AN1176
APPLICATION NOTE

Design Guide
68HC11 and PSD813F1

CONTENTS

■ (See next page)

Contents

1 Introduction 1
2 Simple Design Example 3
2.1 Physical Connections 4
2.2 Memory Map 4
2.3 PSDsoft Design Entry 6
2.3.1 Invoke PSDsoft and Open a New Project 6
2.3.2 Design Flow Window 7
2.3.3 Device Configuration 7
2.3.4 PSDabel Design Entry 8
2.3.5 Edit the template. 9
2.3.6 Logic Synthesis and Fitting 11
2.3.7 C Code Generation 11
2.3.8 MCU Code Mapping 11
2.3.9 Programming the PSD 13
2.3.10 Programming with FlashLINK™ 14
2.3.11 Programming with PSDpro **16**
3 Enhanced Design Example 16
3.1 Physical Connections 17
3.2 Memory Map 17
Memory Map Configuration at Boot-Up/ISP 18
Memory Map Configuration During Normal Application 19
3.3 PSDsoft Design Entry 20
4 Conclusion 21
5 References 21

1 Introduction

FLASH PSD813F 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 development software that makes ISP easy for you, regardless of how much experience you have in embedded flash design.

This document will offer two flash 68HC11 designs using a PSD813F 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 of enhanced features of PSD In-System-Programming as applied to the 68HC11. You can start with the first design, and migrate to the second as your functional requirements grow.

The Generic Problem

Typically, a host computer downloads firmware into an embedded flash system through a communication channel that is controlled by the MCU. This channel is usually a UART, but any communication channel that the 68HC11 supports will do (network, SPI, CAN, J1850, etc.). The 68HC11 must execute the code that controls the ISP process from an independent memory array that is not being erased or programmed. Otherwise, boot code and flash programming algorithms (ISP loader code) will be unavailable to the 68HC11. It is absolutely necessary to use an alternate memory array (an independent memory that is not being programmed) to store the ISP loader code.

A system designer must choose the type of alternate memory to store ISP 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 flash system with external memory is shown in Figure 1.

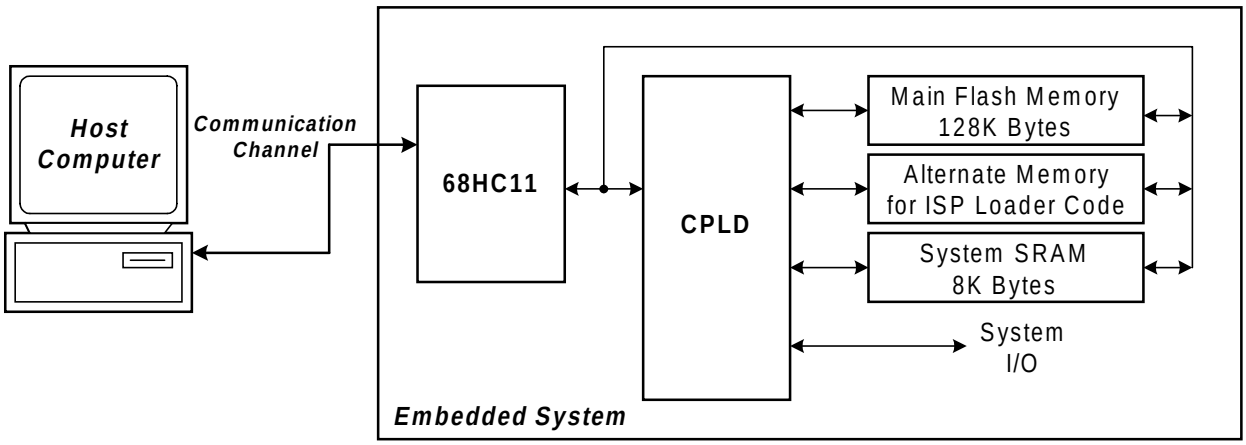


Figure 1 – Embedded flash system capable of ISP (5 devices)

A Common Solution
For ISP, some 68HC11 designers will use the 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 the 68HC11 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, which is difficult to debug and is vulnerable to power outages.

An Better, Integrated Solution
A more robust choice for the type of alternate memory for ISP loader code is Flash or EEPROM. Figure 2 shows a two-chip solution using an FLASH PSD. This system has ample main flash memory, EEPROM for the alternate memory, and some SRAM. All three of these memories can operate independently and concurrently; meaning the MCU can operate from one memory while erasing/writing the other. The system has programmable logic, expanded I/O, and design security. This two-chip solution is 100% ISP.

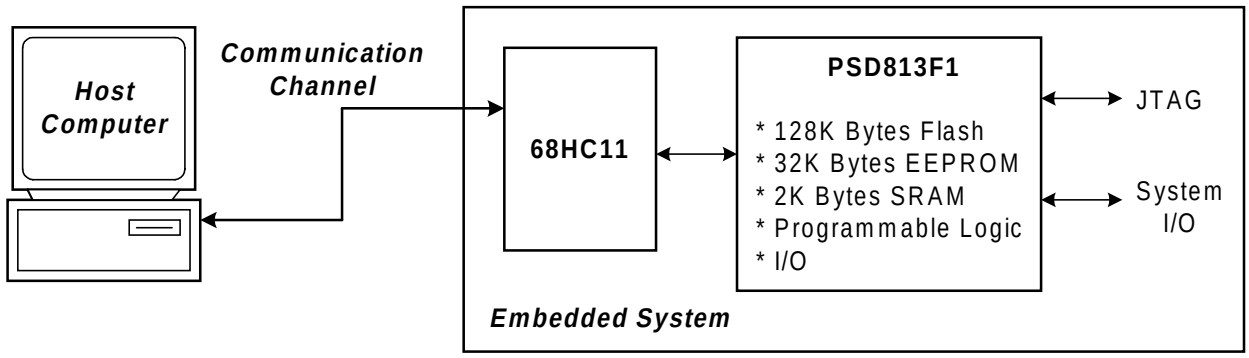


Figure 2 – Embedded flash system capable of ISP (2 devices)

The ISP method described so far requires MCU participation to exercise a communication channel to implement a download to the main flash memory. The PSD813F also offers an alternative ISP method that uses a built-in IEEE-1149.1 JTAG interface, and requires no MCU participation. This means that a completely blank PSD can be soldered into place, and the entire chip can be programmed in-system using ST's FlashLINK™ JTAG cable and PSDsoft development software ($99 USD for both, see www.st.com/psm). No 68HC11 firmware needs to be written, just plug in the FlashLINK™ cable and begin programming memory, logic, and configuration. This is a powerful new feature of the PSD813F that allows immediate development of application code in your lab, smart manufacturing techniques, and easy field updates.

Let's take a quick look inside the FLASH PSD813F1, as shown in Figure 3. You can see the three independent memory arrays, which are selected on a segment basis when the proper MCU address is decoded in the Decode PLD. The page register participates in memory decoding, which greatly simplifies paging. The MCU address, data, and control signals have access to just about everything inside the chip, including the general purpose CPLD. The CPLD has 16 macrocells, and 24 special latches for input signals. All 16 macrocells and 24 input

latches can be accessed directly by the MCU in silicon. No extra design effort on your part is needed. This MCU access feature is great for loadable shift registers, counters, mailboxes, state machines, etc. There are 27 I/O pins that generously replace the 68HC11 I/O pins that are sacrificed to external memory address and data lines. 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. And finally, the self-contained JTAG controller allows ISP of all areas of the chip, even on a completely blank device that is soldered onto a circuit board.

