AN1385
APPLICATION NOTE

PSD913F2 / 68HC11 Design Guide

CONTENTS

*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 68HC11 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 68HC11 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 68HC11 hardware. The second design illustrates the use memory paging. The third covers enhanced features of PSD In-System-Programming, including memory paging and segment swapping. 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 config-

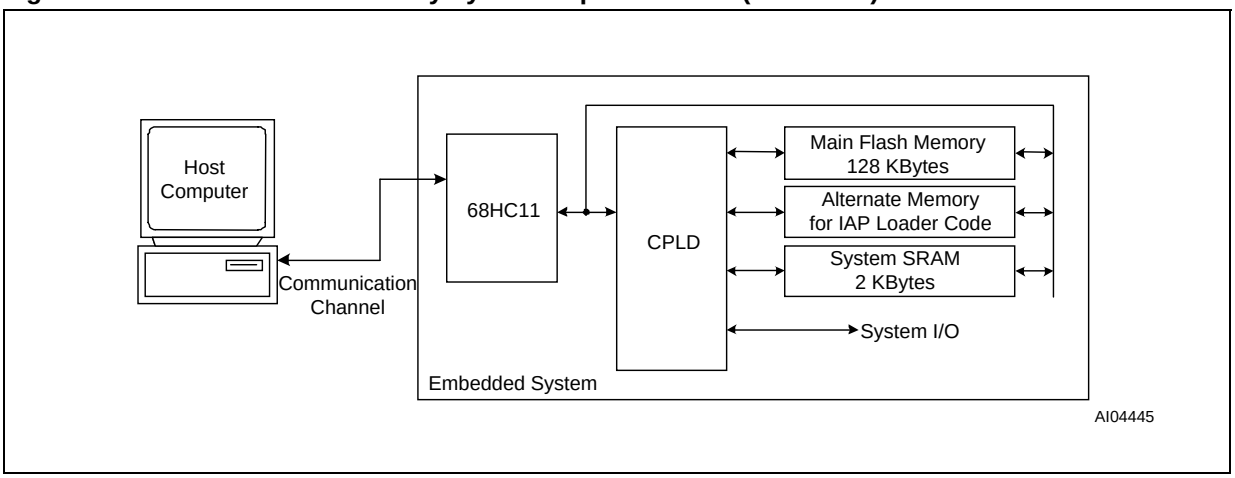
uration and programmable logic portions of the PSD. ISP can program all areas of the PSD.

The Generic Problem 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 68HC11 supports will do (modem, SPI, CAN, J1850, etc.). The 68HC11 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 68HC11. 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 Memory system with external memory is shown in Figure 1 .

Figure 1. Embedded Flash Memory system capable of IAP (5 devices)



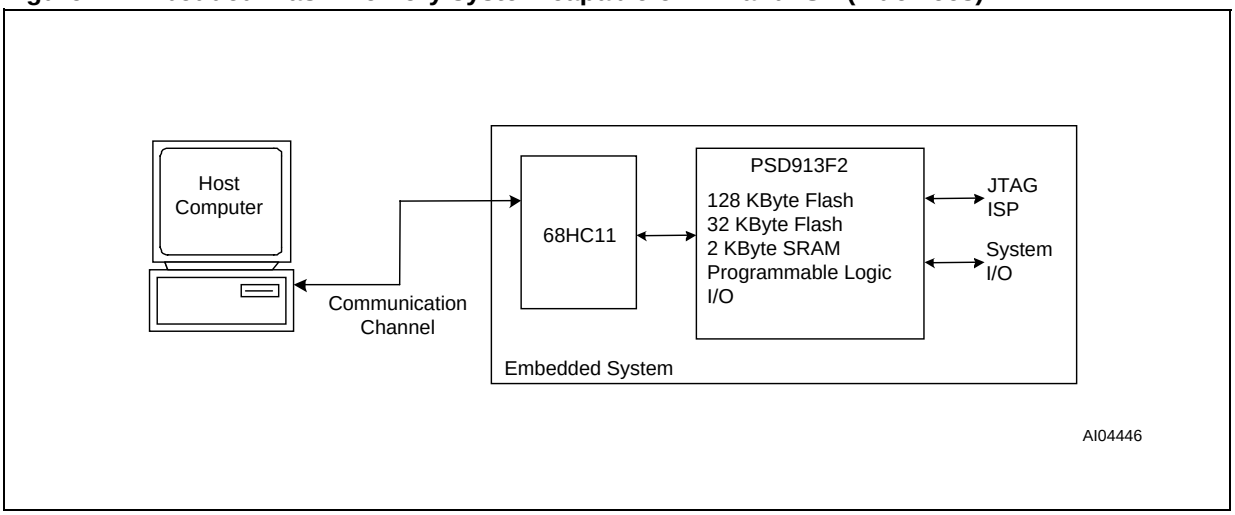
A Common Solution

For IAP, some 68HC11 designers will use the fixed boot-loader feature of the 68HC11 UART to download executable code into SRAM (either the small on-board 68HC11 SRAM or an external SRAM chip) then 68HC11 execution jumps to that SRAM to execute the remainder of the download process for programming the main Flash Memory. This can be a cumbersome and error prone exercise using relocatable code in volatile memory, which is difficult to debug and is vulnerable to power outages. Additionally, this method restricts the designer to use a UART to implement IAP.

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 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 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.

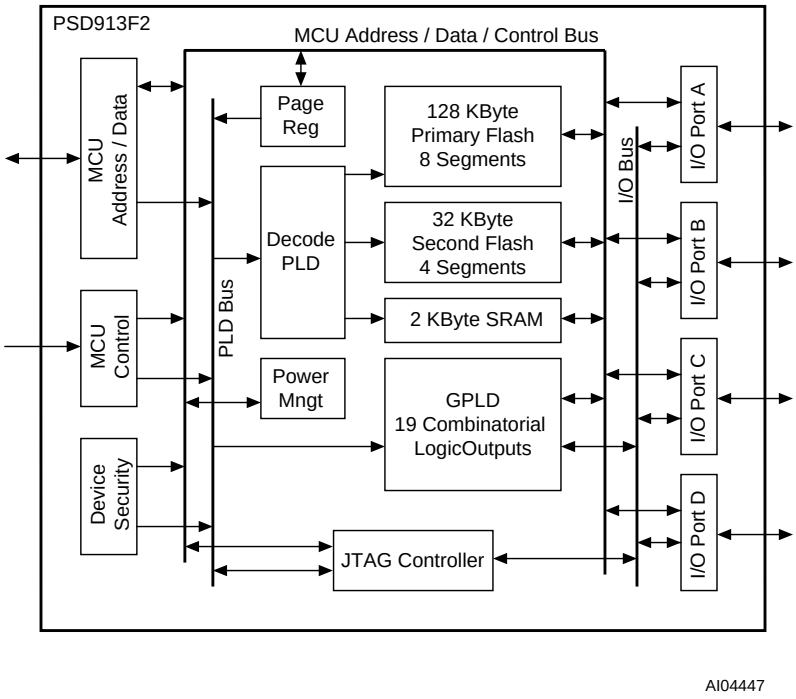
Figure 2. Embedded Flash Memory system capable of IAP and ISP (2 devices)



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 ST's FlashLINK JTAG cable and PSDsoft Express development software. No 68HC11 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. PSDsoft Express is free.

Let's take a quick look inside the *Easy*FLASH 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

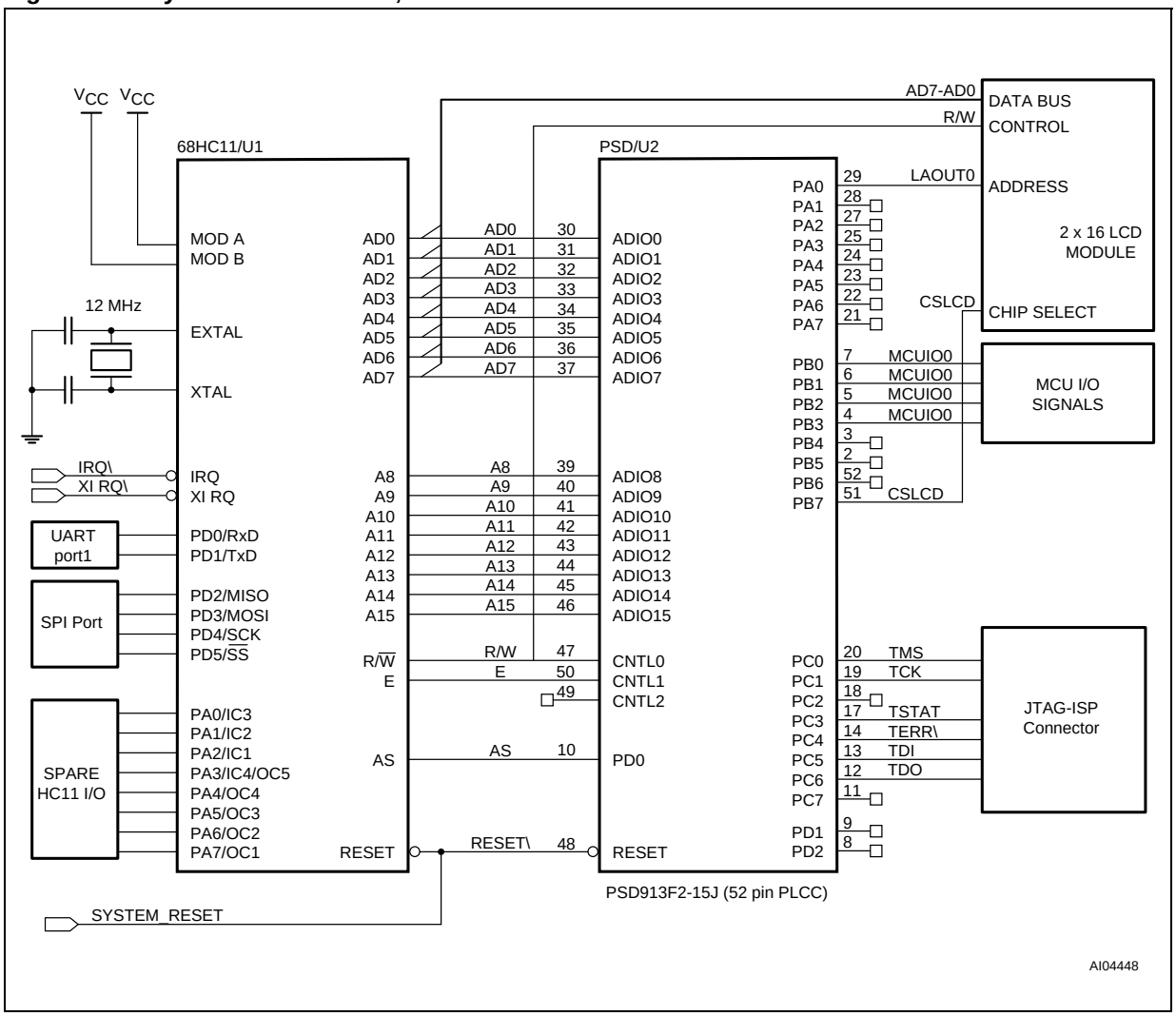


PHYSICAL CONNECTIONS

Connect your 68HC11 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 chip-select for the LCD, four miscellaneous MCU controlled I/O signals, and a six-pin JTAG-ISP interface. There are 15 unused PSD I/O pins shown (should use pullups to V_{CC} with 100KΩ resistor or tie to GND if not used in your design).

