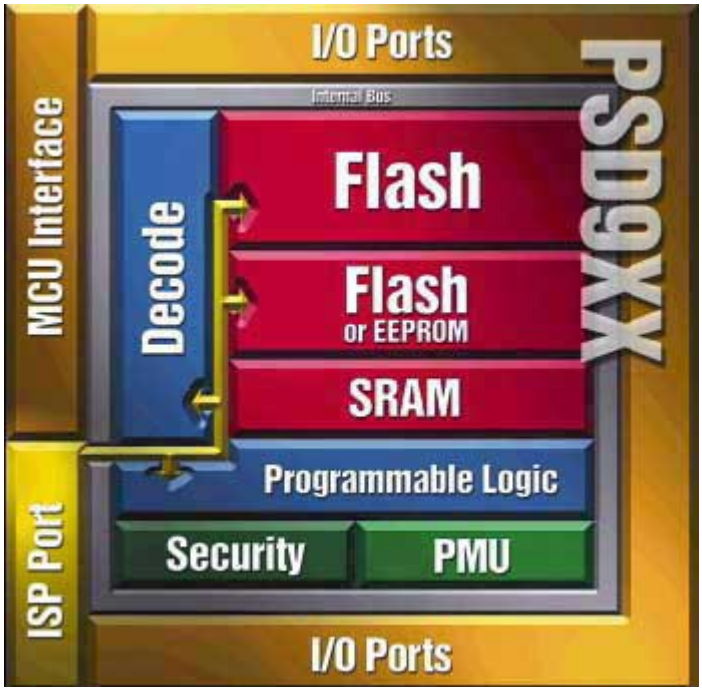
PSD913F2 / 80C32 Design Guide
Application Note 067

By Mark Rootz and Tim Wilkerson

April, 2000
Rev 1.2



Waferscale Integration Inc.
47280 Kato Road, Fremont, CA 94538
Telephone: (510)-656-5400

Contents

1 Introduction

*Easy*FLASH™ PSD9XXF devices are members of a 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, ROM-less 80C32 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 Express development software that make ISP easy for you, regardless of how much experience you have in embedded flash design.

This document offers three flash 80C32 designs using a PSD913F2 device. The first is a simple system to get up and running quickly for basic applications, or to check out your prototype 80C32 hardware. The second design illustrates the use memory paging. The third design covers enhanced features of PSD In-System-Programming, including memory paging and segment swapping (same design as DK-900 kit available at www.waferscale.com for $99 USD). You can start with the first design, and migrate to the second and third as your functional requirements grow. There are other members of the PSD9XXF family, including the PSD913F1 and the PSD934F2. The PSD913F1 contains some EEPROM, and the PSD934F2 has a larger flash memory and a larger SRAM than the PSD913F2. See the PSD9XXF data sheet for details. This application note is applicable to these other PSD9XXF family members, with only slight variations.

In-System Programming and In-Application re-Programming
Our industry uses the term In-System Programming, or 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 re-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. IAP of memory means that the MCU participates in programming 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 updates. PSD9XXF devices provide both ISP and IAP for your system. Keep in mind that IAP can only program the memory sections of the PSD, not the configuration and programmable logic portions of the PSD. ISP can program all areas of the PSD.

Problems with IAP
Typically, a host computer downloads firmware into an embedded flash system through a communication channel that is serviced by the MCU. This channel is usually a UART, but any communication channel that the 80C32 supports will do (modem, SPI, CAN, J1850, etc.). The 80C32 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 programming algorithms (IAP loader code) will be unavailable to the 80C32. 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 reside on-board the MCU. A top-level view of an embedded ISP/IAP flash system with external memory is shown in Figure 1.

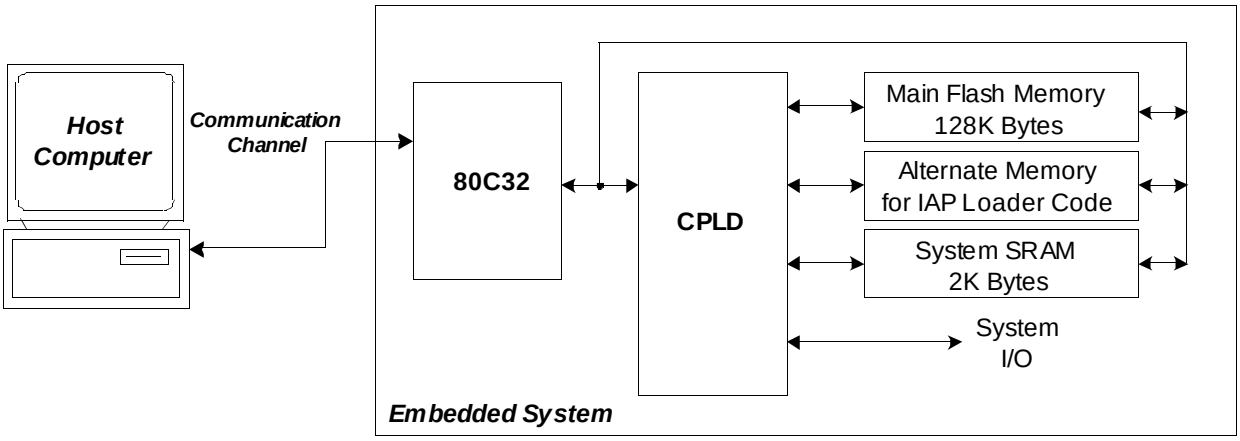


Figure 1 – Embedded flash system capable of IAP (5 devices)

Another problem, which is specific to the 8051 architecture, is related to the separate "Program" and "Data" address spaces. The 80C32 cannot write to Program space, but that is where the flash memory resides that holds 80C32 firmware. How can one program flash memory in-system if the 80C32 cannot write to program space?

A Common Solution

Without a PSD device, implementing IAP with the 80C32 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 method can be a cumbersome and error-prone exercise, which is difficult to debug and vulnerable to power outages.

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 80C32 by 50%.

A Better, Integrated Solution

Figure 2 shows a two-chip solution using an *Easy*FLASH PSD913F2. This system has ample main flash memory, a second smaller 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. The PSD takes care of reclassifying memory as "Program" or "Data" space dynamically with a built-in register that the 80C32 can access at runtime. This easy method allows IAP without having to combine Program and Data spaces, as is commonly done. 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.

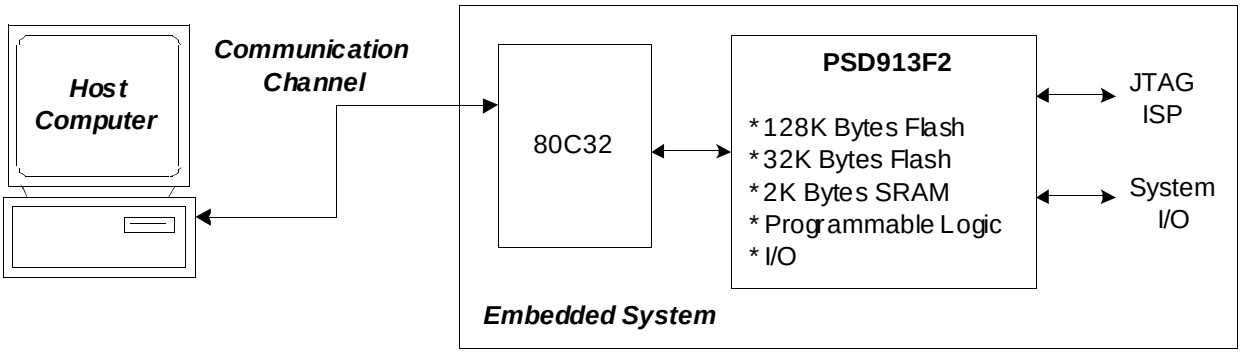


Figure 2 – Embedded flash system capable of IAP and ISP (2 devices)

By design, the IAP method just described requires MCU participation to exercise a communication channel to implement a download to the main flash memory. The PSD9XXF also offers an alternative method (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 using Waferscale's FlashLINK™ JTAG cable and PSDsoft Express development software. No 80C32 firmware needs to be written, just plug in the FlashLINK™ cable to your PC parallel port and begin programming memory, logic, and configuration. This is a powerful feature of the PSD9XXF that allows immediate development of application code in your lab, smart manufacturing techniques, and easy field updates.

> The FlashLINK™ cable and PSDsoft Express are available from our website, www.waferscale.com. The FlashLINK™ is $59 USD (credit cards are accepted) and PSDsoft Express is free.

Let's take a quick look inside the EasyFLASH™ PSD913F2, as shown in Figure 3. There are three independent memory arrays that 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 memory paging. The MCU address, data, and control signals have access to most areas of the chip. The GPLD has 19 combinatorial logic outputs for external device chip-selects or general logic. There are 27 I/O pins. 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. Finally, the self-contained JTAG-ISP controller allows programming of all areas of the chip.

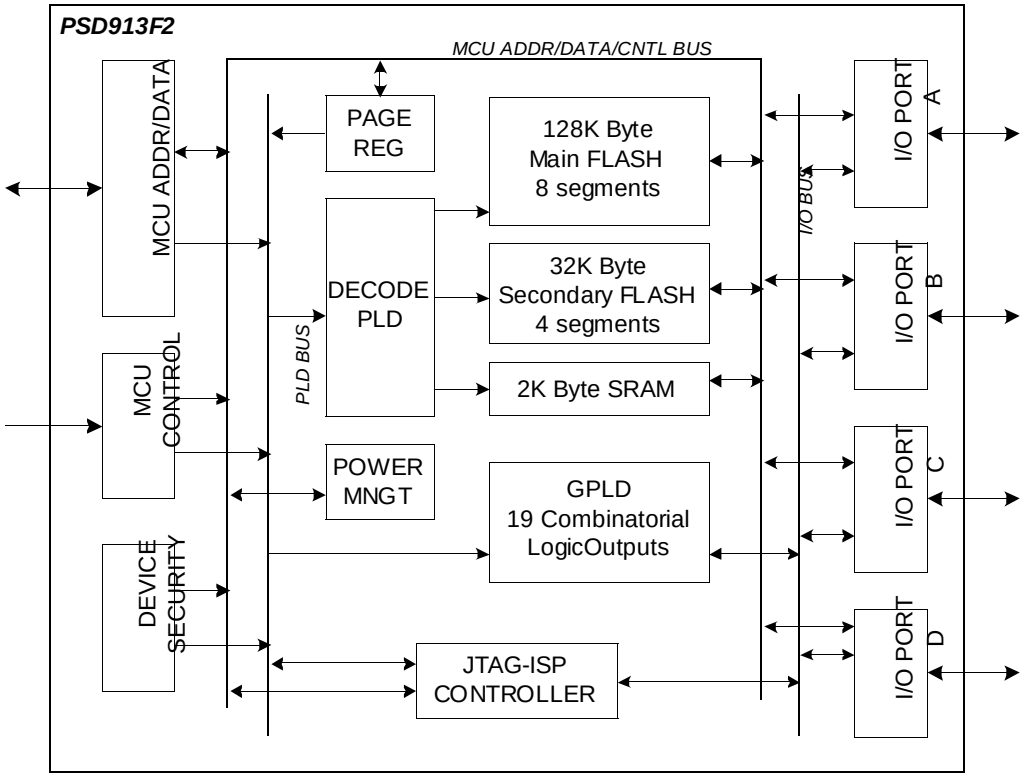


Figure 3 – Top Level Block Diagram of PSD913F2

2 Physical Connections

Connect your 80C32 to the PSD, as shown in Figure 4. A 52-pin PLCC package is used in this example. These same connections can be used for all three design examples in this document. All members of the PSD9XX family share the same pinout.

This example design uses an LCD module, an external LCD chip-select, six MCU controlled I/O signals (two of which control the LCD), and a six-pin JTAG-ISP interface. There are 13 unused PSD I/O pins shown (should use pullups to Vcc with 100K resistor or tie to GND if not used in your design).

