



**DK900-HC11  
DEVELOPMENT KIT  
For PSD9XX Family of Flash PSDs  
Rev 1.1**



**Contents:**

- ❖ PSDsoft Express - Point and Click Windows based Development Software(from web)
- ❖ PSD9XX Sample
- ❖ DK900-HC11 Eval Board
- ❖ FlashLINK JTAG In-System Programmer (ISP)
- ❖ Ribbon and "Flying-Lead" JTAG cables for FlashLINK
- ❖ PSDload – WIN95/98/NT based UART software for IAP
- ❖ Serial UART cable for PSDload
- ❖ CDROM - Data Book, Software and Videos
- ❖ 110V or 220V Power supply

<b>DK900-HC11 DEVELOPMENT KIT .....</b>	<b>4</b>
<b>Introduction .....</b>	<b>4</b>
A couple of definitions:.....	4
Hardware .....	4
Software .....	4
<b>Detailed Descriptions .....</b>	<b>6</b>
Step-By-Step Instructions for ISP Demo: .....	7
Step-By-Step Instructions for IAP Demo:.....	10
<b>Using DK900-HC11 as a Development Platform for HC11 MCU users:.....</b>	<b>15</b>
Concept.....	15
General Board Description .....	15
Downloading to the Development Board .....	15
JTAG - ISP .....	15
PC Software .....	16
UART Support, PSDload .....	16
Definition of Terms .....	16
Serial Interface .....	16
PSD Architecture .....	16
Functions Available .....	17
<b>Memory Map .....</b>	<b>18</b>
Getting started with PSDload .....	18
A few reads and writes.....	19
Download.....	21
<b>How does this swapping stuff work anyway? .....</b>	<b>21</b>
Macro level.....	21
PSDload address translation.....	24
Micro level.....	24
What really happens.....	25
<b>A detailed look at the IAP example implementation.....</b>	<b>26</b>
Top level functional flow.....	26
<b>How to create your own app for UART Download.....</b>	<b>26</b>
<b>References.....</b>	<b>28</b>
<b>Application notes.....</b>	<b>28</b>
<b>APPENDIX .....</b>	<b>29</b>
<b>Appendix A - Jumper configuration on DK900-HC11 eval board .....</b>	<b>30</b>
<b>Appendix B Development Board Schematic and parts list .....</b>	<b>33</b>
Main Schematic .....	33
Serial Port Schematic .....	34
Power Supply Schematic .....	35
Eval Board Parts List.....	36
<b>Appendix C: FlashLINK Users Manual.....</b>	<b>37</b>
Features .....	37
Overview.....	37
Operating considerations.....	37
FLASHlink pinouts.....	39

Loop back connector schematic .....	42
<b>Appendix D crtsi.s routine.....</b>	<b>43</b>
<b>Appendix E evl_init.c routine .....</b>	<b>45</b>

# DK900-HC11 Development Kit

## Introduction

Congratulations on purchasing Waferscale's DK900-HC11 Development kit. The DK900-HC11 (110V or 220 Volt version) is a low cost kit for evaluating the PSD9xx family of FLASH Programmable System Devices (PSD). The kit is extremely versatile, and can be used in several different modes. In its simplest mode, it can be used to demonstrate the PSD9xx's capability of JTAG In-System Programmability (ISP). After ISP is accomplished, the DK900-HC11 can be set-up to update the program while the MCU is running, called In-Application Programming (IAP). And lastly, HC11 family users can utilize the DK900-HC11 as an evaluation platform for code development.

Regardless of how much development work is done on the DK900-HC11, it functions as an extremely low cost complete JTAG ISP programmer for the PSD9xx family.

A couple of definitions:

**In-System Programming (ISP)**- A JTAG interface (IEEE 1149.1 compliant) is included on the PSD enabling the entire device to be rapidly programmed while soldered to the circuit board ( MAIN FLASH, BOOT FLASH, the PLD, all configuration areas). This requires no MCU participation, so the PSD can be programmed or reprogrammed anytime, anywhere, even while completely blank. The MCU is completely bypassed.

**In-Application Programming (IAP)** – Since two independent FLASH memory arrays are included in the PSD, the MCU can execute code from one memory while erasing and programming the other. Robust product firmware updates in the field are possible over any communication channel (CAN, Ethernet, UART, J1850, etc) using this unique architecture. In this case, all code is updated through the MCU.

