



## P51XA/PSD8XX Design Guide

---

### CONTENTS

- In-System Programming and In-Application re-Programming
  - The IAP Problem
  - A Common Solution
- Physical Connections
- Simple Design Example
  - Memory Map
  - PSDsoft Express Design Entry
- Enhanced Design Example
  - Required Changes in the PSDsoft Express Design Entry
  - Memory Map
- Conclusion
- References
- Appendix A: Connecting to a PSD813F with no Secondary Memory

Flash PSD8XX devices are members of a family of Flash memory-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 P51XA MCU, the PSD forms a complete embedded flash system that is 100% In-System Programmable (ISP) and In-Application Programmable (IAP). There are many features in the PSD silicon and in the PSDsoft Express development software that make ISP easy, regardless of how much experience you have with embedded design.

This document offers two designs using a ST PSD813F2 and a Philips P51XA MCU. Note that a variety of 8-bit MCU/MPUs can be used in place of the Philips part. Although the specifics of this document are based on the P51XA, this document can be used as a guide for other MCU/MPU applications. The first design is a simple system to get up and running quickly for basic applications or to check out your prototype P51XA hardware. The second design illustrates the use of enhanced features of PSD In-System Programming as applied to the P51XA. You can start with the first design and migrate to the second as your functional requirements grow. There are other members of the PSD8XX family, including the PSD813F1/F3/F4/F5, the PSD833F2/834F2, and the PSD835G2. See the selector guide on the website for a comparison of the products. This application note is applicable to all PSD8XX family members.

### IN-SYSTEM PROGRAMMING AND IN-APPLICATION RE-PROGRAMMING

Our industry uses the term In-System Programming (ISP) in a general sense. ISP is applicable to programmable logic, as well as programmable Non-Volatile Memory (NVM). However, an additional term will be used in this document: In-Application Programming (IAP). There are subtle yet significant differences between ISP and IAP when microcontrollers are involved. ISP of memory means that the MCU is off-line and not involved while memory is being programmed. For IAP, the MCU participates in programming the memory, which is important for systems that must be online while updating firmware. Often, ISP is well suited for manufacturing, while IAP is appropriate for field

## AN1177 - APPLICATION NOTE

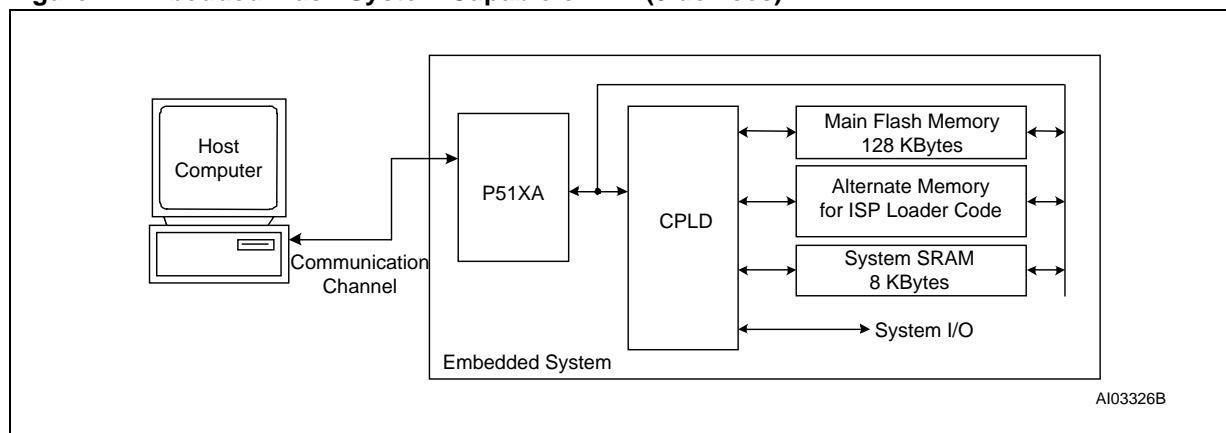
updates. PSD8XX devices are capable of both ISP and IAP. Keep in mind that IAP can only program the memory sections of the PSD and not the configuration and programmable logic portions. With ISP, the entire PSD can be erased or programmed.

### The IAP Problem

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 P51XA supports will do. The P51XA must execute the code that controls the IAP process from an independent memory array that is not being erased or programmed. Otherwise, boot code and Flash memory programming algorithms (IAP loader code) will be unavailable to the P51XA. It is absolutely necessary to use an alternate memory array (an independent memory that is not being programmed) to store the IAP loader code.

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

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



Another problem, which is specific to the P51XA architecture, is related to the separate “Program” and “Data” address spaces. The P51XA cannot write to Program space, but that is where the Flash memory resides that holds P51XA firmware. How can one program Flash memory in-system if the P51XA cannot write to program space?

### A Common Solution

Without a PSD device, implementing IAP with the P51XA can be difficult and time consuming. Philips’ application note *AN440* contains a RAM loader program (bootstrap loader). It shows how to load code into an external RAM over a serial link after power-up and how to switch execution to that RAM to complete the boot sequence. This can be a cumbersome and error prone exercise using re-locatable code in volatile memory, which is difficult to debug, vulnerable to power outages, and not supported by all emulators. Additionally, this method restricts the designer to using a UART to implement IAP.

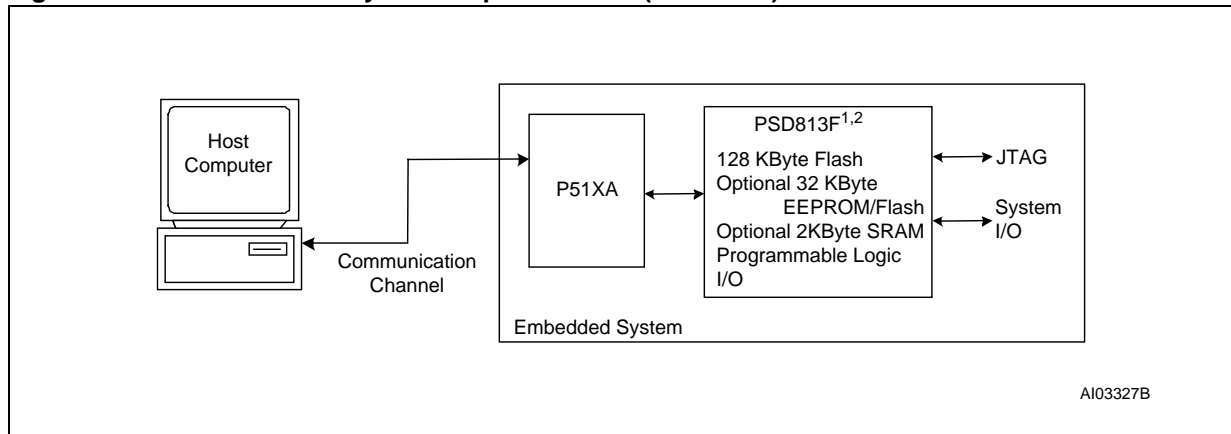
To overcome the issue of Program versus Data space, a common practice is to combine the two address spaces, which reduces the total address space of the P51XA by 50%.

### A BETTER, INTEGRATED SOLUTION

Figure 2 shows a two-chip solution using an Flash PSD813F. This system has ample main Flash memory,

a second alternate Flash memory to hold the IAP loader code and general data, and more SRAM. All three of these memories can operate independently and concurrently; meaning the MCU can operate from one memory while erasing/writing the other. This allows the MCU to continue normal operation during IAP, which is crucial for some applications. This system also has programmable logic, expanded I/O, and design security. The two-chip solution is 100% programmable in the factory or in the field.

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



Note: 1. Other members of the PSD8XX family offer more Flash memory and more SRAM.

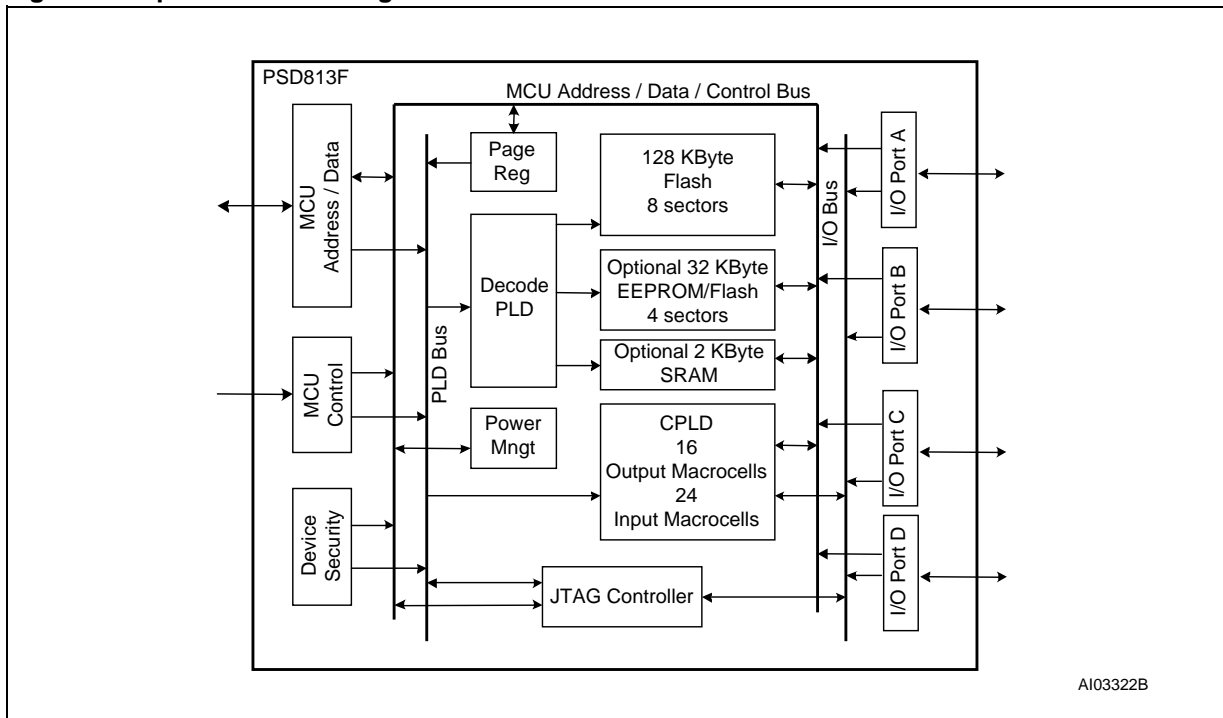
2. Only the PSD813F1 offers EEPROM, while the other members of the PSD8XX family offer secondary Flash memory.