Figure 4. - Physical Connections, 68HC11 and PSD913F2



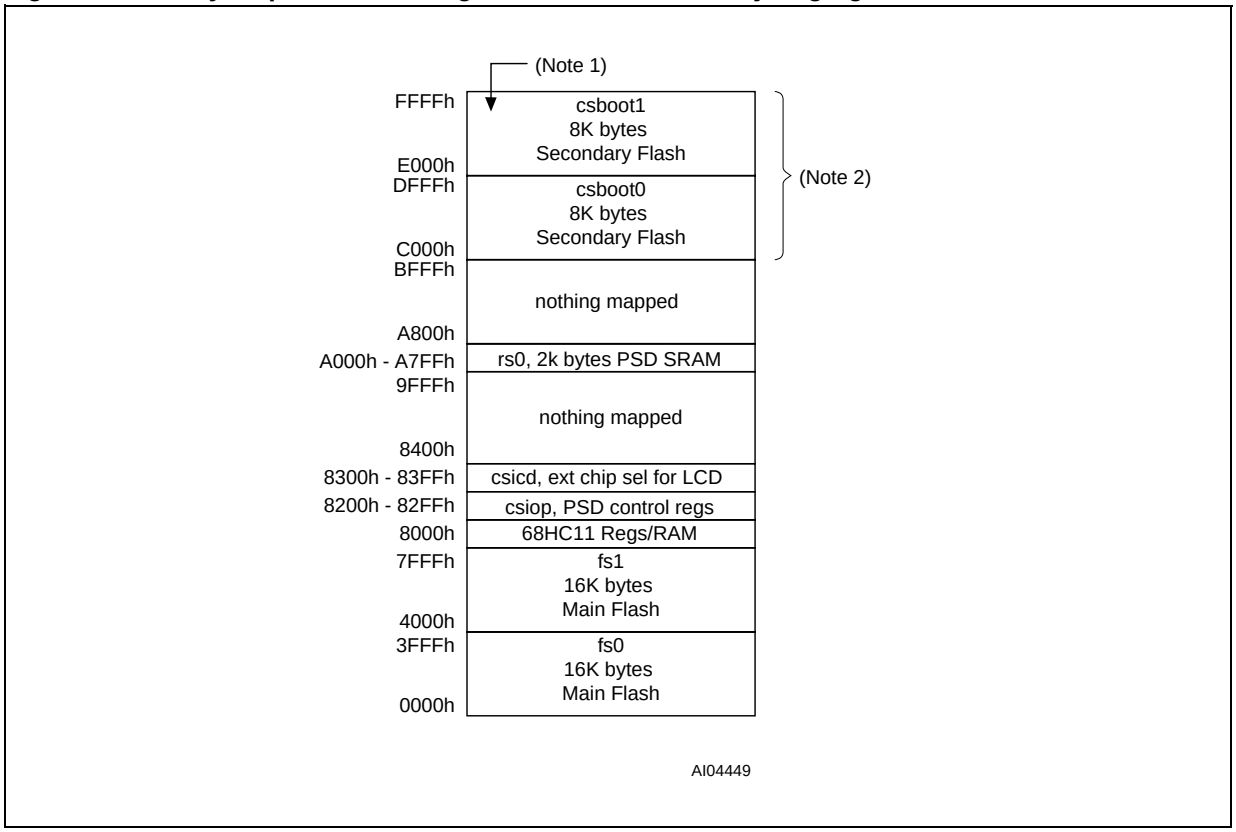
FIRST DESIGN EXAMPLE - IAP WITH NO MEMORY PAGING

The first design example will outline the steps to get a Flash Memory 68HC11 system up and running quickly. No memory paging is used. 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

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, 16 Kbytes of secondary Flash Memory, and 2 Kbytes of SRAM. See the 68HC11 memory map in Figure 5.

Figure 5. Memory Map for First Design - IAP with no Memory Paging



Note: 1. 68HC11 boots from the reset vector stored here.
2. IAP loader code gets programmed here by JTAG-ISP or a convetional programmer tool.

The nomenclature fs0, fs1, csboot0, csboot1, 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 68HC11 may perform IAP by executing from the secondary

Flash Memory segments (csboot0 and csboot1) while erasing and programming the main Flash Memory segments (fs0 and fs1). The secondary Flash Memory is initially programmed though JTAG-ISP or other programming devices with firmware containing the following:

– 68HC11 reset vector and initialization routines
– 68HC11 interrupt vectors and service routines
– I/O management routines
– IAP loader code.

At power-on or after reset, the 68HC11 boots from secondary Flash Memory, runs a checksum of the main Flash Memory, programs and verifies main Flash Memory via the UART if necessary, then execution jumps to main Flash Memory.

Note: The memory map of Figure 5 requires that the placement of internal 68HC11 registers and RAM be relocated from their default base address 0000 hex to a new base address, 8000 hex. This will make your system compatible with memory paging should you decide to add paging later. There is a special register inside the 68HC11 (the INIT register) that facilitates this move. The INIT register must be written with the value of 88 hex within the first 64 MCU oscillator clocks after power up. See the 68HC11 reference manual from Motorola for details on the INIT register.

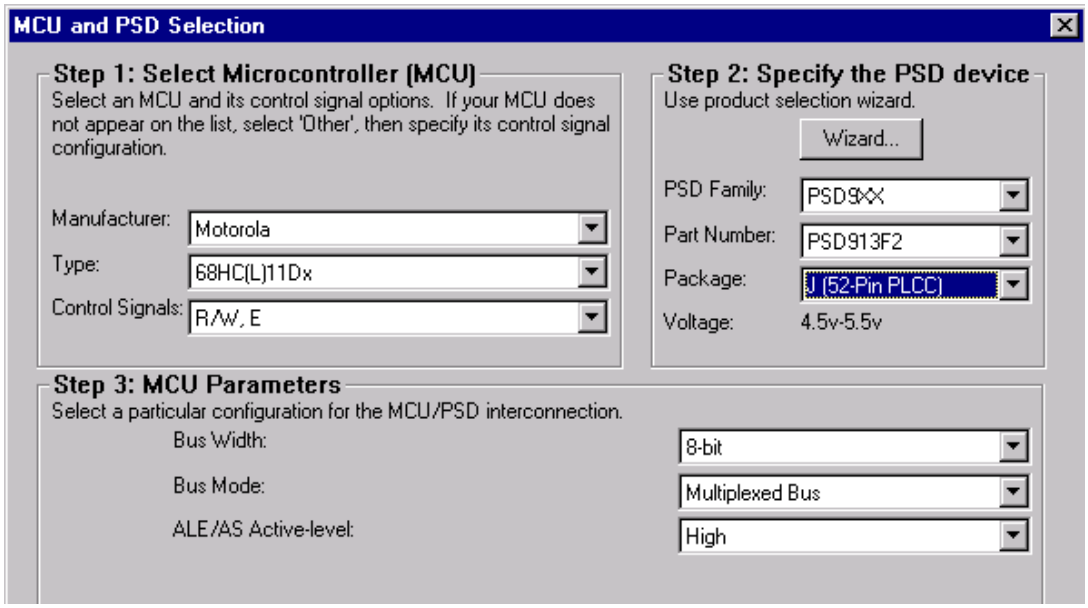
PSDsoft Express Design Entry

Highlights of 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.

Invoke PSDsoft Express and set up your project

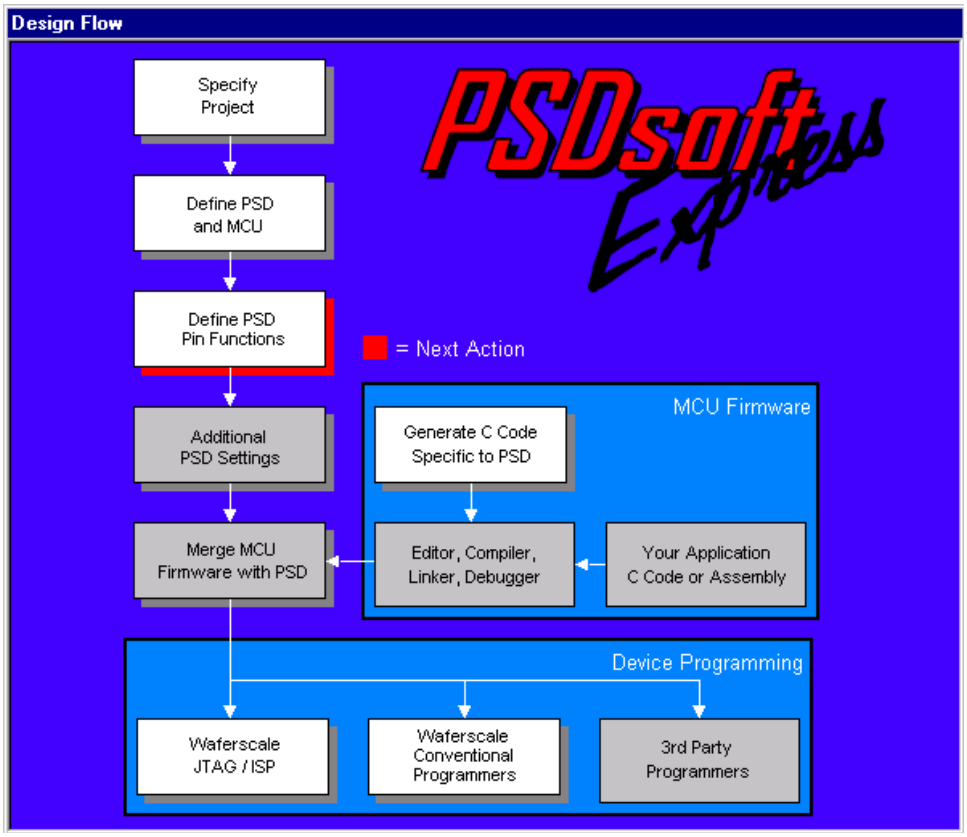
– Start PSDsoft Express.
– Create a new project.
– Select your project folder and name the project (in this example, name the project 'simple11' in the folder PSDexpress\my_project).
– Select an MCU (in this example, we're using a Motorola 68HC11D0, so chose 68HC(L)11Dx).
– Select a PSD913F2 and a 52-pin PLCC package.

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.



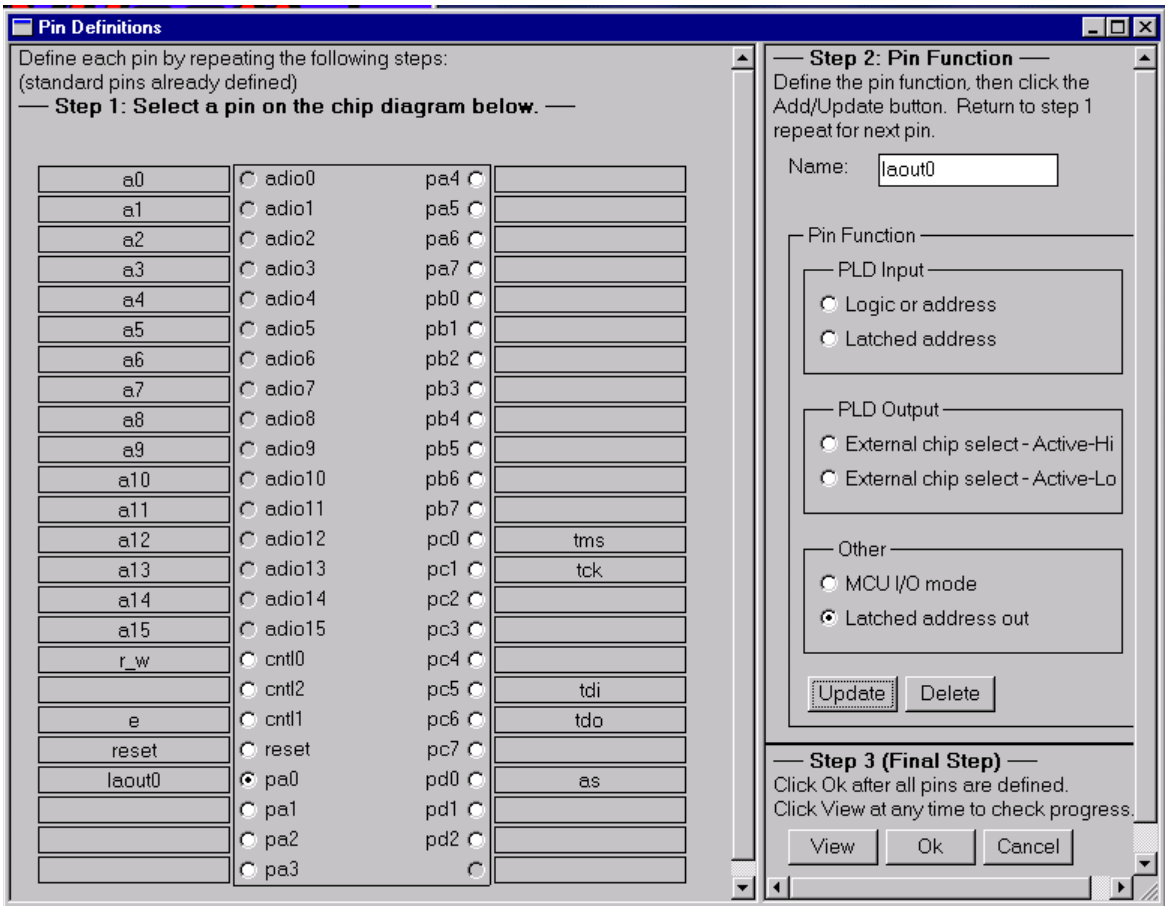
PSD Pin Definition

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 68HC11 are already defined for you. You only have to define the remaining pins. For this example, we'll configure the PSD to use:

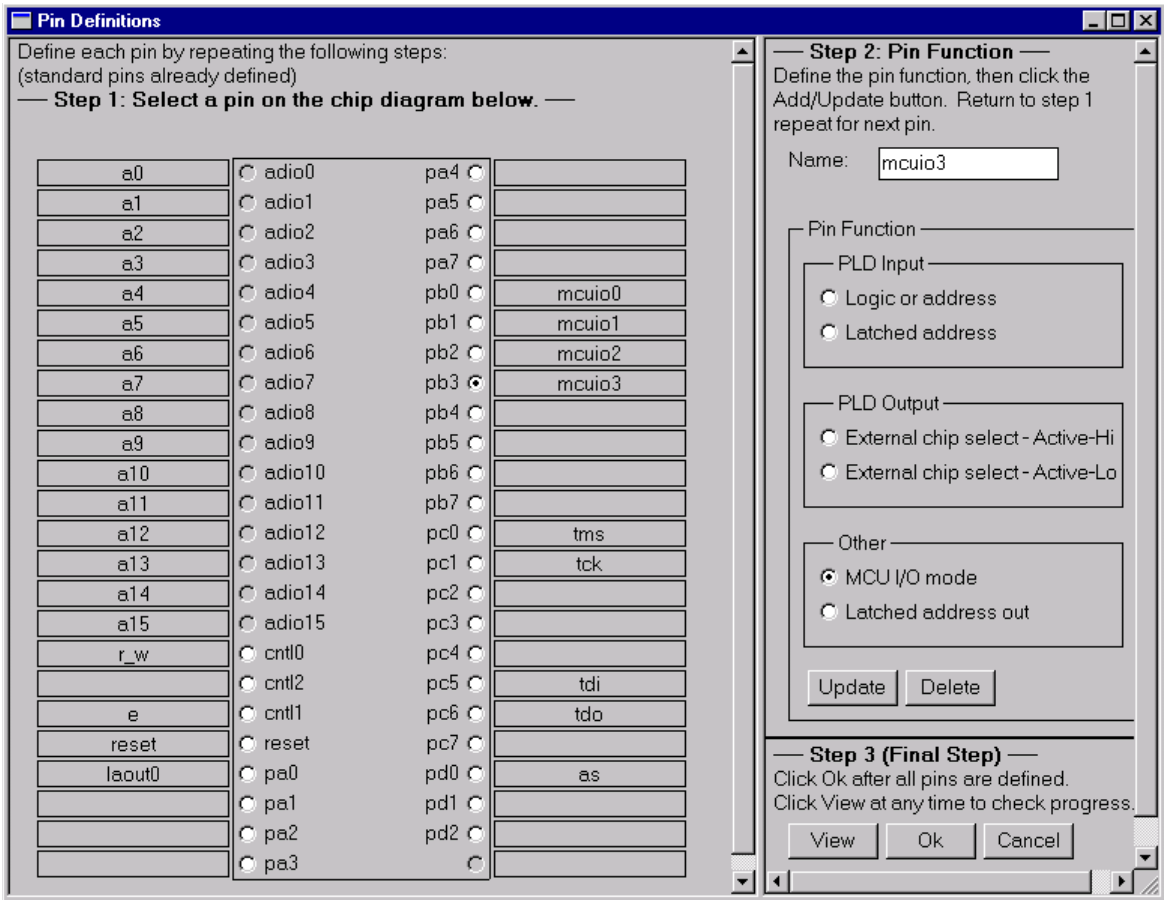
– One pin on Port A to drive a single demultiplexed MCU address line out to an LCD module.
– Four pins on Port B as general purpose MCU I/O
– One pin on Port B as a chip-select output for the LCD module.
– Six pins on Port C as JTAG interface signals for ISP

Click on pa0, type in signal name of "laout0", click on 'Latched address out' in the 'Other' category, then 'Add' or 'Update'. This will produce a demultiplexed 68HC11 address line output from the PSD on pin pa0 that should be routed to the LCD module on the circuit board as shown in the schematic of Figure 4.

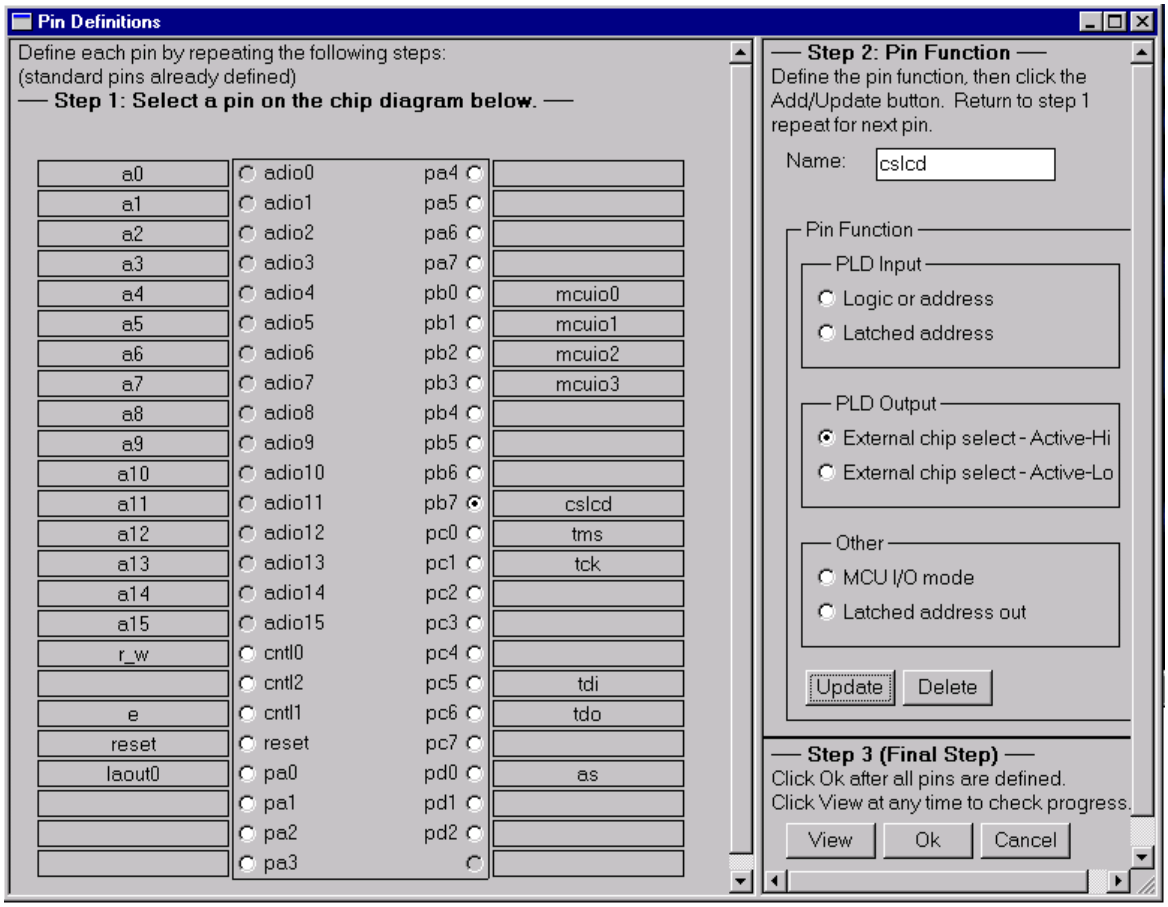
This is what the screen should now look like:



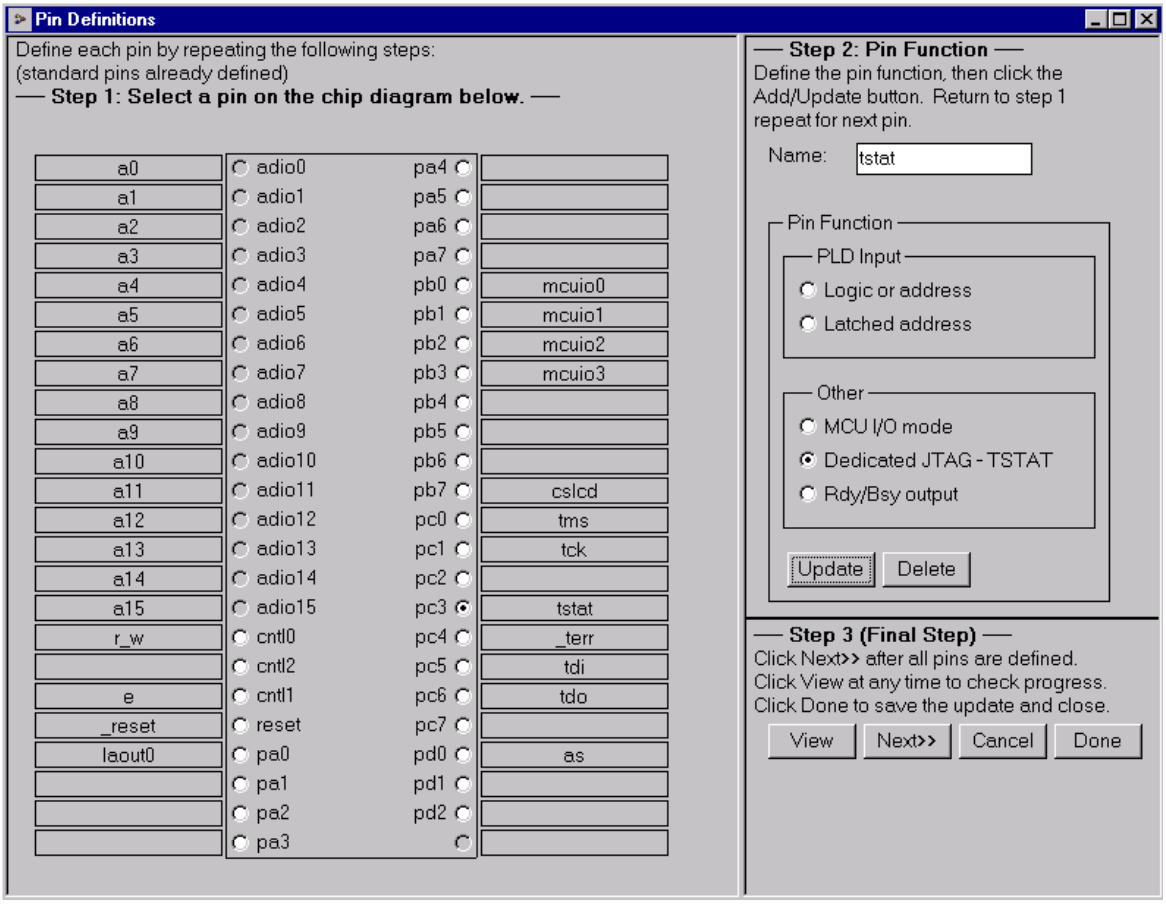
Now click pb0, name the signal "mcuio0", click 'MCU I/O mode' in the 'Other' category, and click 'Add' or 'Update'. Repeat for pins pb1, pb2, and pb3, giving names mcuio1, mcuio2, and mcuio3 respectively. This creates four I/O pins on port B that can be set, cleared, and read by the 68HC11 accessing PSD control registers at runtime. These PSD control registers reside at various address offsets from the base address designated "csiop". In a later section you will see how to place csiop within your system memory map. The screen should now look like this:



Next click pin pb7, name it "cslcd", choose 'External chip-select - Active-Hi'. then click 'Add' or 'Update'. This designates pin pb7 as a chip-select output for the LCD module. The screen should have the following look:



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 Application Note *AN1153* to learn why it is faster with six pins). To do this, 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 Pin Definition.