## Hardware

- PSD9xx FLASH PSD (Programmable System Device) - see [www.waferscale.com](http://www.waferscale.com) for data sheet.  
PSD913F2 - 1Mb MAIN FLASH(128kx8), 256Kb BOOT FLASH(32kx8), 16Kb SRAM(2kx8)  
-or-  
PSD934F2 - 2Mb MAIN FLASH(256kx8), 256Kb BOOT FLASH(32kx8), 64Kb SRAM(8kx8)
- Eval/Demo Board with HC11 MCU, LCD Display, JTAG and UART ports for ISP/IAP
- FlashLINK JTAG ISP Programmer (uses PC's parallel port)
- Null Modem serial cable (Female-Female)
- Power Supply

## Software

- To assure latest version, check our website often.
  1. PSDsoft Express - Point and Click Windows programming development software. This will install to its own directory.
    - MCU Selection by manufacturer and part number
    - Graphical definition of pin functions
    - Easy creation of memory map
    - JTAG ISP Programming
  2. PSDload - Windows 95/98/NT based UART download software. This will also install to its own directory.
    - In-Application Programming
    - Performs erase, fill, read, write, upload and download of PSD
    - All functions performed through MCU's UART channel.
  3. The distribution disk included with the kit contains the following directories, each with executable code. This code is also available from the web site. For convenience, copy each distribution disk directory to your machine under ... \PSDexpress\dk900-HC11\... . For example, ... \PSDexpress\ dk900-HC11\hwtest\, ... \PSDexpress\ dk900-HC11\demo1\, etc
    - Hwtest. Validates DK900-HC11 board hardware including serial port

- IAP. Initial congratulations and also demonstrates serial port functionality.
- DEMO1. Simple program for IAP demo, displays “have no fear...”

Each full code bundle directory(iap, hwt) contains the following

- \*.zip for the psd
- \*.zip for the C level source code
- readme.txt file containing late breaking information
- \*.obj file suitable for direct PSD programming.

Others(demo1) contains the c code subset appropriate for uart download.

Since the \*.obj file is the natural format needed by PSDsoft for direct programming of the PSD, no unzipping is necessary to change the executing code in the development board. A detailed description of each software bundle is included in the appendix.

The following table is a specific listing of the files and their locations on the distribution disk. Place the files listed in the following table under “root” in the following directory  
 ...PSDExpress\DK900-HC11\<table directory>

The files listed under “root”, are all the files that are needed for the demonstrations in this manual. The remaining archives are source information from which these files were constructed.

Directory	Files	Description
IAP		Full code bundle(c level code and psd files)
	Dk9hciap_p_10.zip	Contains all PSD source files
	Dk9hciap_c_10.zip	Contains all C level code files
	Readme.txt	Late breaking information
Demo		No psd or obj files
	UART1-HC11.zip	Contains all C level code files
	Uart1.hex	Directly downloadable via IAP
	Readme.txt	Late breaking information
Hwtest		Full code bundle(c level code and psd files)
	Dk9hchwt_p_10.zip	Contains all PSD source files
	Dk9hchwt_c_10.zip	Contains all C level code files
	Readme.txt	Late breaking information
	hwt.obj	Duplicate obj file (also in PSD file above)
	hwt.mmf	Memory map file used by PSDload
Root		
	lap_HC11.mmf	Memory map file(from PSDsoft project)
	lap_HC11.psd	Configuration file for PSDload
	lap_HC11.obj	Executing code for IAP demo
	Uart1.hex	IAP demo file for direct download