By design, the IAP method described above requires MCU participation to exercise a communication channel to implement a download to the main Flash memory. The PSD8XX also offers an alternative method called In-System Programming (ISP) to program the PSD using a built-in IEEE 1149.1 JTAG interface requiring no MCU participation. This means that a completely blank PSD can be soldered into place and the entire chip can be programmed in-system in just a few seconds using ST's FlashLINK™ JTAG cable and PSDsoft Express development software. No P51XA firmware needs to be written. Just plug in the FlashLINK cable to your PC's parallel port and begin programming memory, logic, and configuration. This is a powerful feature of the PSD8XX that allows immediate development of application code in your lab, smart manufacturing techniques, and easy field updates.

The FlashLINK™ cable and PSDsoft Express software are available in a kit from the website [www.st.com/psd](http://www.st.com/psd).

Figure 3 gives a block diagram of the PSD813F for your reference.

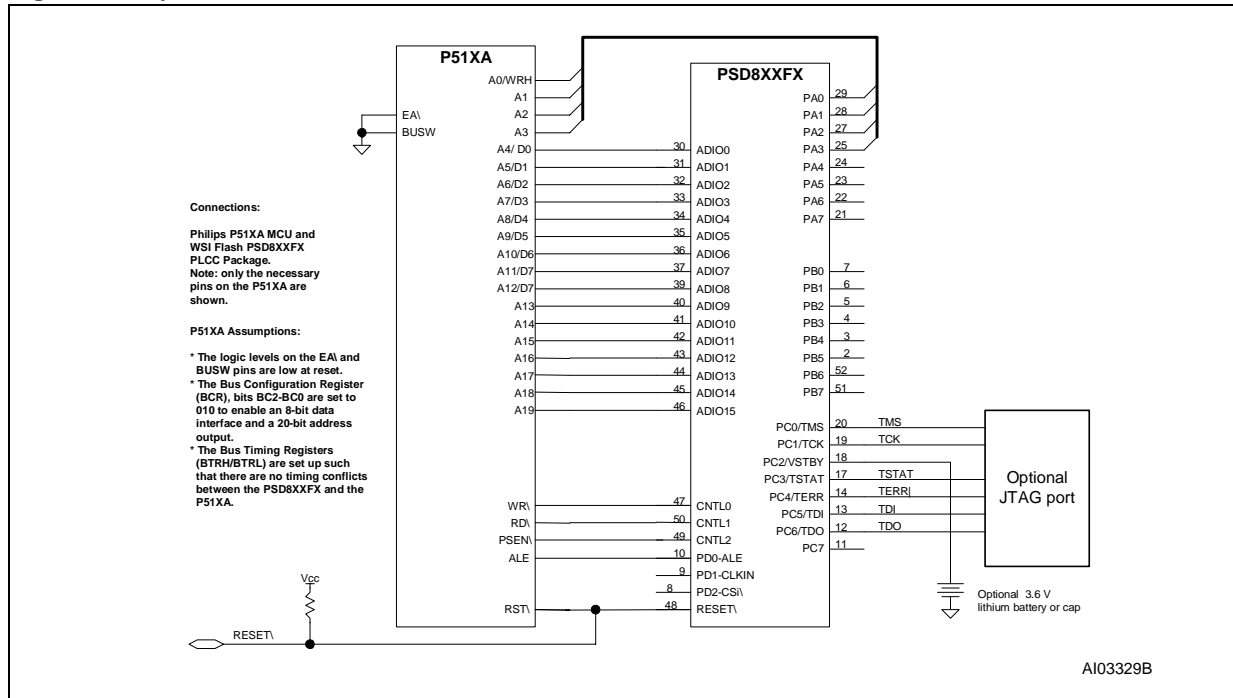
Figure 3. Top Level Block Diagram of PSD8xx



**PHYSICAL CONNECTIONS**

Connect your P51XA to the PSD8XX as shown in Figure 4. The same connections can be used for all of the members of the PSD8XX family except the PSD835G2, which has more I/O. The JTAG programming channel, SRAM with battery backup, LCD module, and MCU I/O connections are all optional.

Figure 4. Physical Connections, P51XA and PSD8XXFX



**SIMPLE DESIGN EXAMPLE**

The first design example outlines the steps required to get a P51XA system up and running quickly. A connection diagram, memory map, and the necessary design file for the PSDsoft Express software development environment are provided. A PSD813F2 was used for this example. However, other members of the Flash PSD family may be used instead, with minor changes to the sample design file. See the selector guide on the website for a comparison of the products.

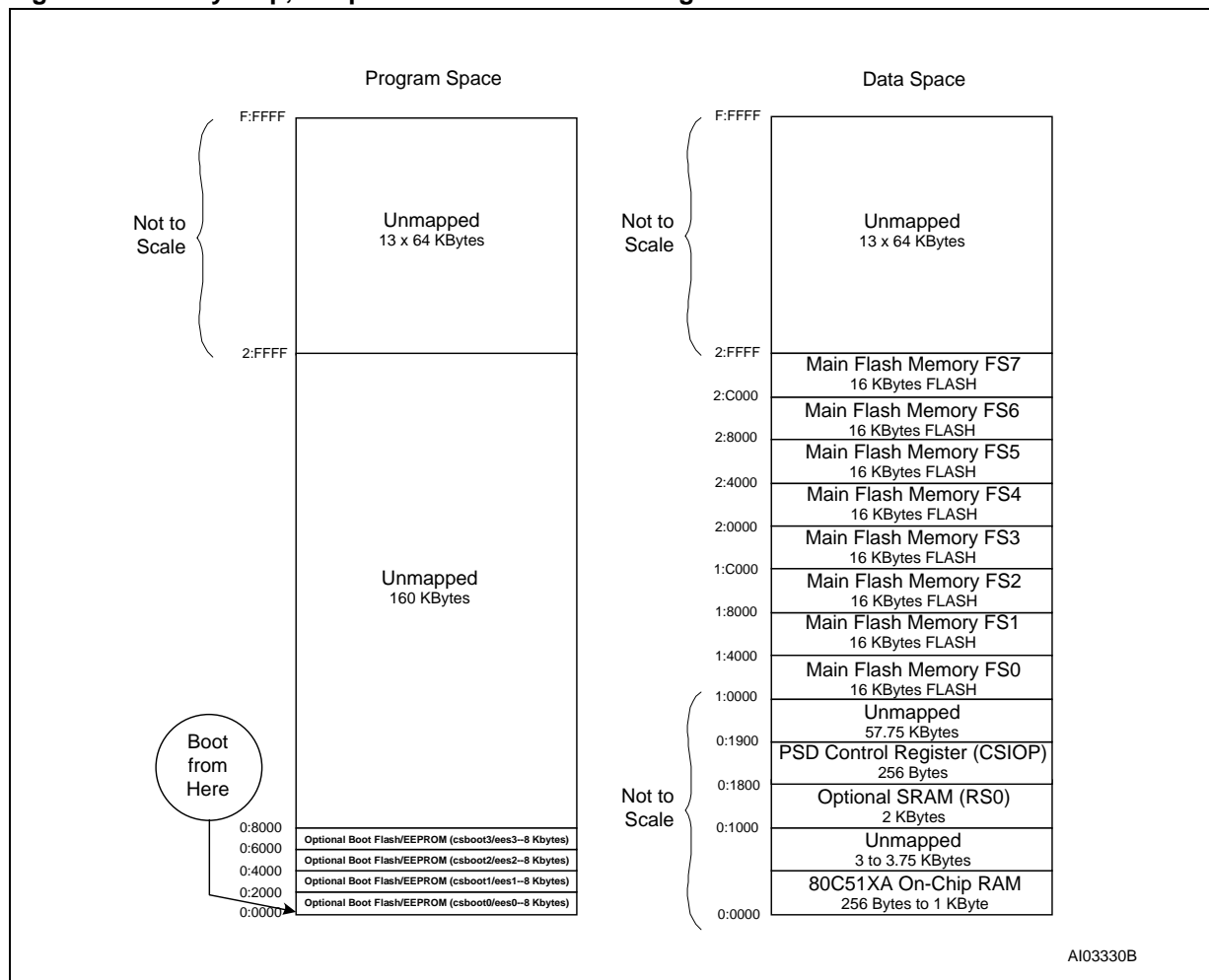
**Memory Map**

For this simple design, we used a PSD813F2 with the following memories:

- 128 KBytes main Flash memory, broken into eight 16 KByte segments denoted  $fs_i$  ( $i = 1-8$ )
- 32 KBytes boot Flash memory, broken into four 8 KByte segments denoted  $csboot_j$  ( $j = 1-4$ ). The PSD813F1 has an EEPROM instead of Flash memory. Therefore,  $ees_j$  ( $j = 1-4$ ) would be used in place of  $csboot_j$ .
- 2 KByte SRAM (rs0)
- 256 byte PSD813F configuration register (csiop).

Note: the PSD memory segments are defined in the “Chip Select Equations” screen in PSDsoft Express. We’ll use the boot memory to hold the ISP boot loader code, P51XA interrupt vectors, and common firm-ware functions. For this example, we’ll execute from secondary Flash memory only and leave the main Flash memory in Data Space. Let’s examine the sample memory map in Figure 5.