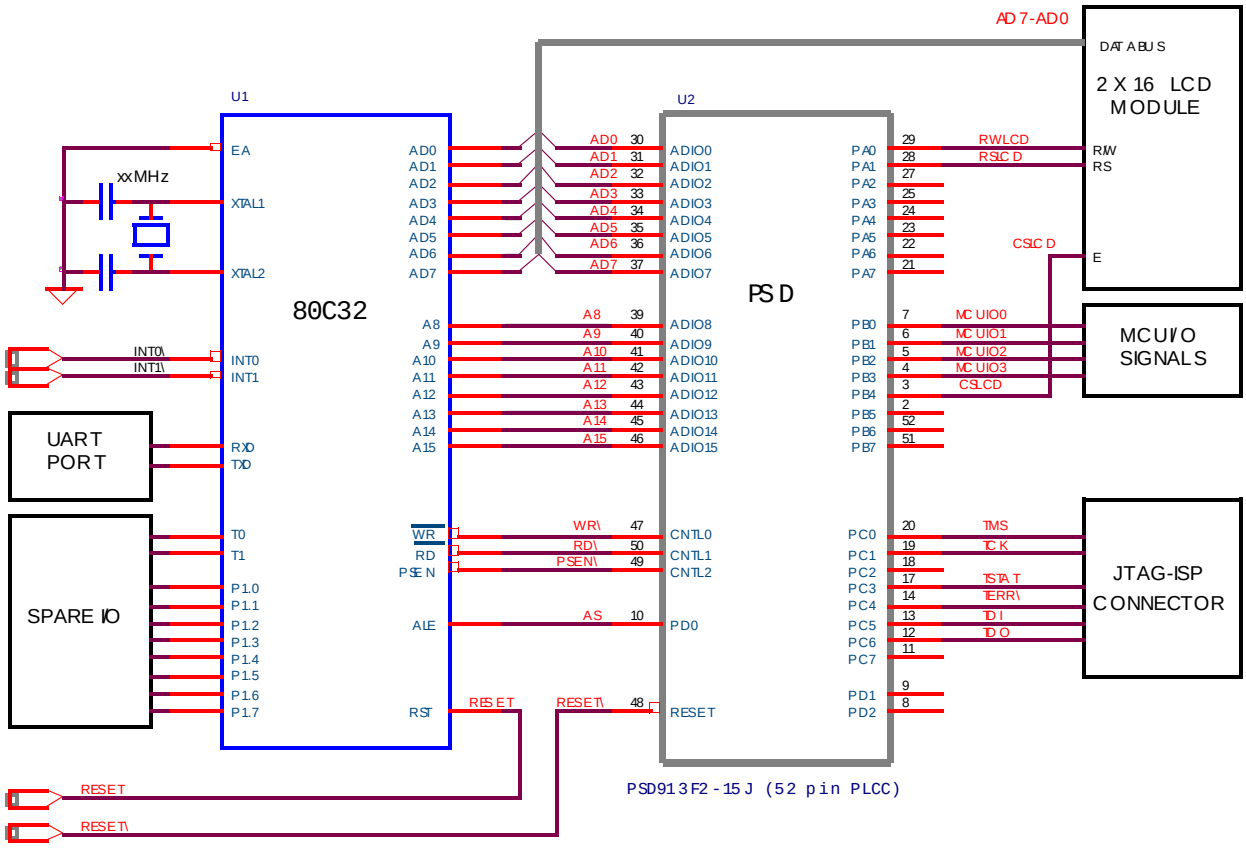


Figure 4 – Physical Connections, 80C32 and PSD913F2

3 First Design Example – IAP with no Memory Paging

The first design example will outline the steps to get a flash 80C32 system up and running quickly. No memory paging is used. Program space and data space for main flash memory is combined because of the small amount of memory used in this simple design. You will see a memory map and the necessary design entry in the PSDsoft Express software development environment. A PSD913F2 is used in this example, but the other members of the *Easy*FLASH™ family may be used instead, with minor changes. See the PSD9XXF data sheet for a comparison of family members.

3.1 Memory Map

We are using a PSD913F2, which provides 128 Kbytes of main flash memory, 32 Kbytes of secondary flash memory, and 2 Kbytes SRAM. However, for this first simple example we will only use 32 Kbytes of main flash memory, 32 Kbytes of secondary flash memory, and 2 Kbytes of SRAM. See the 80C32 system memory map in Figure 5.

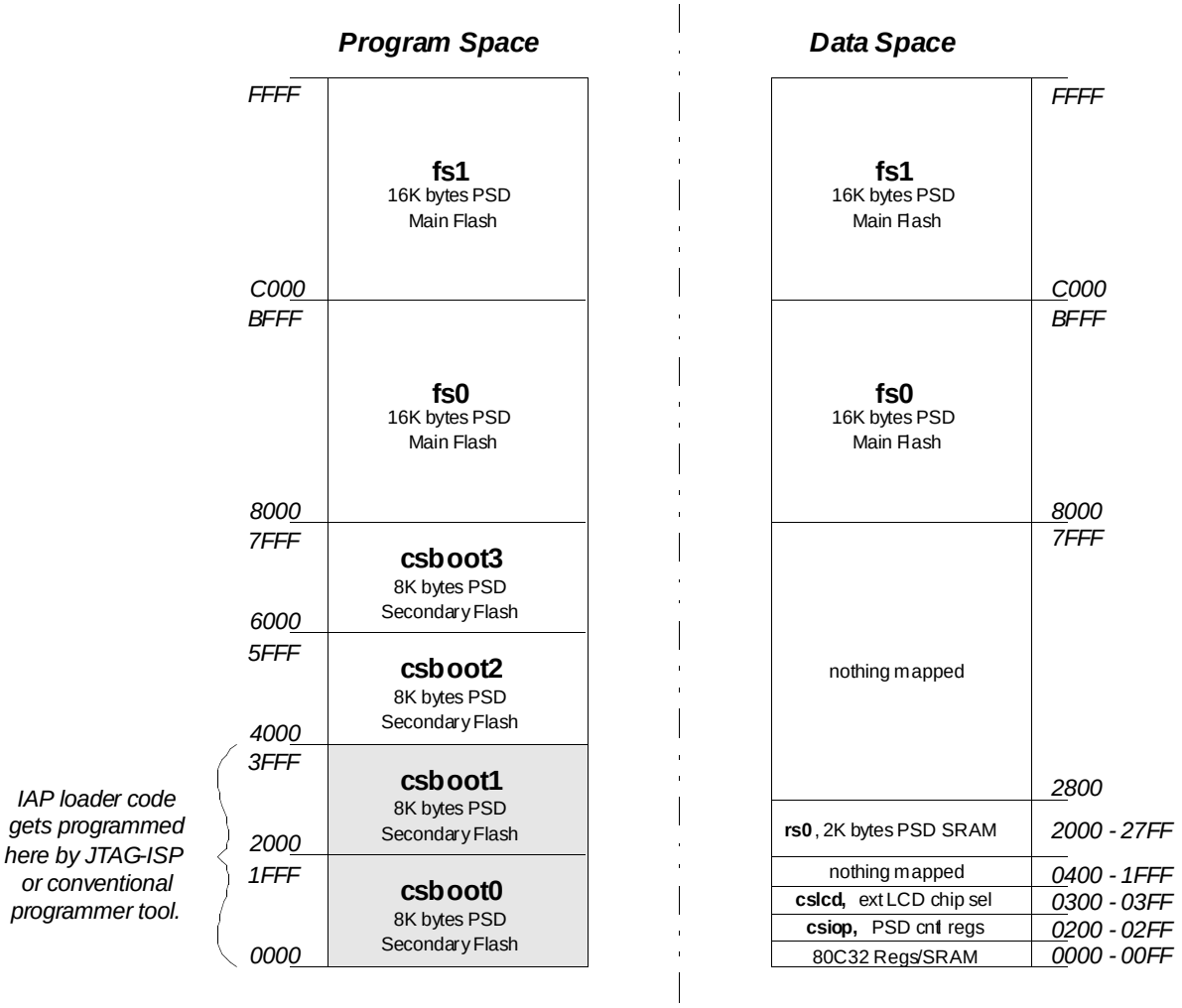


Figure 5 – Memory Map for First Design – IAP with no Memory Paging

The nomenclature fs0, fs1, csboot0-csboot3, rs0, and csiop in Figure 5 refer to the individual internal memory segments of the PSD. The main PSD flash memory has a total of eight 16 Kbyte segments (fs0-fs7). The secondary PSD flash memory has a total of four 8 Kbyte segments (csboot0-csboot3). The 2 Kbyte PSD SRAM has a single segment (rs0). The internal PSD control registers lie in a 256-byte address space named csiop. There is also an external memory chip-select in this example, cslcd, that is used for the LCD module. These PSD memory elements are placed at the desired locations within the system memory map by pointing and clicking choices within PSDsoft Express software.

With the memory arrangement of Figure 5, the 80C32 may perform IAP by executing from the secondary flash segments (csboot0 – csboot3) while erasing and programming the main flash segments (fs0 and fs1). The secondary flash memory is initially programmed though JTAG-ISP or other programming devices with firmware containing the following:

- 80C32 reset vector and initialization routines
- 80C32 interrupt vectors and service routines
- I/O management routines
- IAP loader code.

At power-on or after reset, the 80C32 boots from secondary flash memory, runs a checksum of the main flash memory, programs and verifies main flash via the UART if necessary, then execution jumps to main flash memory.

Notice that the portion of the memory map assigned to the secondary PSD flash memory (csboot0 – csboot3) is quite large, 32 Kbytes. Usually, IAP loader code will fit within less than 8 Kbytes of memory. You may want to designate less secondary flash memory and more main flash memory in your initial design. However, you'll see how all of the large secondary memory in this design is used very efficiently as you proceed on to the second and third designs in this document.

The segments of main PSD flash memory (fs0 and fs1) reside in both Program space and Data space simultaneously while the secondary PSD flash memory segments (csboot0 – csboot3) reside in Program space only, as seen in the memory map of Figure 5. This is done to allow the 80C32 to write to Program storage during IAP. Otherwise, the 80C32 cannot write code to flash memory if it resided in Program space only. PSD silicon architecture and PSDsoft Express allow this kind of flexibility. You will see how to designate the initial configuration of Program space and Data space and you will learn in the second and third designs how to make the 80C32 change the definition of Program space and Data space on the fly for advanced IAP techniques.

3.2 PSDsoft Express Design Entry

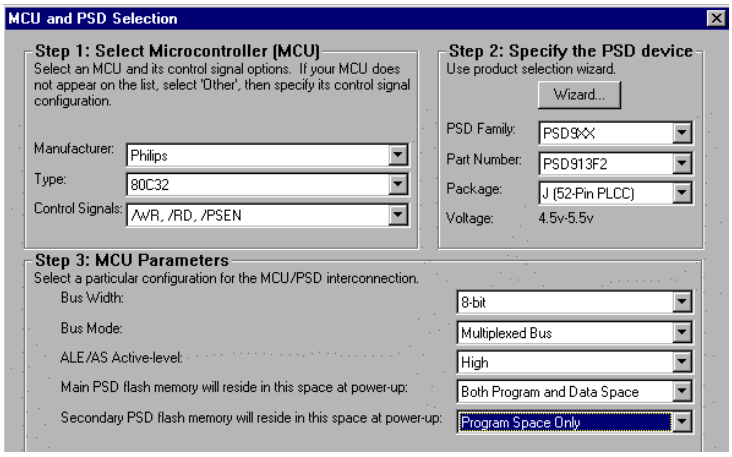
Highlights of the design entry will be given here. The steps are simple and navigation through PSDsoft Express is easy. Invoke PSDsoft Express and follow along if you wish.

3.2.1 Invoke PSDsoft Express and set up your project.

- Start PSDsoft Express.
- Create a new project.
- Select your project folder and name the project (in this example, name the project 'simple32' in the folder PSDexpress\my_project).
- Select an MCU. In this example, we're using a Philips 80C32.
- Select a PSD913F2 and a 52-pin PLCC package.
- Select the main PSD flash memory to reside in both Program and Data space.
- Select the secondary PSD flash memory to reside in Program space only.

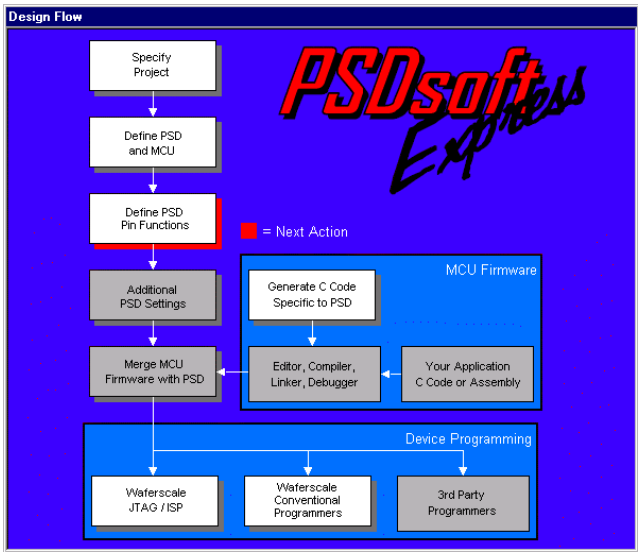
Based on these selections, at power-up and after reset, the main PSD flash memory will reside in both Program and Data space simultaneously, while the secondary PSD flash memory will reside in Program space only. This simple combination allows IAP.

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



Click OK. Now you will be asked if you want to use the Design Assistant or a pre-defined template. Choose Design Assistant. This exercise in the Design Assistant will help you become familiar with the design flow. In the future, you may choose to use a template which will make many of the choices for you, based on your selection of MCU and PSD.

Always reference the main flow diagram shown below to help you navigate through the design process. Clicking on individual boxes within the flow diagram will invoke a process. A box shadowed in red identifies the next process that needs to be completed. PSDsoft Express will automatically invoke the next required process only the first time though a design. When you reenter an existing design, you must choose the next process that you wish to enter from the design flow diagram. If you invoke a process that invalidates other processes downstream, the gray boxes indicate which processes must be invoked again, and the red shadow indicates which process to invoke first.



