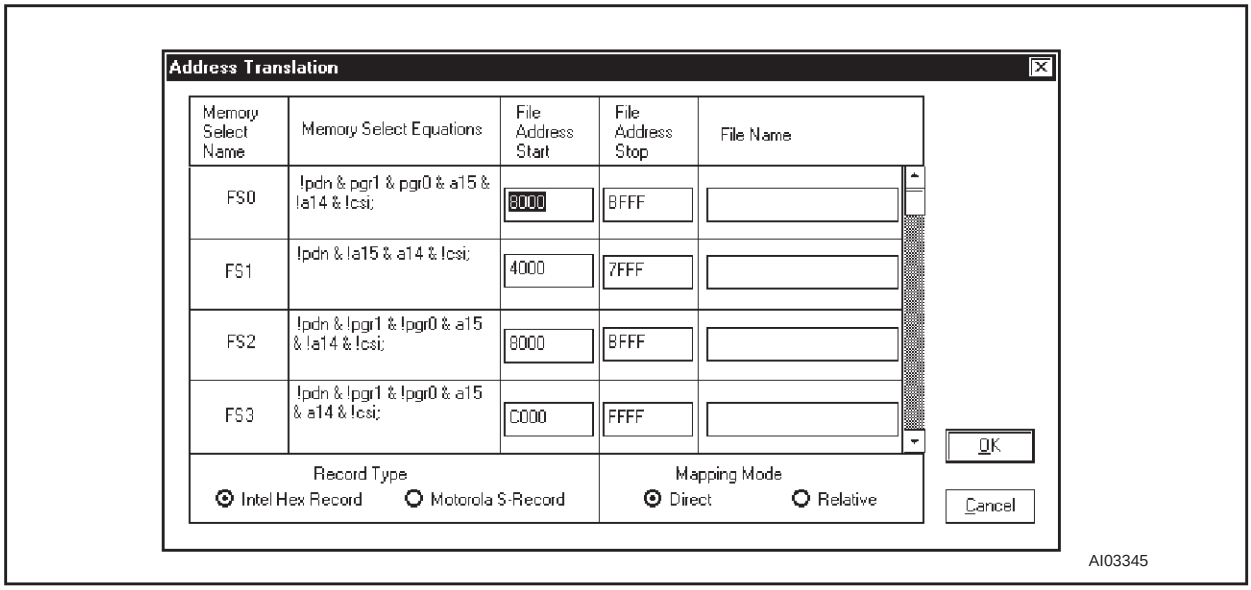
MCU Code Mapping

Now that the fitting process is complete, PSDsoft has created a fuse pattern that reflects the PSD configuration and logic of your design. PSDsoft places this fuse information into a file (the .obj file). However this fuse pattern does not yet contain the 80C51XA firmware. The next step will accomplish this, producing a .obj file that contains the PSD configuration AND the 80C51XA firmware. This final .obj file is what gets programmed into the PSD. Note: the first .obj file that is created without MCU firmware can be used for logic simulation. See Application Note *AN1154* for details. That same .obj file is appended with MCU firmware in the next step below.

For this step, MCU Code Mapping, you will input the firmware file(s) that contain absolute addresses from your 80C51XA compiler/linker (S-record or Intel HEX format). The Address Translator will map these file(s) into the memory segments of the PSD according to the HDL equations that you entered in PSDabel. This mapping process translates the absolute system addresses that 80C51XA uses into physical internal PSD addresses that are used by a programmer to program the PSD. The address translation process is transparent. All you need to do is type in the file name(s) that were generated from your 80C51XA linker into the appropriate boxes, and PSDsoft does the rest.

To perform the address translation, go to the design flow window and click on the “MCU Code Mapping” box, as shown in Figure 10.