Figure 5. Memory Map, Simple P51XA/PSD813FX Design



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

- It is broken up into sixteen 64 KByte segments.
- All areas, except the unmapped regions and the 80C51XA On-Chip RAM region, are resident on the PSD.
- It shows both Program Space and Data Space.
- The 32 KBytes of the PSD813F boot memory is mapped to Program Space. There are several references to “boot” memory in this document, but the “boot” memory is simply a secondary memory that can be used as boot memory or can serve any other purpose.
- 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 memory and secondary memory into Program Space or Data Space is accomplished with the PSD VM Register. PSDsoft Express 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 fusemap that gets programmed into the PSD. At runtime, the VM register can be changed by writing to it with the MCU. This is illustrated in the enhanced design of Section 4.

The boot memory holds the following information:

- P51XA reset vector and initialization routines

- P51XA interrupt vectors and service routines
- I/O management.

Since Figure 5 is a sample memory map, you may wish to change it. To do so, simply change the chip select equations within the Design Assistant in PSDsoft Express. For example, if you have a PSD813F part that doesn't contain the optional secondary memory, you will want to have the main Flash memory located in Program Space. See Appendix A for a sample memory map for parts with no secondary boot memory.

### **PSDsoft Express Design Entry**

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

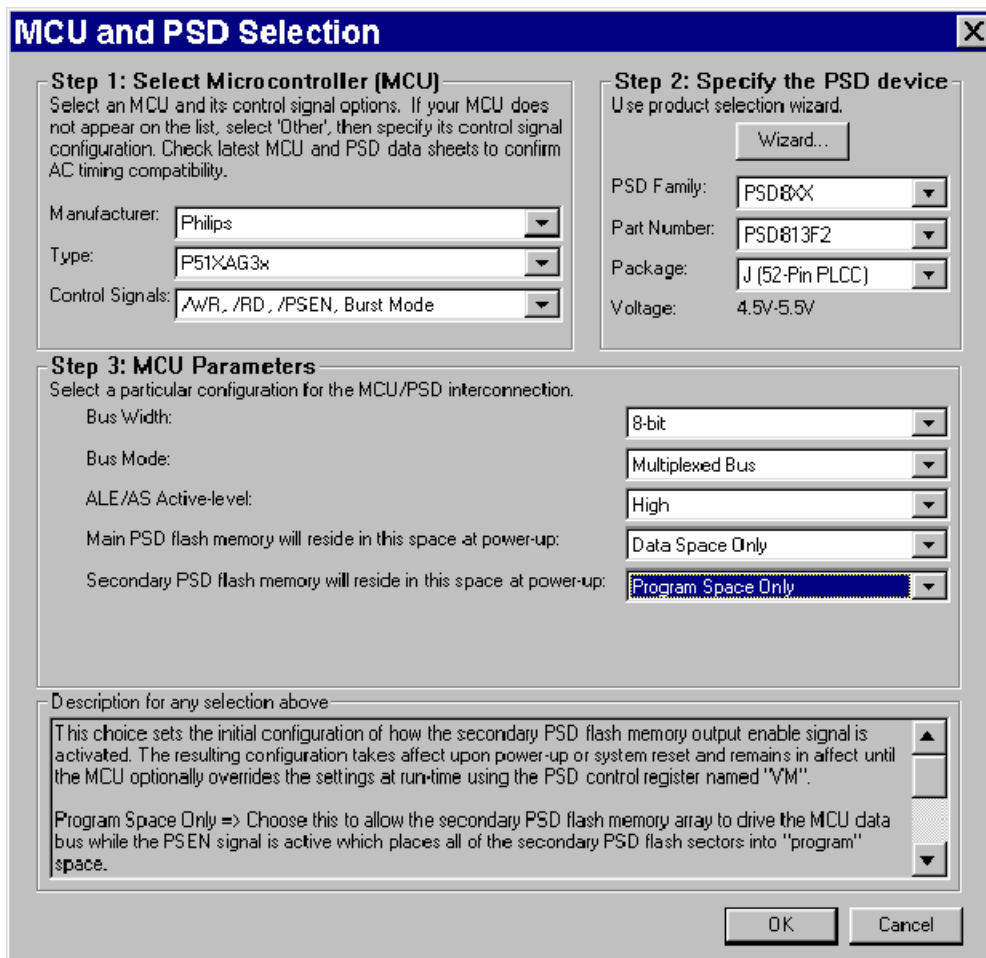
#### **Invoke PSDsoft Express and Open a New Project.**

- Start PSDsoft Express.
- Create a new project.
- Select your project folder and name the project (in this example, name the project "Easy51XA" in the folder PSDsoft\my\_project).

**MCU and PSD Selection.** When you click **OK** in the "New Project" window, the "MCU and PSD Selection" screen appears. When you see this screen, make the following selections:

- Select an MCU manufacturer and part number. In this example, we're using a Philips P51XAG3x.
- For the Control Signals box, select /WR, /RD, /PSEN, Burst Mode
- Select the PSD8XX series for the PSD Family.
- Select a PSD813F2 and use the 52-pin PLCC package (J package).
- Based on the above selections, the "Bus Width", "Bus Mode", and "ALE/AS Active Level" will be set automatically.
- Set the main Flash memory to "Data Space Only" and the secondary Flash memory to "Program Space Only".

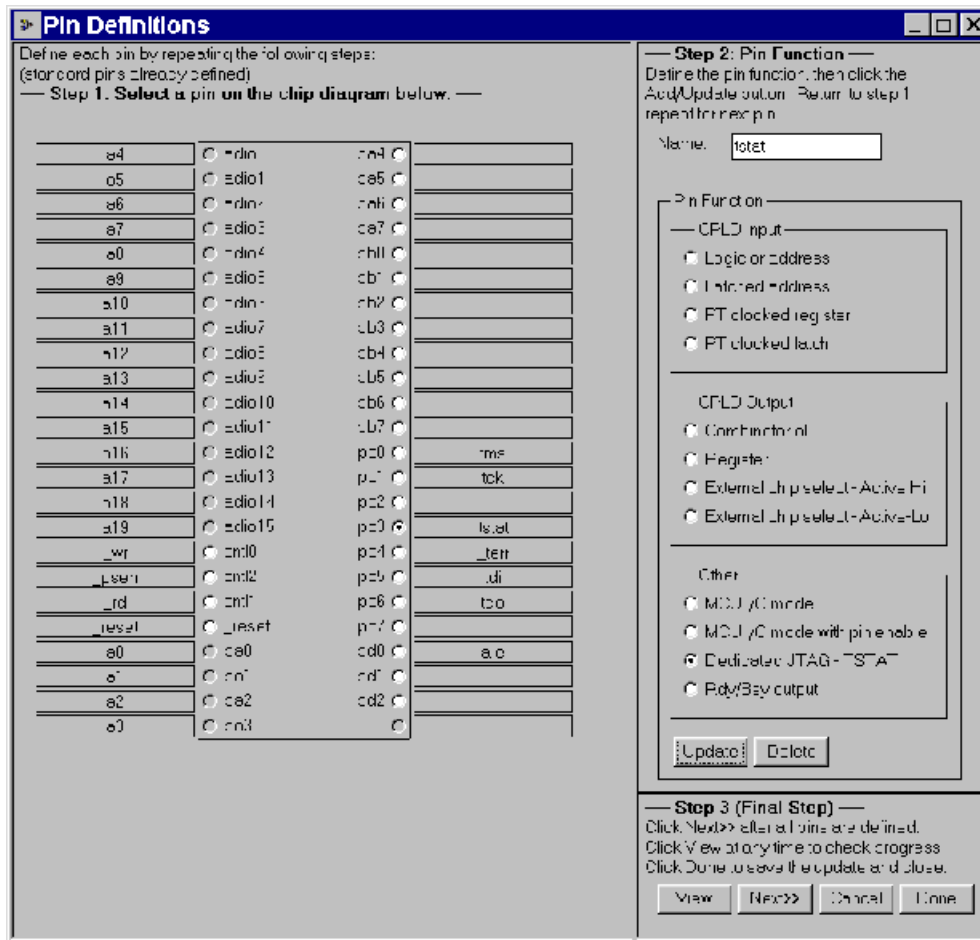
This is what the screen should look like after you've made the selections:



Now you have your project established based on a PSD813F2 and a P51XA. However, there are many other MCU/MPUs you could have chosen in place of the P51XA and still have use of this document. Click **OK** and the “Design Parameters” window will appear.

**Design Parameters.** Choose “Use Design Assistant” and click **OK** to be taken to the Pin Definitions screen. Notice how all the pins functions on the left hand side of the diagram have been assigned for you based on your MCU and PSD selection and configuration. To get an idea of how to add a pin function, click on pc3 in the diagram, then “Dedicated JTAG – TSTAT” in the “Other” box and click the **Add** button. Your screen should match the one below. Continue to add pin functions to match your design. When finished, click **Next >>** to be taken to the “Design Assistant” screen. Note: there are detailed instructions on how to use this screen and other “Design Assistant” screens in the *PSDsoft Express User Manual*.