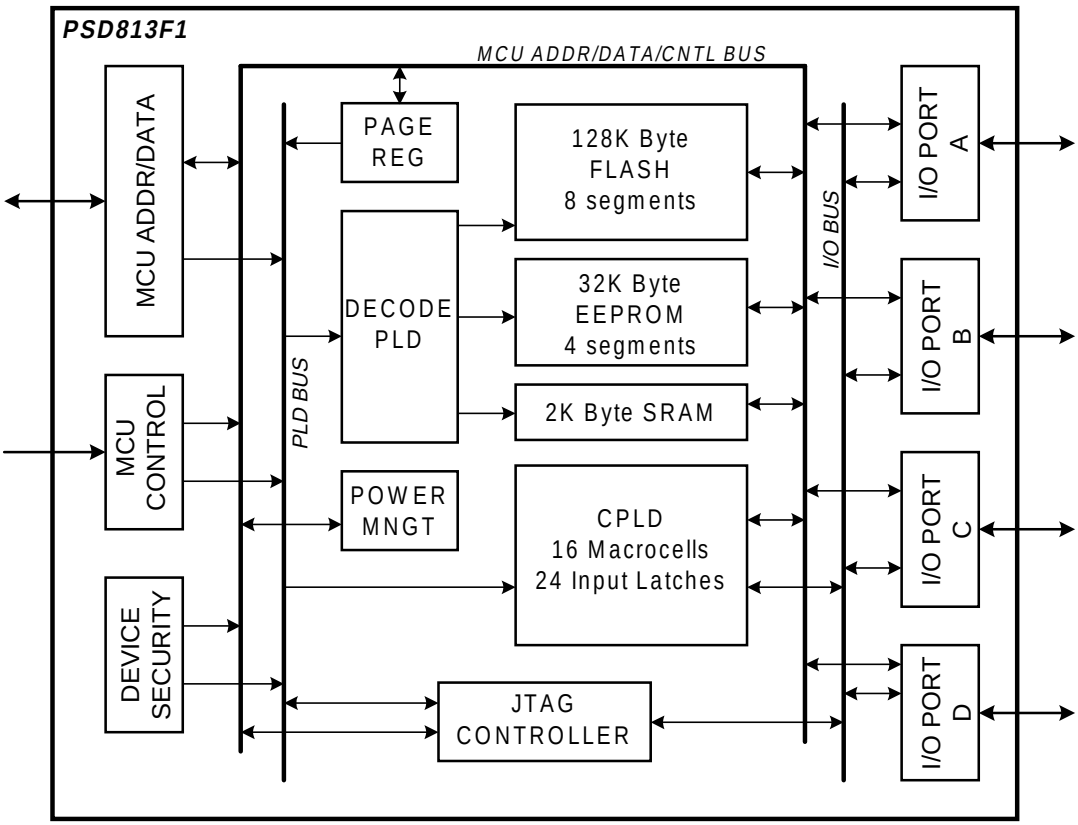


Figure 3 – Top Level Block Diagram of PSD813F1

2 Simple Design Example

The first design example will outline the steps to get a flash 68HC11 system up and running quickly. You will see a connection diagram, a memory map, and the necessary design entry in the PSDsoft software development environment. A PSD813F1 is used in this example, but the other members of the FLASH family may be used instead, with minor changes. See the PSD813F data sheet for a comparison of family members.

2.1 Physical Connections

Connect your 68HC11 to the PSD as shown. The same connections can be used for all five members of the PSD813F family (PSD813F1 though PSD813F5). Notice that the 16 general purpose I/O lines that were lost on the 68HC11 for external address and data signals are replaced by the 16 I/O lines on ports A and B of the PSD. The JTAG programming channel connection is optional, as well as the battery backup feature for the internal PSD SRAM.