System Memory Map: PSD Page Register and Chip-select Definitions

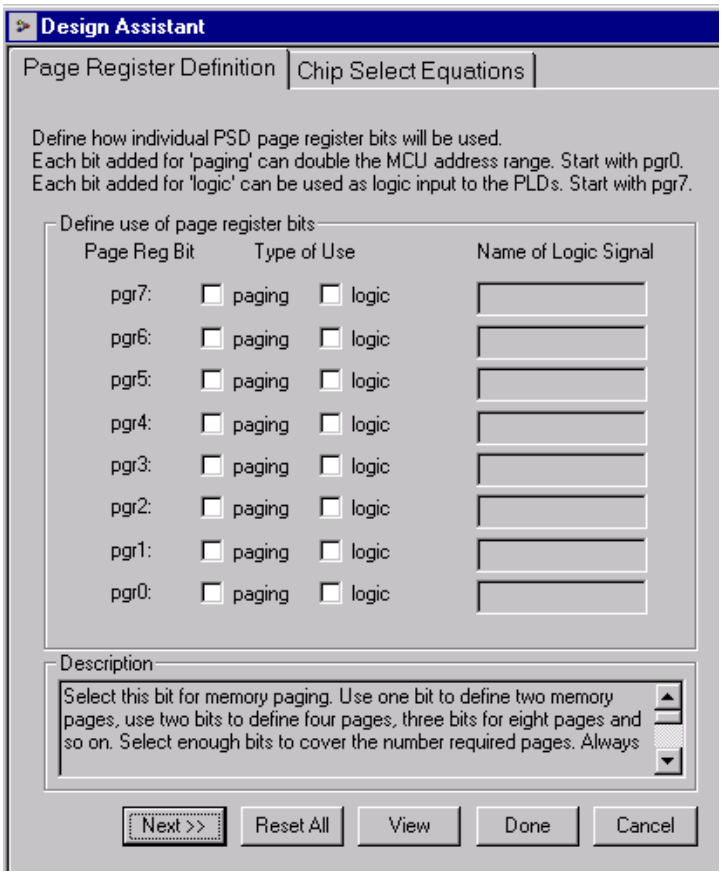
Now that the PSD pins are defined, you will need to define the system memory map. This is accomplished by defining all the chip-selects in the system (both internal to the PSD and external chip-selects), and also defining the function of the PSD page register.

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 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, csboot1, 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.

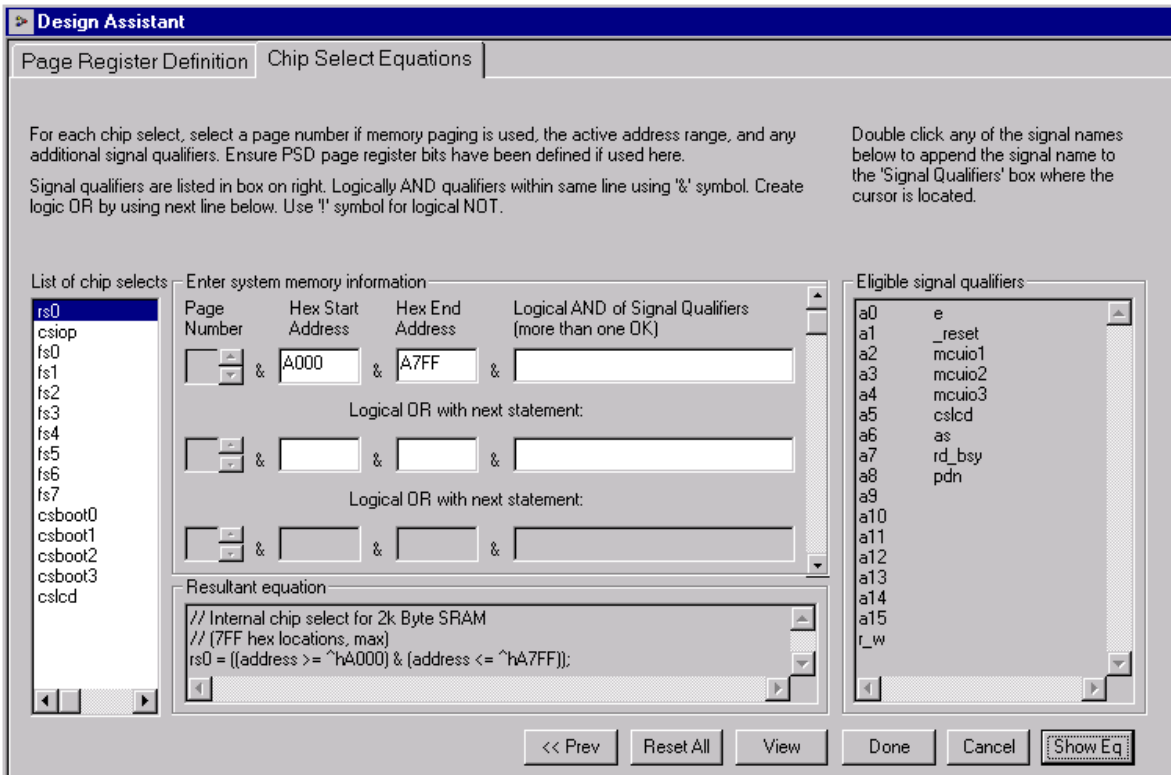
In many 68HC11 system designs, memory paging is used to address more than 64 Kbytes of address space. However, for this simple design, no paging is used, so no PSD page register bits need to be defined.

You should see the following Page Register Definition screen appear. Simply 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 the PSD SRAM, which is "rs0". Then enter start and stop MCU addresses to match the memory map of Figure 5. Additional signal qualifiers (68HC11 control signals e, r/w, as) are NOT needed for internal PSD memory chip-selects as this is taken care of in silicon. Your screen should match the one shown below:

rs0:



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

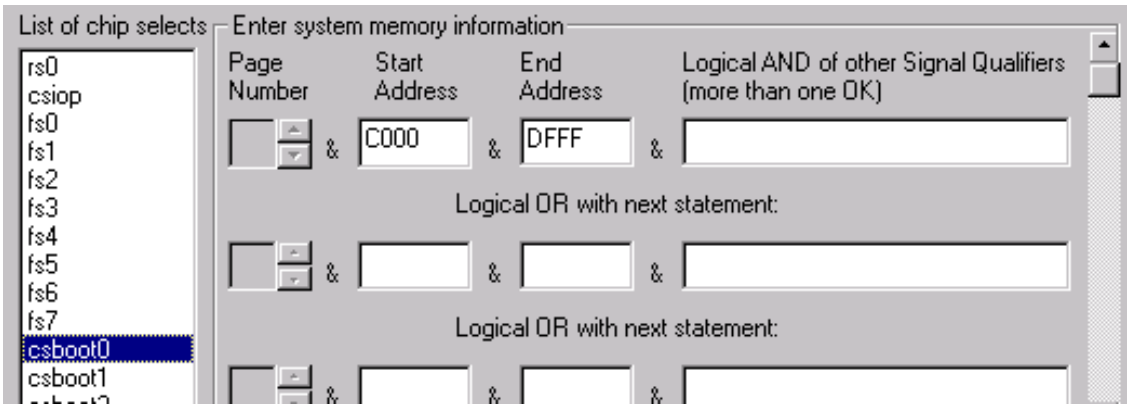
csiop:

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 and csboot1. 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:

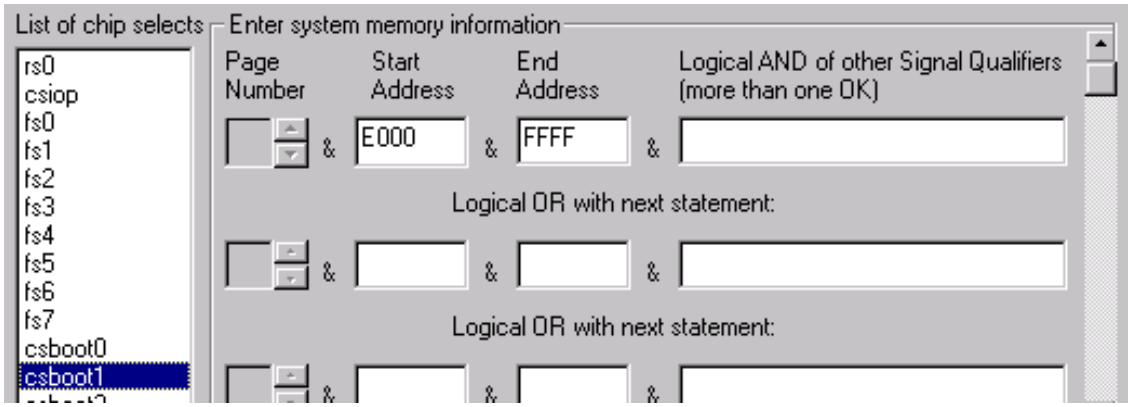
fs0:

fs1:

csboot0:

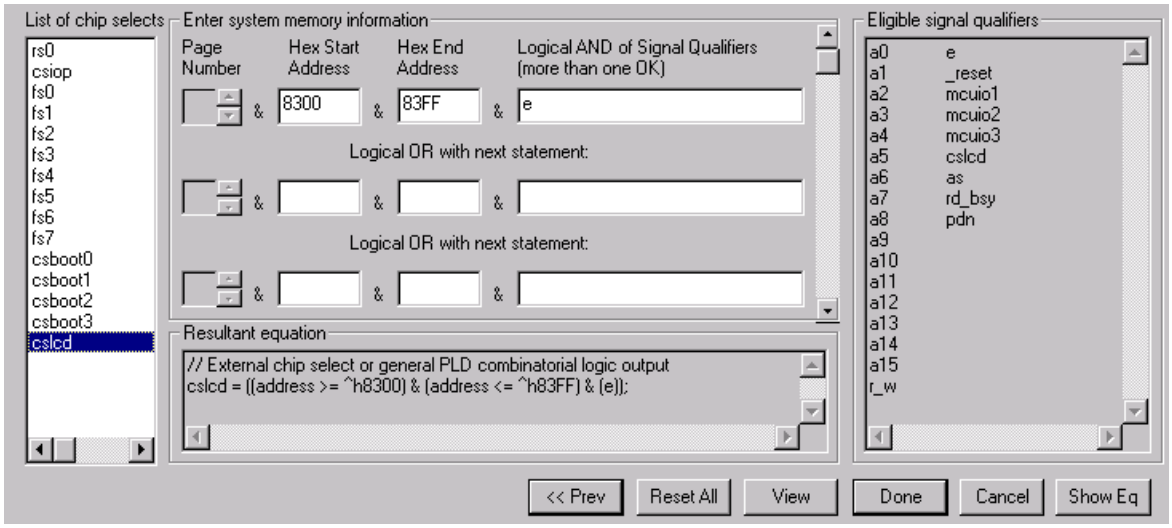


csboot1:



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 a qualifier, meaning that this logic signal is true only for a given MCU address range AND only when another signal is active. In this case "cslcd" is true only when the MCU presents an address in the range of 8300 to 83FF hex AND when the 68HC11 control signal "e" is true. Notice that the signal "e" is in the list of Available Signal Qualifiers. Be sure to qualify "cslcd" with the signal "e" as shown below, then click 'Done':

cslcd1:



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 USER-CODE value and set sector protection on PSD Non-Volatile memory segments.

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 68HC11 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 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.h¾ANSI-C .....header file to define particular PSD registers
- map913F2.h¾ANSI-C .....header file to define locations of system memory

elements (Main and Secondary Flash Memory, 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 68HC11 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.