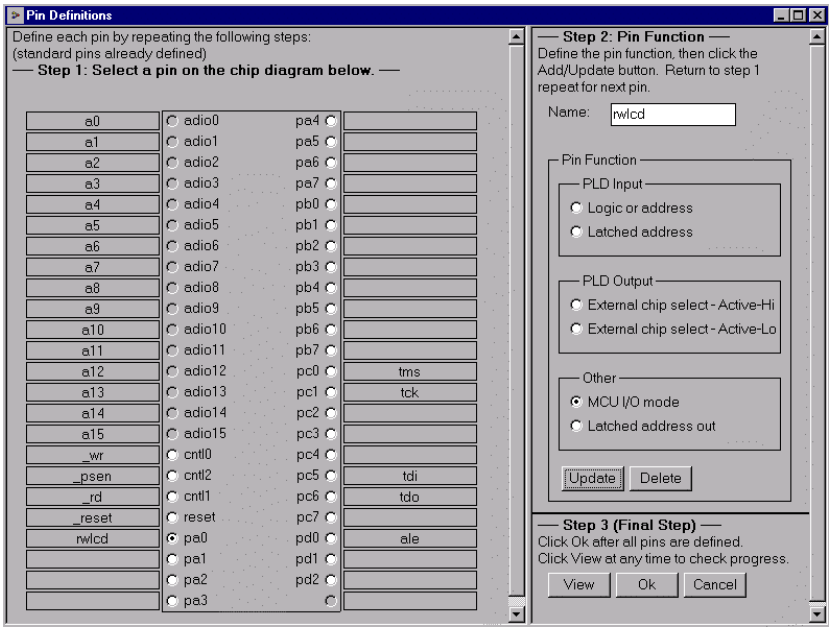
3.2.2 PSD Pin Definition

Next you will see the Pin Definition Screen which allows you to define each PSD pin function on a point and click basis. Notice that all of the PSD pins that connect to the 80C32 are already defined for you. You need only define the remaining pins. For this example, we'll configure the PSD to use:

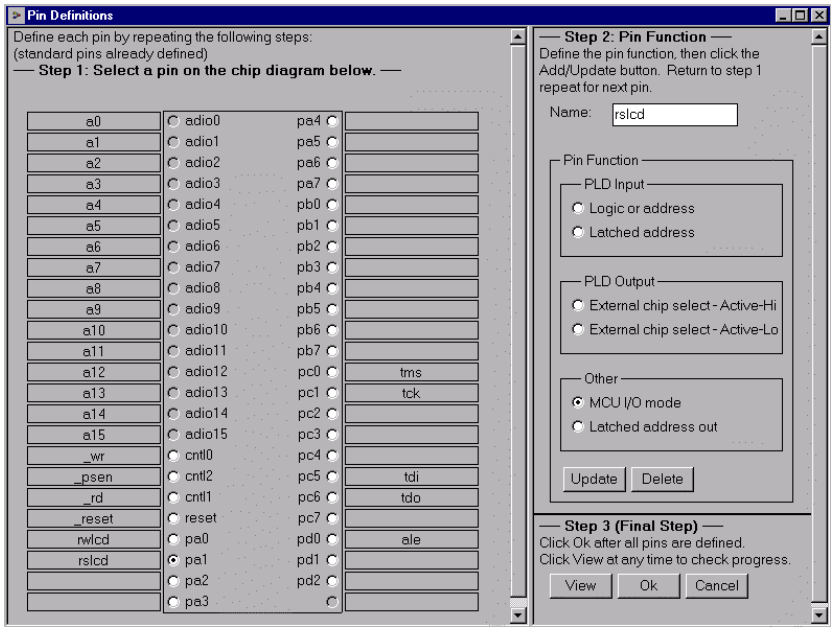
- Two pins on Port A to drive control signals to the LCD module.
- Four pins on Port B as general purpose MCU I/O.
- One pin on Port B as a chip-select output for the LCD module.
- Six pins on Port C as JTAG interface signals for ISP.

Click on pin pa0, type in signal name of "rwlcd", click on 'MCU I/O' in the 'Other' category, then 'Add' or 'Update'. This will produce an output controlled by MCU firmware at run-time, which controls the LCD module read/write input. The 80C32 will control this output signal by writing PSD control registers at run-time. These PSD control registers reside at various address offsets from the base address designated as "csiop". In a later section of this document you will see how to place csiop in your system memory map. The signal rwlcd should be routed to the LCD module on the circuit board as shown in the schematic of Figure 4.

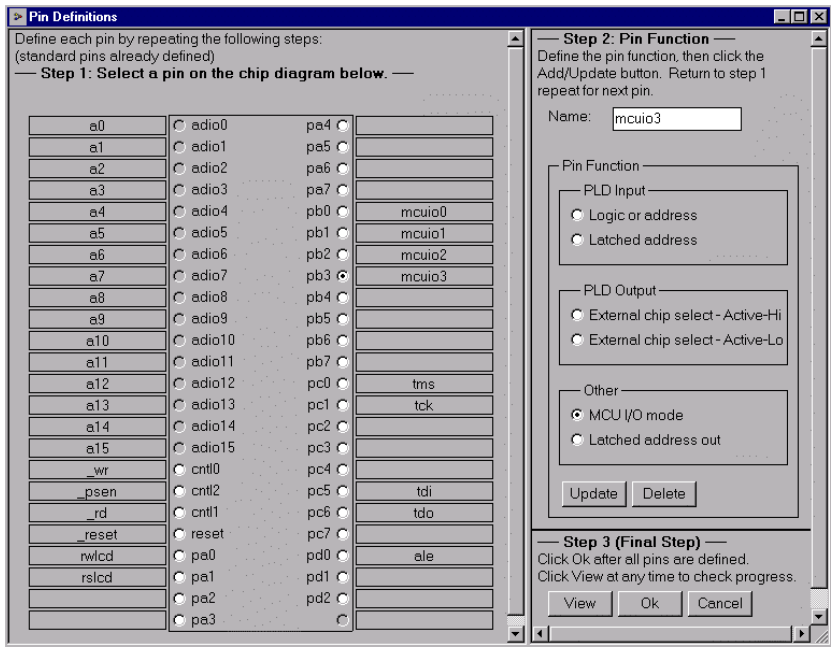
This is what the screen should now look like:



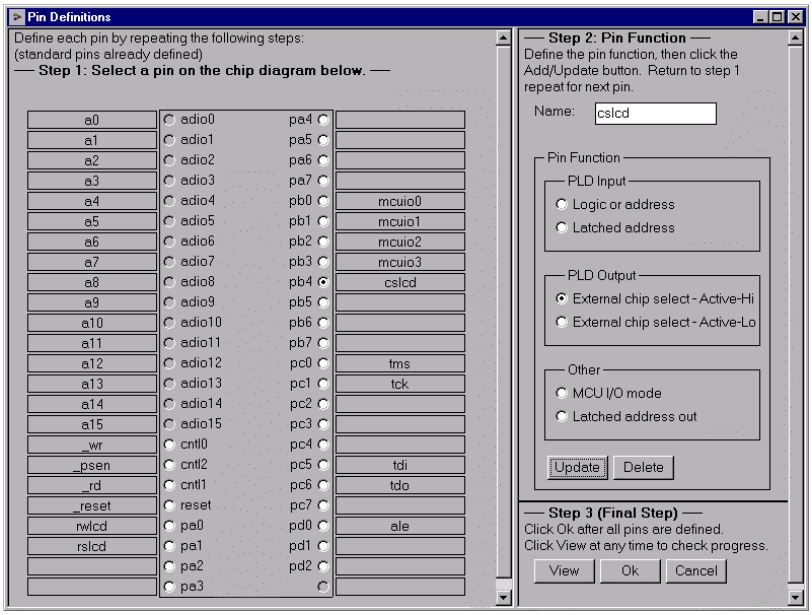
Now click pa1, name the signal “rslcd”, click ‘MCU I/O mode’ in the ‘Other’ category, and click ‘Add’ or ‘Update’. This will produce another output controlled by MCU firmware at run-time, which controls the LCD module register select input. This scheme to control the LCD module allows a fast MCU to run with a slow LCD module. The screen should now look like this:



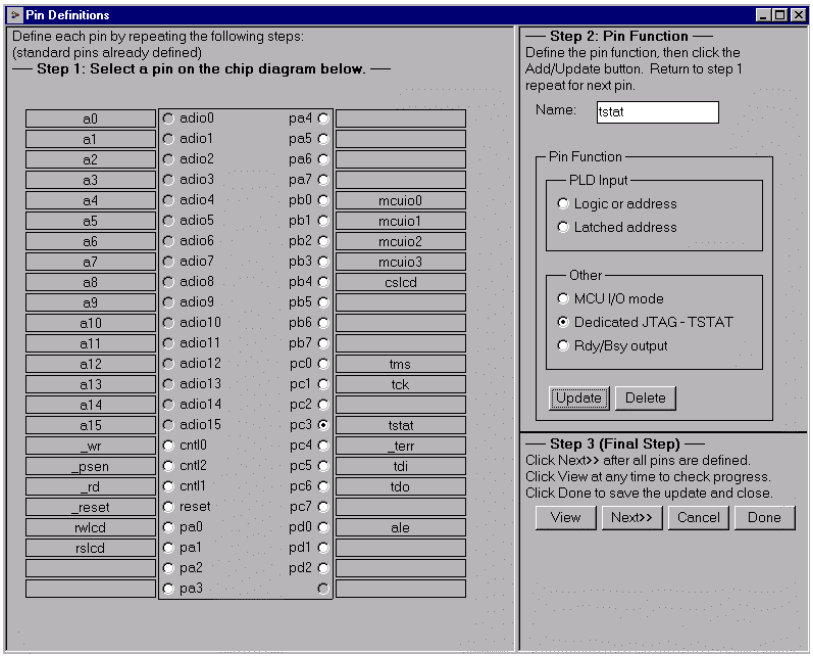
Now click pb0, name the signal “mcuio0”, click ‘MCU I/O mode’ in the ‘Other’ category, and click ‘Add’ or ‘Update’. Repeat for pins pb1, pb2, and pb3, giving names mcuio1, mcuio2, and mcuio3 respectively. This creates four I/O pins on port B that can be set, cleared, and read by the 80C32 accessing PSD control registers at runtime (register base address at “csiop”). The screen should have the following look:



Next click pin pb7, name it “cslcd”, choose ‘External chip-select – Active-Hi’. Click ‘Add’ or ‘Update’. This designates pin pb7 as a chip-select output for the LCD module. See below:



Finally, set up six pins on Port C for JTAG-ISP. Four standard JTAG pins are defined by default (tms, tck, tdi, tdo). For this example, add two more JTAG signals, tstat and terr, to speed the ISP process (see App Note 54 to learn why it is faster with six pins). Click on pin pc3, choose ‘Dedicated JTAG – TSTAT’ then click ‘Add’ or ‘Update’. The signal name is automatically filled in. Also, the signal TERR on pin pc4 is automatically added since tstat and terr must be used as a pair. The screen should now look like this:



That’s all there is to it. Click ‘View’ to see a summary, click ‘Next>>’ to exit the Pin Definition.

3.2.3 System Memory Map: PSD Page Register and Chip Select Definitions.

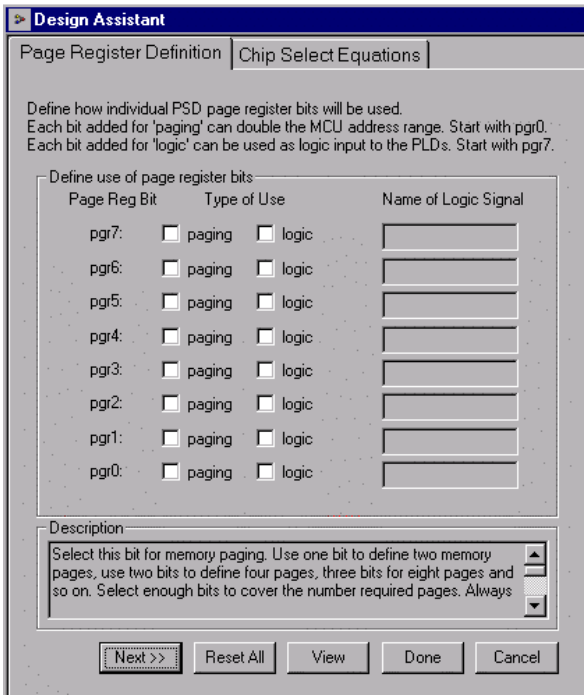
Now that the PSD pins are defined, you will need to define the system memory map. This is accomplished by defining all the chip-selects in the system (both internal to the PSD and external chip-selects), and also defining the function of the PSD page register.

The three memories inside the PSD are individually selected segment-by-segment when MCU addresses are presented to the Decode PLD (DPLD). Each internal PSD memory segment has its own individual chip-select signal name. For example, the main PSD flash memory has eight individual chip-selects (one for each sector) named fs0 – fs7. See the PSD9XXF data sheet for details. Each PSD memory segment must be defined in PSDsoft Express if it is to be accessed by the MCU.

For this example, we must define the internal PSD memory segment chip-selects: fs0, fs1, csboot0 - csboot3, rs0, and csiop to match the memory map of Figure 5. The external chip-select for the LCD module, cslcd, must also be defined, as shown in Figure 5.

In many 80C32 system designs, memory paging is used to address more than 64 Kbytes of address space. However, for this simple design, no paging is used so no PSD page register bits need to be defined.

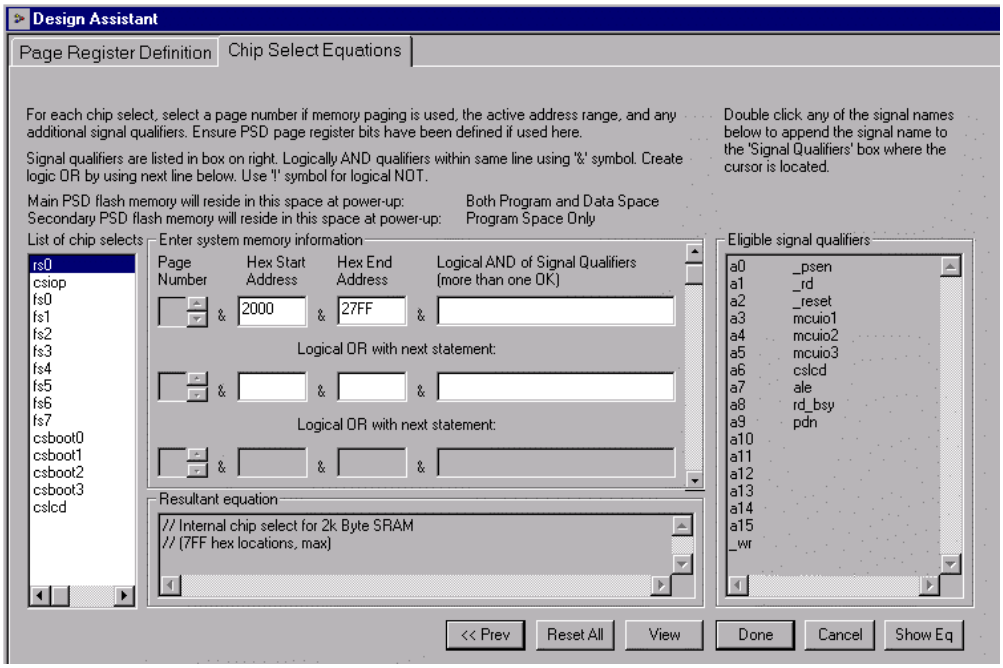
The following Page Register Definition screen should be present on your monitor, just click ‘Next >>’ since we are not using memory paging in this first design example.