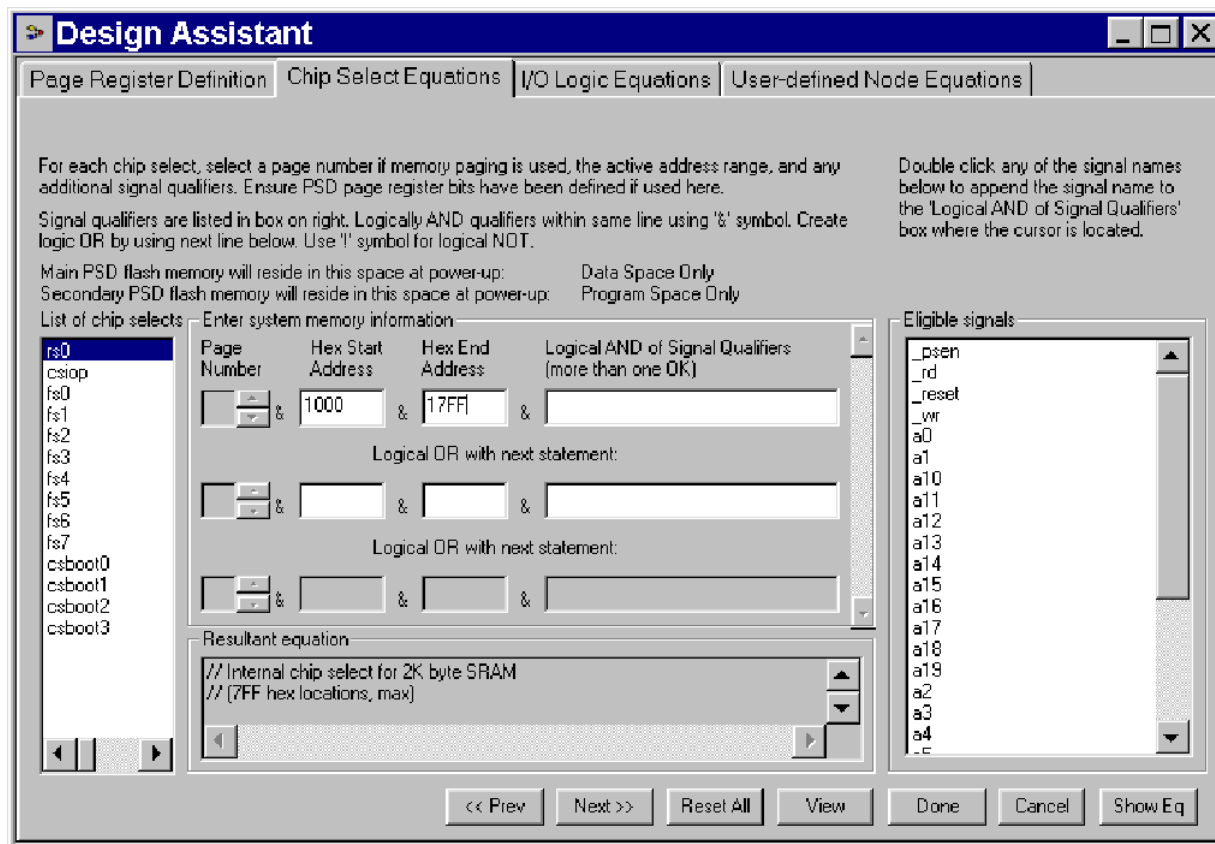




**Page Register Definition.** In this example, the P51XA is assumed to be outputting 20 address bits, providing a one megabyte address space. As such, no page bits are required to extend the address space, so there is nothing to do on this screen at this time except to move on. However, later, you will see how the Page Register can be used for general logic inputs to the PLD. Click **Next >>** when finished.

**Chip Select Equations.** Use this screen to enter chip-select equations to match your memory map. The entry for the PSD SRAM (rs0) is shown below.

# AN1177 - APPLICATION NOTE



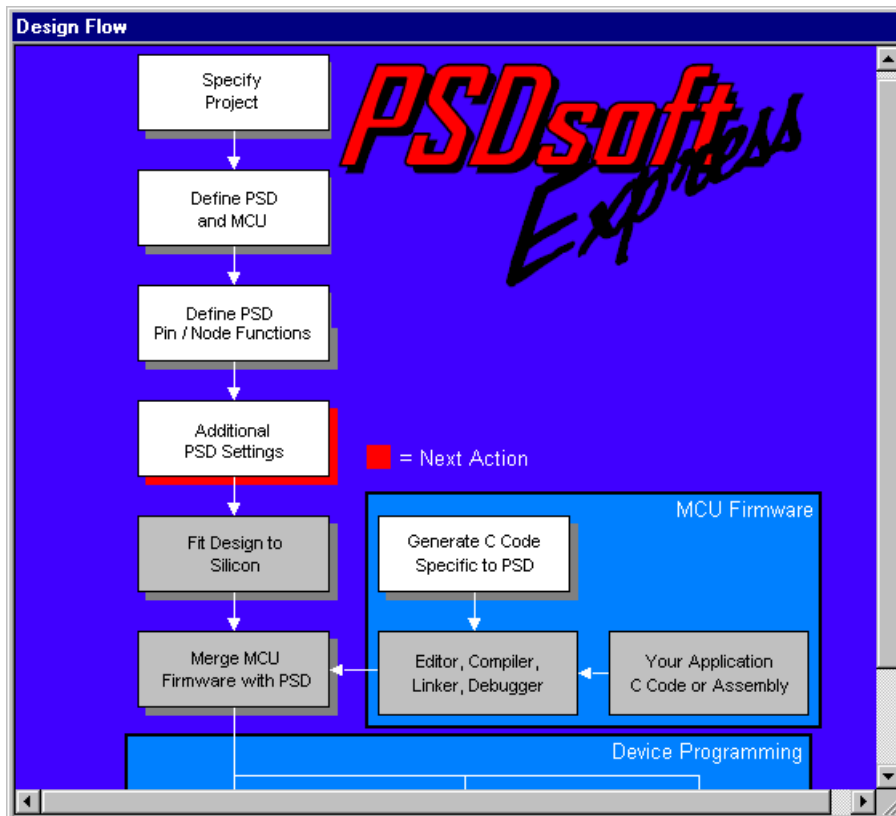
Use the following table to fill in the rest of the Chip Select equations:

**Table 1.**

Chip Select Segment	Hexadecimal Start Address	Hexadecimal End Address
csiop	01800	018FF
fs0	10000	13FFF
fs1	14000	17FFF
fs2	18000	1BFFF
fs3	1C000	1FFFF
fs4	20000	23FFF
fs5	24000	27FFF
fs6	28000	2BFFF
fs7	2C000	2FFFF
csboot0	00000	01FFF
csboot1	02000	03FFF
csboot2	04000	05FFF
csboot3	06000	07FFF

**I/O Logic and User Defined Node Equations.** The “I/O Logic Equations” and “User Defined Node Equations” screens are used to enter equations for the registered logic within the PSD. Since this document focuses on issues related to ISP and IAP, registered logic equations are not covered. However, for more information on entering registered logic equations, refer to the *PSDsoft Express User Manual*. Also, see *Application Note AN1356—Design Guide: PSDsoft Express*, section 5.2 for a tutorial on implementing logic in the CPLD.

Click **Done** and the software will check your design for errors. If you have any errors, go back and fix them. Otherwise, you should now see the “Design Flow” window:



Click on **Additional PSD Settings** in the “Design Flow” window and a dialog box will appear.

**Additional PSD Settings.** There are three functions that can be accomplished in this dialog box:

1. Setting the security bit—blocks all access to the contents of the PSD’s memories by means of JTAG or a conventional programmer. That is, once the security bit is set, no programmer can read or copy the configuration or memory contents of the PSD. The only way to erase the security bit is to completely erase the PSD.
2. Specify the IEEE 1149.1 JTAG user code—allows you to enter a 32-bit code, which can be used for various functions. Click on the “JTAG/ISP” tab for more details
3. Set the internal memories’ sector protections—allows the individual memory sectors within the PSD to be write protected to prevent accidental data loss. The MCU/MPU cannot change these settings at run-time; only a device programmer can alter these settings.

Click **OK** and you will see the Design Flow again. Next, we need to fit the design to silicon.

**Fitting the Design to Silicon.** To fit the design to silicon, click the **Fit Design to Silicon** box in the Design Flow. PSDsoft Express will compile and synthesize the design and create part of the program data

## AN1177 - APPLICATION NOTE

file (.obj) that will later be programmed into the PSD813F2 silicon. When this process is complete, a report will pop up that shows the resulting pin assignments PSD usage. This is the fitter report, which you can use to document your design. Since you created a project from scratch, you might receive a fitter error. If this is the case, you should check the *PSDsoft Express User Manual* for further instructions.

**C Code Generation.** You can take advantage of the provided low-level C code for accessing memory elements within the PSD by clicking on the **Generate C Code Specific to PSD** 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 P51XA C compiler environment in the folder you specify. Simply tailor the code to meet your system needs. See the *PSDsoft Express User Manual* for details on the C code generation feature.

**Merge MCU Firmware with PSD.** Now that the fitting process is complete, PSDsoft Express has created a fuse pattern that reflects the PSD configuration and logic of your design. PSDsoft Express places this fuse information into a file (the .obj file). However this fuse pattern does not yet contain the P51XA firmware. The next step will accomplish this, producing an .obj file that contains the PSD configuration and the P51XA firmware. This final .obj file is what gets programmed into the PSD. The same .obj file is appended with MCU firmware in the next step below.

For this step, “Merge MCU Firmware with PSD”, you will input the firmware file(s) that contain absolute addresses from your P51XA compiler/linker in Intel Hex format. The Merger will map these file(s) into the memory segments of the PSD according to the chip select equations that you entered in the Design Assistant. This mapping process translates the absolute system addresses that P51XA 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 enter the file(s) that were generated from your P51XA linker into the appropriate boxes and PSDsoft Express does the rest.

Go to the design flow window and click the **Merge MCU Firmware with PSD** box and you should see this:

**Merging of MCU Firmware with PSD**

**Step 1: MCU firmware placement**

Specify name of MCU firmware file for each PSD memory segment below. Scroll to see all segments. You may need to edit/add the start and stop addresses if paging or other memory manipulation is used. More Info...