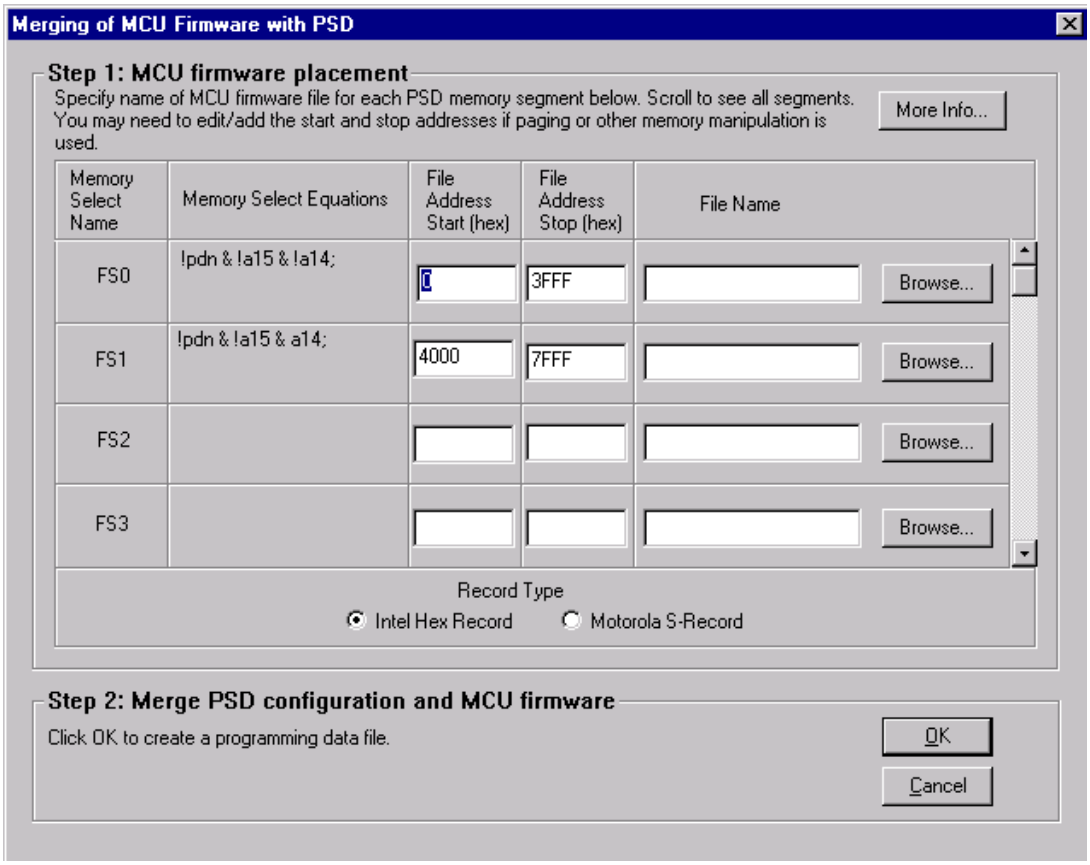
Merging MCU Firmware

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 68HC11 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 simple11.obj for this design example.

During this merging process, PSDsoft Express will input firmware files from your 68HC11 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 68CH11 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 68HC11 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 68HC11 linker has created your firmware file(s). For each PSD chip-select in which you have specified a

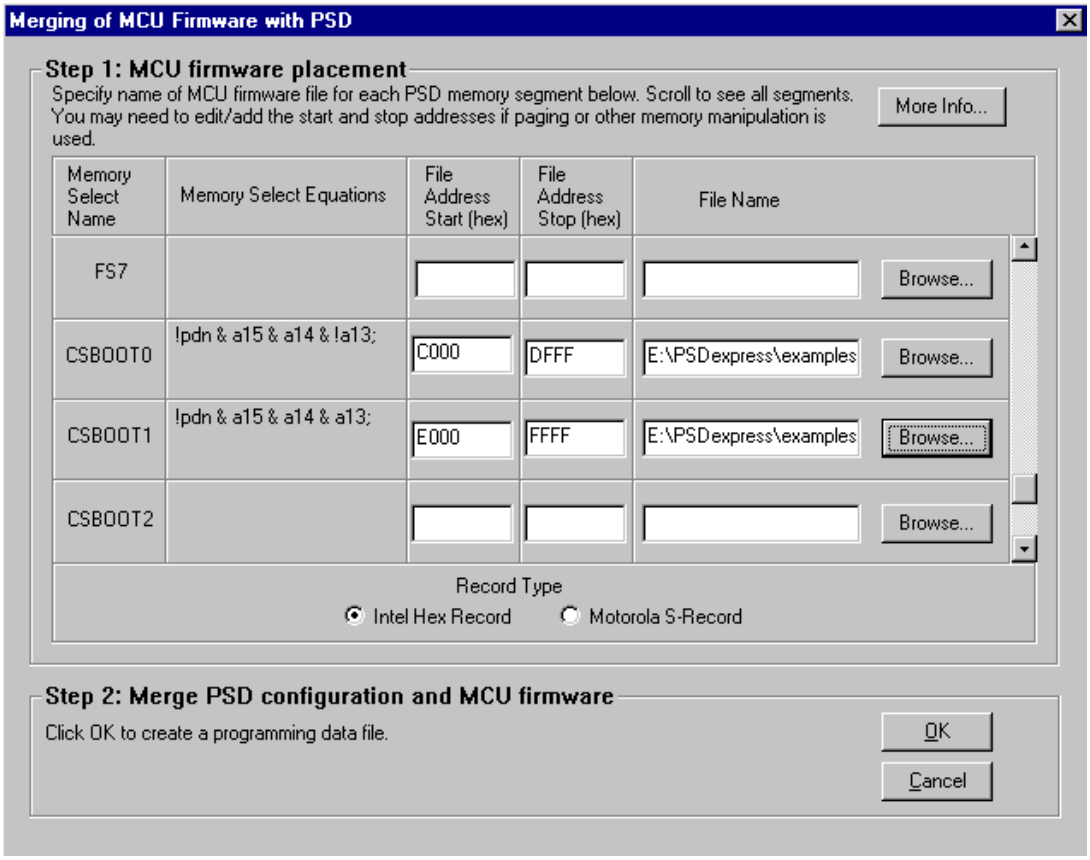
firmware file name, PSDsoft Express will extract firmware from that file only between the specified start and stop addresses, and ignore firmware outside of the start and stop addresses.

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 the 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_hc11.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_hc11.hex between MCU addresses C000 and DFFF and place it in PSD memory segment csboot0. It will also extract any firmware that lies inside the file isp_hc11.hex between MCU addresses E000 and EFFF and place it in PSD memory segment csboot1. Click OK to generate the composite object file, simple11.obj.

Note: The file isp_hc11.hex will run on the DK68HC11 development board from ST, 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 16 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 to for advanced tasks, including IAP of main PSD Flash Memory.

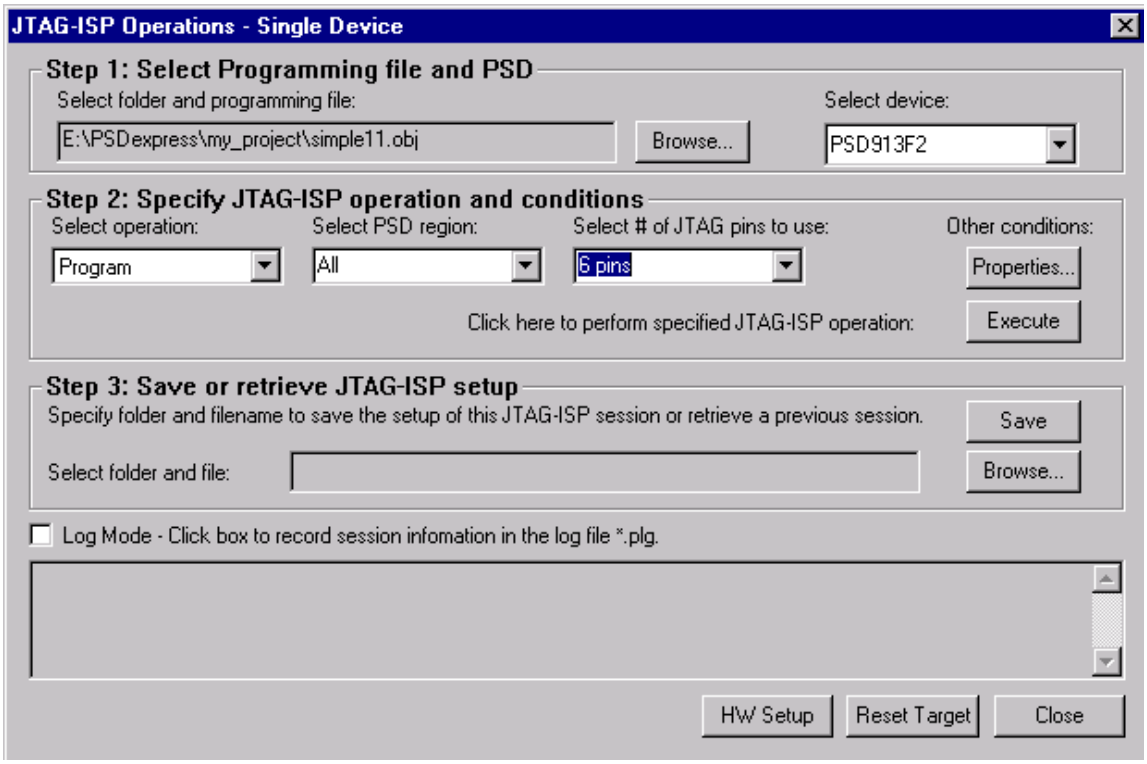
Programming the PSD

The simple11.obj file can be programmed into the PSD by one of three ways:

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

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 *AN1153* 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.

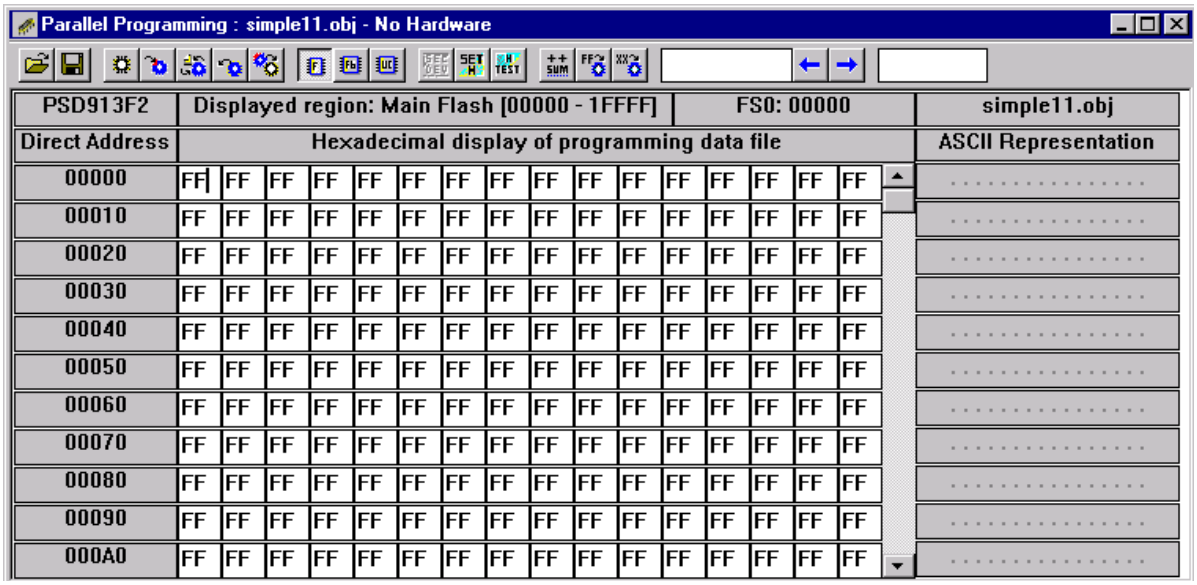
There are optional choices available when the 'Properties.." button is clicked. One choice includes setting

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.

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 simple11.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 68HC11 firmware within physical memory of the PSD. For this design example, click on the secondary PSD Flash Memory icon "Fb" in the tool bar, then scroll to the end of segment csboot1 to see the 68HC11 reset vector at absolute MCU addresses FFFEh and FFFFh, which translates to direct physical addresses 23FFEh and 23FFFh respectively. To see how all of your 68HC11 absolute addresses translated into direct physical PSD memory addresses, see 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).

SECOND DESIGN EXAMPLE - IAP WITH MEMORY PAGING

The second design example builds upon the first by adding memory paging which allows the 68HC11 to access all of the memory resident on the PSD913F2. The physical connections between the 68HC11 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.

Memory Map

The PSD913F2 provides 128 Kbytes of main Flash Memory, 32 Kbytes of secondary Flash Memory, and 2 Kbytes SRAM. We'll use all of the main Flash Memory, all of the SRAM, and one half of the secondary Flash Memory (16 Kbytes) to hold the IAP bootloader code, 68HC11 interrupt vectors, and common firmware functions. We'll allocate the remaining 16 Kbytes of secondary Flash Memory for general data storage.

Since the 68HC11 cannot address more than 64 Kbytes of address space directly, we will use paging (or banking) to take full advantage of the 162 Kbytes of total memory on the PSD913F2. The PSD has a built-in page register for this purpose. Many MCU cross-compilers support paging today.