Now define the internal and external chip-selects. Start with the internal chip-select for PSD SRAM, which is “rs0”. Then enter start and stop MCU addresses to match the memory map of Figure 5. Additional signal qualifiers (80C32 control signals psen, rd, wr, ale) are NOT needed

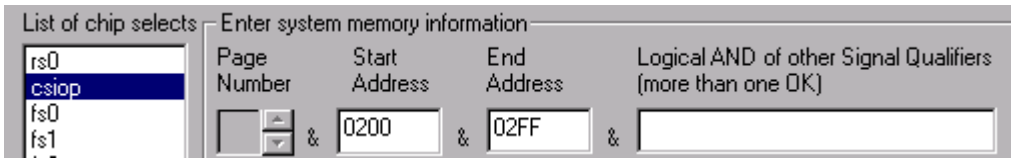
for internal PSD memory chip-selects as this is taken care of in silicon. The screen should look like the following:

rs0:



Next, define the chip-select for the base address of the internal PSD control registers by clicking on “csiop” in the left side of the screen. Enter the address range as shown:

csiop:

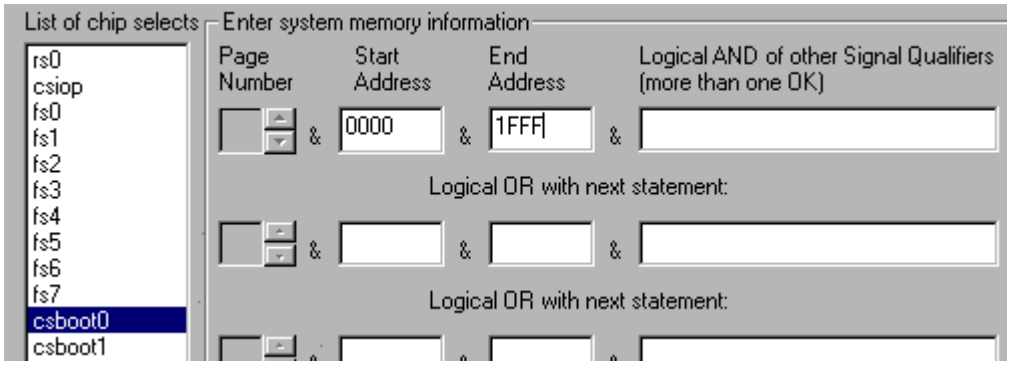


Continue to define internal PSD memory chip-selects for the main flash memory segments fs0 and fs1, and then the secondary flash memory segments csboot0 – csboot3. Use Figure 5 as a guide for address ranges. Again, no signal qualifiers are needed for internal PSD memory chip-selects. This is what the screen should look like for each chip-select:

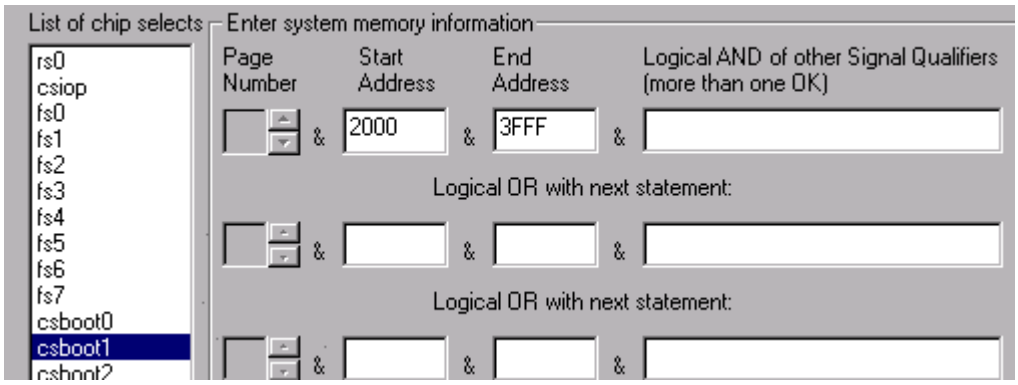
fs0:

fs1:

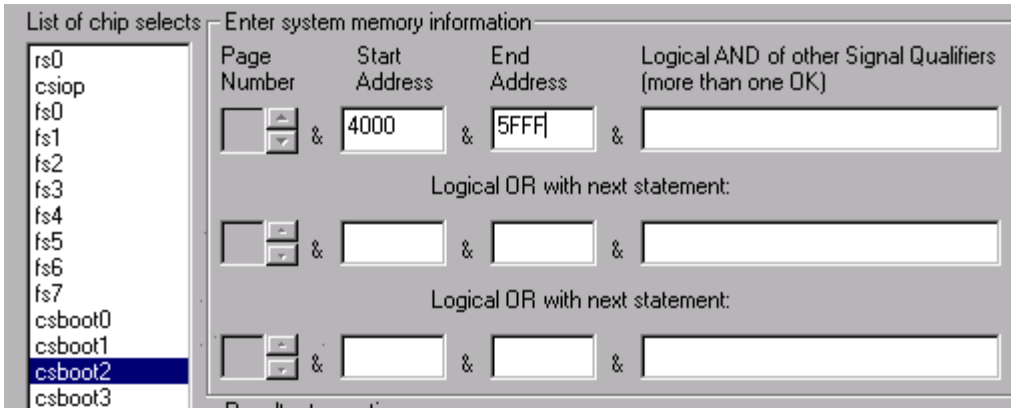
csboot0:



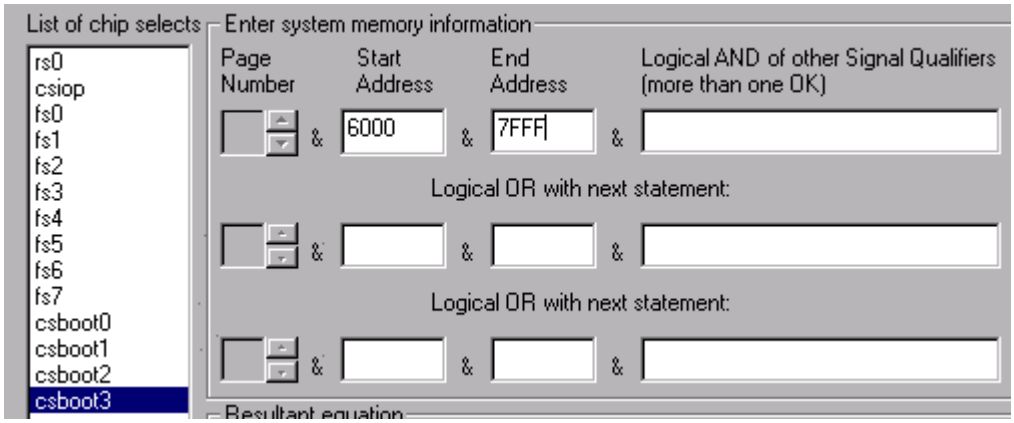
csboot1:



csboot2:



csboot3:

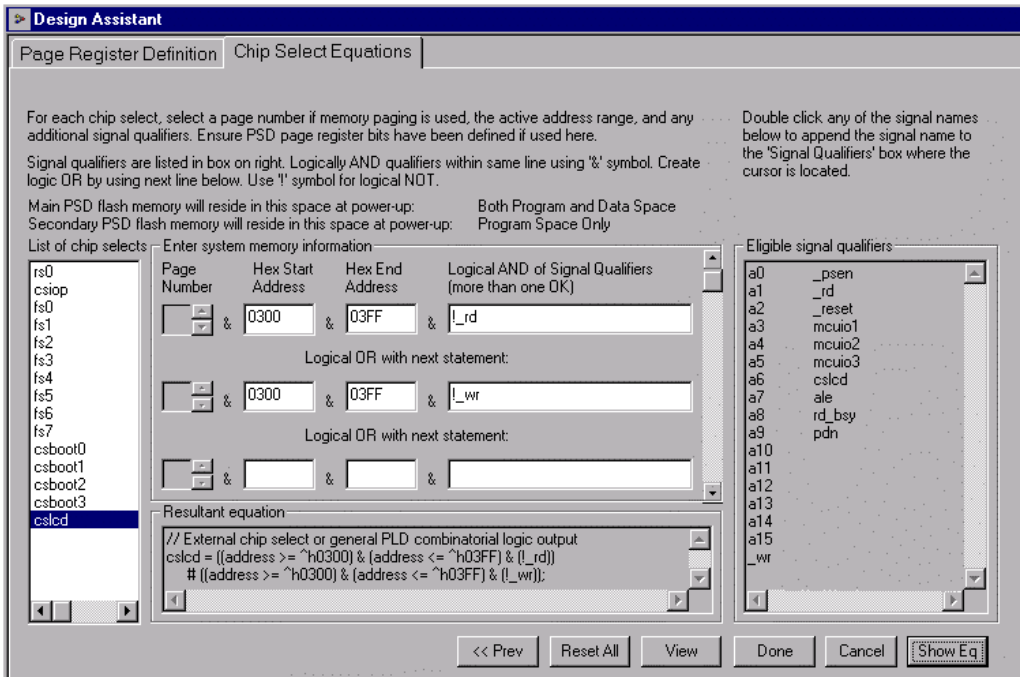


Finally, define the external chip-select for the LCD module, “cslcd’. This chip-select is different for two reasons. First, it is an external chip-select that does not activate any memory element inside the PSD because the signal “cslcd” is output on a PSD I/O pin. And second, this chip-select requires qualifiers, meaning that this logic signal is true only for a given MCU address range AND only when one of two other another signals are active.

In this design, “cslcd” is true only when the MCU presents an address in the range of 0300 to 03FF hex AND when either the 80C32 control signal “_rd” is true, OR when 80C32 signal “_wr” is true. To create this logic, enter information as shown in the screen below. Since both signals, “_rd” and “_wr”, are active low as they leave the 80C32, the logical NOT operator (!) is used when they are specified as qualifiers.

Signal qualifiers may be added by setting the cursor where you want the signal name to go then just double click on the signal name in the list of qualifiers.

cslcd:



Click ‘Done’.

3.2.4 Additional PSD Configuration

Now you should see the main flow diagram again. Click on the box ‘Additional PSD Configuration’. This is where you may choose to set the security bit to prevent a device programmer from examining or copying the contents of the PSD. You can also click through the other sheets on this screen to set the JTAG USERCODE value and set sector protection on PSD Non-Volatile memory segments.

3.2.5 C Code Generation

You can take advantage of the low level C code drivers that are generated by PSDsoft Express for accessing memory elements within the PSD by clicking on the 'C code Generation' box in the design flow window. ANSI C code functions and headers are generated for you to paste into your 80C32 C compiler environment. Just tailor the code to meet your system needs and compile. C code generation can be performed anytime after a project is opened.

To generate ANSI C functions and headers, simply specify the folder(s) in which you want the header files and the C source file to be written, and name the C source file. Select the categories of functions that you would like to include, then click "Generate". Three files will be written to your specified folder(s):

- <your_specified_name>.c ….ANSI-C source for all of the selected functions
- psd913F2.h—ANSI-C ……..header file to define particular PSD registers
- map913F2.h—ANSI-C …….header file to define locations of system memory elements (Main and Secondary Flash, PSD registers, etc.).

Notice that you do not have a choice to rename the two generated header files. This is because those header files are specified by name within the generated C function source file. If you edit the names of the generated header files, be sure to edit the generated C function source file to match the new header file names.

The three generated files may now be tailored and integrated into your 80C32 compiler environment. The file psd913F2.h contains a #define statement for each individual C function within the <your_specified_name>.c file. Edit psd913F2.h and simply remove the comment delimiters (//) from the #define statement for each generated C function that you would like to be compiled with the rest of your C source code.

There are also coded examples available. Click on the 'Coded Examples' tab at the top of the C Code Generation screen. This sheet contains several examples that you may use as a basis for building your own C code application. These are complete projects (main, functions, and headers) targeted toward a particular MCU. You may copy these files to some folder to browse them for ideas, or cut and paste sections from the examples into your own MCU cross-compiler environment.