## Detailed Descriptions

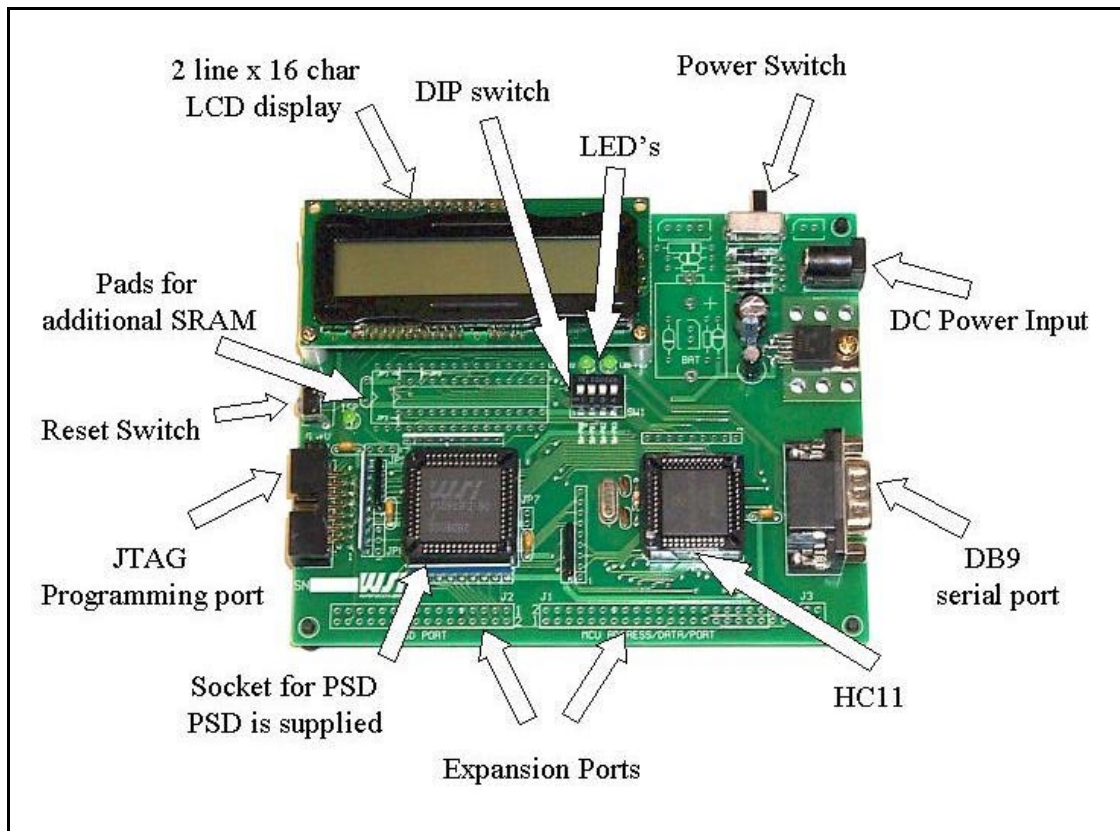
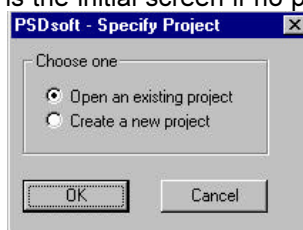


Figure 1 DK900-HC11 Development Board

- **Display** - A two line by 16 character LCD display is included on the Development Board.
- **Power switch**
- **UART Serial Port(male)** - Connected to MCU serial port; used for In-Application Programming
- **HC11 MCU** - Low cost MCU HC11, 44 pin PLCC
- **Socket for PSD9xx** - Blank PSD9xx is supplied, user installs and performs initial JTAG ISP.
- **JTAG programming Port** - Used in conjunction with FlashLINK programmer for ISP.
- **Reset Button** - For resetting the MCU and PSD
- **DIP switch** for IAP control
- **LED's** for functional annunciation
- **Pads for additional SRAM** - The resident PSD9xx contains either 2KB or 8KB SRAM. This site is for additional SRAM.

### Step-By-Step Instructions for ISP Demo:

- Locate and install PSDsoft Express and PSDload. The latest version is always on the web.
- Plug the blank PSD9XX device into the Eval board socket.
- Plug the FlashLINK Programmer into your PC's parallel port and plug in the ribbon cable to the JTAG port on the eval board (for help see the Appendix C, FlashLINK manual).
- Plug in power supply and turn on power. Typically you will observe that the top row of characters are black boxes. This indicates no code is running on the board. You may need to adjust the contrast control located on the left side of the board under the LCD.
- Run PSDsoft Express. Here is the initial screen if no project is open.



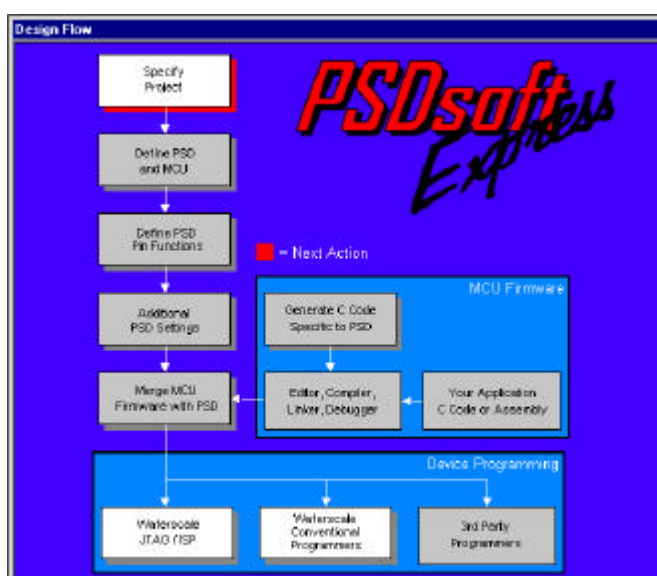
**Figure 2 Opening screen upon PSDsoft Express invocation**