Memory Select Name	Memory Select Equations	File Address Start (hex)	File Address Stop (hex)	File Name
FS0	!pdn & !a19 & !a18 & !a17 & a16 & !a15 & !a14;	10000	13FFF	<input type="text"/> <span>Browse...</span>
FS1	!pdn & !a19 & !a18 & !a17 & a16 & !a15 & a14;	14000	17FFF	<input type="text"/> <span>Browse...</span>
FS2	!pdn & !a19 & !a18 & !a17 & a16 & a15 & !a14;	18000	1BFFF	<input type="text"/> <span>Browse...</span>
FS3	!pdn & !a19 & !a18 & !a17 & a16 & a15 & a14;	1C000	1FFFF	<input type="text"/> <span>Browse...</span>

Record Type:  Intel Hex Record  Motorola S-Record

Mapping Mode:  Direct  Relative

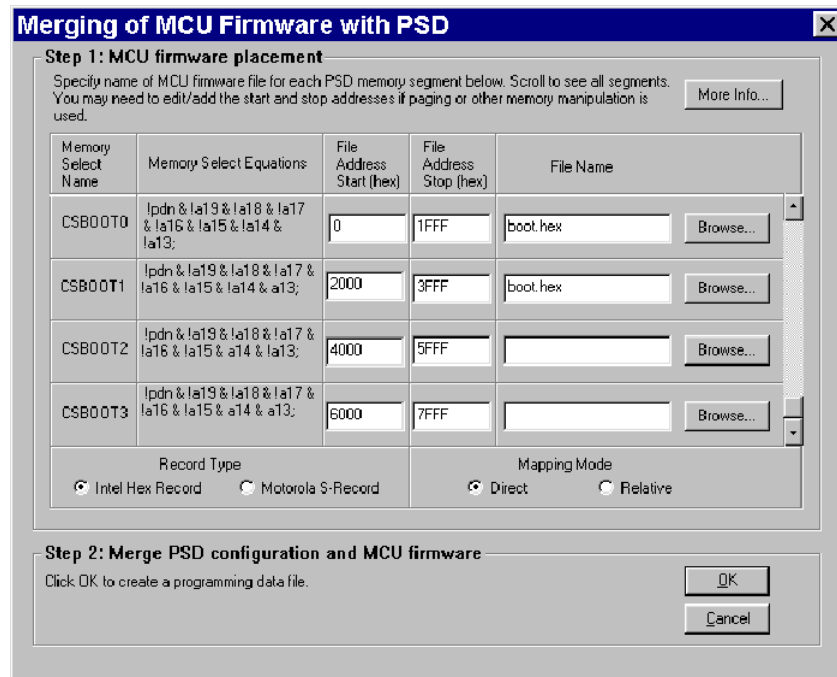
**Step 2: Merge PSD configuration and MCU firmware**

Click OK to create a programming data file. OK Cancel

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 “Chip Select Equations” screen to create the memory map. PSDsoft Express filled in these address fields for you. PSDsoft Express expects to find these absolute MCU addresses within your P51XA linker file(s) when they are imported. On the right are boxes where you can type in (or browse for) the name of the file(s) (including path) that indicates the location of your P51XA linker files. 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.



Type in the name of the file from your P51XA 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 P51XA 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 secondary Flash memory 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 of Section 4.

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

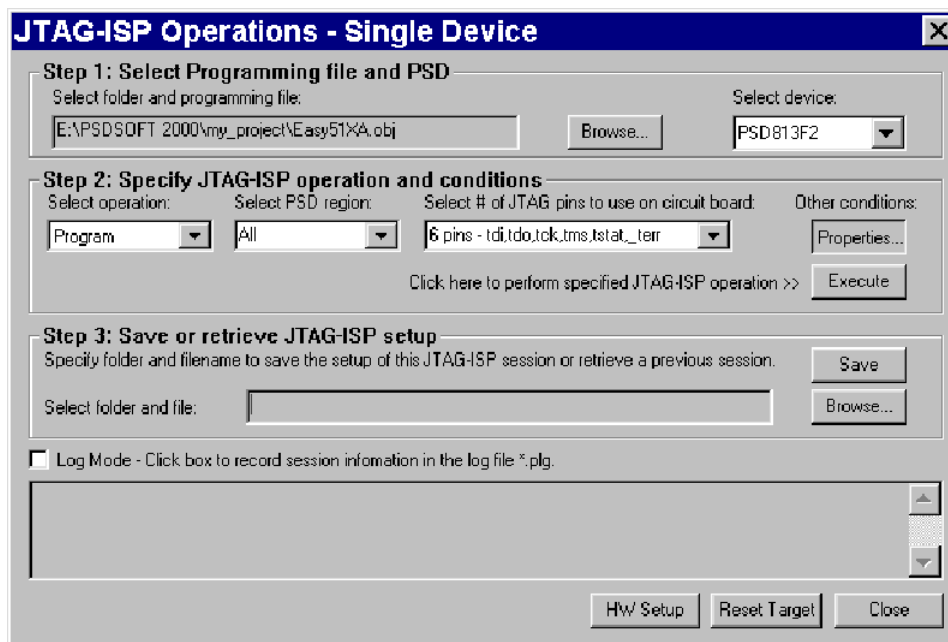
Third-party programmers, such as Stag and Needhams. See the website at [www.st.com/psd](http://www.st.com/psd) for a list of compatible third-party programmers.

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

**Programming with FlashLINK™.** Connect the FlashLINK™ cable to your PC's parallel port. Click the **ST JTAG/ISP** box in the Design Flow window. You will be prompted for the number of devices in the JTAG chain on your circuit board. Make the appropriate selection and click **OK**. This document assumes only

## AN1177 - APPLICATION NOTE

one device is in the JTAG chain. If you have more than one device, refer to the *PSDsoft Express User Manual*. For single device JTAG chains, the window will look similar to the following one:



To use this window, ensure that the correct programming data file and PSD device appear in Step 1. For Step 2, select the desired operation, the regions of the PSD that the operation affects, and the number of JTAG pins (4 or 6) to use on the circuit board.

Before you perform the selected operation, click the **Properties...** button. This dialog box allows you to do the following:

**Set Port Pins:** with this screen, you can set up the PSD's I/O pins during JTAG operations. The default (except for the JTAG pins) is Input, which is usually fine for most pins. (Note that the PSD will not respond to any non-JTAG I/O.) However, sometimes it may be desirable to set a pin or pins to output during JTAG. For example, if you have chip-select signal being generated from the PSD that selects a device that potentially could drive signals on the JTAG lines (if you are multiplexing the pins), you would want that chip-select to be inactive during the JTAG operation.

**JTAG-ISP Attributes:** this screen allows you to view the device name and Instruction Register length. This information may be useful to other design programs.

**User Code:** basically, by clicking on the "User Code" tab, you are provided with a space to enter an IEEE 1149.1 User Code that will be compared to the value previously entered in the "Additional PSD Settings" screen.

Once you are satisfied with your property settings, click **OK** to return to the "JTAG-ISP Operations" window. You can now perform the selected operation by clicking **Execute**.

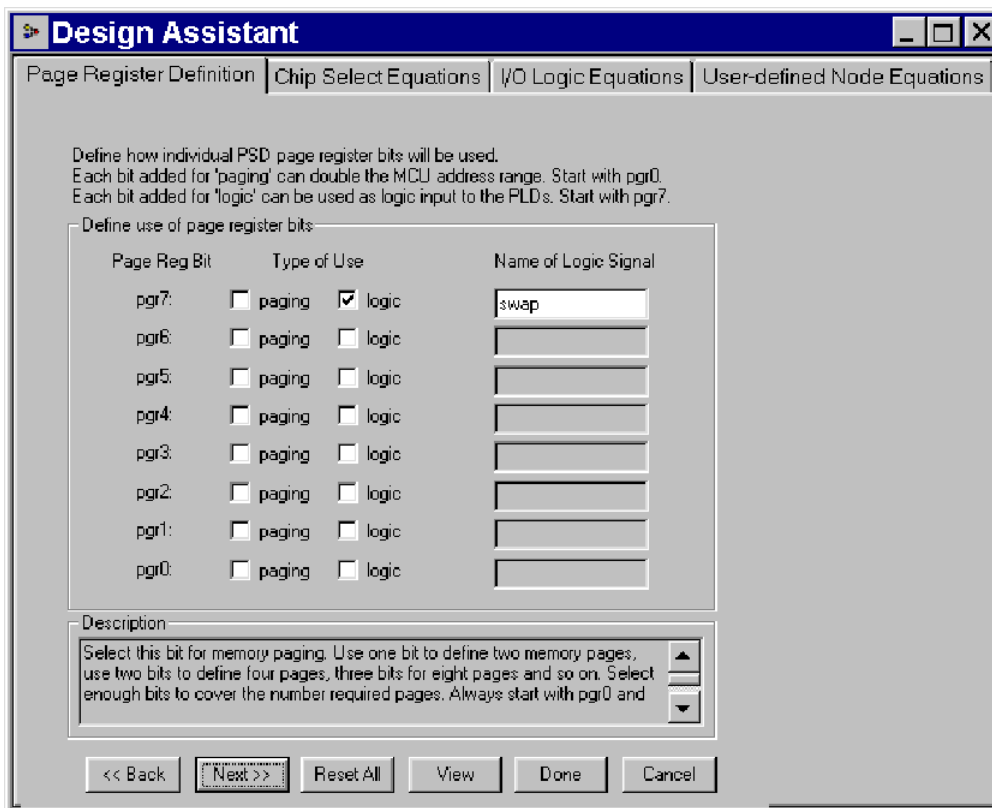
Before you leave this screen, you may wish to save your JTAG configuration. This can be done in Step 3 by clicking on the **Save** button and specifying a file name. This file can be used next time by clicking the **Retrieve** button.