Examine the memory map in Figure 6. At power on or reset, the 68HC11 boots from secondary Flash Memory, runs a checksum of the main Flash Memory, programs and verifies main PSD Flash Memory (IAP) via the UART if necessary, then execution jumps to main Flash Memory. The 68HC11 can access all of Flash Memory now since it is paged across four memory pages.

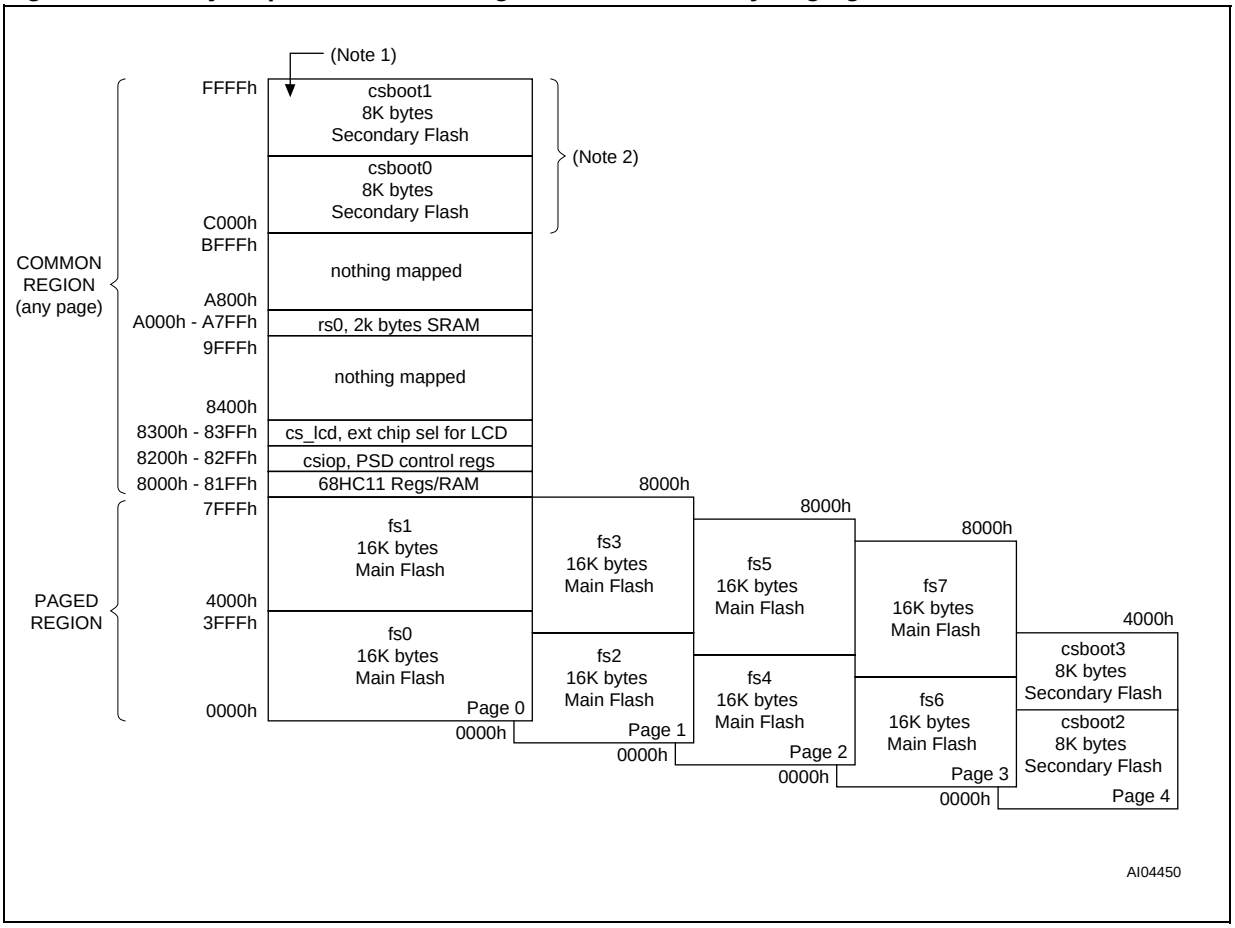
Notice that the memory map is divided into two areas. The upper memory (8000 to FFFF) is a common area, meaning the MCU will have access to this region regardless of what memory page is active. This common area holds the following:

- 68HC11 reset vector and initialization routines
- 68HC11 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.

The lower half of memory (0000 to 7FFF) is the region that will be "paged" or "banked". This paged area of 32 Kbytes allows the 68HC11 to address large amounts of memory. We'll place all 128 Kbytes of main PSD Flash Memory across four 32K-byte pages (memory pages 0 through 3). The remaining half of the secondary PSD Flash Memory will be placed on memory page 4 and used for general data.

To make this division of the memory map possible (that is, common upper half, paged lower half), the 68HC11 initialization firmware must move the internal 68HC11 SRAM and 68HC11 registers from their default location of 0000 hex, to the new location of 8000 hex, as shown in Figure 6. There is a special register within the 68HC11 (the INIT register) that facilitates this move. The INIT register must be written with the value 88 hex within the first 64 MCU oscillator clocks after power up. See the 68HC11 reference manual from Motorola. This move is necessary to "clean-up" the memory map, meaning that the entire region from 0000 - 7FFF is now free to use for paging, once these internal MCU locations are moved up and out of the way to 8000 hex. Also note that the common area (non-paged) had to be located in the upper half of memory (8000h - FFFF) because the 68HC11 boots from high memory (FFFE). Keep in mindthat all of the PSD page register bits are zero at power-up and after a system reset.

Figure 6. Memory Map for Second Design - IAP with Memory Paging

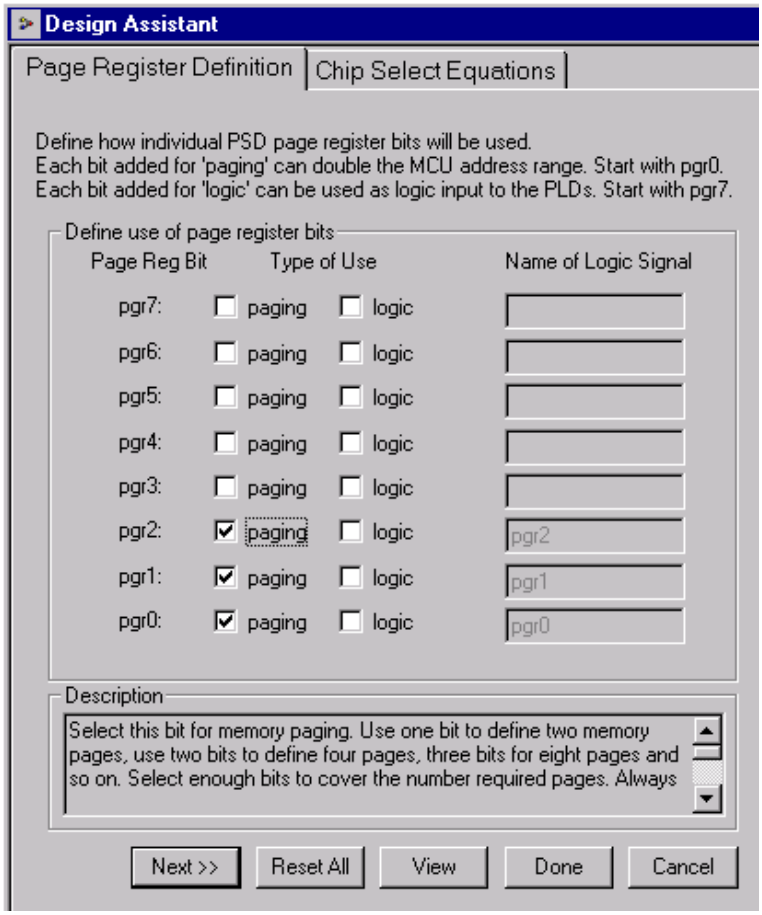


Note: 1. 68HC11 boots from the reset vector stored here.
2. IAP loader code gets programmed here by JTAG-ISP or a convetional programmer tool.

PSDsoft Express Design Entry

To implement the memory map with paging techniques shown in Figure 6, invoke PSDsoft Express, open the project "simple11" 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 "page11". 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 five memory pages used in Figure 6, so you will need to define three PSD page register bits for paging (23 = 8), as two PSD page register bits are not enough (22 = 4). To do so, click on pgr0, pgr1, and pgr2 as shown below.



Click 'Next >>'.

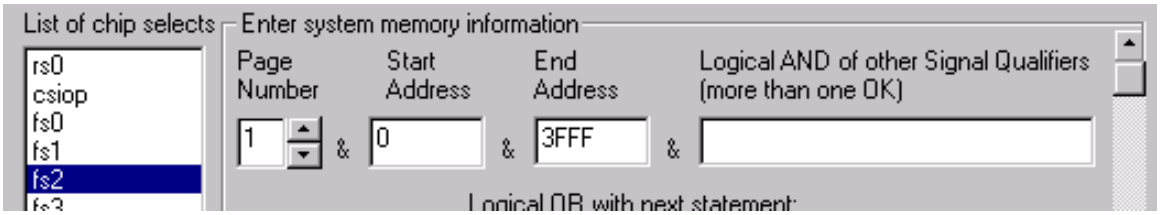
The chip-select equations for PSD SRAM (rs0), PSD control registers (csiop), and the external LCD module (cslcd) do not change from the first design example. Only chip-selects for main PSD Flash Memory and some of the secondary PSD Flash Memory need to be changed for this second design because they are affected by paging.

Define the internal PSD memory chip-select signals to implement the memory map of Figure 66. The following illustrates how the chip-selects will look when you enter their definitions:

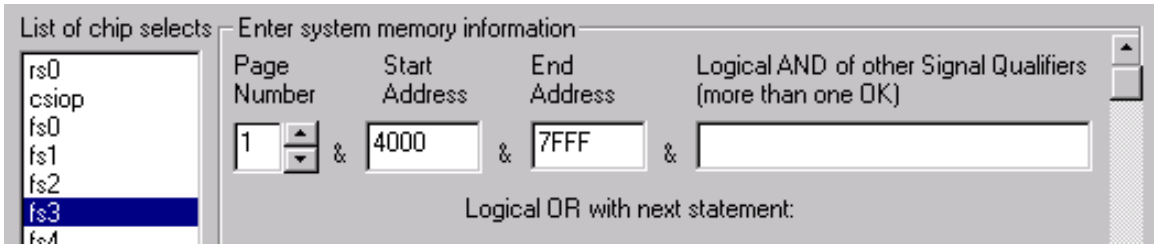
fs0:

fs1:

fs2:



fs3:

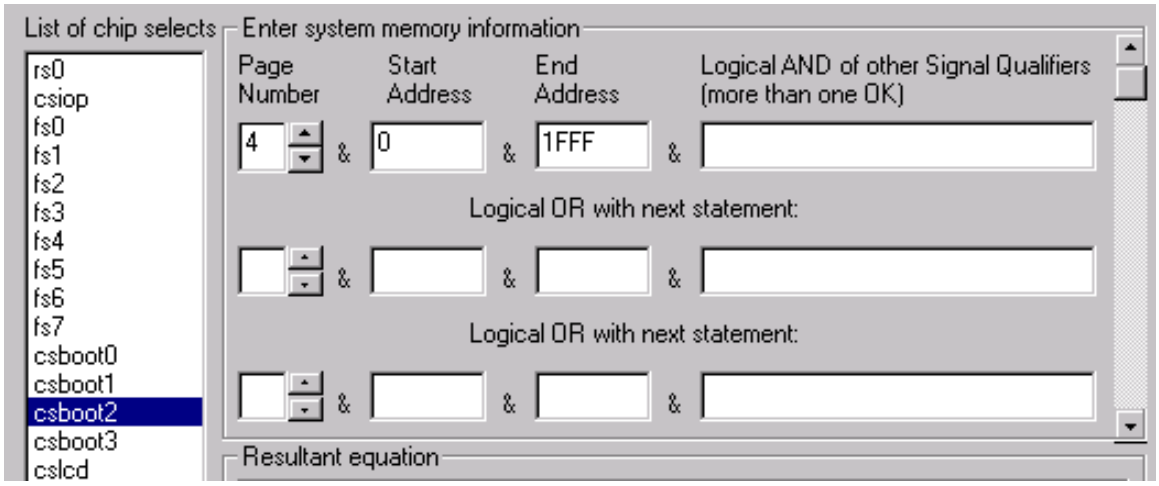


Continue defining fs4 and fs5 on memory page 2, and fs6 and fs7 on memory page 3 per the memory map in Figure 6.