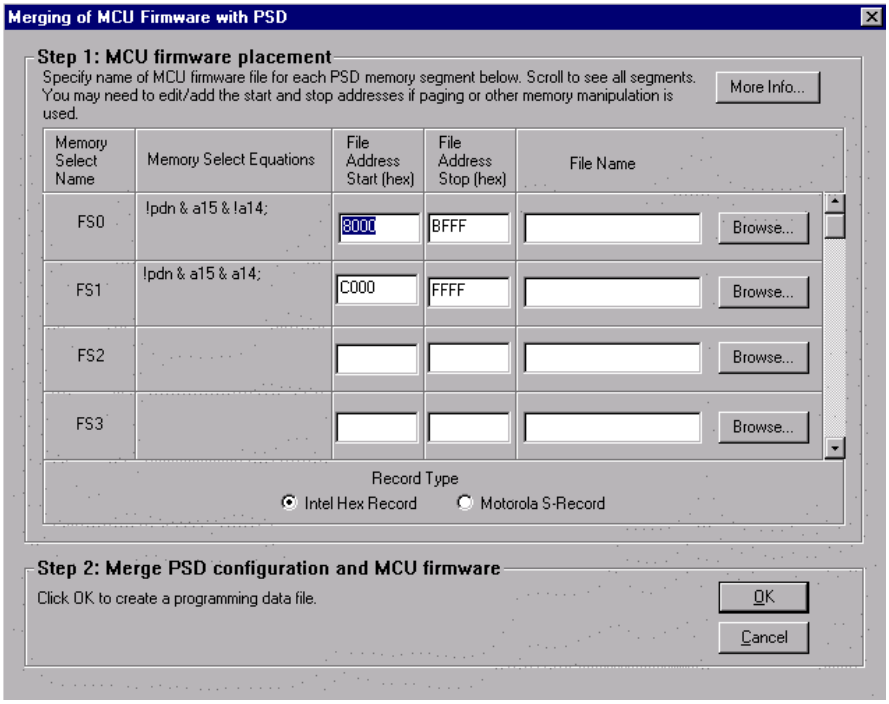
3.2.6 Merging MCU Firmware

Now that all PSD pins and internal configuration settings have been defined, PSDsoft Express will create a single object file (*.obj) that is a composite of your 80C32 firmware and the PSD configuration. FlashLINK™, PSDpro, and third party programmers can use this object file to program a PSD device. PSDsoft Express will create simple32.obj for this design example.

During this merging process, PSDsoft Express will input firmware files from your 80C32 compiler/linker in S-record or Intel HEX format. It will map the content of these files into the physical memory segments of the PSD according to the choices you made in the 'Chip Select Equations' screen. This mapping process translates the absolute system addresses inside 80C32 firmware files into physical internal PSD addresses that are used by a programmer to program

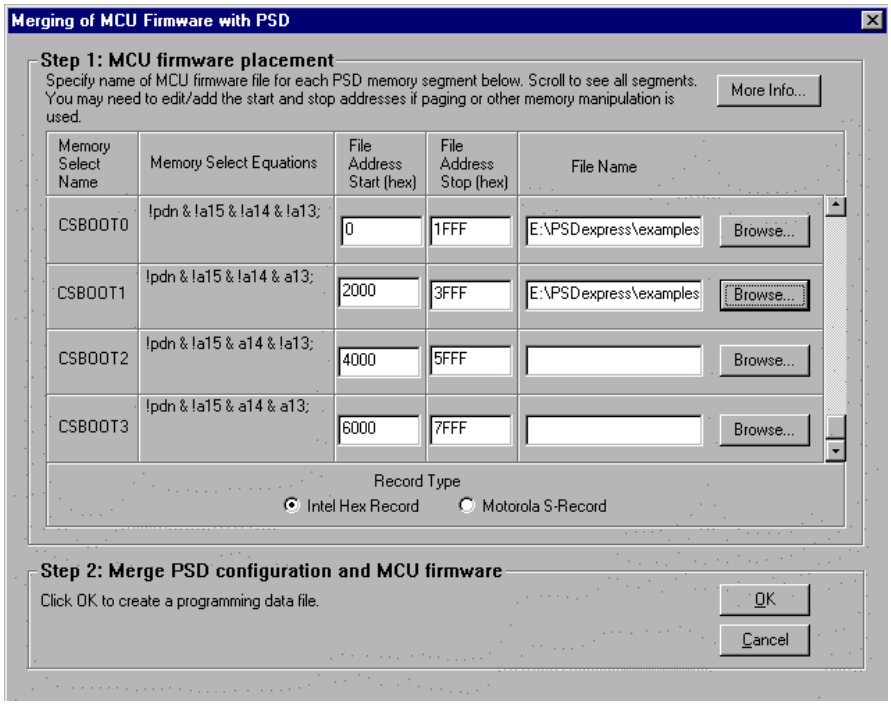
the PSD. This address translation process is transparent. All you need to do is type (or browse) the file names that were generated from your 80C32 linker into the appropriate boxes and PSDsoft Express does the rest. You can specify a single file name for more than one PSD chip-select, or a different file name for each PSD chip-select. It depends on how your 80C32 linker has created your firmware file(s). For each PSD chip-select in which you have specified a firmware file name, PSDsoft Express will extract firmware from that file only between the specified start and stop addresses, and ignore firmware outside of the start and stop addresses.

Click on 'Merge MCU Firmware' in the main flow diagram. First you will notice that PSDsoft Express will "Fit" your PSD configuration to the silicon architecture of the PSD. After the fitting process is complete, you'll see this screen:



In the left column are individual PSD memory segment chip-selects (FS0, FS1, etc). The next column shows the logic equations for selection of each internal PSD memory segment. These equations reflect the choices that you made while defining PSD internal chip-select equations in an earlier step. In the middle of the screen are hexadecimal start and stop addresses that PSDsoft Express has filled in for you based on your chip-select equations. On the right are fields to enter (browse) the MCU firmware files.

Select ‘Intel Hex Record’ for ‘Record Type’ as shown. Now scroll down to the bottom until you see CSBOOT0. Use the ‘Browse’ button and select the firmware file, PSDexpress\examples\isp_8032.hex. Now do the same for CSBOOT1. The screen should look like this:



This specification places firmware in secondary PSD flash memory segments csboot0 and csboot1. PSDsoft Express will extract any firmware that lies inside the file isp_8032.hex between MCU addresses 0000 and 1FFF and place it in PSD memory segment csboot0. It will also extract any firmware that lies inside the file isp_8032.hex between MCU addresses 2000 and 3FFF and place it in PSD memory segment csboot1. Click OK to generate the composite object file, simple32.obj.

Note: The file isp_hc32.hex will run on the DK900 development board from Waferscale, and display some messages on the LCD screen to indicate a successful ISP session. For your own prototype project, create a simple firmware file that configures your system hardware and performs rudimentary tasks to check out your new hardware. In this design example, there are 32 Kbytes available in secondary flash memory segments csboot0 – csboot3, 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 to for advanced tasks, including IAP of main PSD flash memory.

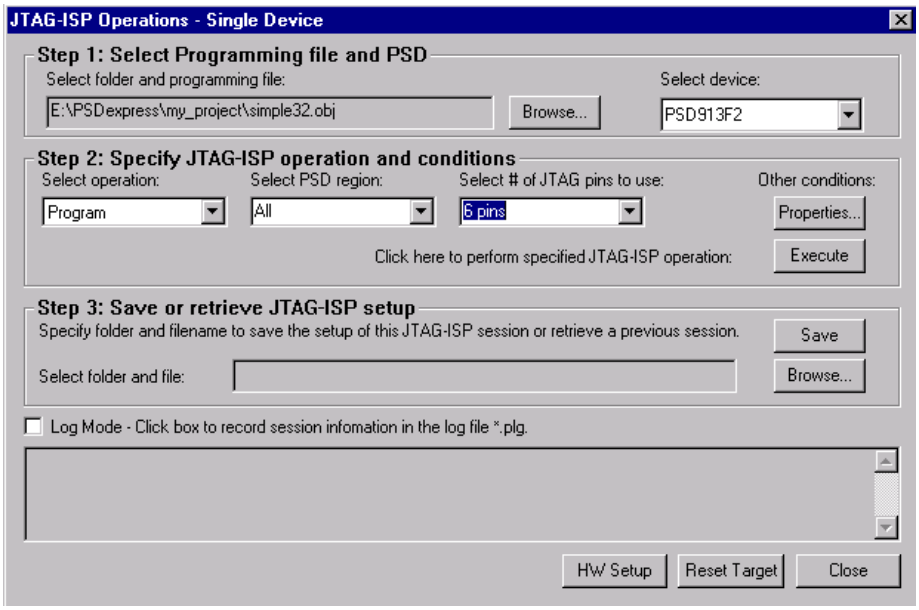
3.2.7 Programming the PSD

The simple32.obj file can be programmed into the PSD by one of three ways:

- The Waferscale FlashLINK™ JTAG cable, which connects to the PC parallel port.
- The Waferscale PSDpro device programmer, which also uses the PC parallel port.
- Third-party programmers, from Stag, Needhams, and others. See our web site at www.waferscale.com for compatible third-party programmers.

3.2.7.1 Programming with FlashLINK™

Connect the FlashLINK™ JTAG-ISP cable to your PC parallel port. Click the 'JTAG-ISP' box in the design flow window. You should see the following screen:



This window enables you to perform JTAG-ISP operations and also offers a loop back test for your FlashLINK™ cable. If this is your first use, test your FlashLINK™ cable and PC parallel port by clicking the 'HW Setup' button, then click 'LoopTest' button and follow the directions.

Now let's define our JTAG-ISP environment. For this example project, PSDsoft Express should have filled in the folder and filename of the object file to program, the PSD device, and the JTAG-ISP operation, as shown in the screen above. For this design example, we have chosen to use all six JTAG-ISP pins (instead of four). Be sure to indicate "6 pins" as shown above to achieve minimum JTAG-ISP programming times (refer to Application Note 54 for details on six pins vs. four)

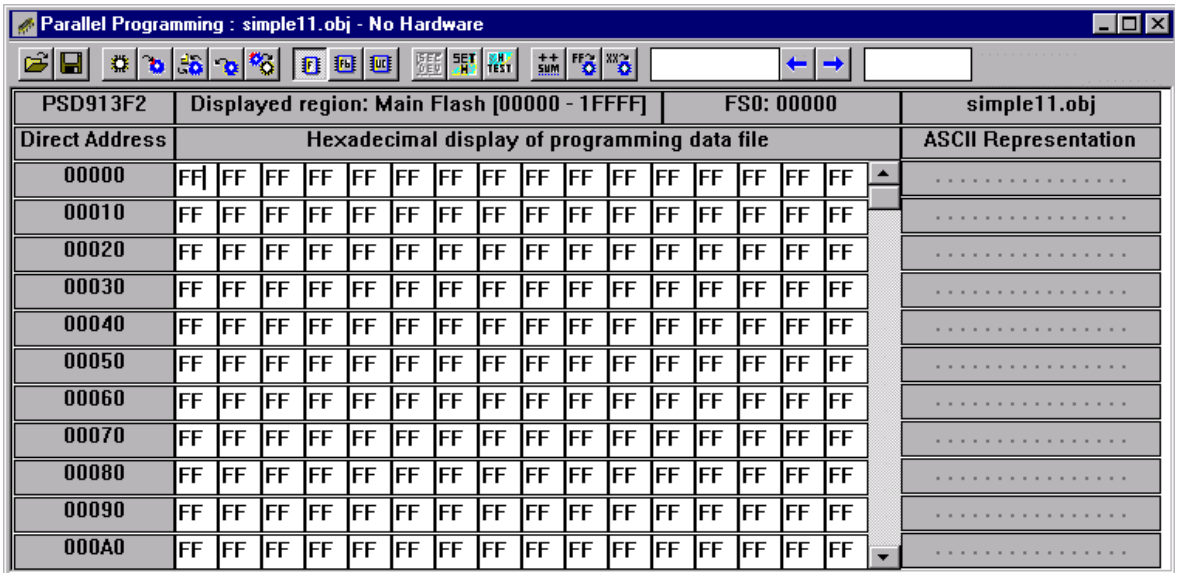
To begin programming, connect the JTAG cable to the target system, power-up the target system, and click 'Execute' on the JTAG screen. The Log window at the bottom of the JTAG screen shows the progress. Programming should just take a few seconds.

There are optional choices available when the 'Properties..' button is clicked. One choice includes setting the state of all non-JTAG PSD I/O pins during JTAG-ISP operations (make them inputs or outputs). The default state of all non-JTAG PSD I/O pins is "input", which is fine for this design example. The other choice allows you to specify a USERCODE value to compare before any JTAG-ISP operation starts. This is typically used in a manufacturing environment (see on-screen description for details).

After JTAG-ISP operations are completed, you can save the JTAG setup for this programming session to a file for later use. To do so, click on the 'Save' button. To restore the setup of a different previous session, click the 'Browse..' button.

3.2.7.2 Programming with PSDpro

Connect the PSDpro device programmer to your PC parallel port per the installation instructions. Click on the ‘Conventional Programmer’ box in the design flow window. You will see this:



If this is the first use of the PSDpro, you’ll need to designate the PSDpro as the device connected to your parallel port. To do this, click the “SET H” icon button at the top of the “Conventional Programming” screen and choose the PSDpro. Then click on the ‘H TEST’ icon to perform a test of the PSDpro and the PC parallel port. After testing, place a PSD913F2 into the socket of the PSDpro and click on the ‘Program’ icon (the simple32.obj file is automatically loaded when this process is invoked). The messaging of PSDsoft will inform you when programming is complete.

> Note: This window is also helpful even if you do not have a PSDpro device programmer. Use this window to see where the “Merge MCU Firmware” utility has placed 80C32 firmware within physical memory of the PSD. For this design example, click on the secondary PSD flash memory icon “Fb” in the tool bar to see the 80C32 reset vector at absolute MCU addresses 0001h and 0002h, which translates to direct physical PSD addresses 20001h and 20002h respectively. To see how all of your 80C32 absolute addresses translated into direct physical PSD memory addresses, veiw the report that PSDsoft generates under ‘Reports’ from the main toolbar, then select ‘Address Translation Report’. Within the report, the ‘start’ and ‘stop’ addresses are the absolute MCU system addresses that you have specified. The addresses shown in square brackets are the 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).