**Programming with PSDpro.** Ensure that the PSDpro device programmer is connected to your PC's parallel port. Click on the **ST Conventional Programmers** box in the Design Flow window. You will see this:



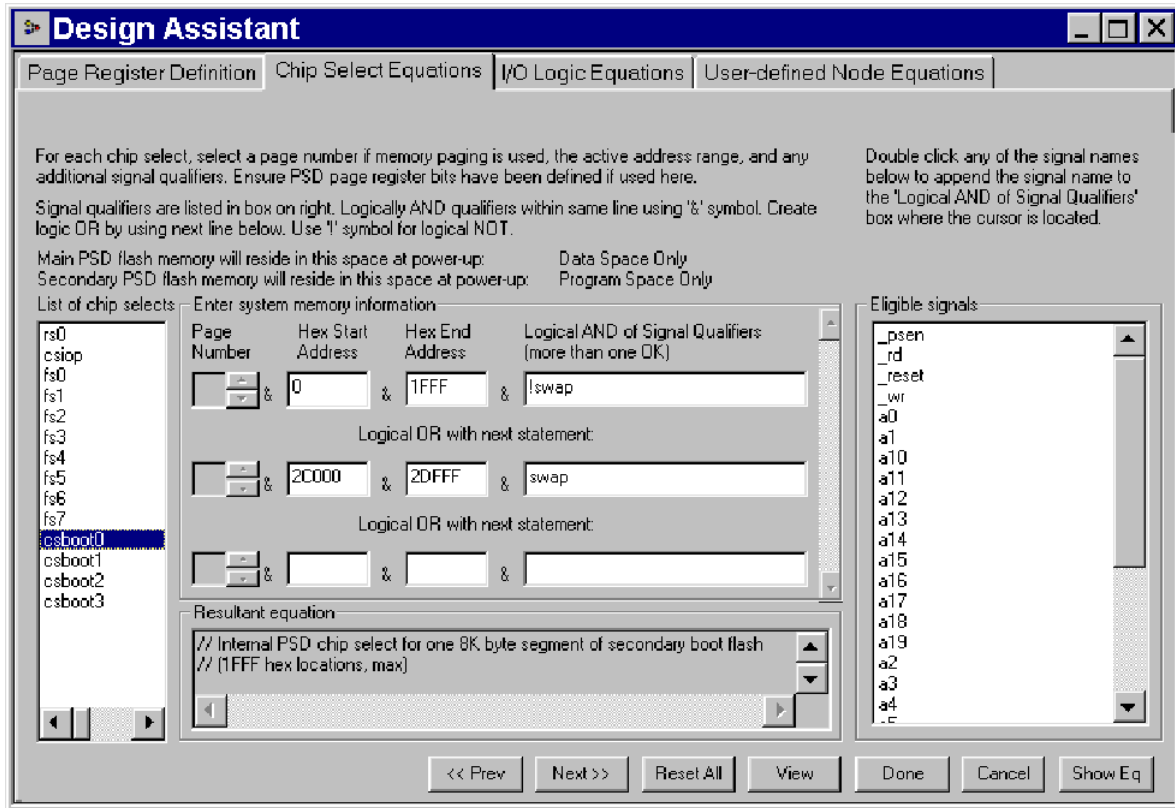
## AN1177 - APPLICATION NOTE

equations to implement memory swapping (as shown in the next subsection). This bit can be modified at runtime by writing to its location in the Page Register within the CSIOP address space. See the *PSD8XX Family Data Sheet* for details. When you have made the addition, your screen should look like this:

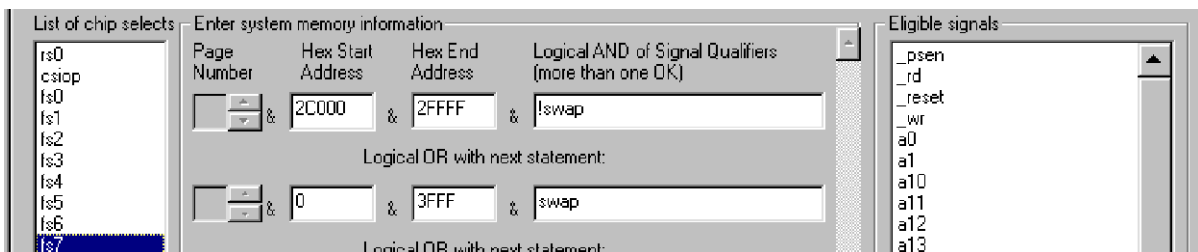
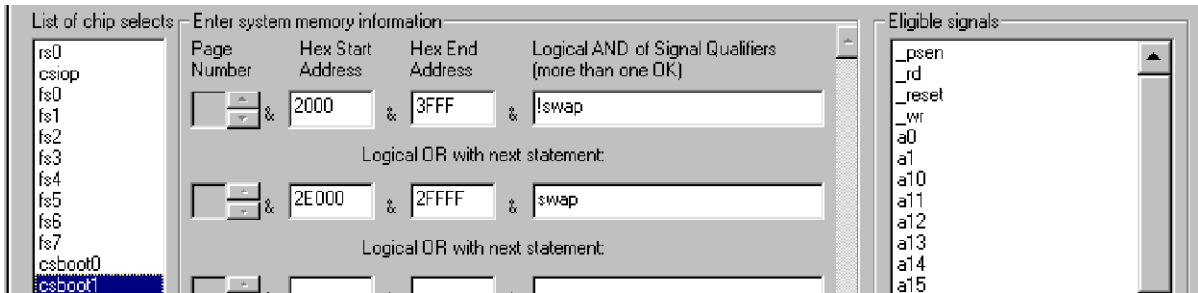


**Modify the Chip Select Equations.** The chip selects equations need to be modified (from what they were in Section 3.2.5) to match the situational memory maps outlined in Figure 6 to Figure 9 in Section 4.2. That is, the memory map that is presented to the MCU will vary dynamically based on the settings of the VM Register and pgr7 (“swap”) of the Page Register. In order to make the memory maps of Section 4.2 work, csboot0, csboot1, and fs7 need to be modified. Below, the modified csboot0 is shown as an example.





Continue to modify csboot1 and fs7 according to the following screen captures:



The steps outlined in Sections 3.2.6 to 3.2.9 can be repeated for the Enhanced Design Example at this time. For Section 3.2.10, when mapping the P51XA firmware in the Address Translate utility of PSDsoft

Express for this second design example, you still do not need to specify any Hex file for the PSD main Flash memory area. You only need to specify the P51XA linker file(s) for the secondary Flash memory area (as in the first simple design) because the P51XA will execute code from secondary Flash memory and download to main Flash memory. See the next subsection for more details.

### Memory Map

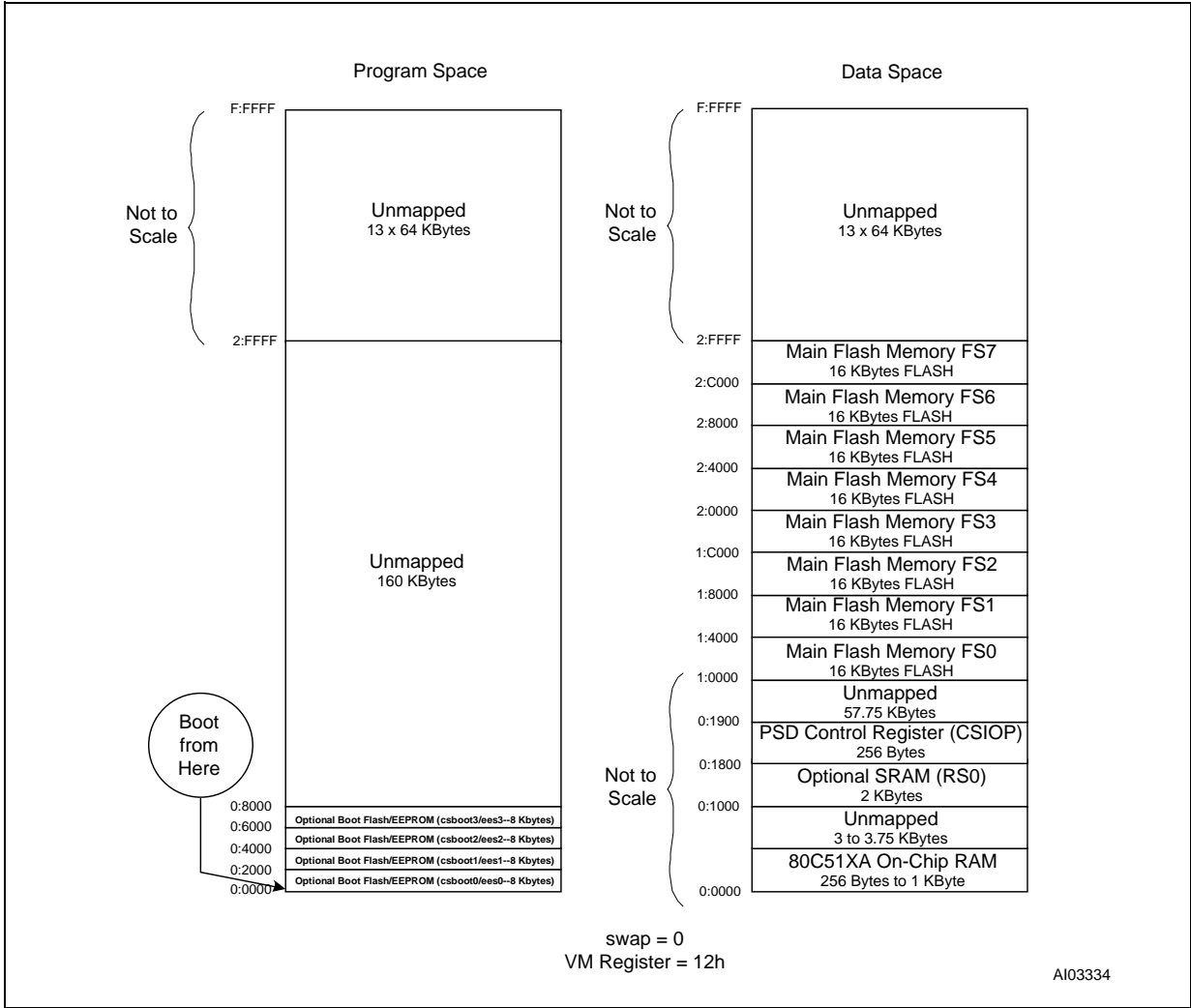
The boot sequence and memory swap is a four-step process, as shown in Figure 6 to Figure 9.

For more information on both the Page Register and VM Register, see the data sheets and the *PSDsoft Express User Manual*.