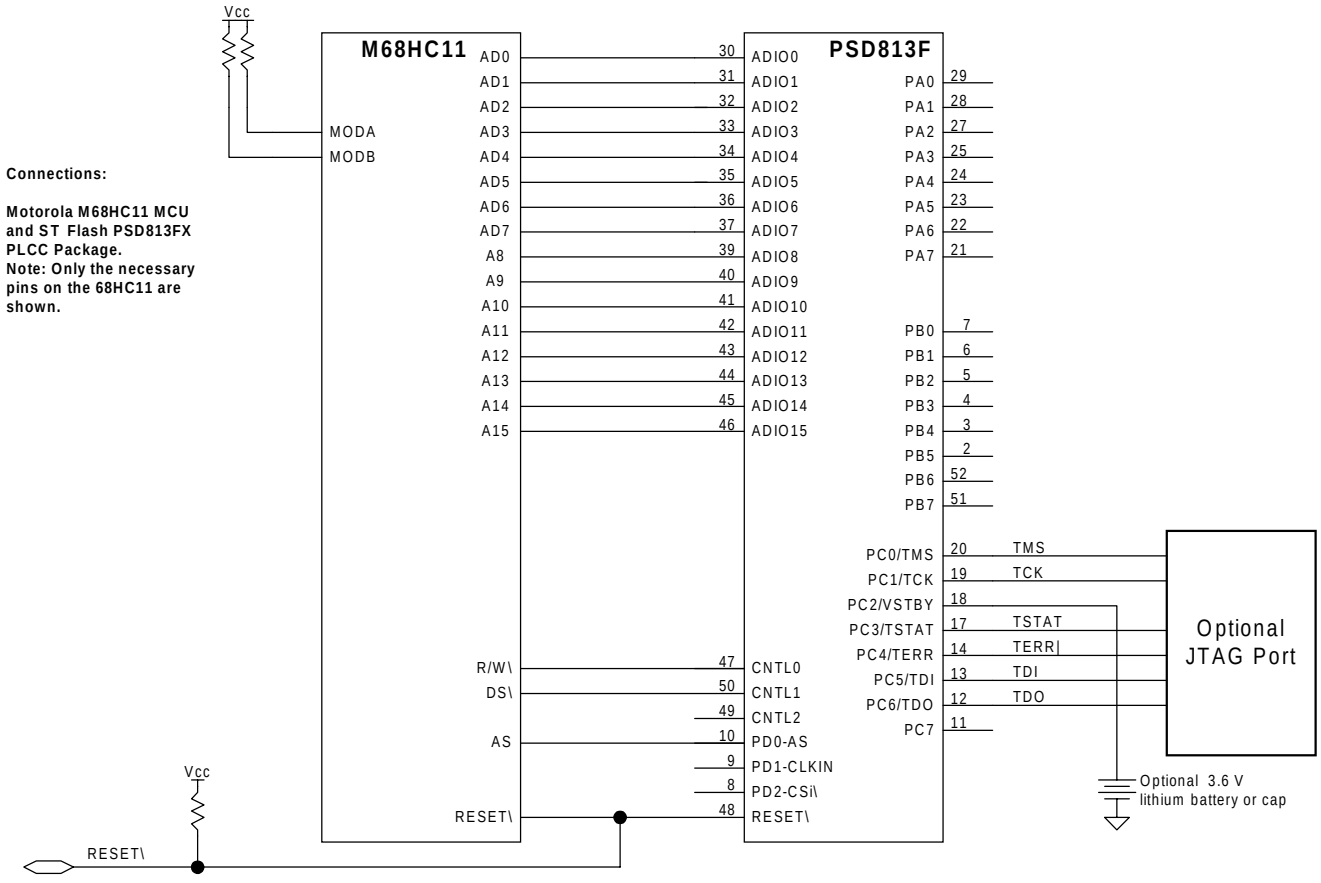


Figure 3 – Physical Connections, 68HC11 and PSD813F

2.2 Memory Map

For this simple design, we'll use a PSD813F1, which provides 128K bytes flash memory, 32K bytes EEPROM, and 2K bytes SRAM. We'll use one half of the EEPROM (16K bytes) to hold the ISP bootloader code, 68HC11 interrupt vectors, and common firmware functions. We'll allocate the remaining 16K bytes of EEPROM for general data storage.

Since the 68HC11 cannot address more than 64K bytes of address space directly, we will use paging (or banking) to take full advantage of the 162K bytes of total memory on the PSD. The PSD has a built-in page register for this purpose. Please examine the memory map in Figure 4.

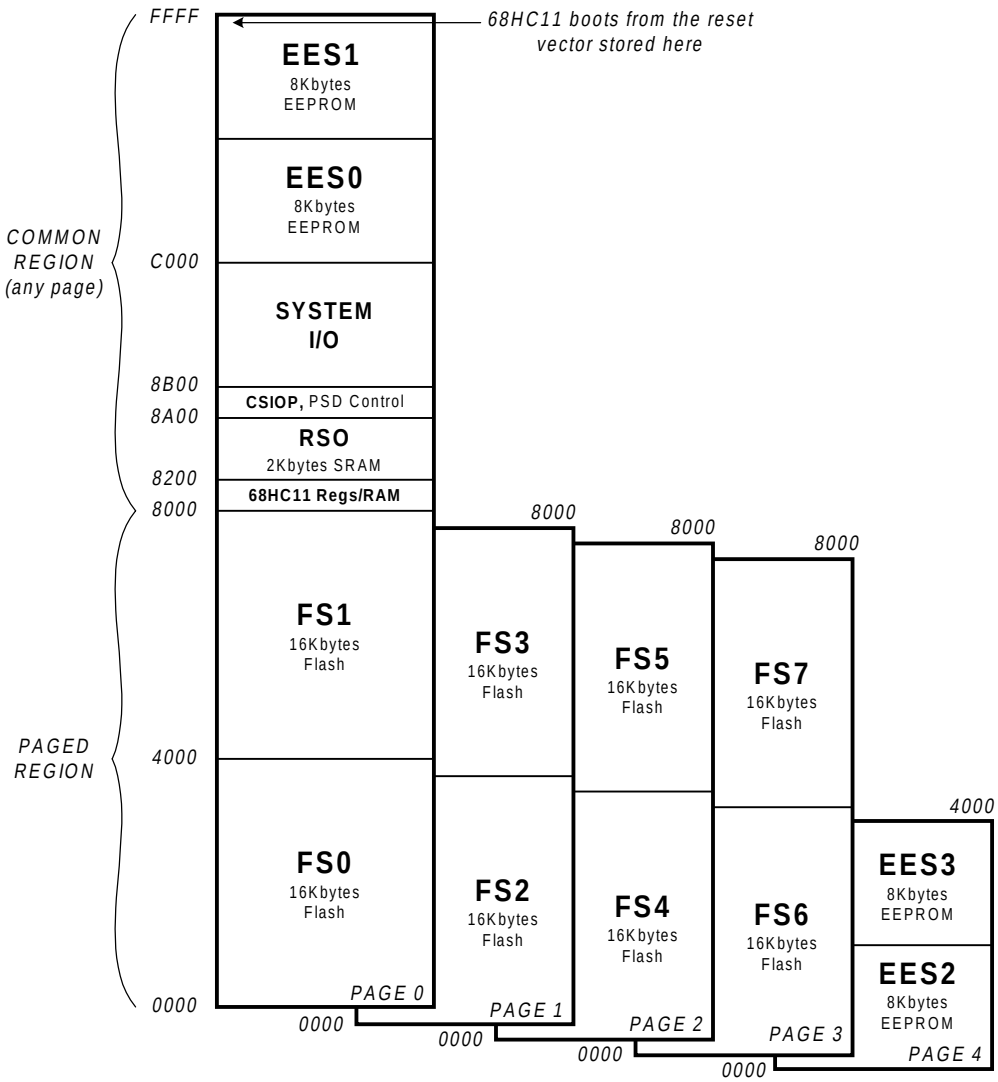


Figure 4 – Memory Map, Simple 68HC11/PSD813F1 design

Notice that the memory map is basically divided into two areas. The upper memory (8000h to FFFFh) 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:

- 68HC11 reset vector and initialization routines
- 68HC11 interrupt vectors and service routines
- I/O management
- Memory page management
- SRAM variables and SRAM stack
- Anything else that must be accessible no matter what memory page is selected.

The lower half of memory (0000h to 7FFFh) is the region that will be "paged" or "banked". This paged area of 32K bytes allows the 68HC11 to address large amounts of memory (well beyond the 64K byte limit of the 68HC11). We'll place all 128K bytes of PSD main flash memory across four 32K-byte pages (memory pages 0 through 3). The remaining half of the PSD EEPROM will be placed on memory page 4 and used for general data.

To make this division of the memory map possible (i.e. common upper half, paged lower half), the 68HC11 initialization firmware must move the internal 68HC11 SRAM and registers from their default location of 0000h, to the new location of 8000h, as shown in Figure 4. There is a special register in the 68HC11 (the INIT register) that facilitates this move. See the 68HC11 reference manual from Motorola. This move is necessary to "clean-up" the memory map, meaning that the entire region from 0000h – 7FFFh is free to use for paging, once these internal MCU locations are moved up and out of the way to 8000h. Also note that the common area (non-paged) had to be located in the upper half of memory (8000h – FFFFh) because the 68HC11 boots from high memory (FFFEh).

The nomenclature FS0-FS7, EES0-EES3, RS0, and CSIOP in Figure 4 refer to the internal memory segments of the PSD. The main flash memory has eight 16K byte segments (FS0-FS7). The EEPROM has four 8K byte segments (EES0-EES3). The 2K byte SRAM has a single segment (RS0). The internal PSD control registers lie in a 256-byte address space, named CSIOP. You place these PSD memory elements to the desired location within the memory map by writing Hardware Description Language (HDL) equations for the Decode PLD for each memory segment. Equations are written and compiled in the PSDsoft software development environment, using the standard Hardware Description Language, ABEL (called "PSDabel" as packaged in PSDsoft).

Note that the page register bits are zero at power-up or after a system reset.