Use cancel at this point since all we need to do is program the PSD and there is no need to create a project. Later, in the "Using the DK900-HC11 as a development platform" section, a further tutorial is given on using PSDsoft Express with the DK900-HC11 for development.



**Figure 3 Invocation reminder screen**

- In the Design Flow (shown below), click on the Waferscale JTAG/ISP button. Bottom row of boxes left side.



**Figure 4 PSDsoft Express flow**

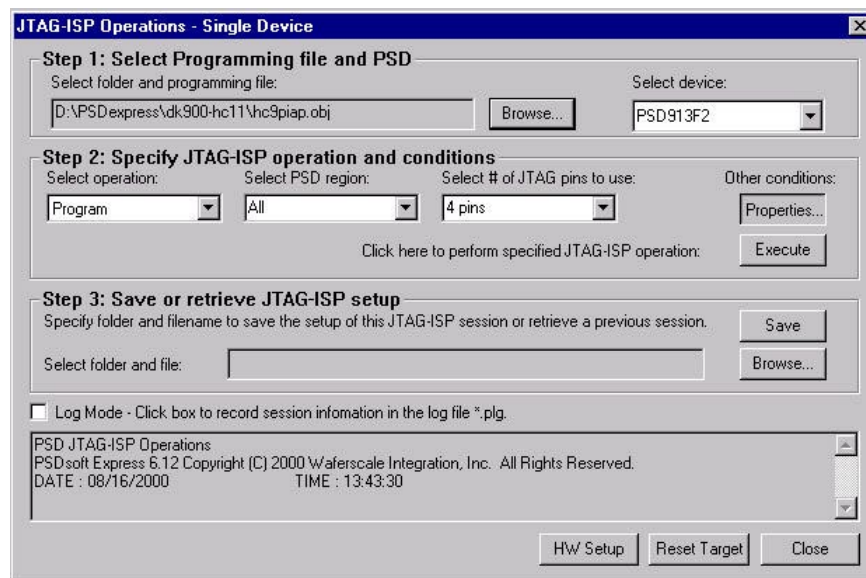
Clicking on this box yields the JTAG Operations- Single device dialog shown below.

The following screen appears inquiring if it's desired to program a single device or multiple devices in the JTAG chain. Select "Only one" as shown below and click OK.



**Figure 5 JTAG-ISP Operations dialog**

- g) Clicking OK brings up the JTAG Operations –Single Device dialog shown in the following figure.
- h) Browse to the \*.obj file shown, and click on this file. The information will be filled in for you.
- i) In Step 2, click Execute.

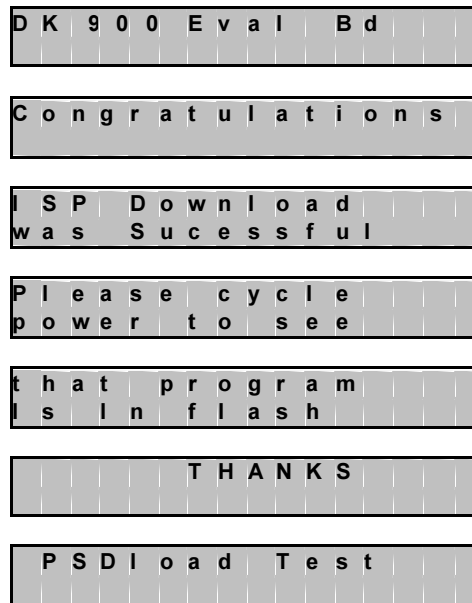


**Figure 6 PSDsoft Express, JTAG Operations dialog**

- j) Observe in the lower pane the JTAG activities that occur while programming your device. When activities stop here, observe the LCD display on the Development Board itself.



k) When the download is completed the Development Board will boot automatically, showing the displays below: This display will sequence one time, ending with the last screen, PSDload Test. This is the screen that needs to be active for the following IAP demo.