**Memory Map Configuration at Boot-Up.** Figure 6 (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 runtime using PSDsoft Express Configuration. The VM register resides in the PSD and can be accessed at any time by the P51XA. (See the PSD8XX data sheets.) Here’s what the P51XA does upon power-up or reset:

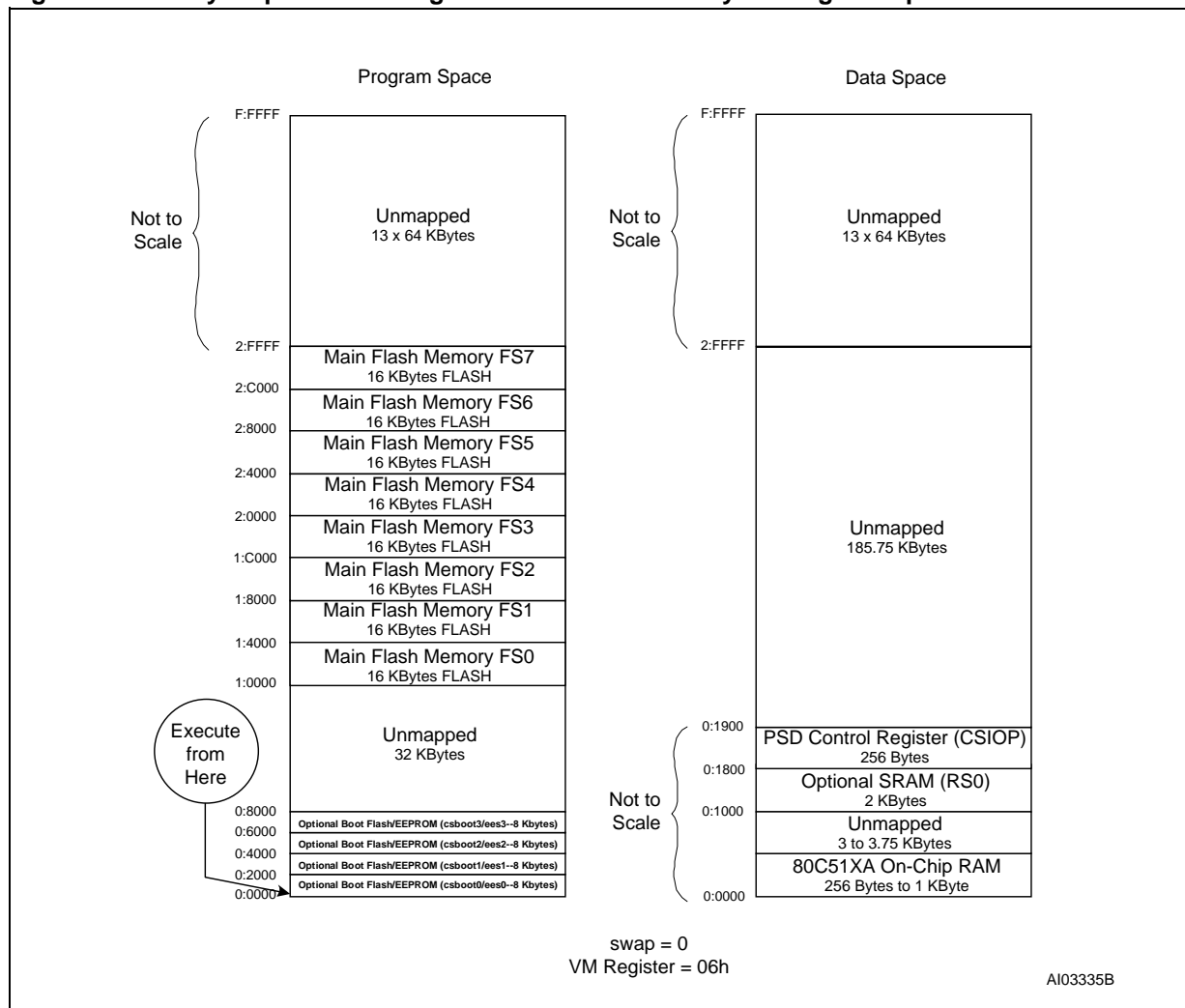
- Boot from Flash memory boot csboot0 at address 0h
- Perform a checksum of main Flash memory
- Download main Flash memory from a host computer if needed and validate contents.

Figure 6. Memory Map, Enhanced Design at Boot-Up/ISP



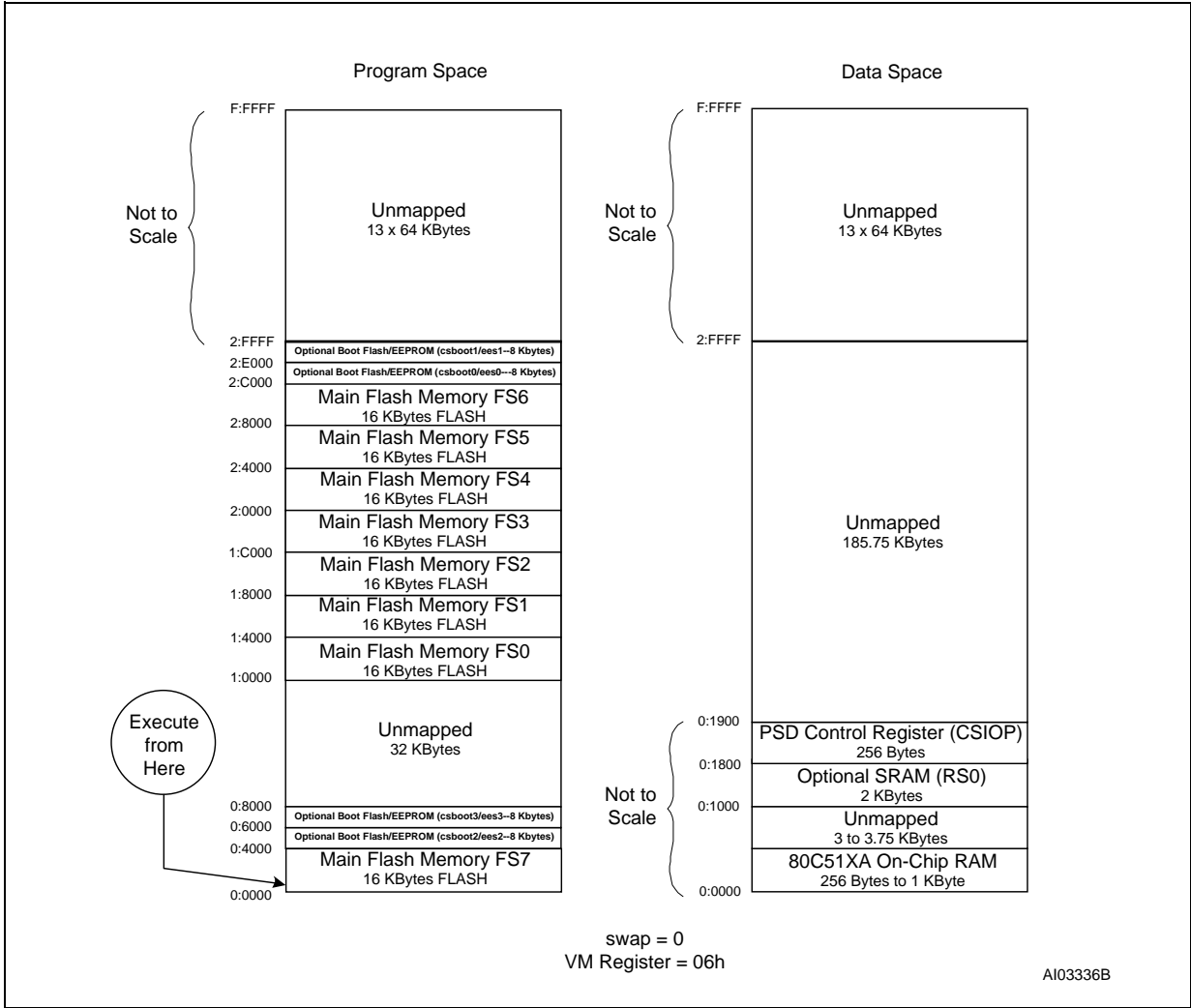
**Memory Map Configuration After Moving the Main Flash.** The next step is to move the main Flash memory from Data Space to Program Space. To do so, while executing out of the boot Flash memory, write 06h to the VM register. You will now have the memory map shown in Figure 7.