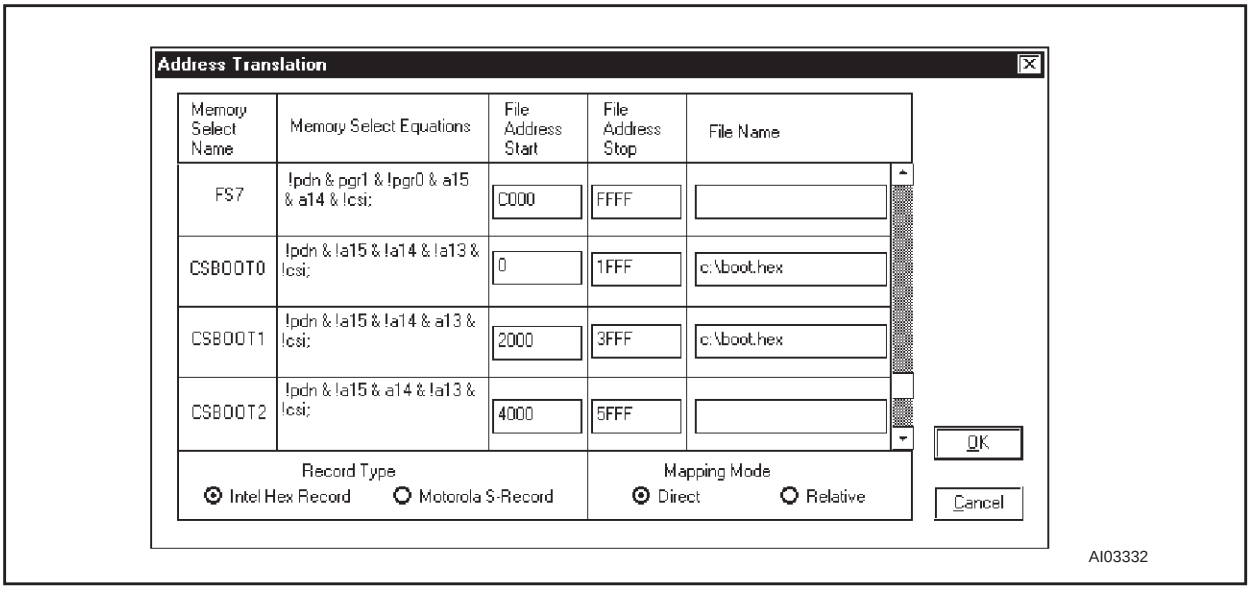
Figure 10. Address Translation Window



The far left column contains individual PSD memory segments. The next column shows the logic equations for selection of each memory segment (shown for reference only). In the middle are the address ranges that were specified in the PSDabel file to create the memory map. PSDsoft filled in these address fields for you. PSDsoft expects to find these absolute MCU addresses within your 80C51XA linker file(s) when they are imported. On the right are boxes where you type in the name of the file(s) (including path) that your 80C51XA linker created. Notice that you can select Motorola S-Record or Intel Hex Record for the input type. Leave the “Mapping Mode” set to “Direct”.

Now slide the scroll bar down until you see csboot0 and csboot1, as shown in Figure 11.

Figure 11. Address Translation Window – after scrolling



Type in the name of the file from your 80C51XA linker that contains the firmware that will boot up your system. For this example we called it boot.hex. This file can contain very simple 80C51XA code that configures your system hardware and performs rudimentary tasks to check out your new hardware. In this example, there are 32 KBytes available in flash boot segments csboot0 and csboot1, which is more than enough for this simple boot and test code. After your new hardware is proven, you can add more code to the boot area for advanced tasks, such as implementing a download to main flash memory from a host computer, as shown in the enhanced design (starting on page 17).

No file names are required for the main flash regions (fs0-fs7) because we are only operating out of boot flash for now.

Click OK, and the address translate process will produce the final .obj file that you can use to program the PSD.

Programming the PSD

The .obj file can be programmed into the PSD in one of three ways:

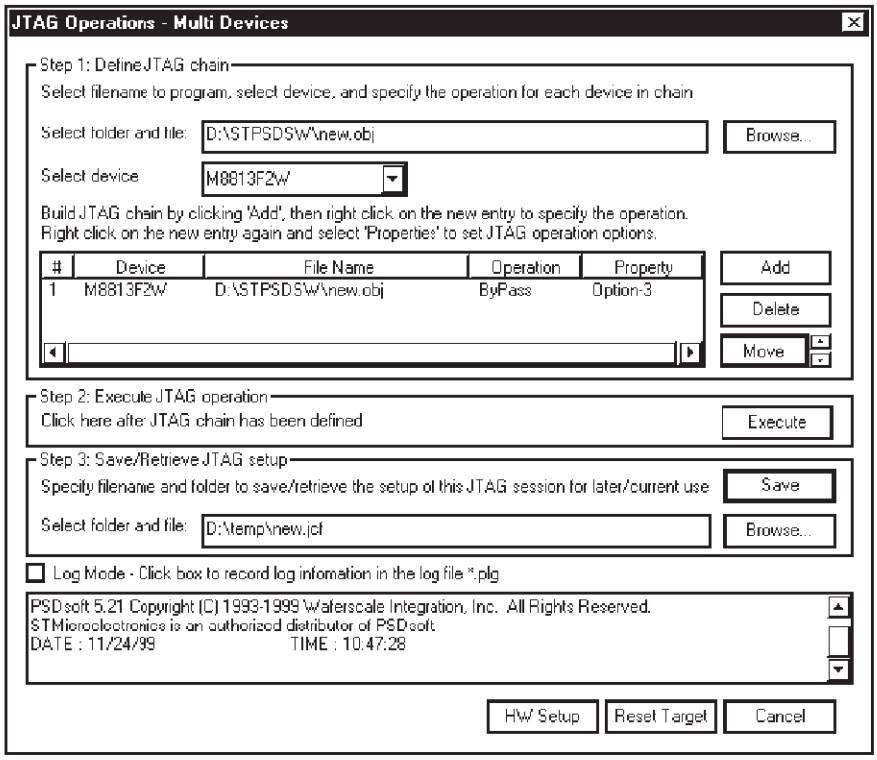
- The ST FlashLINK JTAG cable, which connects to the PC parallel port.
- The ST PSDpro device programmer, which also connects to the PC parallel port.
- Third-party programmers, such as DATA I/O, Stag, and Needhams. Please see the web site at *www.st.com/flashpsd* for a list of compatible third-party programmers.

First we show you how to use the FlashLINK JTAG cable to program the PSD.

Programming with FlashLINK

Connect the FlashLINK JTAG cable to the PC parallel port. Click the “JTAG Programming” box in the design flow window, to make the window, as shown in Figure 12, pop up.

Figure 12. JTAG Chain Setup Window



This window allows you to describe the JTAG chain that exists on your circuit board, your desired operation, and also offers a loop-back test for your FlashLINK cable. If this is your first time using the FlashLINK cable, you should test it by clicking on the “HW Setup” button, then click the “LoopTest” button and follow the directions.