2.3 PSDsoft Design Entry

Highlights of design entry will be given here. Please refer to Application Note 57 for a thorough coverage of all the features of PSDsoft. This section is meant to show you just the essentials to get you going. Here are the steps:

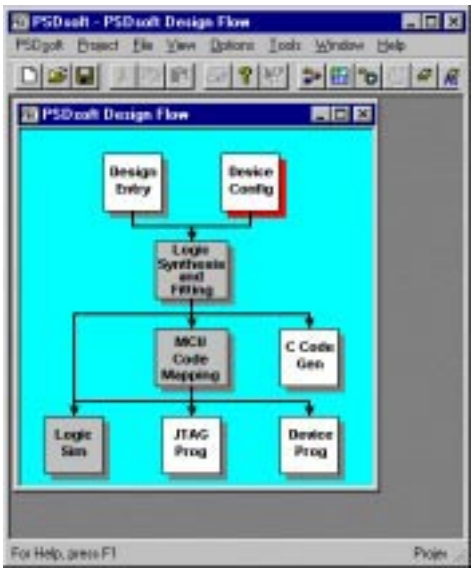
2.3.1 Invoke PSDsoft and Open a New Project

- Start PSDsoft.
- Open a new project.
- Enter your project name and directory (in this example, it is named 'yourfile' in the directory PSDSOFT\YOUR_PROJECT).
- Select a PSD813F1
- Select the 68HC11 template
- Click OK.

Now you have your project established, based on a PSD813F1 and a 68HC11.

2.3.2 Design Flow Window

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



2.3.3 Device Configuration

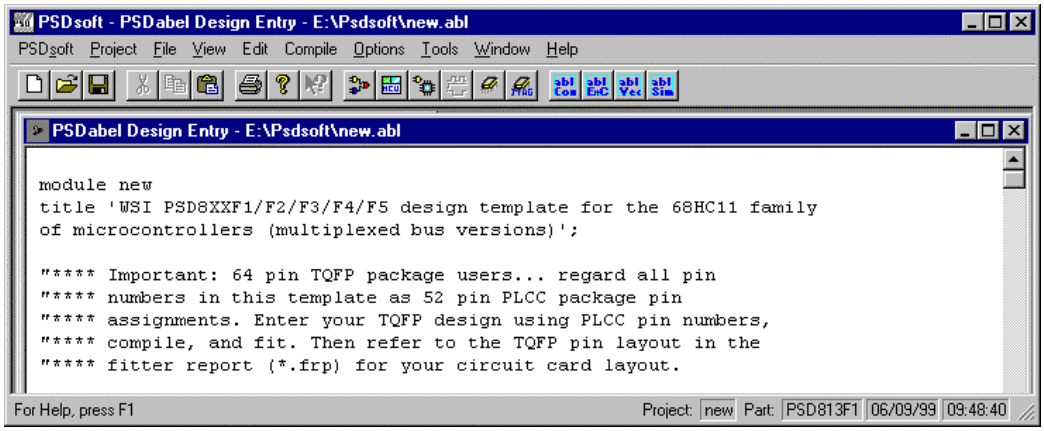
Click on the Device Config box in the design flow window to configure the PSD for connection to a 68HC11. You should make selections with your mouse to match this picture (this is a multiplexed 68HC11):

Now go to the JTAG Configuration tab and click the boxes as shown if you want to dedicate six pins of Port C to be the JTAG channel for In-System-Programming. See Application Note 54 for options regarding JTAG.

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

2.3.4 PSDabel Design Entry.

Click on the Design Entry box of the design flow window so you can enter your HDL equations using PSDabel. Since you selected a 68HC11 template previously, the basis of the design will already be there; you just have to edit the PSDabel template. You should see this:



2.3.5 Edit the template.

The default 68HC11 template that you see on the screen has numerous elements inside to illustrate many features of the FLASH PSD813F. To simplify things, edit this template file that you see to look like the following simple PSDabel file. You can cut and replace the entire body of the PSDabel text in the template on your PC directly from this document if you desire. Be sure to change the module name in the very first line from 'yourfile' to whatever you named your project when you first created the project.

```
module yourfile
title 'Simple 68HC11 embedded flash design';


"**** Pin Declarations

r_w      pin;    "CNTL0 Input:(pin 47)- read/write indicator
e        pin;    "CNTL1 Input:(pin 50)- E clock
as       pin;    "PD0 Input:(pin 10)- address strobe
reset    pin;   "Input:(pin 48)- system reset
a15..a0  pin;   "Input:(pins 46..39,37..30)- demuxed address


"**** Port pin assignments. See Application Notes 55 & 57
"**** for details on assigning port pins. No I/O pins are
"**** used in this simple HC11 example.

"**** Be aware that you have clicked the boxes in
"**** 'Device Config' to dedicate 6 of the 8 Port C
"**** pins for JTAG use. See Application Note 54
"**** for details on JTAG and Port C


"**** Internal node declarations

fs7..fs0 node; "Internal chip selects, Main Flash memory segs.
                "Eight segments, 16K bytes each

ees3..ees0 node;   "Internal chip selects, EEPROM segments.
                    "Four segments, 8K bytes each

rs0 node;  "Internal chip select, SRAM, one segment, 2K bytes

csiop node;   "Internal chip select PSD Control Regs, 256 bytes

pgr2..pgr0 node;  "Internal PSD page register bits. Eight are
                   "available, we are only using three
                   "bits in this example to access five
                   "memory pages.
```

```
"**** Definitions for convenience and readability

X  = .x.;             "Don't care symbol

page = [pgr2..pgr0];  "page symbol based on three page bits

address = [a15..a0];  "De-muxed MCU address signals

EQUATIONS

"******** DPLD equations ************************
" Write chip select equations to implement the memory map

fs0 = (address >= ^h0000) & (address <= ^h3FFF) & (page == 0);
fs1 = (address >= ^h4000) & (address <= ^h7FFF) & (page == 0);
fs2 = (address >= ^h0000) & (address <= ^h3FFF) & (page == 1);
fs3 = (address >= ^h4000) & (address <= ^h7FFF) & (page == 1);
fs4 = (address >= ^h0000) & (address <= ^h3FFF) & (page == 2);
fs5 = (address >= ^h4000) & (address <= ^h7FFF) & (page == 2);
fs6 = (address >= ^h0000) & (address <= ^h3FFF) & (page == 3);
fs7 = (address >= ^h4000) & (address <= ^h7FFF) & (page == 3);

ees0 = (address >= ^hC000) & (address <= ^hDFFF) & (page == X);
ees1 = (address >= ^hE000) & (address <= ^hFFFF) & (page == X);
ees2 = (address >= ^h0000) & (address <= ^h1FFF) & (page == 4) ;
ees3 = (address >= ^h2000) & (address <= ^h3FFF) & (page == 4) ;

rs0 = (address >= ^h8200) & (address <= ^h89FF) & (page == X);

csiop = (address >= ^h8A00) & (address <= ^h8AFF) & (page == X);

"******** GPLD/ECSPLD Equations ***************
" Write equations for general logic and external chip
" select equations here. See Application Notes 55 and 57
" for details.

end
```

The following are things to notice about this simple PSDabel file (in order):

- The pin declarations for connection to the 68HC11 occur at the beginning of the file.
- No PSD general-purpose I/O pins are declared in this example for simplicity. These features can be easily implemented, see App Notes 55 & 57 for examples using I/O pins.
- Internal PSD nodes are declared for memory selection and PSD control.
- Some logic definitions are stated to make the file more readable.
- The HDL equations appear for the selection of internal PSD memory segments and the PSD control register. Each memory segment is assigned to an address range according to the memory map of Figure 4.
- Observe how the page register is used to participate in memory selection.
- No equations for the GPLD are used here. See App Notes 55 & 57 for examples.