Figure 7. Memory Map After Moving the Main Flash Memory to Program Space



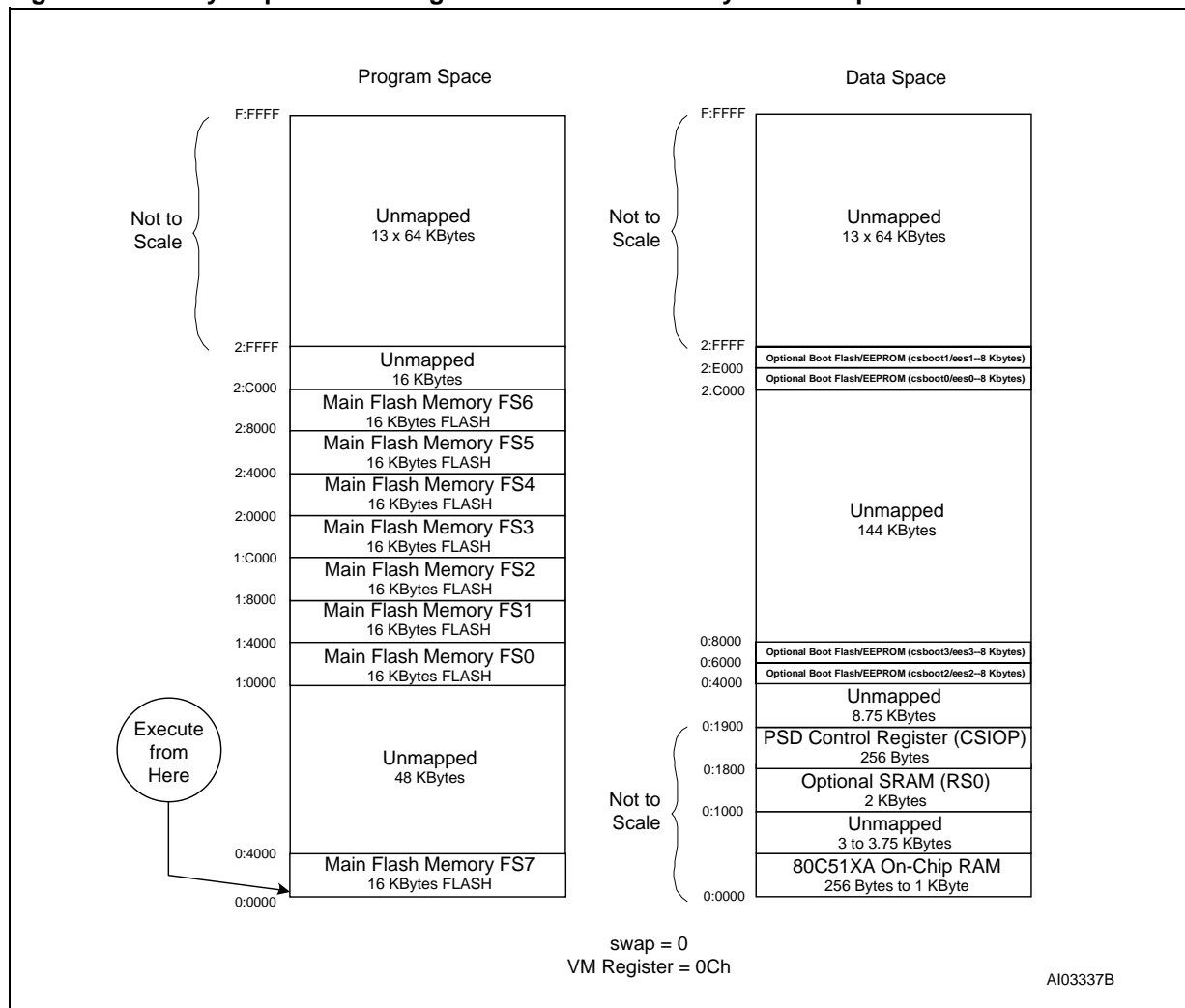
**Memory Map Configuration After Setting the swap bit.** Next, we want to swap main and secondary Flash memory and transfer execution to main Flash memory segment fs7. To do so, the “swap” bit must be set to HI to re-map the Flash memory boot segments csboot0/csboot1 out of the MCU boot area and replace it with main Flash memory segment fs7, as shown in Figure 8. So that no program continuity is lost, the instruction that sets the “swap” bit is executed from csboot0 and the next contiguous instruction must be in fs7. For example, if the instruction that executes the swap is at location 1000h in csboot0, then fs7 must contain the next instruction to be executed at location 1002h.

Figure 8. Memory Map After Setting the SWAP bit



**Memory Map Configuration After Moving the Boot Flash memory to Data Space.** The final step is to move the secondary Flash memory to Data Space so that it can be updated if desired. To move the secondary Flash memory to Data Space, write 0Ch to the VM register. Once the VM register has been written, you can program either half of the secondary Flash memory, depending on how the “unlock” bit is set. Figure 9 shows the final state of the memory map.

Figure 9. Memory Map After Moving the Boot Flash Memory to Data Space



In this final configuration, the P51XA has available:

- 16 KBytes main Flash memory in the boot area (00000h-03FFFh)
- 112 KBytes main Flash memory in Program Space (10000h-2BFFFh)
- 2 KBytes of SRAM in addition to the SRAM that resides on the P51XA
- 16 KBytes of boot Flash memory for general data storage (04000h-07FFFh)
- 16 KBytes of boot Flash memory for boot and IAP loader code (2C000h-2FFFFh).

Each time this P51XA system gets reset or goes through a power-on cycle, the PSD presents the memory map of Figure 6 to the MCU, and the boot sequence is repeated.

**CONCLUSION**

These examples are just two of an endless number of ways to configure the 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 crosschecked under the PSDsoft Express development environment to minimize your effort to design a Flash memory P51XA system capable of IAP.

### REFERENCES

*PSD8XX Family Data Sheets* for detailed PSD8XX information

*PSDsoft Express User Manual* for details on how to use the design software

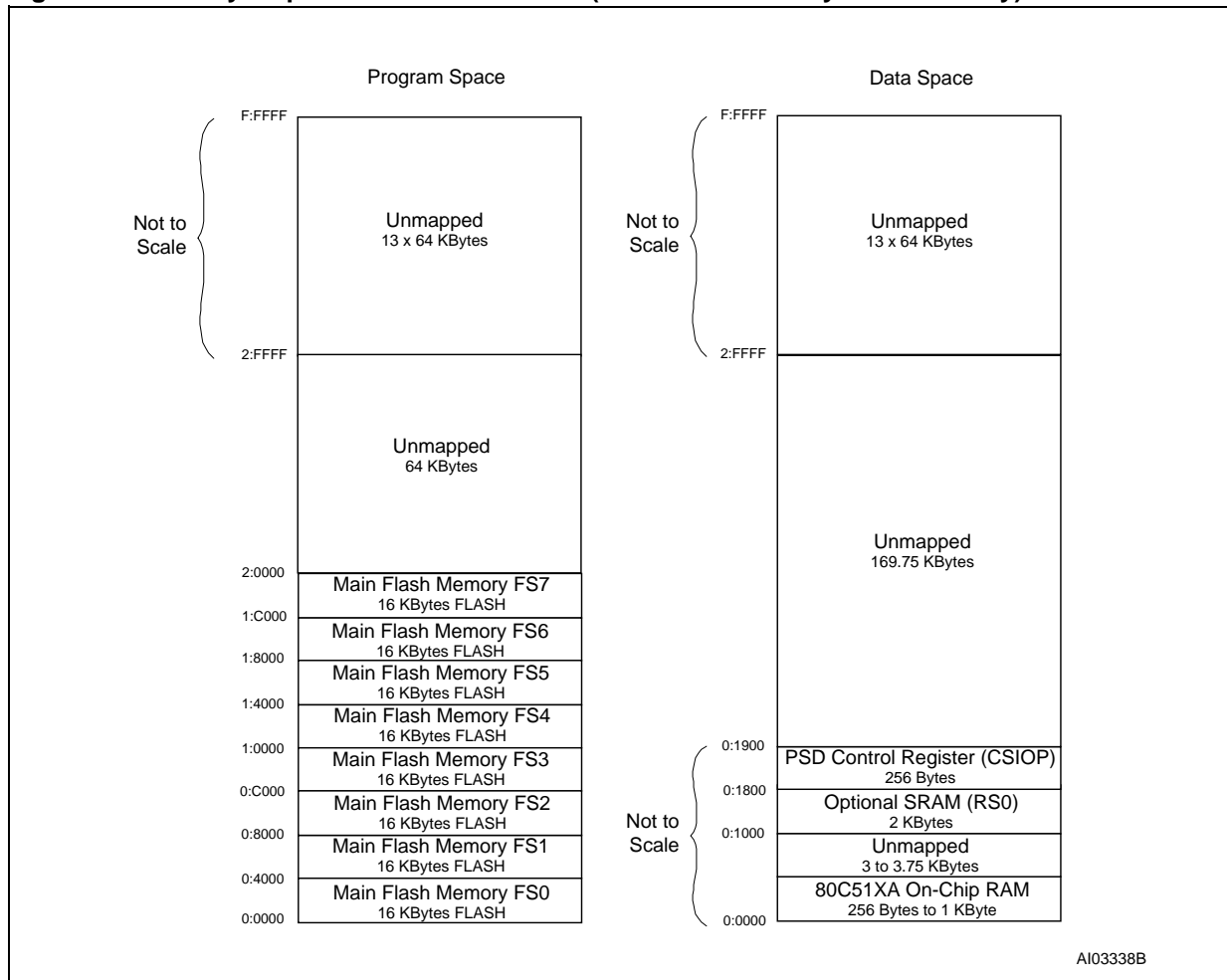
*Application Note AN1153: JTAG ISP Information: Flash PSD* for detailed use of the JTAG port

*Application Note AN1171: Flash PSD CPLD Primer*

### APPENDIX A: CONNECTING TO A PSD813F WITH NO SECONDARY MEMORY

The following is a sample memory map for connecting to a PSD813F with no secondary memory (such as the PSD813F3 or PSD813F5). This memory map assumes you have downloaded the main Flash memory with the FlashLINK cable or you have booted from a separate PROM and have downloaded the Flash memory using the MCU. In either case, you must change your design to account for the different segment locations.

Figure 10. Memory Map for a PSD813F Device (with No Secondary Boot Memory)





**Table 2. Document Revision History**

<b>Date</b>	<b>Rev.</b>	<b>Description of Revision</b>
Nov-2000	2.0	Document written in the WSI format
30-Oct-2001	3.0	Document converted to the ST format

## AN1177 - APPLICATION NOTE

---

For current information on PSD products, please consult our pages on the world wide web:

*[www.st.com/psd](http://www.st.com/psd)*

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

*[apps.psd@st.com](mailto:apps.psd@st.com)* (for application support)  
*[ask.memory@st.com](mailto:ask.memory@st.com)* (for general enquiries)

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

Information furnished is believed to be accurate and reliable. However, STMicroelectronics assumes no responsibility for the consequences of use of such information nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of STMicroelectronics. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. STMicroelectronics products are not authorized for use as critical components in life support devices or systems without express written approval of STMicroelectronics.

The ST logo is registered trademark of STMicroelectronics  
All other names are the property of their respective owners

© 2001 STMicroelectronics - All Rights Reserved

STMicroelectronics group of companies Australia - Brazil - Canada - China - Finland - France - Germany - Hong Kong -  
India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States.  
[www.st.com](http://www.st.com)