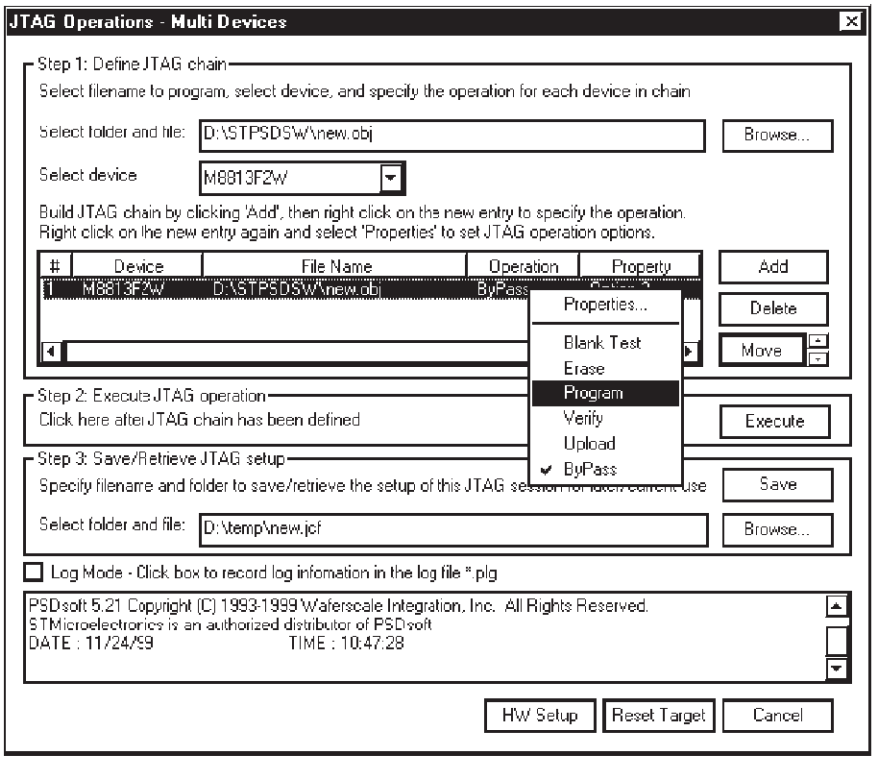
This design example uses a JTAG chain of one device, which is the most typical case. Now to define the JTAG chain:

- Specify the .obj file to be programmed into the PSD. Click the “browse” button within the “Chain Information” area to find your file. In this example, it is yourfile.obj, as shown in Figure 12.
- Choose the device name, M8813F2x, also as shown in Figure 12.
- Click the “Add” button to add the .obj file and the PSD device to the chain definition, as shown.

Now that the file and part are specified, you must specify the operation that you want to perform. In this case we want to program the device. Notice in the highlighted line for device number one in the window above, the operation is currently “ByPass”. This is the default operation for all devices that are added to the JTAG chain.

Let us change that operation by clicking the right mouse button on the highlighted line, as shown in Figure 13.

Figure 13. JTAG Chain Setup Window – after pulling down the Bypass menu



Now choose “Program” from the list that appears in the small pop up box. Once you select “Program”, you will be given a choice to program all of the PSD, or just certain regions. Click “All” for this example.

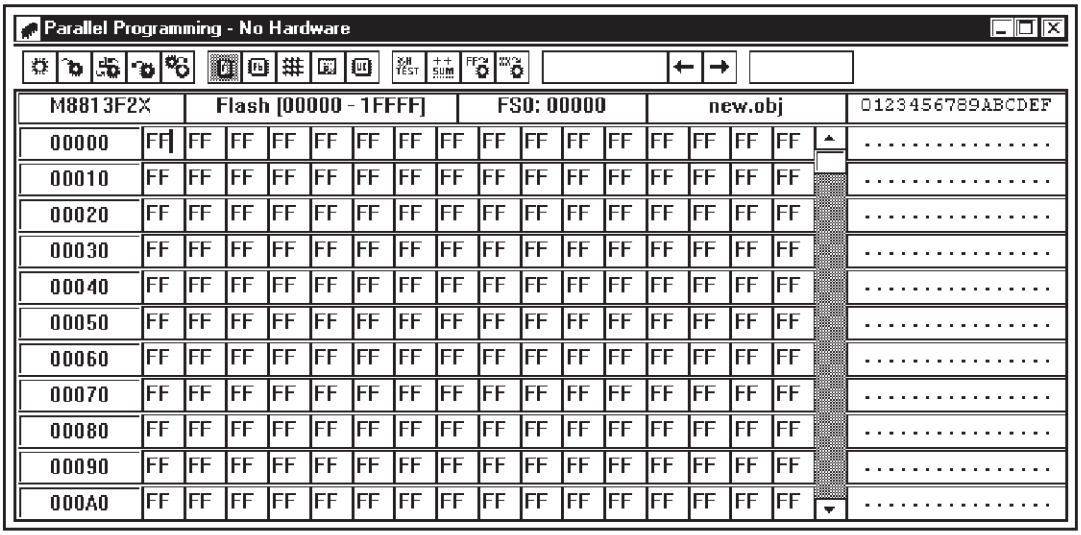
Again, right-click on the highlighted line for device number one. This time, just for reference, click on “Properties”. This is where you can specify use of 6 pins or 4 pins for the JTAG channel. You can also specify the default state of I/O pins while JTAG operations are occurring. Notice that the default JTAG setup in this window is “Option 3” (6 JTAG pin channel), and all I/O pins are in high-impedance input mode. Just click “Cancel” to move on, since we are using all the default choices on this screen.

Now everything has been defined in the JTAG chain of one device: the file name, the type of PSD, and the desired operation. To begin programming the device, just click the “Go” button. After programming is complete, you can save the JTAG setup for this programming session to a file for later use. To do so, click on the “Save” button in the “JTAG Chain File” section of the window. See Application Notes *AN1153* and *AN1154* for details on all the features of the JTAG channel and this JTAG Chain Setup window.

Programming with PSDpro

Connect the PSDpro device programmer to your PC parallel port per the installation instructions. Click on the “Device Programmer” box in the design flow window, as shown in Figure 14.