2.3.6 Logic Synthesis and Fitting

After you have edited the PSDabel template file to look like the above file, go to the design flow window and click on the 'Logic Synthesis and Fitting' box. Now PSDsoft will compile the PSDabel file and then synthesize the PSDabel statements into reduced logic that fits the PSD813F1 silicon. When this process is complete, a report will pop up that shows the resulting pin assignments and the resulting reduced equations. This is the 'fitter report', which you can use to document your design. Notice the pin assignments for the JTAG channel in the fitter report.

2.3.7 C Code Generation

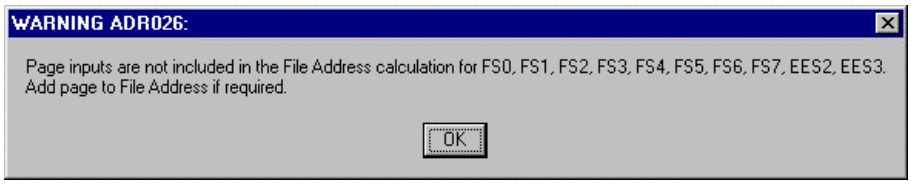
You can take advantage of the provided low level C code drivers for accessing memory elements within the PSD by clicking on the 'C code Generation' box in the design flow window. ANSI C code functions and headers are generated for you to paste into your 68HC11 C compiler environment. Just tailor the code to meet your system needs and compile. To get the C functions and headers, simply specify the folder in which you want the ANSI C files to be written. See Application Note 57 for details on the C code generation feature.

2.3.8 MCU Code Mapping

Now that the fitting process is complete, PSDsoft has created a fuse pattern that reflects the PSD configuration and logic of your design. PSDsoft places this fuse information into a file (this is the *.obj file). However this fuse pattern does not yet contain the 68HC11 firmware. The next step will accomplish this, producing a *.obj file that contains the PSD configuration AND the 68HC11 firmware. This final *.obj file is what gets programmed into the PSD. Note: the first *.obj file that is created without MCU firmware can be used for logic simulation, see Application Note 57 for details. That same *.obj file is appended with MCU firmware in the next step below.

This next step, MCU Code Mapping, will input the firmware file(s) that contain absolute addresses from your 68HC11 compiler/linker (S-record or Intel HEX format), and it will map these file(s) into the memory segments of the PSD according to the HDL equations that you entered in PSDabel. This mapping process translates the absolute system addresses that 68CH11 uses into physical internal PSD addresses that are used by a programmer to program the PSD. This address translation is transparent to you, the user. All you need to do is type in the file name(s) that were generated from your 68HC11 linker into the appropriate boxes, and PSDsoft does the rest.

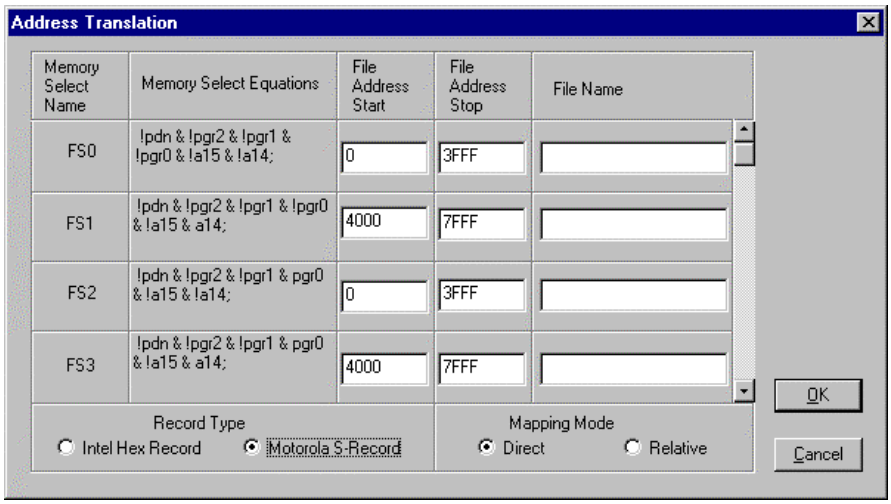
Go to the design flow window, click the 'MCU Code Mapping' box. You should see the following warning as the utility starts.



For this simple example, you can ignore the warning and click ‘OK’ because we are only placing 68HC11 code into EEPROM segments EES0 and EES1, which are not paged. But here is an explanation of this warning, as it will apply later on if you page your firmware.

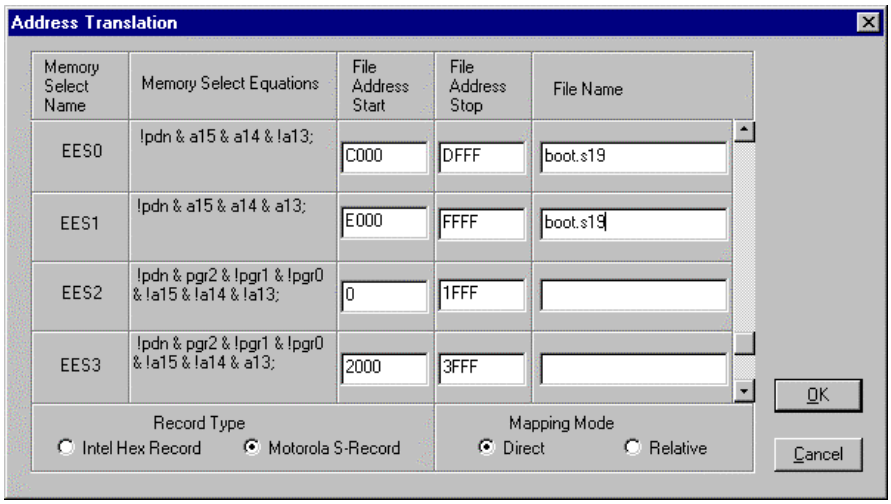
> PSDsoft attempts to populate all of the address range boxes for you, based on your PSDabel equations. These are the absolute address ranges that PSDsoft will expect to see inside the file(s) generated by your 68HC11 linker. However, if PSDsoft sees that paging information is used in your PSDabel memory segment equations, PSDsoft warns you it has filled in address ranges that may be ambiguous (overlapping or dependent on non-address signals, like page register bits) and it is up to you to resolve them. How you resolve them is purely a function of your 68HC11 compiler/linker and how it handles paging. For example, some MCU linkers will generate a different file for each memory page of firmware, and each of these files will contain MCU addresses that do not exceed 16 bits. Other linkers will put all of the memory pages of firmware into a single file using MCU addresses greater than 16 bits to represent multiple pages (extends 16-bit addresses with page bits). Either method requires you to type the appropriate file name(s) and address ranges into the window based on how your particular 68HC11 linker operates.