4 Second Design Example – IAP with Memory Paging

This second design example builds upon the first by adding memory paging which allows the 80C32 to access all of the memory resident on the PSD913F2. The physical connections between the 80C32 and PSD913F2 do not change, but the memory map, PSD page register definition, and some PSD chip-select equations do change. A PSD913F2 is still used in this example, but the

other members of the *Easy*FLASH™ family may be used instead with minor changes. See the PSD9XXF data sheet for a comparison of family members.

4.1 Memory Map

The PSD913F2 provides 128 Kbytes of main flash memory, 32 Kbytes of secondary flash memory, and 2 Kbytes SRAM. In this second design, we'll use all of this memory. Since the 80C32 cannot address more than 64 Kbytes of address space directly, we will use paging (or banking) to access all 162 Kbytes of memory. The PSD has a built-in page register for this purpose. Many MCU cross-compilers support paging today.

Figures 6 and 7 represent the system memory maps for this design. These maps have paged memory in the upper address range of 8000 to FFFF hex, and common memory in the lower range of 0000 to 7FFF hex. In Data space, the 80C32 has access to system SRAM and I/O regardless of what memory page is active. In Program space, the firmware in the secondary PSD flash memory (csboot0 – csboot3) is available to the 80C32 regardless of what memory page is active. This common Program area holds the following:

- 80C32 reset vector and initialization routines
- 80C32 interrupt vectors and service routines
- I/O and memory page management routines
- SRAM variables and SRAM stack
- IAP loader code
- Anything else that must be accessible no matter what memory page is selected.

Figure 6 represents the system memory map at power-up and after reset. This map is also valid during IAP. Notice that all of the main PSD flash memory is initially in Data space so that the 80C32 can write to it during IAP. Also notice that all of the secondary PSD flash memory is initially in Program space so the 80C32 can execute code from it during IAP. The choice for this initial placement of memory in Program or Data space is made within PSDsoft Express ('Define MCU and PSD' in flow diagram).

Figure 7 represents the system memory map after IAP is complete. All of main PSD flash memory has moved to Program space. The PSD has a control register (named the VM register) that allows the 80C32 to change the definition of Program space and Data space at run-time for IAP purposes. This VM register is accessed at an address offset from the base address, "csiop".

Sequence of events for IAP:

- Fig 6 - at power on or after reset, the 80C32 boots from secondary PSD flash memory
- Fig 6 - 80C32 runs a checksum of the main PSD flash memory in Data space
- Fig 6 - If needed, 80C32 programs and verifies main PSD flash in Data space via the UART
- Fig 6 - 80C32 writes 06 hex to the VM register to place main PSD flash into Program space
- Fig 7 – main flash has moved to program space as a result of writing 06 hex to VM register
- Fig 7 - 80C32 can now execute application code from either main or secondary PSD flash

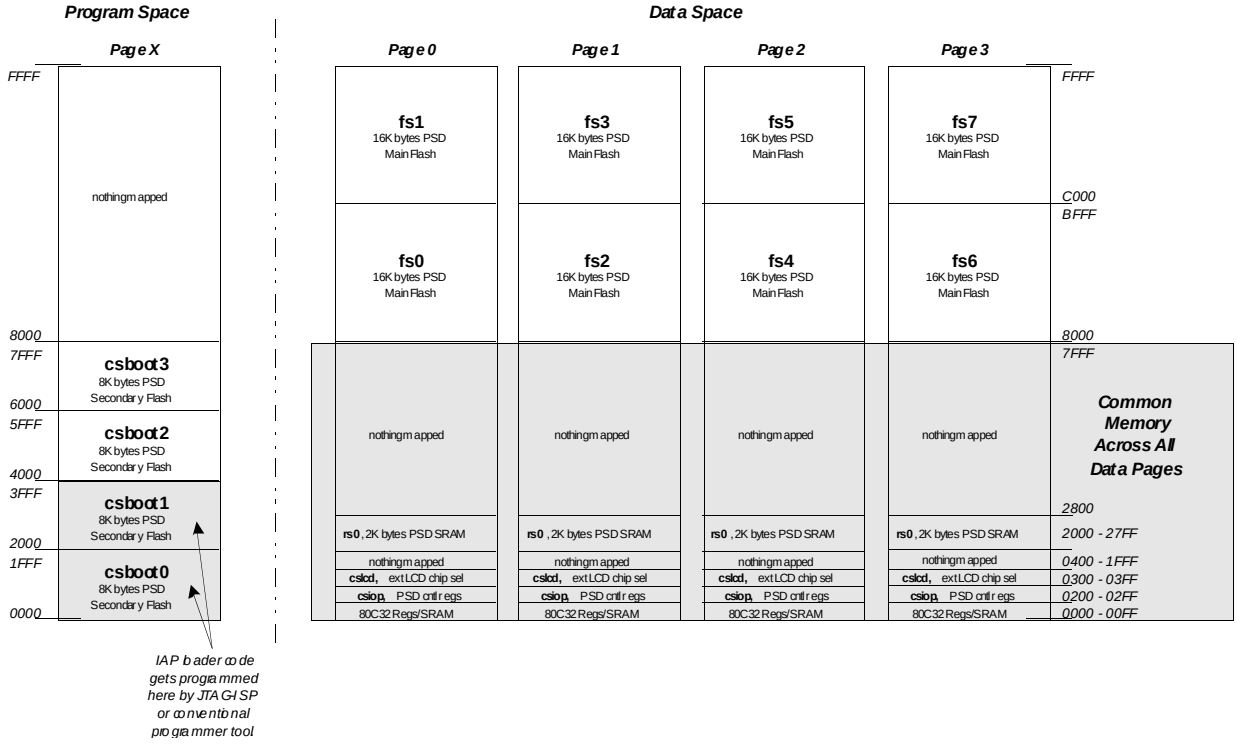


Figure 6 – Memory Map at Boot-Up or Reset and During IAP.
PSD VM register initially 12 hex, Main PSD flash in Data space

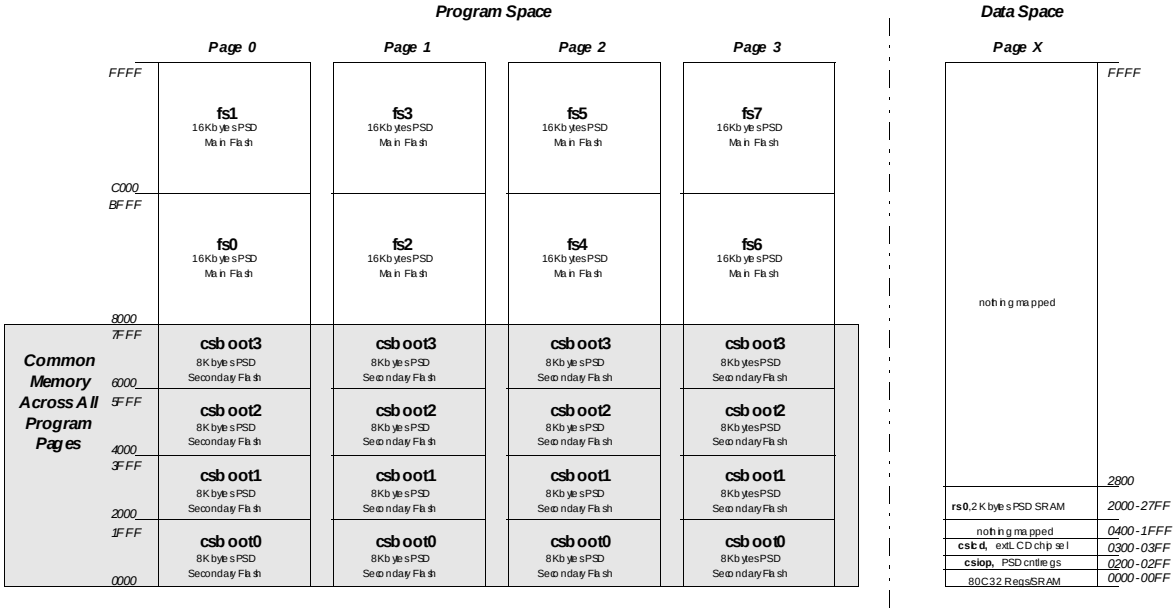


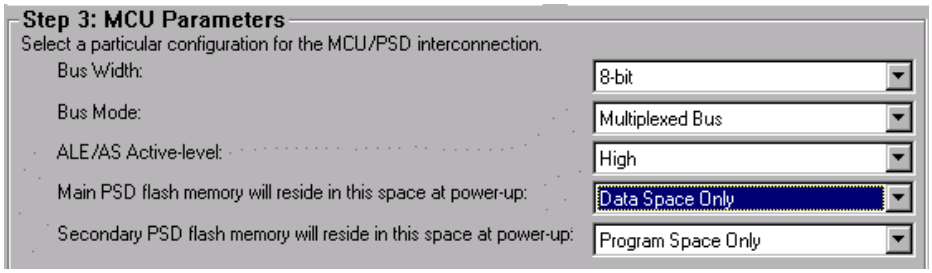
Figure 7 – Memory Map just after 80C32 wites 06 hex to PSD VM register .
IAP complete, main PSD flash moves to Program space

Your system design may require that you operate application code completely from main PSD flash memory after IAP is complete. This means swapping the secondary PSD flash memory (containing IAP loader code) out of Program space, and replacing it with main PSD flash memory (containing application code). This is explained in the third design example.

4.2 PSDsoft Express Design Entry

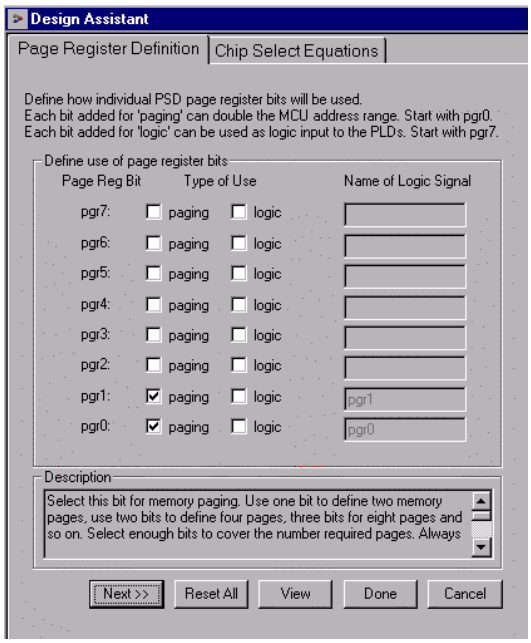
To implement the memory map with paging techniques shown in Figures 6 and 7, invoke PSDsoft Express, open the project "simple32" from the first design example. Now pull down the menu 'Project' from the top of the screen, and select 'Save As'. For this second design example, save the first project under the new name "page32".

Click on the 'Define PSD and MCU' box in the design flow diagram. Change the settings for initial placement of Program and Data space in Step 3 as shown here, then click OK:



Now click on the Pin Definition box in the design flow diagram. Click OK to get to the Page Register Definition screen since no pin assignment needs to be changed for this second design.

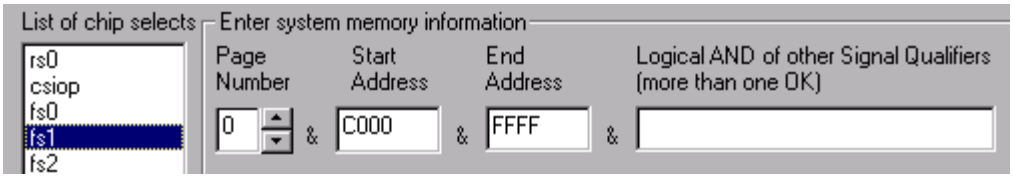
There are a total of four memory pages used in Figures 6 and 7, so you will need to define two PSD page register bits for paging ($2^2 = 4$). To do so, click on pgr0 and pgr1 as shown below, then click 'Next>>'.



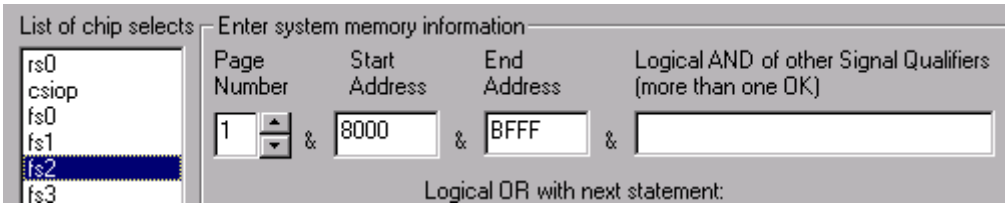
The chip-select equations for PSD SRAM (rs0), PSD control registers (csiop), and the external LCD module (cslcd) do not change from the first design example. Only chip-selects for main PSD flash memory will change because they are now paged in this design. Define the internal PSD main flash memory chip-select signals to implement the memory map of Figures 6 and 7. The following illustrates how the chip-selects will look when you enter their definitions:

fs0:

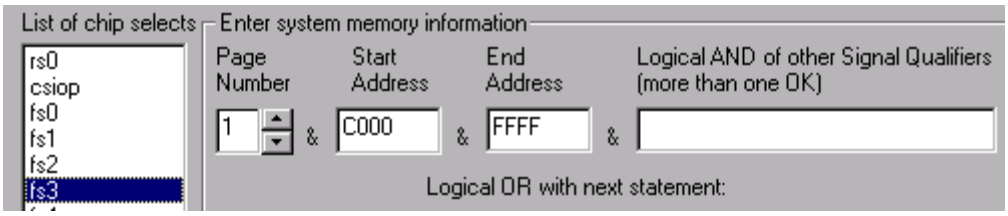
fs1:



fs2:



fs3:



Continue defining fs4 and fs5 on memory page 2, and fs6 and fs7 on memory page 3. The chip-selects for csboot0 through csboot3 did not change from the first design example because no page number was specified in their definition. This means they will appear to the 80C32 on any memory page as indicated in Figures 6 and 7. Click ‘Done’ to get the main flow diagram.