Figure 14. Parallel Programming Window



If this is the first use of the PSDpro, click on the “Htest” icon to perform a test of the PSDpro and the PC port. After testing, place an M8813F2x into the socket of the PSDpro and click on the “Program” icon. (The .obj file is automatically loaded when this process is invoked). The messaging of PSDsoft will inform you when programming is complete. See Application Note *AN1154* for details on all the features of this “Parallel Programming” window.

This window is also helpful even if you do not have a PSDpro programmer. You can use this window to see where the Address Translate utility of PSDsoft has placed the 80C51XA firmware within physical memory of the PSD. For this design example, you can click on the boot flash icon in the tool bar. Notice the 80C51XA reset vector at absolute MCU addresses 00000h and 00002h, translates to PSD boot flash physical addresses 20000h and 20002h, respectively. To see how all of your 80C51XA absolute addresses translated into physical PSD memory addresses, click “View” from the main tool-bar and select “Address Translation Report”. The start and stop addresses in the report are the absolute MCU system addresses that you have specified. The addresses shown in square brackets are direct physical addresses used by a device programmer to access the memory elements of the PSD in a linear fashion (a special device programming mode that the MCU cannot access).

ENHANCED DESIGN EXAMPLE

This second design example builds upon the first to add enhanced features to this ISP capable system. The physical connections between the 80C51XA and M8813F2x do not change, but the memory map and PSDabel equations do. The focus of this enhanced design is to show how the memories of the M8813F2x can be used concurrently. This means swapping the boot code out of Program Space after the initial boot sequence has completed. The boot code can then be updated if desired.

Physical Connections

Same as the simple design example (on page 4).

Memory Map

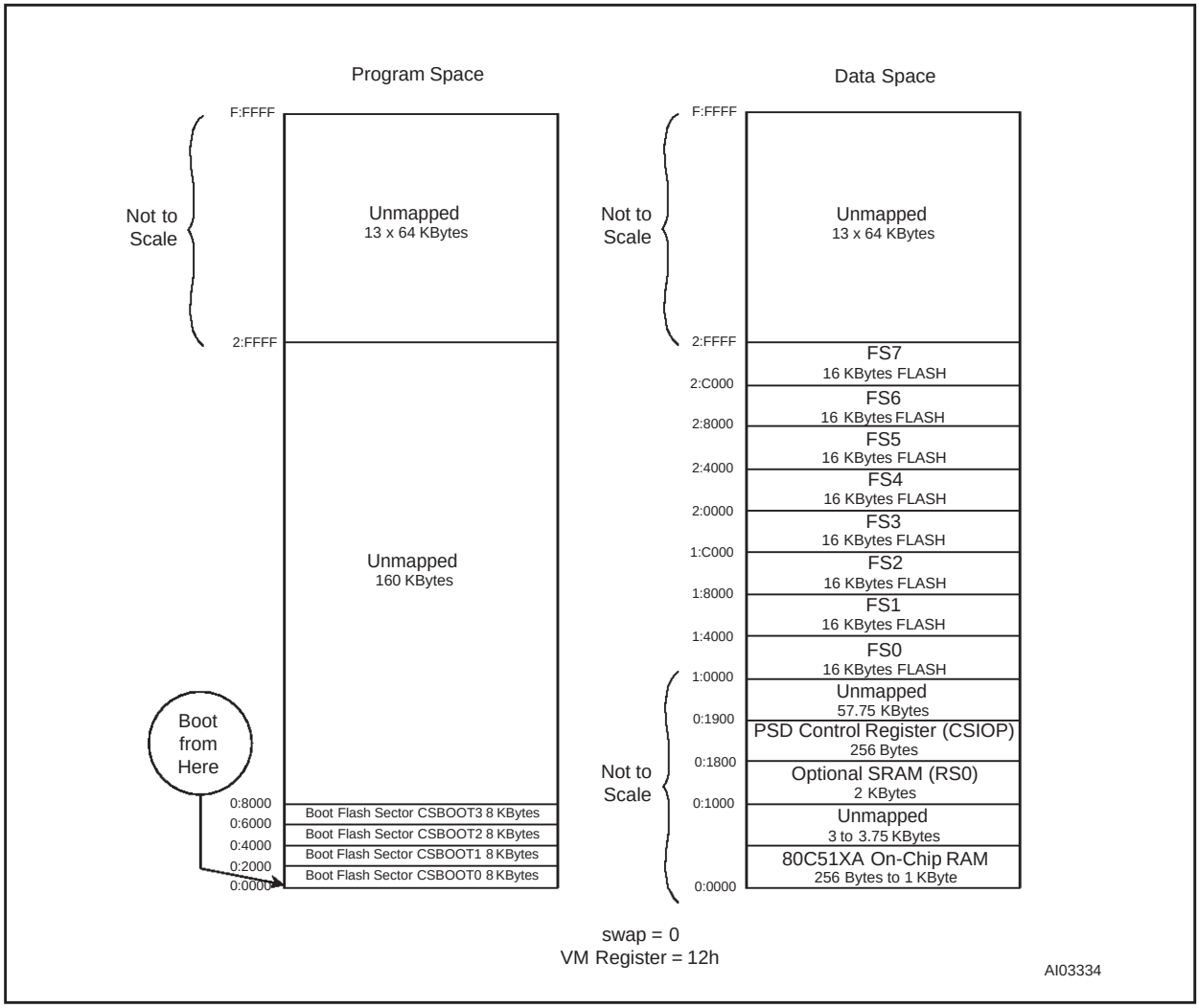
The boot sequence and memory swap is a four step process, as shown in Figure 15 to Figure 19.