After clicking OK to the warning message, you should see this:



On the left are the individual PSD memory segments (FSO, FS1, etc). The next column holds the logic equations for selection of each memory segment (from PSDabel) are shown for reference. In the middle are the address ranges that you specified in the PSDabel file to create the memory map (PSDsoft filled in these address fields for you). PSDsoft will expect to find these absolute MCU addresses within your 68HC11 linker file(s) when the file(s) are imported. On the right are boxes where you type in the name of the file(s) that your 68HC11 linker creates. Notice that you can select Motorola S-Record or Intel HEX format for input. Leave the ‘Mapping Mode’ set to ‘Direct’.

Now slide the scroll bar down until see the following:



Type in the name of the file from your 68HC11 linker that contains the firmware that will boot up your system. Use your own file name. For this example we'll call it boot.s19. This file can contain very simple 68HC11 code that just configures your system hardware and performs rudimentary tasks to check out your new hardware. In this example, there are 16K bytes available in EEPROM segments EES0 and EES1, 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, such as implementing a download to main flash memory from a host computer.

No file names need to be typed in for the main flash regions (FS0-FS7) because we are only operating out of EEPROM (EES0, EES1) for now.

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

2.3.9 Programming the PSD

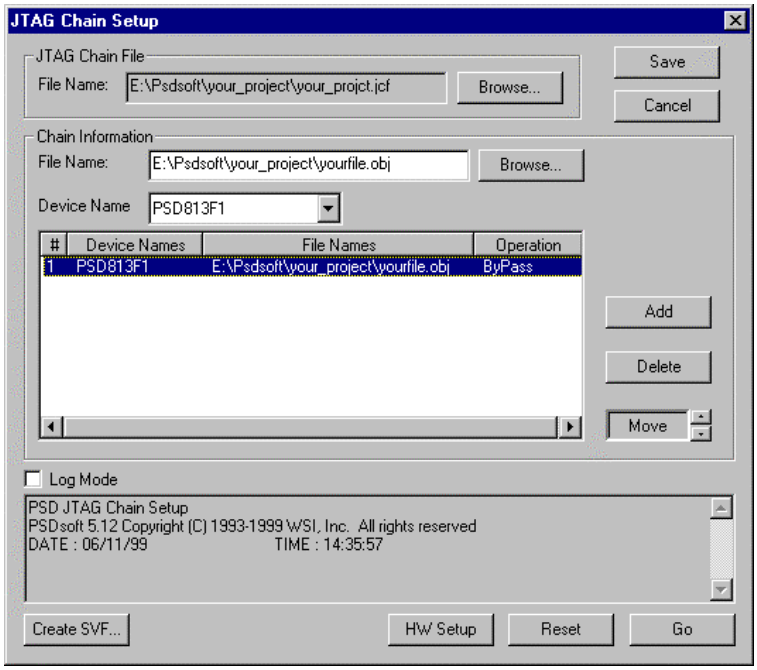
The *.obj file can be programmed into the PSD by one of three ways:

- 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, such as DATA I/O, Stag, Needhams, etc. Please see the web site at www.st.com/psm for compatible third-party programmers.

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

2.3.10 Programming with FlashLINK™

Connect the FlashLINK™ JTAG cable to the PC parallel port. Click the 'JTAG Programming' box in the design flow window. You should see the following window pop up:



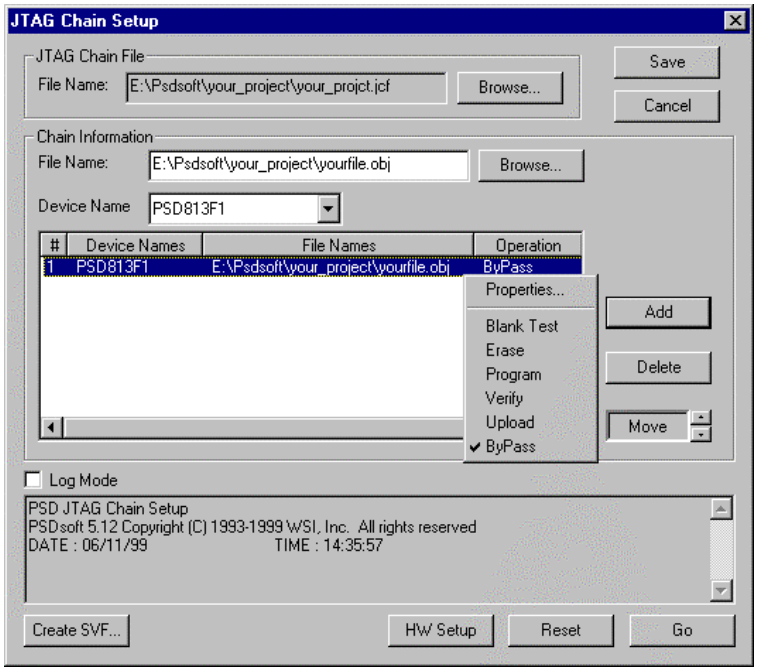
This window allows you to describe the JTAG chain that exists on your circuit board, your desired operation, and also offers a loop back test for your FlashLINK™ cable. If this is your first use, test your FlashLINK™ cable and PC port by clicking on the 'HW Setup' button, then click the 'LoopTest' button and follow the directions.

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

- Specify the *.obj file to be programmed into the PSD. Click the browse button within the 'Chain Information' area to find your file. In this example, it is yourfile.obj, as shown in the window above.
- Choose the device name, PSD813F1, also shown above.
- Click the 'Add' button to add the *.obj file and the PSD device to the chain definition as shown.

Now that the file and part are specified, you must specify the operation that you want to perform. In this case we want to program the device. Notice in the highlighted blue line for device number one in the window above, the operation is 'ByPass'. This is the default operation for all devices that are added to the JTAG chain. Lets change that operation by clicking the right mouse button on the highlighted blue line.

You should see this:



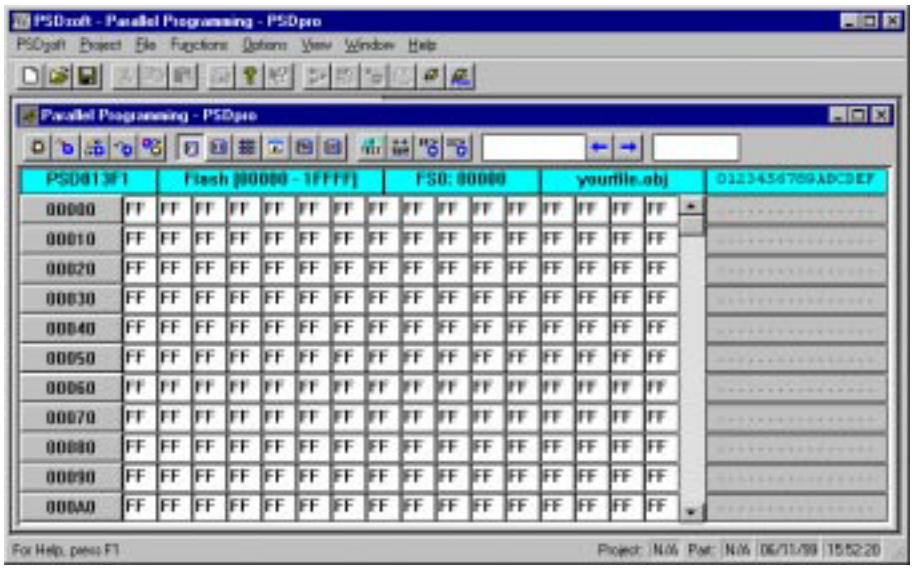
Now click on the operation ‘Program’ that appears in the small pop up box, with either mouse button. Once you select ‘Program’, you will be given a choice to program all of the PSD, or just certain regions. Click ‘All’ for this example.