Click the ‘Merge MCU Firmware’ box in the design flow diagram. You will see an informational dialog box pop up that indicates memory paging is used and that the firmware file(s) you specify should be set up to handle paging. Click OK, since for this design example the firmware that would run the IAP process is not paged. It resides in csboot0 and csboot1 of the PSD and is active on all pages (independent of what memory page is selected).

> Click the “More Info” button in Step 1 of the Merge Firmware screen if your future 80C32 system design will execute code from different pages, and that code will be programmed into the PSD with a device programmer (for example, you specify filename(s) in this screen that go to PSD memory segments that are paged).

Now specify the name of the 80C32 firmware file to place into the secondary flash memory segments csboot0 and csboot1. This can be any file that you create to implement IAP with paging. No firmware filename needs to be designated for the main PSD flash segments (fs0 – fs7) since they will be programmed by the 80C32 during IAP. Click OK in the merging screen to create a composite object file for programming. Program the PSD913F2 as in section 3.2.7.

5 Third Design Example – Advanced IAP with Paging & Swapping

The third design example adds enhanced IAP features. The physical connections between the 80C32 and PSD913F2 do not change, but the memory map, PSD page register definition, and some PSD chip-select definitions do change. This enhanced design gets the most out of the 64 Kbyte address space that the 80C32 can access directly. This means swapping the IAP loader code out of Program space after IAP is complete, and replacing it with application code, leaving the maximum amount of address space available for the application. Swapping out the IAP loader code not only frees up address space for application code, it also allows the software designer the option of having two sets of interrupt vectors and associated service routines; one set during IAP, and a different set after IAP during the normal application. In addition, this swapping technique allows the IAP loader code itself to be programmed in the field while the 80C32 operates out of main PSD flash memory. Additionally, this design allows half of the secondary PSD flash memory to be used for IAP loader code, and the other half for general data.

> To actually use this third design, purchase the DK-900 kit from www.waferscale.com for $99 USD. It includes all hardware and firmware, and also a PC Windows program to implement IAP with a UART.

5.1 Memory Map

The memory map is a sequence of four steps shown in Figures 8 through 11. Figure 8 is the memory map at system power-on or system reset. The swap bit is defined as one of the eight internal PSD page register bits. The swap bit is an example of how page register bits can be implemented for uses other than memory paging. Here's the sequence after power-up or reset:

- Fig 8: 80C32 boots from secondary flash memory (csboot0/csboot1) at address 0000
- Fig 8: 80C32 performs a checksum of main flash memory while it resides in Data space
- Fig 8: 80C32 downloads to main flash from host computer if needed and validate contents
- Fig 8: 80C32 writes 06 hex to PSD VM register
- Fig 9: Main Flash memory has moved to Program space because of 06 hex in VM register
- Fig 9: 80C32 sets swap bit to logic one (writes to PSD page register)
- Fig 10: Secondary flash memory has moved out of MCU address range 0000 to 7FFF and main flash memory has moved into its place because of the swap bit. This swapping action is implemented by qualifying the chip-selects with the swap signal.
- Fig 10: 80C32 writes 0C hex to PSD VM register
- Fig 11: Secondary flash memory has moved to Data space because of 0C hex in VM register.

Figure 11 shows the final memory map. At this point, the 80C32 can download new IAP loader code to csboot0/csboot1 if needed. Another one of the eight PSD page register bits is used for logic in this design, named unlock. The 80C32 must first set the unlock bit to logic one before updating IAP loader code. All page register bits are cleared to zero at power-up or at reset.

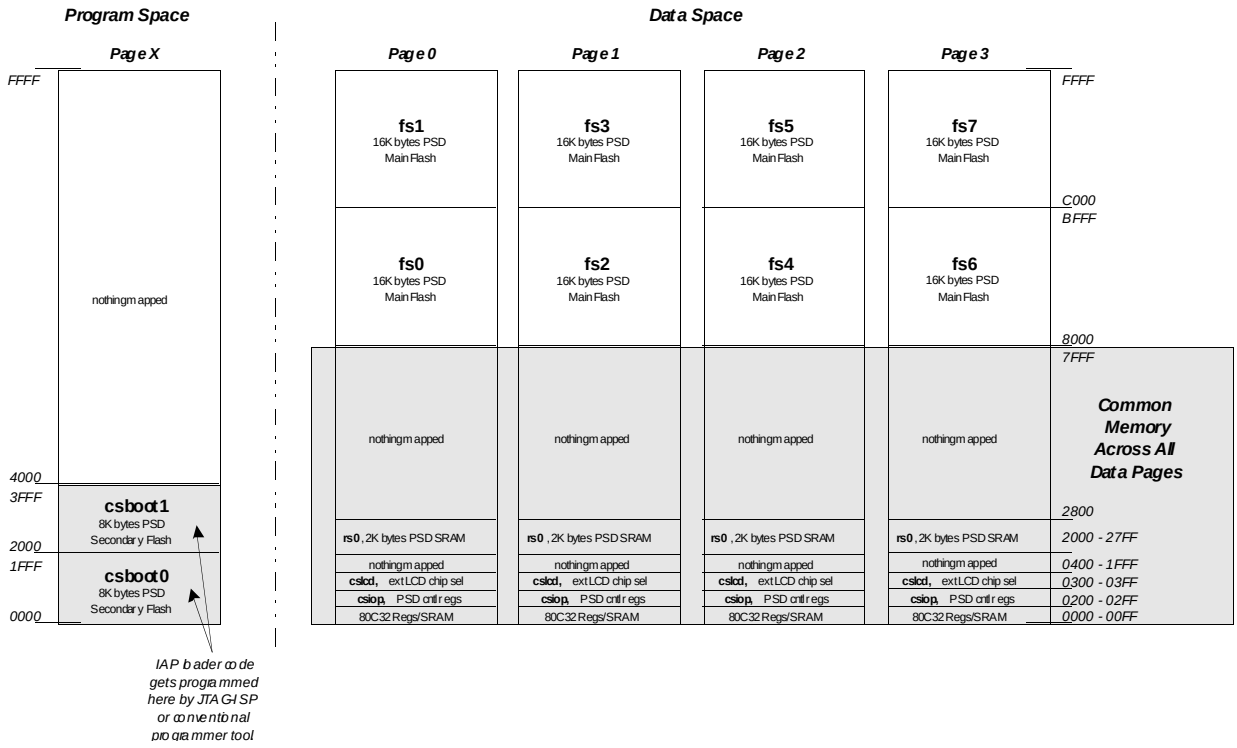


Figure 8 – Memory Map at Boot-Up or Reset and During IAP.
PSD VM register = 12 hex, swap = 0, unlock = 0

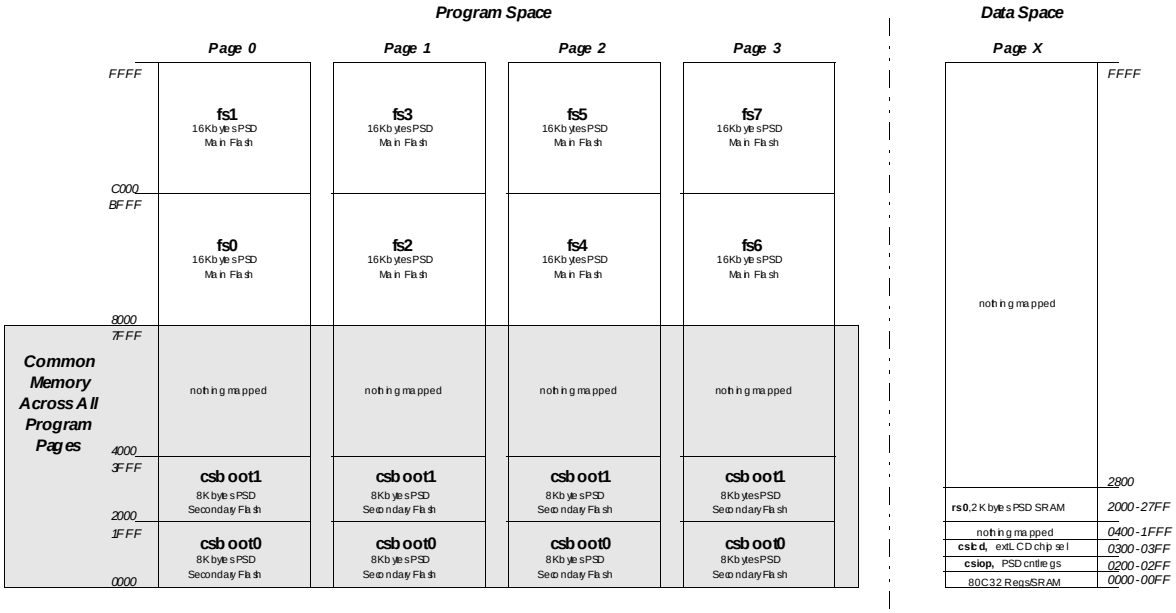


Figure 9 –Memory Map just after 80C32 writes 06 hex to PSD VM register.
Main PSD flash moves to Program space, swap = 0, unlock = 0

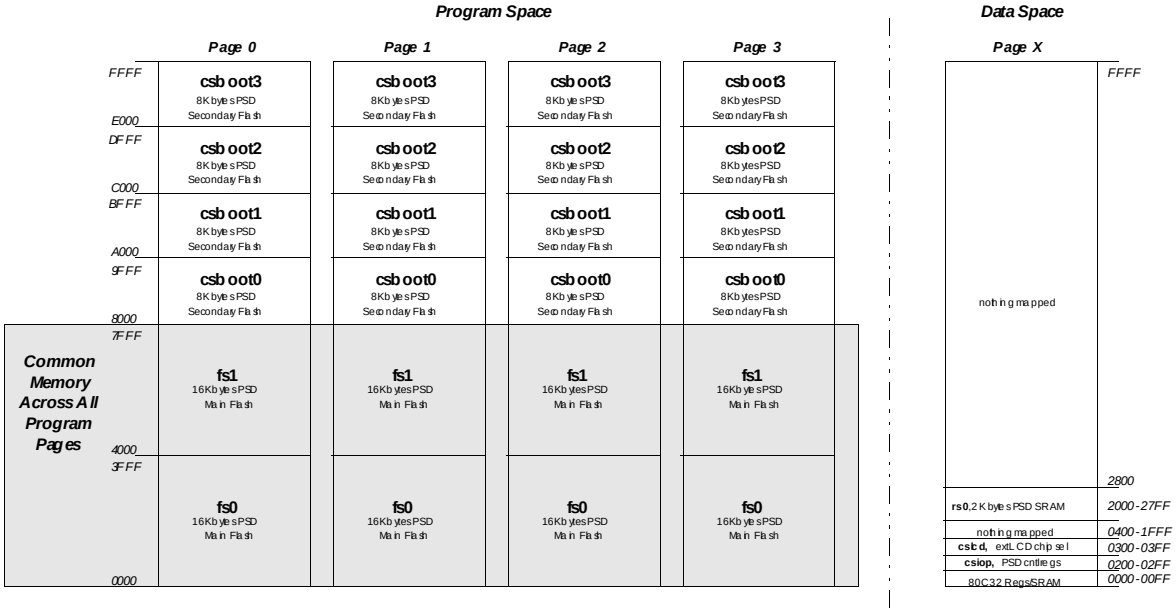


Figure 10 – Memory Map just after 80C32 sets swap = 1.

IAP loader code is swapped away, main PSD flash takes its place.VM reg = 12, unlock = 0

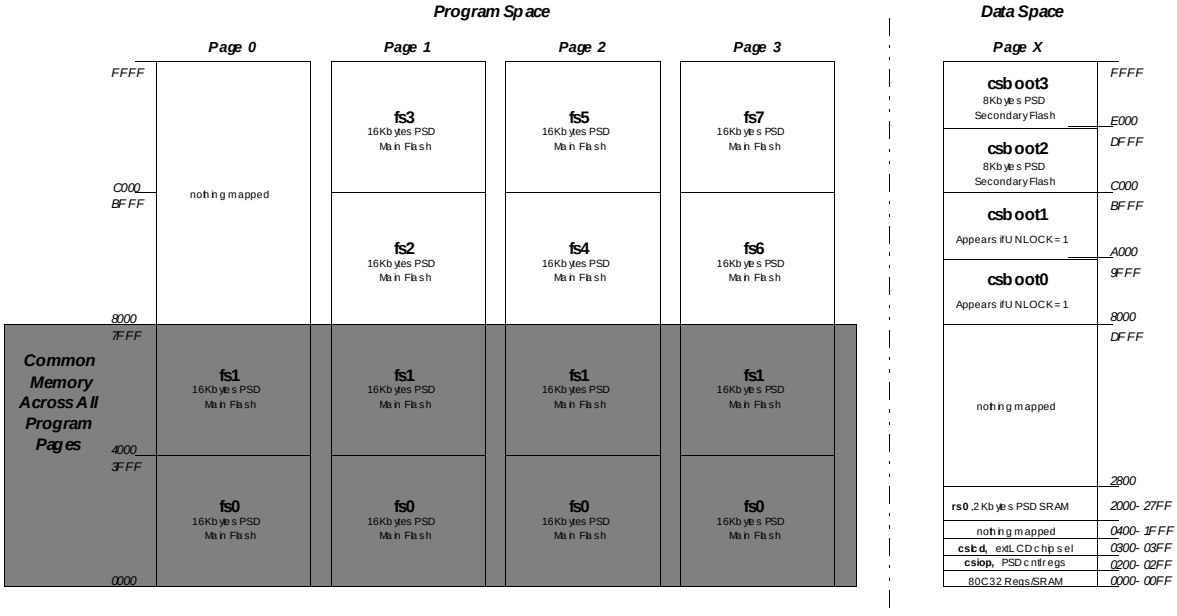


Figure 11 – Memory Map just after 80C32 writes 0C hex to PSD VM register.