Memory Map Configuration at Boot-Up

Figure 15 on the next page shows how the memory map looks at system power-on or at system reset. The SWAP bit is one of the eight internal PSD page register bits, whose value is zero by default. The SWAP bit is an example of how the page register bits can be implemented for uses other than memory paging. The VM Register controls which space (Program or Data) the PSD memories appear in and can be set prior to run-time using PSDsoft Configuration. Here is what the 80C51XA does upon power-up or reset:

- Boot from flash boot csboot0 at address 00000h
- Perform a check-sum of main flash memory
- Download main flash memory from a host computer if needed and validate contents.

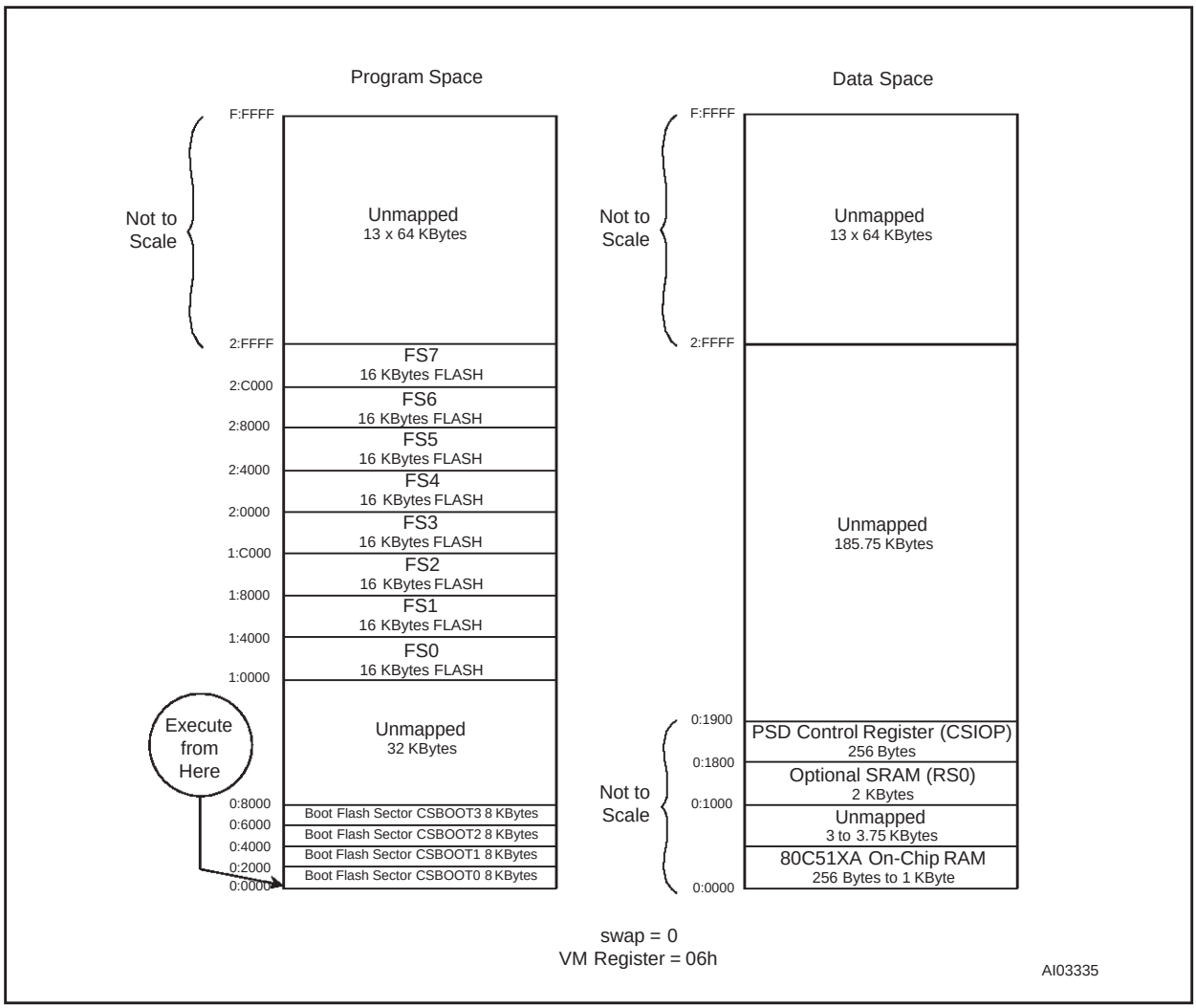
Figure 15. Memory Map, Enhanced Design at Boot-Up/ISP



Memory Map Configuration After Moving the Main Flash

The next step is to move the main flash from Data Space to Program Space. To do so, while executing out of the boot flash, write 06h to the VM register. You will now have the memory map shown in Figure 16.