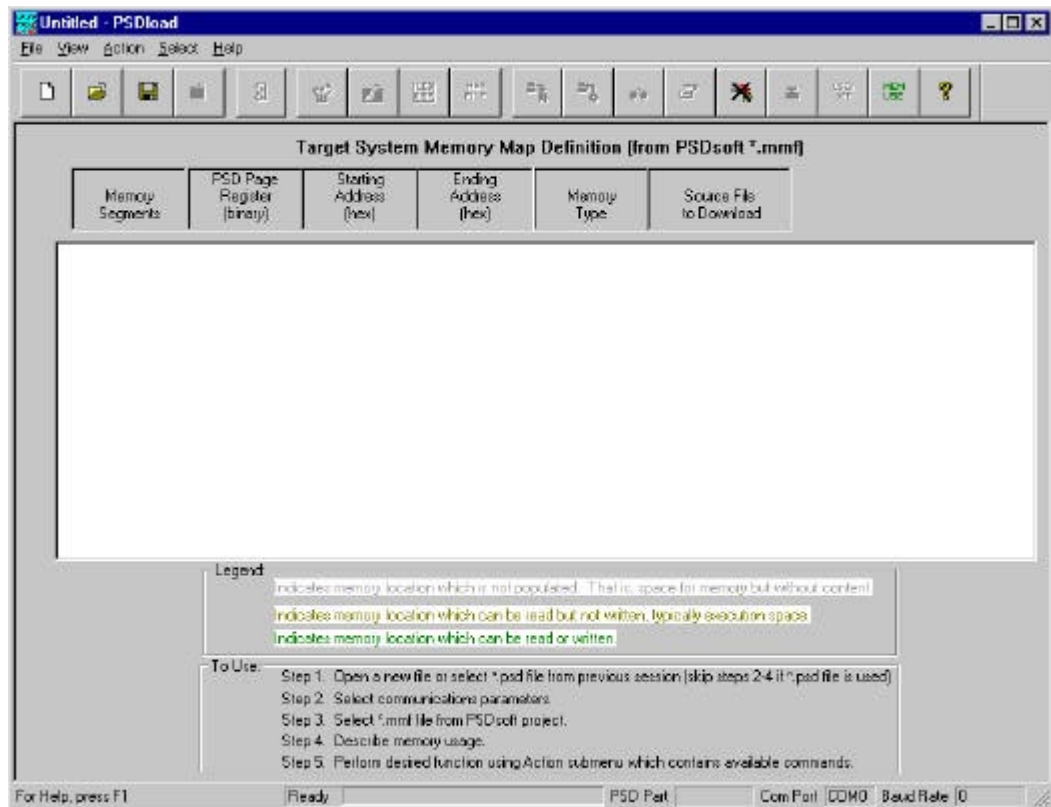
**Figure 7 Eval Board Displays for ISP**

If you power off/on the board, you will see that the display will resequence, confirming that the program and all configuration information are stored in the PSD's non-volatile memory.

- l) For better understanding of the program you may want to examine the following resources:
1. System memory map. Figure 18.
  2. PSDsoft Express project
  3. The file source code (included) to see the flow of the executing code

### Step-By-Step Instructions for IAP Demo:

- a) Now, let's perform an In-Application Programming (IAP). Disconnect the FlashLINK programmer and close PSDsoft Express. Connect the serial cable to the serial port on the PC, and the Dsub connector on the Development Board. Note that this cable is a null modem cable(F-F).
- b) Once the Development Board displays PSDload Test, proceed to the next step.
- c) Invoke PSDload on the PC. At invocation of PSDload, most buttons will be greyed out indicating the PC communications port is not configured as shown below.



**Figure 8 Initial PSDload invocations screen (no comm)**

- d) From within PSDload, choose File, then Open. Find the file as follows; \DK900-HC11\iap\*.psd. This is a configuration file for PSDload that's been constructed for this demo containing the particulars of the design.

- e) Observe the buttons become active (colorful) when this file is selected indicating the communications port is configured. If the button colors do not appear, change the comm port (while retaining 19.2Kbaud) using the Select, Communications submenu or the Comm Port hot button. In this case, you will also be prompted for the \*.mmf file from the same directory. Do not leave this step until you've achieved active buttons as shown below.