Secondary PSD flash memory moves to Data space. swap = 1, unlock = 0

In this final configuration, the 80C32 has available:

- 32 Kbytes of main flash memory in the common area (0000h-7FFFh)
- 96 Kbytes of main flash across three pages of upper memory (8000h-FFFFh)
- 2 Kbytes of SRAM in addition to the SRAM that resides on the 80C32
- 16 Kbytes of secondary flash for general data storage
- 16 Kbytes of secondary flash for IAP loader code.

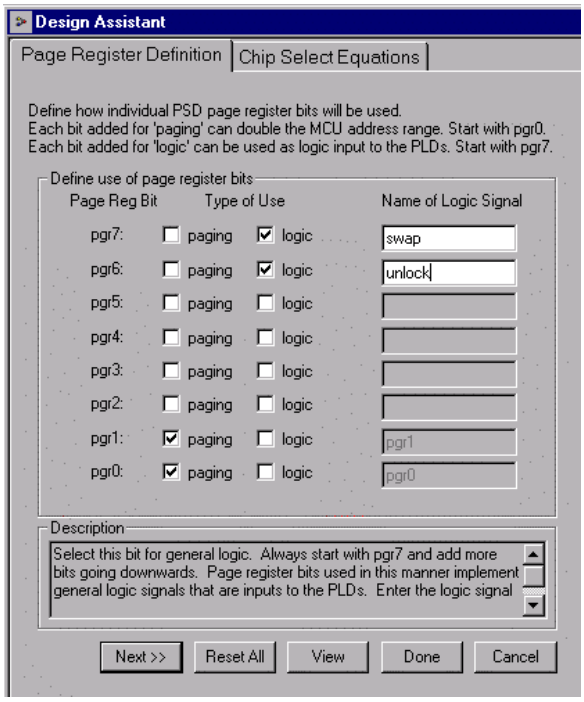
Each time this 80C32 system gets reset or goes through a power-on cycle, the PSD presents the memory map of Figure 8 to the MCU, and the boot sequence is repeated.

> Note: When the 80C32 is executing code from the secondary PSD flash memory (csboot0 and csboot1), and then it sets the swap bit, it is very important that the 80C32 firmware linker has set up "synchronized" code in the segment of main PSD flash memory that replaces the secondary PSD flash memory. This is necessary to create seamless MCU operation during the actual swap of memory since the 80C32 is completely unaware that there is a swap going on. It just continues to fetch opcodes and operands during the memory swap. This requires that the operands and opcodes in main PSD flash that follow the MCU instructions that actually set the swap bit in the secondary PSD flash, are continuos. This means that the remainder of the instructions to complete setting the swap bit is present in main PSD flash memory so there is continuos operation throughout the memory swapping process. To see example 80C32 code for this, go to the main flow diagram, then 'Generate C Code', then 'Coded Examples', then 'DK-900'.

5.2 PSDsoft Express Design Entry

To implement the advanced memory maps of Figures 8 - 11, invoke PSDsoft Express, open the project "page32" from the second design example. Now pull down the menu 'Project' from the top of the screen, and select 'Save As'. For this third design example, save the second project under the new name "advanc32". Now click on the 'Pin Definition' box in the design flow diagram. Click "OK" to get to the 'Page Register Definition' screen since no pin assignment needs to be changed for this third design.

You will need to define two additional PSD page register bits to be used for logic. The two page register bits that were used for memory paging in the second design example stay just as they are. Define page register bits for logic as shown below, labeling one bit "swap" and the other bit "unlock":

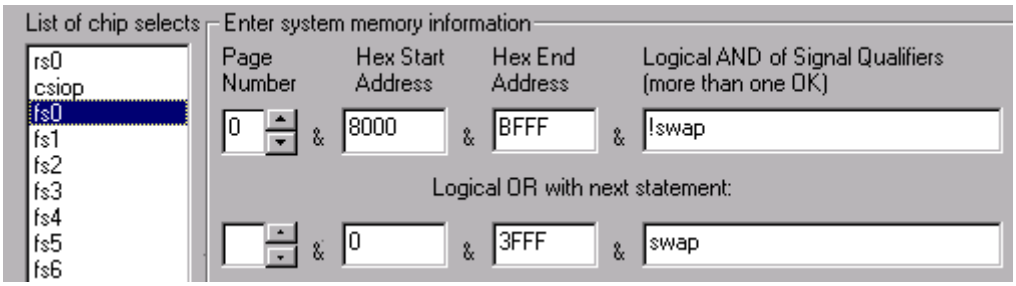


Click ‘Next >>’.

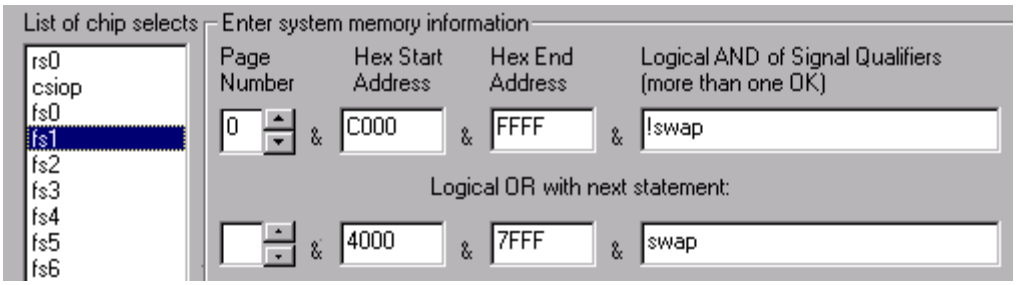
The chip-select equations for PSD SRAM (rs0), PSD control registers (csiop), and the external LCD module (cslcd), and most of the internal PSD memory segments do not change from the second design example. Only chip-selects for main PSD flash memory segments fs0 and fs1, and the secondary PSD flash memory segments csboot0 – csboot3 need to change for this third design because they are affected by memory swapping.

These internal memory chip-selects must be qualified with the page register bit “swap” as shown below. The secondary PSD memory segments, csboot0 and csboot1, must be additionally qualified by “unlock” to prevent the MCU from inadvertently writing to IAP boot and loader code. The following illustrates how the chip-selects will look when you enter their definition:

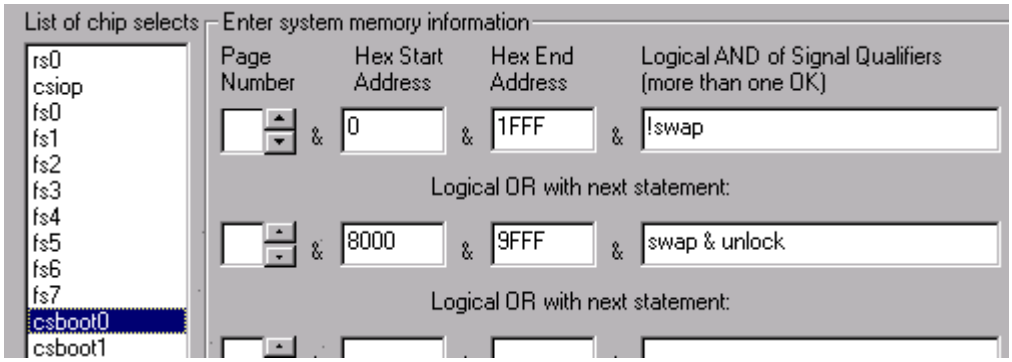
fs0:



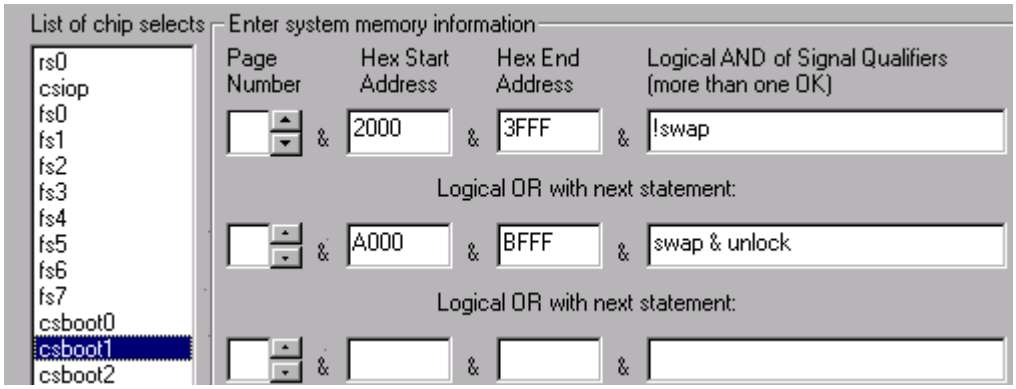
fs1:



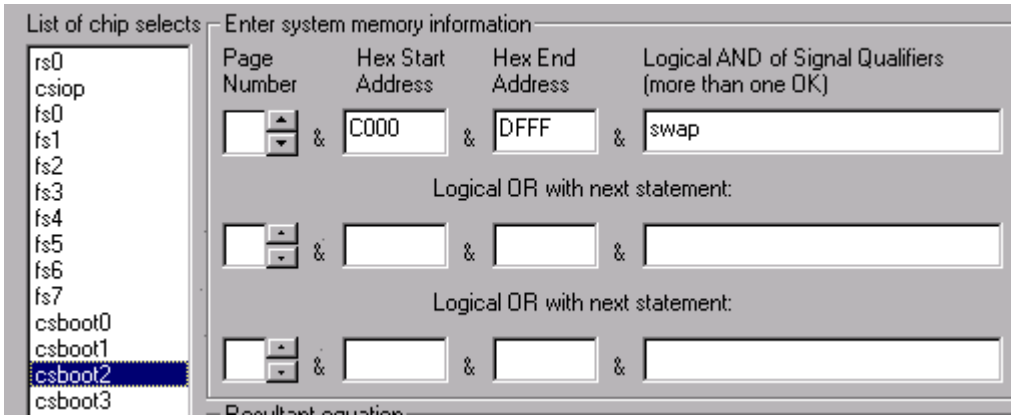
csboot0:



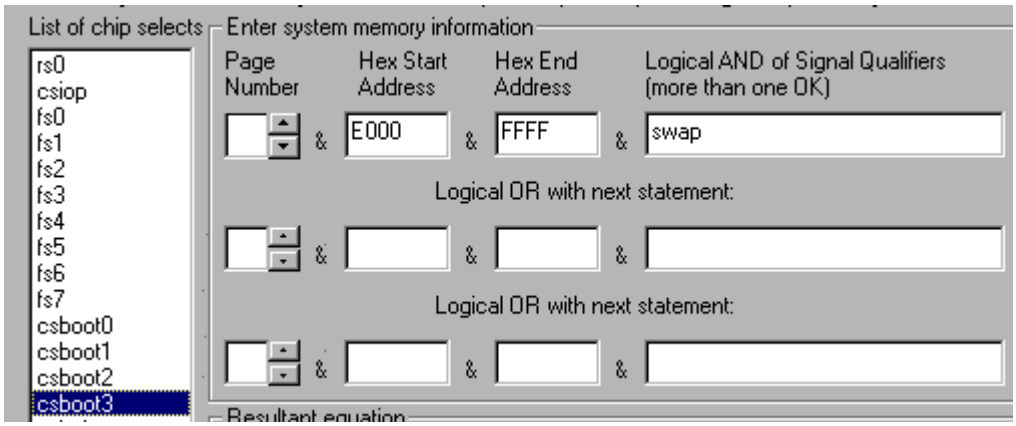
csboot1:



csboot2:



csboot3:



Notice that these PSD physical memory segments can appear in more that one MCU address space depending on the memory page and the “swap” and “unlock” qualifiers. Now the memory maps of Figures 8 - 11 have been implemented. Click ‘Done’ and you should see the main flow diagram.

Click the “Merge MCU Firmware” box in the design flow diagram. You will see an informational dialog box pop up that indicates memory paging is used and that the firmware file(s) you specify should be set up to handle paging. Click OK, since for this design example the firmware that would run the IAP process example is not paged. When the MCU is executing IAP code, it resides in csboot0 and csboot1 of the PSD and is active on all pages (independent of what memory page is selected).

> Click the “More Info” button in Step 1 of the Merge Firmware screen if your future 80C32 system design will execute code from different pages, and that code will be programmed into the PSD with a device programmer (for example, you specify filename(s) in this screen that go to PSD memory segments that are paged).

Now specify the name of the 80C32 firmware file to place into the secondary flash memory segments csboot0 and csboot1. This can be any firmware file that you create to implement IAP with paging and swapping.

> If you have purchased the DK-900 kit you can use its firmware for IAP. The DK-900 kit also includes a PC Windows program (Win95, Win8, Win NT) called PSDload that will download (IAP) the PSD913F over a serial (RS-232) cable. It works directly with the memory map of this third design example. The DK-900 is $99 USD and can be purchased from www.waferscale.com with a credit card.

No firmware filename needs to be designated for the main PSD flash segments (fs0 – fs7) since they will be programmed by the 80C32 during IAP. Click “OK” in the merging screen to create a composite object file for programming. You are now ready to program your PSD913F2. See section 3.2.7.

6 Conclusion

These examples are just three of an endless number of ways to configure the *Easy*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 enhance 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 Express development environment to minimize your effort to design a flash 80C32 system capable of ISP and IAP.

7 References

1) PSD9XXF Family Data Sheet
2) Application Note AN054 for detailed use of the JTAG channel