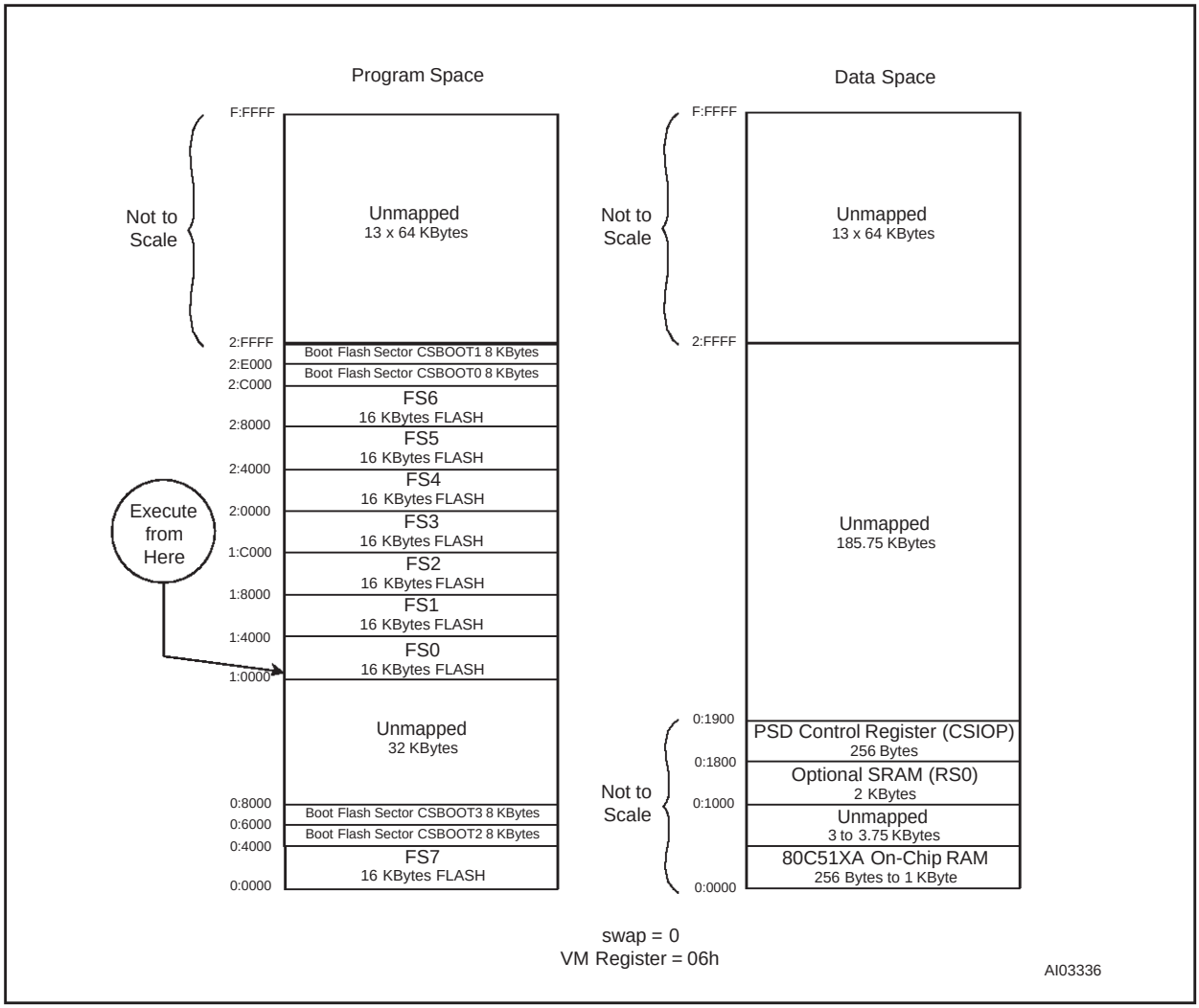
Figure 16. Memory Map After Moving the Main Flash to Program Space



Memory Map Configuration After Setting the SWAP bit

Next, we want to transfer execution to main flash segment fs0. Once we are executing out of main flash, it is desired to set the SWAP bit to re-map the flash boot segments csboot0/csboot1 out of the MCU boot area and replace it with main flash segment fs7, as shown in Figure 17. This swapping action is implemented by including the SWAP bit in the PSDabel equations for the memory chip select equations. (please see the PSDabel file, on page 4)

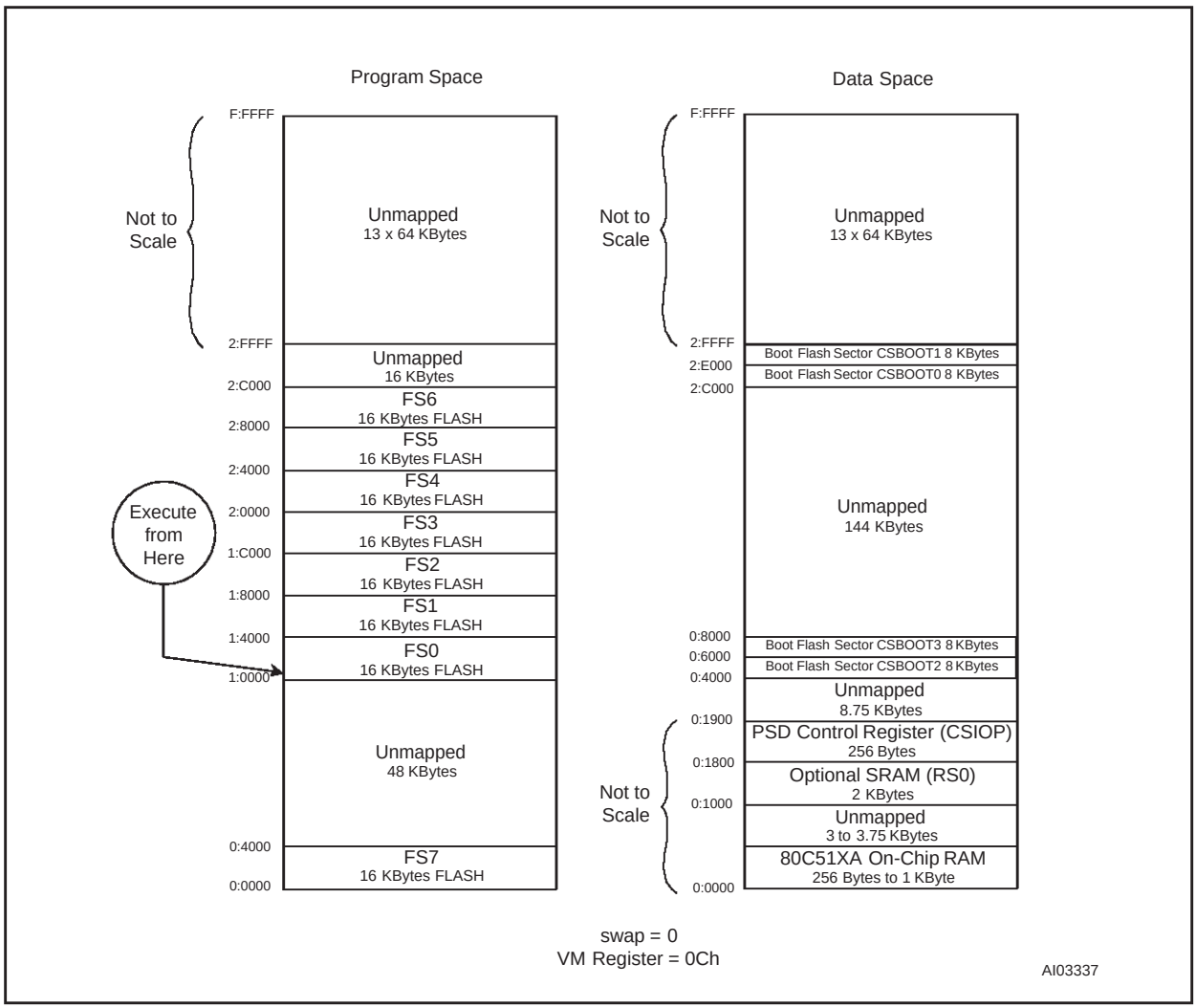
Figure 17. Memory Map After Moving the Boot Flash to Data Space