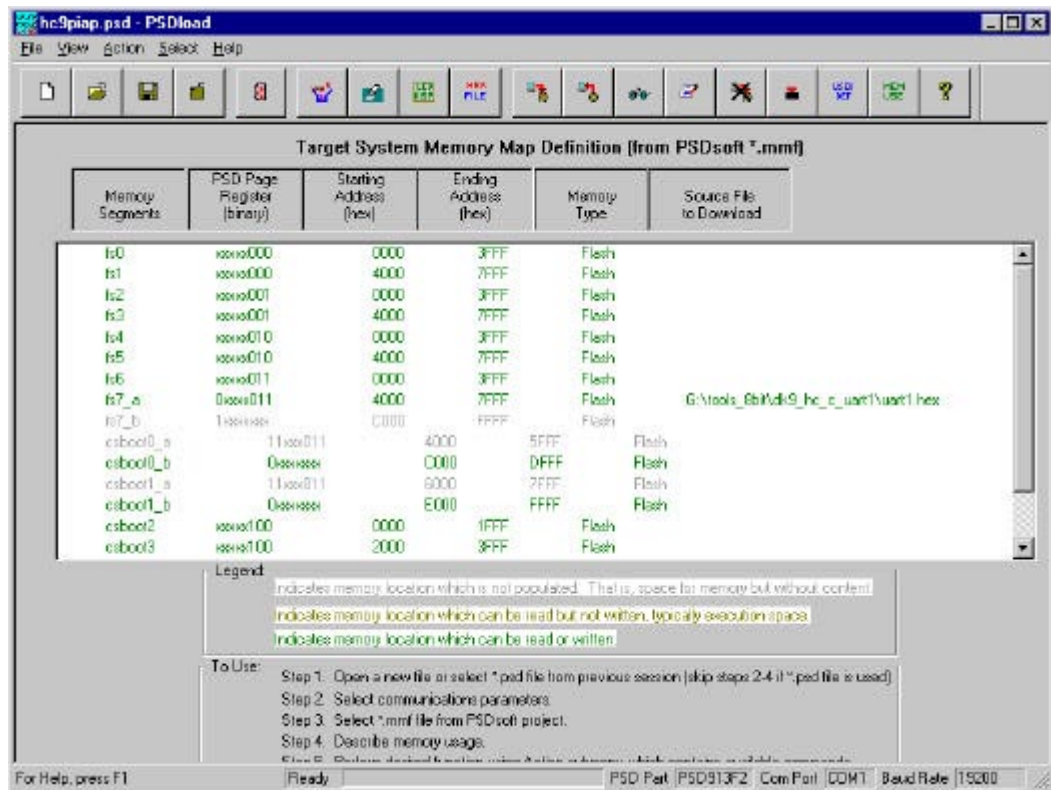


Figure 9 Initial PSDload invocations screen (with comm)

As well as the active buttons, notice that the main window is now populated with the active design. The entries are effectively the equations used to determine the memory map. This information was entered in PSDsoft Express during the design phase of the project and conveyed to PSDload via the \*.psd file (mmf file derivative).

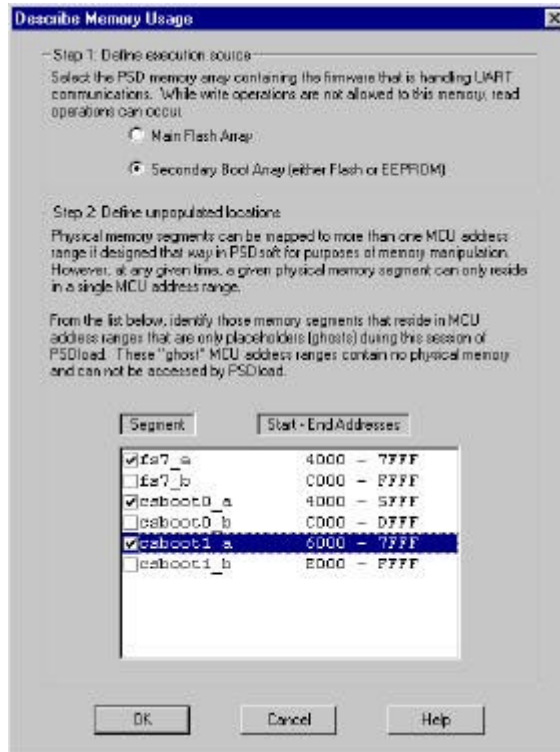
If you must use the \*.mmf file, the following two dialogs will appear;

The first is to setup the communications parameters.