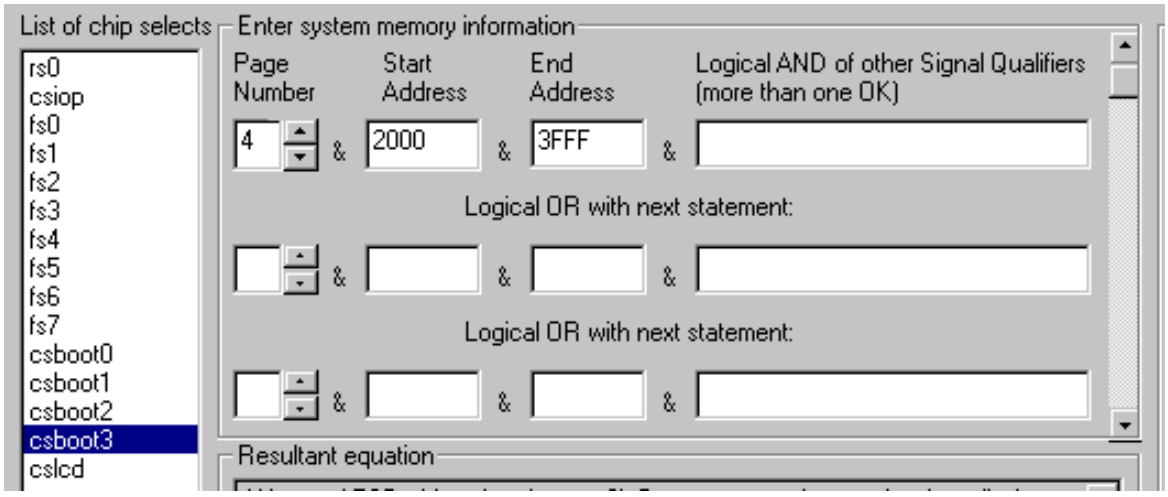
The chip-selects for csboot0 and csboot1 did not change from the first design example because no page number was specified in their definition. This means they will appear to the 68HC11 on any memory page, as indicated in Figure 6.

Next, define chip-selects as shown below for the secondary PSD Flash Memory csboot2 and csboot3 to match Figure 6:

csboot2:



csboot1:



Click 'Done' after defining all chip-selects. You should now 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 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 68HC11 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 68HC11 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. No firmware filename needs to be designated for the main PSD Flash Memory segments (fs0 - fs7) since they will be programmed by the 68HC11 during IAP.

Click OK in the merging screen to create a composite object file for programming. You are now ready to program the PSD913F2. See the section entitled “Programming the PSD“ on page 20.

THIRD DESIGN EXAMPLE - ADVANCED IAP WITH PAGING & SWAPPING

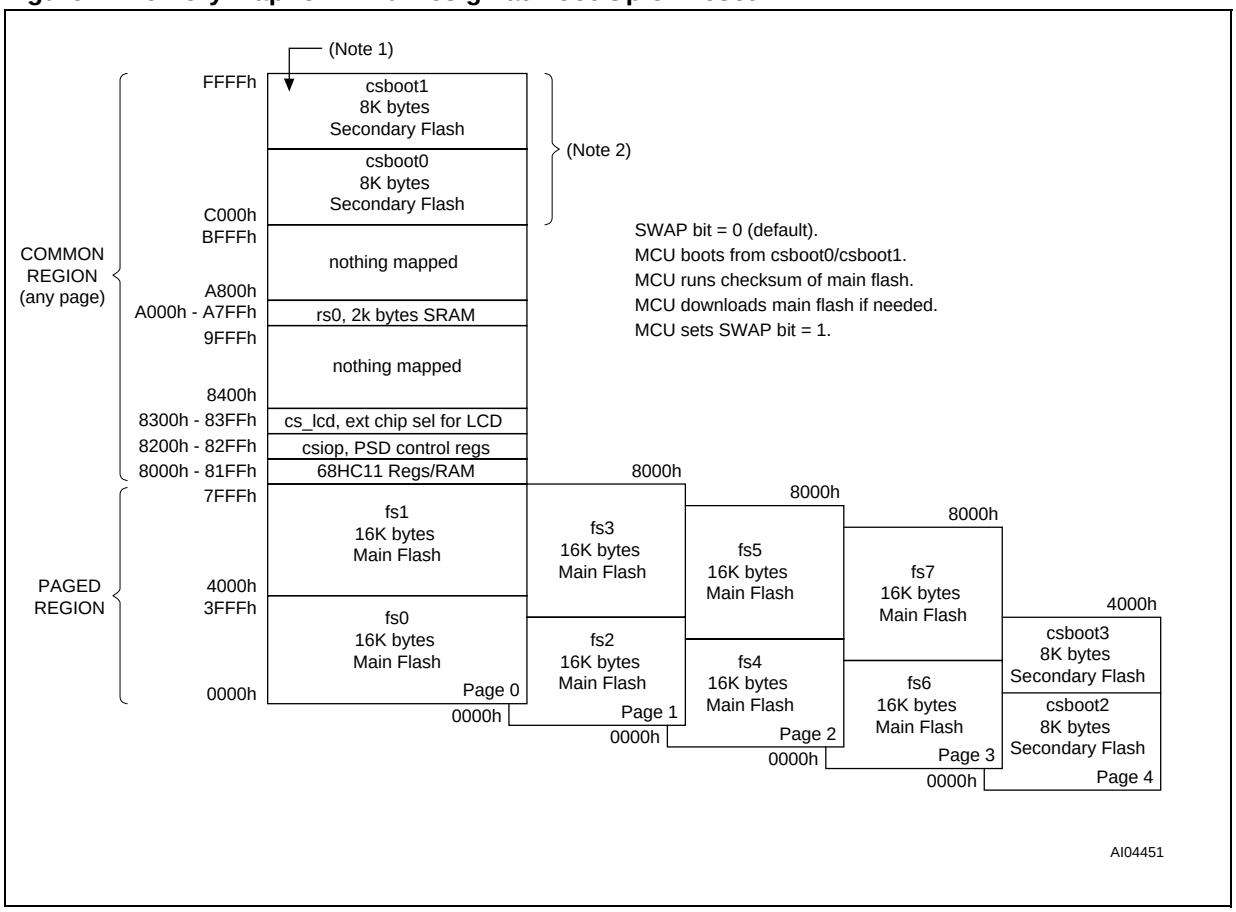
The third design example adds enhanced IAP features. The physical connections between the 68HC11 and PSD913F2 do not change, but the memory map, PSD page register definition, and some PSD chip-select definitions do change. The focus of this enhanced design is to get the most use out of the 64 Kbyte address space that the 68HC11 can access directly. This means swapping the IAP loader code out of the memory map 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 downloaded in the field if necessary while the 68HC11 operates out of main PSD Flash Memory.

Memory Map

The memory map has two basic configurations: boot-up/IAP and normal application. See Figures 7 and 8.

Memory Map Configuration at Boot-up or Reset

Figure 7. Memory Map for Third Design at Boot-Up or Reset



Note: 1. 68HC11 boots from the reset vector stored here.
2. IAP loader code gets programmed here by JTAG-ISP or a convetional programmer tool.

Figure 7 shows the memory map at system power-on or at 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 the page register bits can

be implemented for uses other than memory paging. Here's what the 68HC11 does upon power-up or reset:

– Boot from secondary Flash Memory (csboot0/csboot1) at address FFFEh
– Perform a checksum of main Flash Memory
– Download main Flash Memory from host computer if needed (IAP) and validate contents
– Set the swap bit to logic one.

Note that all PSD page register bits are cleared to zero at power-up and a reset.

Memory Map Configuration After IAP and During Normal Application

Setting the swap bit in the PSD page register swaps the location of the secondary Flash Memory segments csboot0/csboot1 (which were in the MCU boot area) with a segment of main Flash Memory, fs7, as shown in Figure 8. (Note: fs7 is 16 Kbytes and csboot1 & 2 are 8 Kbytes each - thus the 2:1 ratio). This swapping action is implemented by qualifying the chip-selects for csboot0, csboot1, and fs7 with the PSD page register bit named swap.

Figure 8. - Memory Map, Enhanced Design, Normal Operation

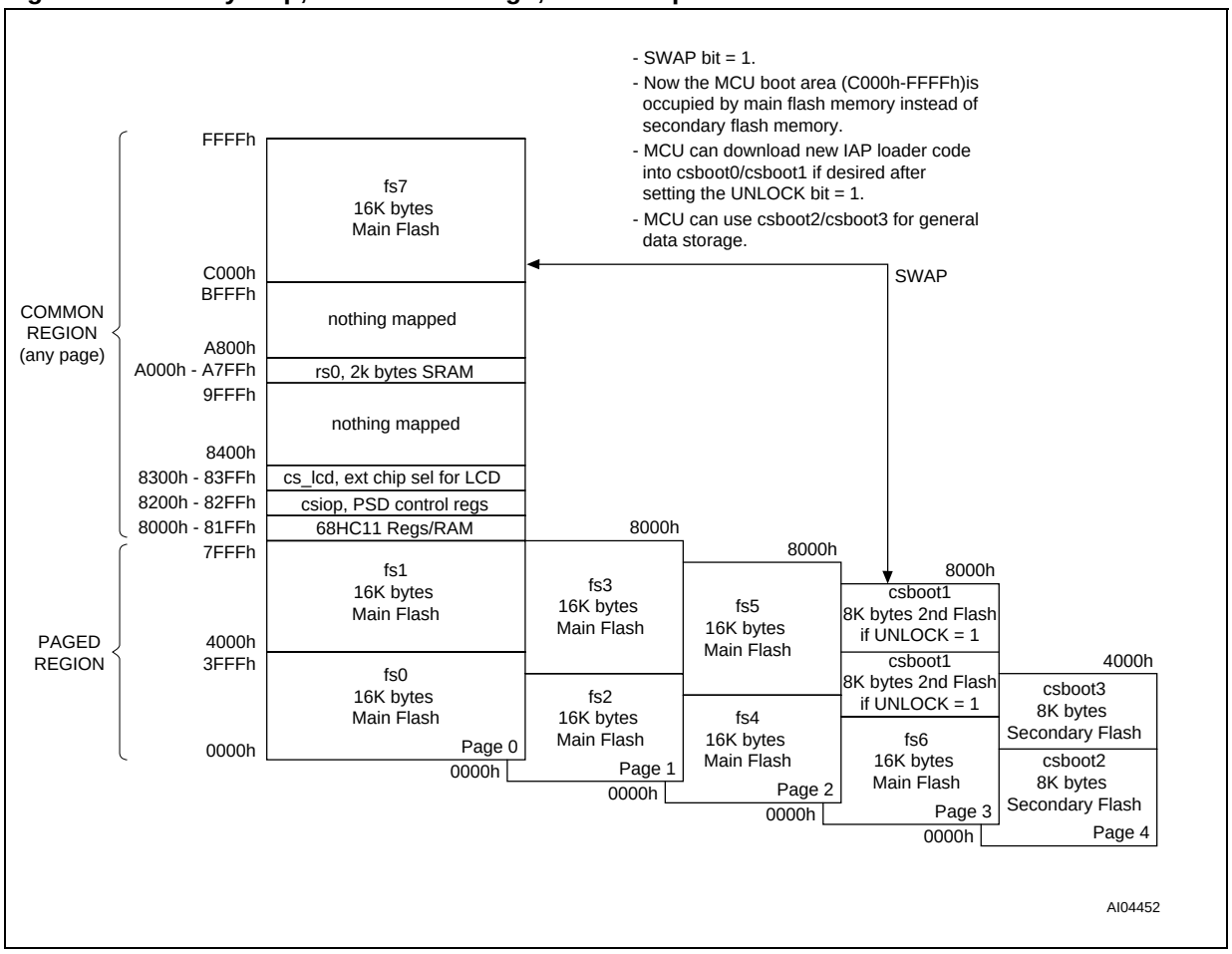


Figure 8 shows the memory map after IAP is complete and after the 68HC11 has set the swap bit to logic one. At this point, the 68HC11 can download new IAP loader code to secondary Flash Memory segments csboot0/csboot1 if needed. Another one of the eight PSD page register bits is used for logic in this design,

named 'unlock'. The 68HC11 must first set the unlock bit to logic one before updating IAP loader code. In this final configuration, the 68HC11 has available:

* 16 Kbytes of main Flash Memory in the common area (C000h-FFFFh)
* 112 Kbytes of main Flash Memory across four pages of lower memory (0000h-7FFFh)
* 2 Kbytes of SRAM in addition to the SRAM that resides on the 68HC11
* 16 Kbytes of secondary Flash Memory for general data storage on memory page 4
* 16 Kbytes of secondary Flash Memory for IAP loader code.