Memory Map Configuration After Moving the Boot Flash to Data Space

The final step is to move the boot flash to Data Space so that it can be updated if desired. To move the boot flash to Data Space, write 0Ch to the VM register. Once the VM register has been written, you can program either half of the boot flash, depending on how the UNLOCK bit is set. Figure 18 shows the final state of the memory map.

Figure 18. Memory Map After Setting the SWAP Bit



In this final configuration, the 80C51XA has available:

- 16 KBytes main flash memory in the boot area (00000h-03FFFh)
- 112 KBytes main flash in Program Space (10000h-2BFFFh)
- 2 KBytes of SRAM in addition to the SRAM that resides on the 80C51XA
- 16 KBytes of boot flash for general data storage (04000h-07FFFh)
- 16 KBytes of boot flash for boot and ISP loader code (2C000h-2FFFFh).

Each time this 80C51XA system gets reset or goes through a power-on cycle, the PSD presents the memory map of Figure 15 to the MCU, and the boot sequence is repeated.

PSDsoft Design Entry

The 80C51XA design template that is installed with your copy PSDsoft contains the PSDabel file needed to implement this second design example. Just do the following:

- Open a new project
- Select an M8813F2x
- Select the 80C51XA design template to get the core of this design

- Tailor the PSDabel template to meet your system needs
- Synthesize and fit the design
- Map the MCU Code (80C51XA firmware)
- Program the part.

When mapping the 80C51XA firmware in the Address Translate utility of PSDsoft for this second design example, you still do not need to specify any HEX or S-Record file for the PSD main flash area. You only need to specify the 80C51XA linker file(s) for the boot flash area (as in the first simple design) because the 80C51XA will execute code from boot flash and download to main flash memory. For reference, the following are the only PSDabel statements that differ between the simple design and this enhanced design.

```
swap node 117;   ”This is an unused page register bit ’pgr7’
unlock node 116;  ”This is an unused page register bit ’pgr6’

// Node numbers are used the declarations above instead of reserved
// the names, pgr7 and pgr6. This allows the actual names, SWAP and
// UNLOCK, to appear throughout the reduced logic equations, making
// reports more readable also making files compatible with future
// PSDsoft features.

// The following memory select equations match the memory maps of
// Figure 16 to Figure 19. Notice the use of the SWAP and UNLOCK bits.

fs0 = (address >= ^h10000) & (address <= ^h13FFF);
fs1 = (address >= ^h14000) & (address <= ^h17FFF);
fs2 = (address >= ^h18000) & (address <= ^h1BFFF);
fs3 = (address >= ^h1C000) & (address <= ^h1FFFF);
fs4 = (address >= ^h20000) & (address <= ^h23FFF);
fs5 = (address >= ^h24000) & (address <= ^h27FFF);
fs6 = (address >= ^h28000) & (address <= ^h2BFFF);
fs7 = ((address >= ^h2C000) & (address <= ^h2FFFF) & !swap)
  # ((address >= ^h00000) & (address <= ^h03FFF) & swap);

csboot0 = ((address >= ^h00000) & (address <= ^h01FFF) & !swap)
    # ((address >= ^h2C000) & (address <= ^h2DFFF) & swap & !unlock);
csboot1 = ((address >= ^h02000) & (address <= ^h03FFF) & !swap)
    # ((address >= ^h2E000) & (address <= ^h2FFFF) & swap & !unlock);
csboot2 = (address >= ^h04000) & (address <= ^h05FFF) & swap & unlock;
csboot3 = (address >= ^h06000) & (address <= ^h07FFF) & swap & unlock;
```

CONCLUSION

These examples are just two of an endless number of ways to configure the M88 FLASH+PSD for your system. Concurrent memories with a built-in programmable decoder at the segment level offer excellent flexibility. Also, as you have seen with the SWAP and UNLOCK bits, the page register bits do not have to be used just for paging through memory. The ability to expand your system does not require any physical connection changes, as everything is configured internal to the PSD. And finally, the JTAG channel can be used for ISP anytime, and anywhere, with no participation from the MCU. All of these features are cross-checked under the PSDsoft development environment to minimize your effort to design a flash 80C51XA system capable of ISP.