Figure 10 PSDload comm parameter dialog

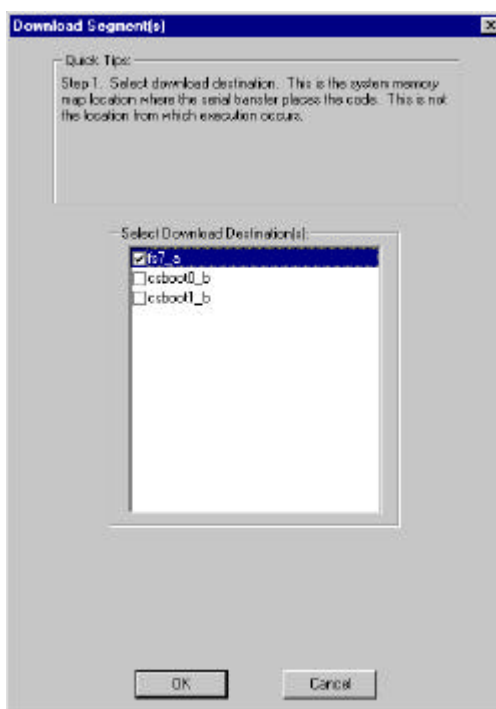
The second is the Describe Memory Usage dialog box. Here the user is to declare how the PSD memory is used as well as which memory locations are unpopulated at the present time(ghosts). The unpopulated locations occur from the desire to swap memory; in these cases there is typically only one resident location for the memory at any particular time. The alternate location also exists and is used after memory is swapped.



**Figure 11 PSDload Describe Memory Usage dialog**

- f) Now, do a Write To Display using the Action, Write Display submenu or the LCD Display hot key. Type something in the dialog, press OK and see if it comes up on the Development Board display. If it does, you've successfully established communications between the PC and Development Board. If this doesn't work, check the following;
1. cable is plugged in
  2. cable is of correct type(Null modem, F-F)
  3. the correct comm port is selected on the PC

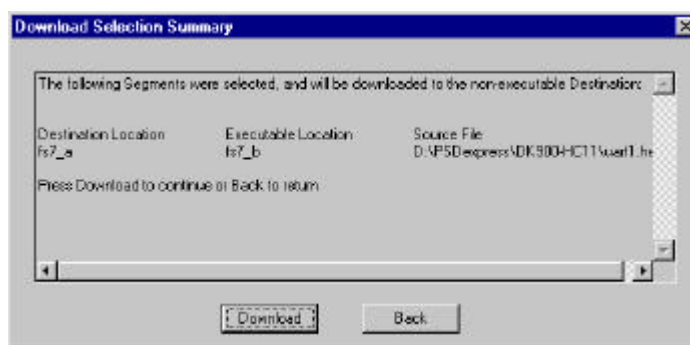
- g) Select Action, download to observe the Download Segments dialog. The following screen will appear.



**Figure 12 Download Segments dialog, PSDload**

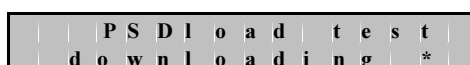
Selecting the download destination (Step 1) to be fs7\_a. Behind the scenes fs7\_b will automatically be selected as the execution location. This will be confirmed in the next screen. Click OK.

- h) Now the Download Selection Summary screen, below, pops up. The intent is to validate the settings chosen in the last screen. You should see fs7\_a as the download destination and fs7\_b as the execution location. Click Download to start the process or back to change.



**Figure 13 Download Summary screen**

- i) Observe the progress bar at the bottom of the PSDload window for activity. Also, observe the display on the Development Board as follows.



**Figure 14 Development Board display for download in process**

During the download, you'll observe the \* character position changing between the following -, \, |, and /. A change from one character to the next occurs with each new packet received by the Development Board. When the download is complete you will see the following.

```

| | P S D | o a d | t e s t |
| | d o w n l o a d | d o n e |

```

**Figure 15 Eval Board display for download complete**


Next, observe the results of the checksum calculation covering the entire downloaded contents as shown below. Of course this was a successful download. This particular display does not persist, so watch the display intently.

```

| | P S D | o a d | t e s t |
| | c h e c k s u m | g o o d |