Again, right click on the highlighted blue line for device number one. This time, just for reference, click on ‘Properties..’. This is where you can specify whether or not to use 6 pins or 4 pins for the JTAG channel. You can also specify the default state of I/O pins while JTAG operations are occurring. Notice that the default JTAG setup in this window is ‘Option 3’ (6 JTAG pin channel), and all I/O pins are in high-impedance input mode. Just click ‘Cancel’ to move on, since we are using all the default choices on this screen.

Now everything has been defined in the JTAG chain of one device: the file name, the type of PSD, and the desired operation. To begin programming the device, just click the ‘Go’ button. After programming is complete, you can save the JTAG setup for this programming session to a file for later use. To do so, click on the ‘Save’ button in the ‘JTAG Chain File’ section of the window. See Application Notes 54 and 57 for details on all the features of the JTAG channel and this JTAG Chain Setup window.

2.3.11 Programming with PSDpro

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



If this is the first use of the PSDpro, click on the 'Htest' icon to perform a test of the PSDpro and the PC port. After testing, place a PSD813F into the socket of the PSDpro and click on the 'Program' icon (the *.obj file is automatically loaded when this process is invoked). The messaging of PSDsoft will inform you when programming is complete. See Application Note 57 for details on all the features of this 'Parallel Programming' window.

This window is also helpful even if you do not have a PSDpro programmer. You can use this window to see where the Address Translate utility of PSDsoft has placed the 68HC11 firmware within physical memory of the PSD. For this design example, you can click on the EEPROM icon in the tool bar, then scroll to the end segment EES1 to see the 68HC11 reset vector at absolute MCU addresses FFFEh and FFFFh, which translates to PSD EEPROM physical addresses 23FFEh and 23FFFh respectively. To see how all of your 68HC11 absolute addresses translated into physical PSD memory addresses, see the report that PSDsoft generates under 'View' from the main toolbar, then select 'Address Translation Report'. The 'start' and 'stop' addresses in the report are the absolute MCU system addresses that you have specified. The addresses shown in square brackets are 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).

3 Enhanced Design Example

This second design example builds upon the first to add enhanced features to this ISP capable 68HC11 system. The physical connections between the 68HC11 and PSD813F1 do not change, but the memory map and PSDabel equations do. The focus of this enhanced design is to get the

most use out of the 64K byte address space that the 68HC11 can access directly. This means swapping the ISP loader code out of the memory map after ISP is complete, and replacing it with application code, leaving the maximum amount of address space available for the application. Swapping the ISP 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 ISP, and a different set during the normal application. In addition, this swapping technique allows the ISP loader code itself to be downloaded in the field if necessary while the 68HC11 operates out of main flash memory.

3.1 Physical Connections

Same as the simple design example (section 2.1).

3.2 Memory Map

The memory map has two basic configurations, boot-up/ISP and normal application. See Figures 5 and 6.

3.2.1 Memory Map Configuration at Boot-Up/ISP

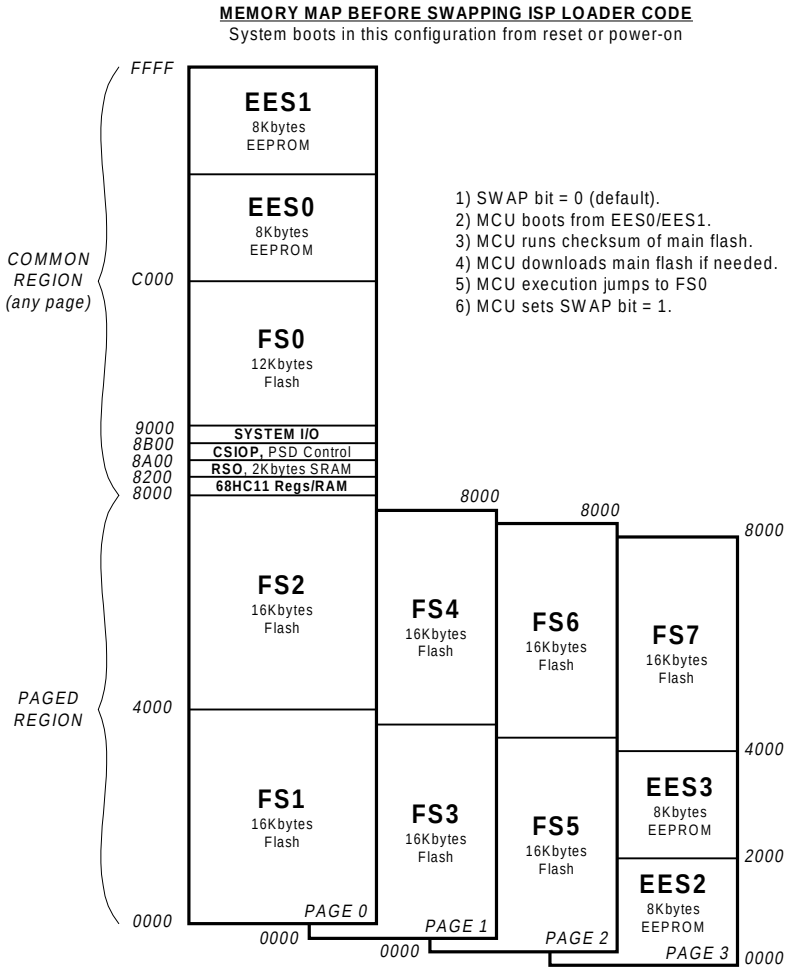


Figure 5 – Memory Map, Enhanced Design at Boot-Up/ISP

Figure 5 shows how the memory map looks at system power-on or at system reset. The SWAP bit is one of the eight internal PSD page register bits, whose value is zero by default. The SWAP bit is an example of how the page register bits can be implemented for uses other than memory paging. Here's what the 68HC11 does upon power-up or reset:

- Boot from EEPROM (EES0/EES1) at address FFFEh
- Perform a checksum of main flash memory
- Download main flash memory from a host computer if needed and validate contents
- Perform an execution jump from EEPROM to main flash segment FS0
- Set the SWAP bit to logic one.

Setting the SWAP bit maps EEPROM segments EES0/EES1 out of the MCU boot area and maps in a segment of main flash memory, FS7, in its place as shown in Figure 6. This swapping action is implemented by including the SWAP bit in the PSDabel equations for the memory chip select equations.