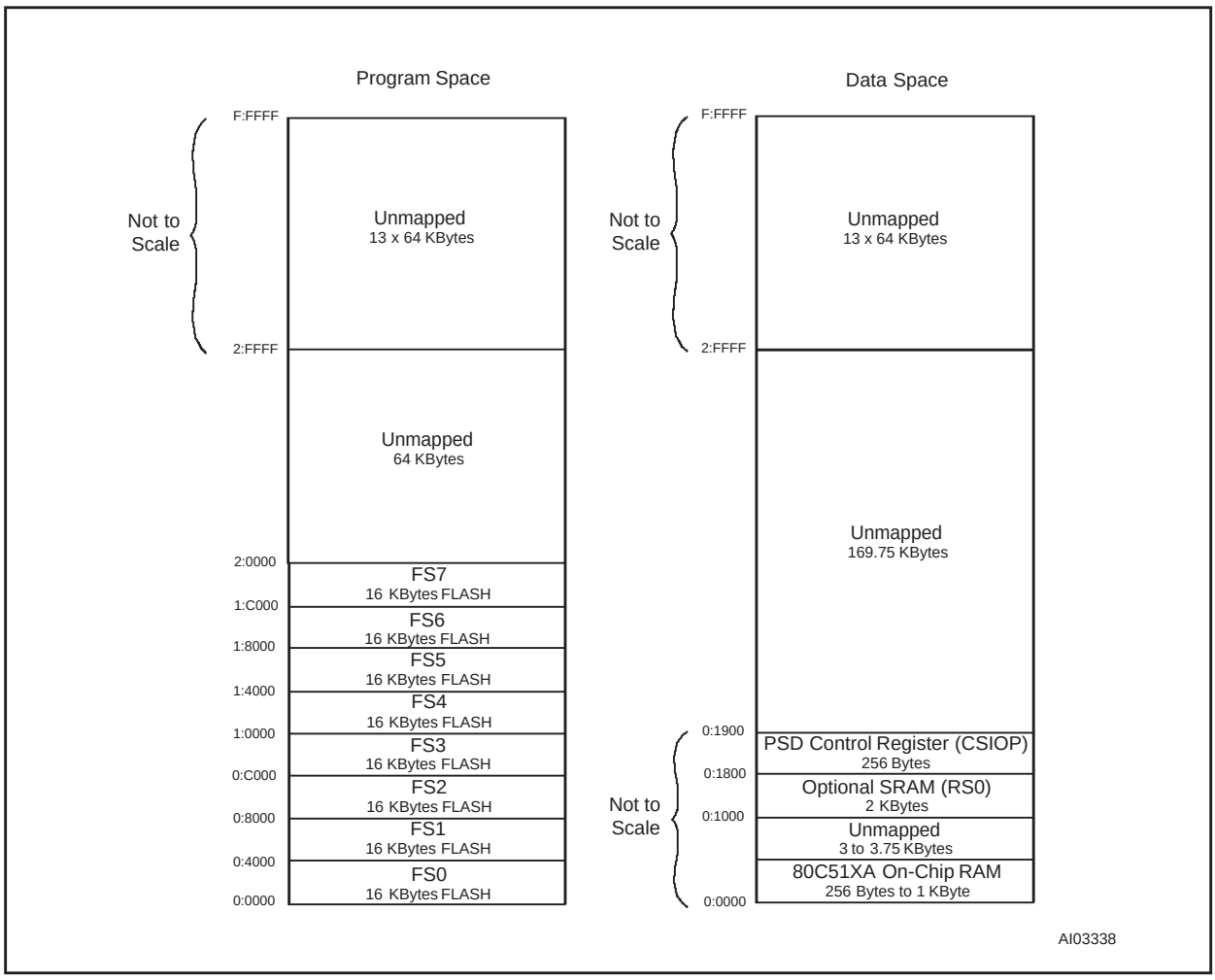
REFERENCES

- *M88 FLASH+PSD* Family Data Sheet
- Application Note *AN1153* for detailed use of the JTAG channel
- Application Note *AN1171* for details on the CPLD and I/O pins
- Application Note *AN1154* for a design tutorial, and details on PSD I/O, CPLD, logic simulation, and PSDsoft features.
- Application Note *AN1176* for a design guide for the 68HC11 and M8813F1x.
- Application Note *AN1178* for a design guide for the 80C51 and M8813F2x.

APPENDIX A—CONNECTING TO AN M88X3F3X (WITH NO BOOT MEMORY)

Figure 19 shows a sample memory map for connecting to an M88x3F3x (with no secondary boot memory). This memory map assumes you have downloaded the main flash with the FlashLINK cable or you have booted from a separate PROM and have downloaded the flash using the MCU. In either case, you must change your design file (.abl) to account for the different segment locations.

Figure 19. Memory Map for an M88x3F3x Device (with No Secondary Boot Memory)



For current information on M88 FLASH+PSD products, please consult our pages on the world wide web:
www.st.com/flashpsd

If you have any questions or suggestions concerning the matters raised in this document, please send them to the following electronic mail addresses:

apps.flashpsd@st.com (for application support)
ask.memory@st.com (for general enquiries)

Please remember to include your name, company, location, telephone number and fax number.