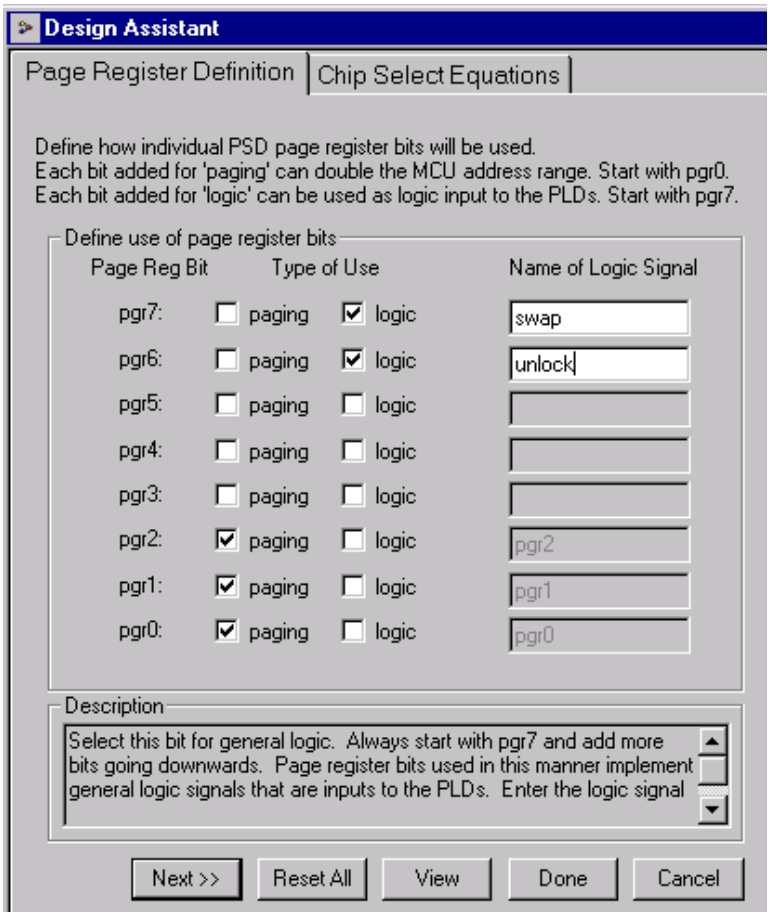
Each time this 68HC11 system gets reset or goes through a power-on cycle, the PSD presents the memory map of Figure 7 to the MCU, and the boot sequence is repeated.

Note: When the 68HC11 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 68HC11 firmware linker has set up "synchronized" code in the segment of main PSD Flash Memory (fs7) that replaces the secondary PSD Flash Memory. This is necessary to create seamless MCU operation during the actual swap of memory since the 68HC11 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 Memory that follow the MCU instructions that actually set the swap bit in the secondary PSD Flash Memory, 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.

PSDsoft Express Design Entry

To implement the advanced memory maps of Figures 7 and 8, invoke PSDsoft Express, open the project "page11" 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 "advanc11". 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.

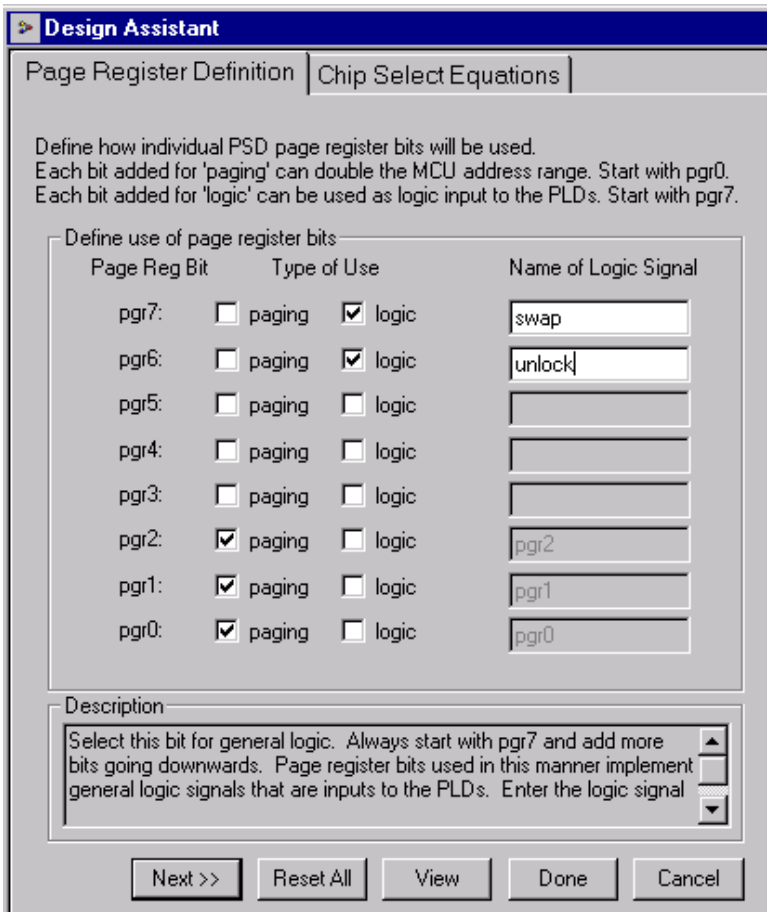
You will need to define two additional PSD page register bits to be used for logic instead. The three page register bits that were used for memory paging in the second design example stay just as they are. Define page register bits for logic, as shown below, labeling one bit "swap" and the other bit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



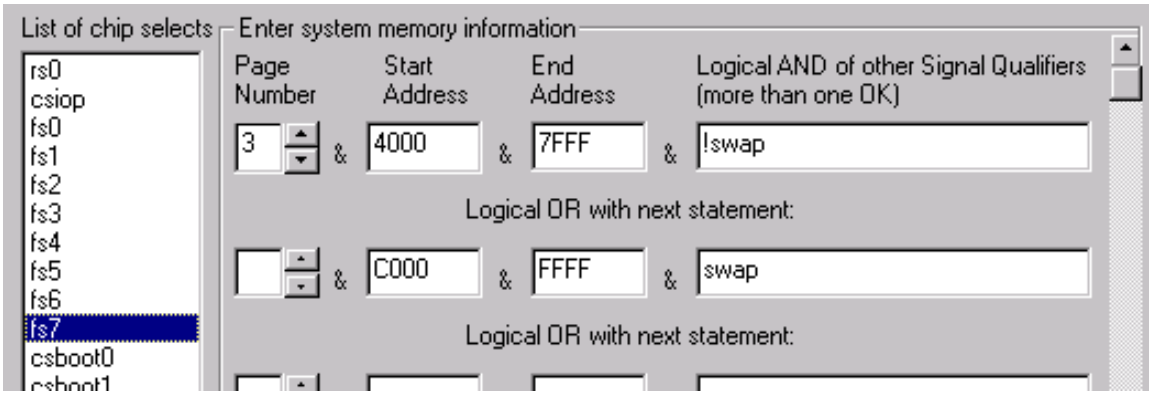
Click 'Next >>'.

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 three chip-selects need to change for the third design because they are affected by memory swapping.: main PSD Flash Memory segment fs7, and two secondary PSD Flash Memory segments csboot0 and csboot1.

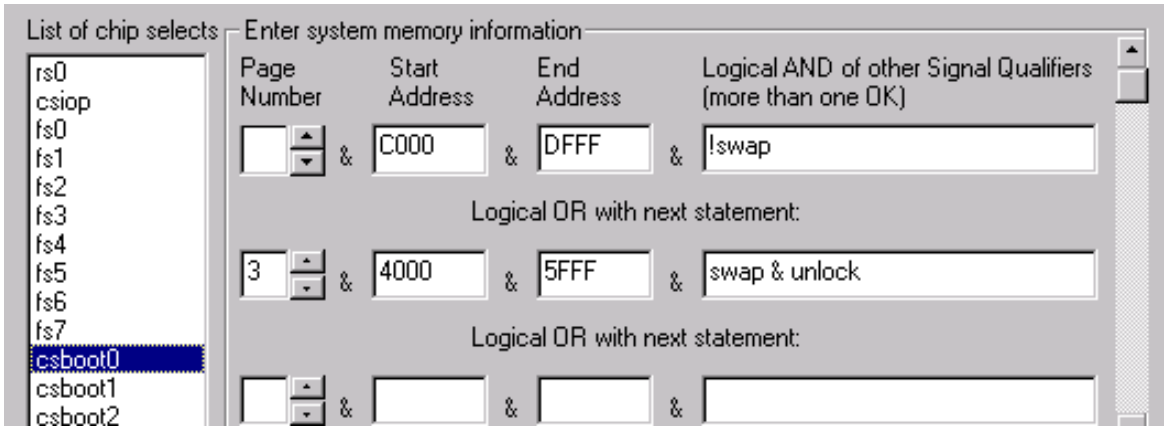
These three internal memory chip-selects must be qualified with the page register bit "swap", as shown below. The secondary PSD memory segments, csboot0 and csboot1, must be additionally qualified by "unlock" to prevent the MCU from inadvertently writing to IAP boot and loader code. Also notice the new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



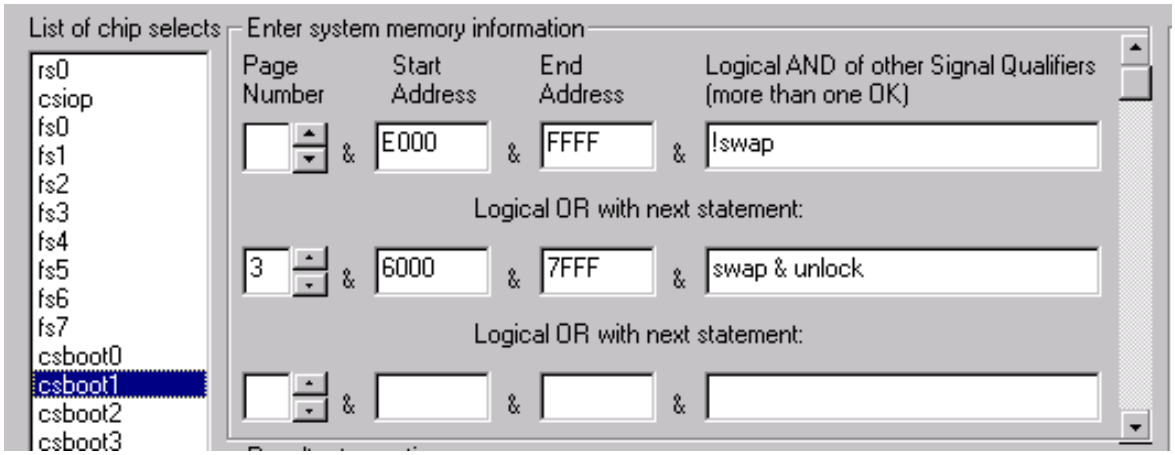
fs7:



csboot0:



csboot1:



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 7 and 8 have been implemented.

Click 'Done' and 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 68HC11 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 68HC11 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. No firmware filename needs to be designated for the main PSD Flash Memory segments (fs0

- fs7) since they will be programmed by the 68HC11 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 the section entitled “Programming the PSD“ on page 20.

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 Memory 68HC11 system capable of ISP and IAP.

REFERENCES

1. *PSD9XXF* Family Data Sheet

2. Application Note *AN1153* for detailed use of the JTAG channel

For current information on PSD products, please consult our pages on the world wide web:
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 (for application support)
ask.memory@st.com (for general enquiries)

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