3.2.2 Memory Map Configuration During Normal Application

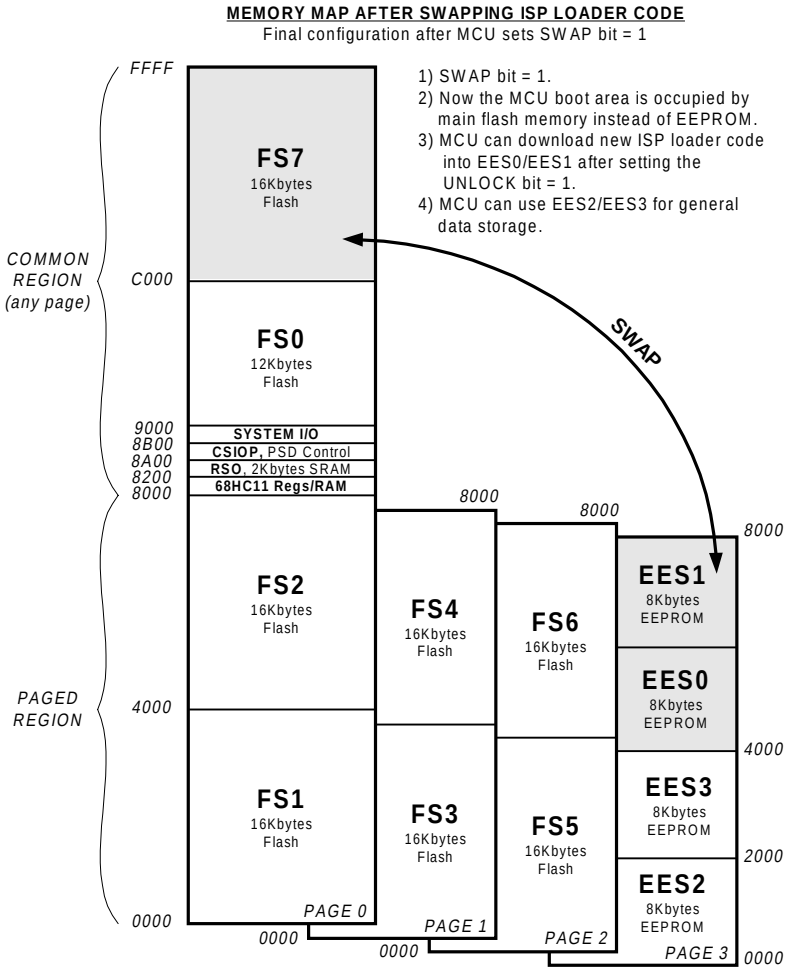


Figure 6 – Memory Map, Enhanced Design, Normal Operation

Figure 6 shows the memory map after the 68HC11 has set the SWAP bit to logic one. At this point, the 68HC11 can download new ISP loader code to EEPROM segments EES0/EES1 if needed. The 68HC11 must first set the UNLOCK bit to logic one before updating ISP loader code. The UNLOCK bit is another one of the eight internal PSD page register bits. In this final configuration, the 68HC11 has available:

- 48K bytes of main flash memory in the common area (9000h-FFFFh)
- 96K bytes of main flash across three pages of lower memory (0000h-7FFFh)
- 2K bytes of SRAM in addition to the SRAM that resides on the 68HC11
- 16K bytes of EEPROM for general data storage on memory page 3
- 16K bytes of EEPROM for ISP loader code.

Each time this 68HC11 system gets reset or goes through a power-on cycle, the PSD presents the memory map of Figure 5 to the MCU, and the boot sequence is repeated.

3.3 PSDsoft Design Entry

The 68HC11 design template that is installed with your copy PSDsoft contains the PSDabel file needed to implement this second design example. Just do the following:

- Open a new project
- Select a PSD813F1
- Select the 68HC11 design template and you have the core of this design (all PSDabel equations are there)
- Tailor the PSDabel template to meet your system needs
- Synthesize and fit the design
- Map the MCU Code (68HC11 firmware)
- Program the part.

When mapping the 68HC11 firmware in the Address Translate utility of PSDsoft for this second design example, you still do not need to specify any HEX or S-Record file for the PSD main flash area. You only need to specify the 68HC11 linker file(s) for the EEPROM area (as in the first simple design) because the 68HC11 will execute code from EEPROM and download flash memory. For reference, the following are the only PSDable statements that differ between the simple design and this enhanced design. No changes to Device Configuration are needed.

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

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

"**** The following memory select equations match the memory maps of
"**** Figures 5 and 6. Notice the use of SWAP and UNLOCK.

fs0 = (address >= ^h9000) & (address <= ^hBFFF) & (page == X);
fs1 = (address >= ^h0000) & (address <= ^h3FFF) & (page == 0);
fs2 = (address >= ^h4000) & (address <= ^h7FFF) & (page == 0);
fs3 = (address >= ^h0000) & (address <= ^h3FFF) & (page == 1);
fs4 = (address >= ^h4000) & (address <= ^h7FFF) & (page == 1);
fs5 = (address >= ^h0000) & (address <= ^h3FFF) & (page == 2);
fs6 = (address >= ^h4000) & (address <= ^h7FFF) & (page == 2);
fs7=((address >= ^h4000)&(address <= ^h7FFF)&(page == 3)& !swap)
   #((address >= ^hC000)&(address <= ^hFFFF)&(page == X)&  swap);

ees0=((address >= ^hC000)&(address <= ^hDFFF)&(page == X)& !swap)
    #((address >= ^h4000)&(address <= ^h5FFF)&(page == 3)& swap & unlock);
ees1=((address >= ^hE000)&(address <= ^hFFFF)&(page == X)& !swap)
    #((address >= ^h6000)&(address <= ^h7FFF)&(page == 3)& swap & unlock);
ees2=(address >= ^h0000)&(address <= ^h1FFF)&(page == 3) ;
ees3=(address >= ^h2000)&(address <= ^h3FFF)&(page == 3) ;
```

4 Conclusion

These examples are just two of an endless number of ways to configure the FLASH PSD for your system. Concurrent memories with a built-in programmable decoder at the segment level offer excellent flexibility. Also, as you have seen with the SWAP and UNLOCK bits, the page register bits do not have to be used just for paging through memory. The ability to grow your system does not require any physical connection changes, as everything is configured internal to the PSD. And finally, the JTAG channel can be used for ISP anytime, and anywhere, with no participation from the MCU. All of these features are cross-checked under the PSDsoft development environment to minimize your effort to design a flash 68HC11 system capable of ISP.

5 References

1) PSD813F Family Data Sheet
2) Application Note 54 for detailed use of the JTAG channel
3) Application Note 55 for details on the CPLD and I/O pins
4) Application Note 57 for details on PSD I/O, GPLD, logic simulation, and PSDsoft features.

Table 1. Document Revision History

Date	Rev.	Description of Revision
Jun-1999	1.0	Document written (AN061) in the WSI format
03-Jan-2002	1.1	Front page, and back two pages, in ST format, added to the PDF file References to Waferscale, WSI, EasyFLASH and PSDsoft 2000 updated to ST, ST, Flash+PSD and PSDsoft Express

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

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.