```

**Figure 16 Eval Board display for checksum validation**

- j) On the Development Board, place SW-PB3 in the on(up) position. This switch is read when the board boots and indicates to the software the desired execution location. On(up) indicates the desire to execute from the main flash area which you just downloaded. Off(down) indicates the desire to continue executing from the default boot area.
- k) Now click the reset button  and observe the Development board display. The program you just downloaded to the main flash area will boot showing the displays listed below.

```

Y o u | h a v e | j u s t | |
p e r f o r m e d | |

```

```

I n - A p p l i c a t i o n |
P r o g r a m m I n g ( I A P ) |

```

```

T h e | M C U | o p e r a t e d |
d u r i n g | d o w n l o a d |

```

```

o f | a | n e w | p r o g r a m |
i n t o | t h e | F l a s h | |

```

```

N o w | p o w e r | c y c l e |
o f f | a n d | o n | t o |

```

```

s e e | t h e | n e w | |
p r o g r a m | e x e c u t e |

```

```

G O O D | J O B ! | | |

```

**Figure 17 Eval Board display sequence for In Application Programming(IAP)**

You can cycle power or press the reset button again to see that this code also persists in non volatile FLASH memory. Note that this code bundle contains less communications capability than the IAP code run previously.

- l) Now, let's reinvoke the original program that was running prior to the IAP download. This is done by placing SW-PB3 in the off(down) position. Now press the reset button and observe the original, ISP program execute again.

## **Using DK900-HC11 as a Development Platform for HC11 MCU users:**

### Concept

The Wafer-scale DK900-HC11 Development Board provides the following capabilities

- Demonstrate design concepts early, optimizing “time to market”
- Jump start user application with proven framework (hardware and software)
- Substitute for user target system until target prototypes are available
- Gives instant platform for testing ISP and IAP demonstration.
- Allows programming the PSD using included Flashlink cable

### General Board Description

The DK900-HC11 Development Board is specific to the HC11 microcontroller family. The board contains an empty socket for the PSD9xx, which can be populated with the included PSD9xx family component. Programming of the PSD is required since the component provided is blank.

### Downloading to the Development Board

Executable code can be downloaded to the Development Board two different ways; via the JTAG (ISP) or via the UART (IAP). Both methods are described and demonstrated in the Step by Step demos for ISP and IAP earlier in this manual.

The ISP programming can program all elements within the PSD (PLD, MAIN FLASH, secondary FLASH memory and all configuration elements) using the 2x7 JTAG connector. That is, all internal PSD components can be programmed via this channel.

The IAP method uses a standard null modem PC serial cable (F-F) and PSDload PC software downloaded from the web as well as the UART of the installed MCU. The IAP method allows only data and executable code to be downloaded over a PC serial link. The PSD, PLD cannot be updated by the IAP channel.

The IAP method is not restricted in destination to the PSD. The destination can be any resources on the Board itself; PSD components or the external SRAM (SRAM not supplied, user must solder in standard 32Kx8 SRAM if you desire more SRAM than is contained in the PSD).

PSDload, a win95/98/NT compatible application for the PC, administers the PC side of the serial link. The protocol used is described in PSDstep document on the web.

### JTAG - ISP

The PSD813F JTAG interface provides the capability of programming all memory within the PSD ( PLD, configuration, MAIN and secondary FLASH memory and BOOT areas ). This interface can also be used to program a completely blank component as JTAG enabled is the default PSD state. See Application Note 54 (AN054) for further description on our CD or website at [www.waferscale.com](http://www.waferscale.com).

The LCD will be non operational during JTAG - ISP, since the MCU is not operating. During this interval, the PSD is not connected to the MCU bus.

Waferscale provides a FlashLINK programmer to facilitate this JTAG programming operation. The FlashLINK programmer connects the PC parallel port to the JTAG connector (2x7) and is driven by PSDsoft Express, the PSD development tool.

## PC Software

### UART Support, PSDload

PSDload is a PC application (WIN95/98/NT) which allows serial communications between the PC and the Waferscale's series of Development Boards. This application utilizes the microcontroller UART on the target system side and a standard serial PC channel. The protocol utilizes commands to perform the following functions on the resident PSD, and potentially, other Development Board resources.

1. Read and write registers, memory
2. Erase and fill memory areas
3. Write to the LCD display
4. Download files from the PC to the target system(any system area)
5. Program the downloaded file into the PSD memory in circuit(MAIN or BOOT areas)
6. Upload files from the PSD or development board resources
7. Reset the target system.

The primary target of this interface is FLASH based PSD's from the standpoint of in circuit programmability. However, the capability is also applicable to the OTP family of PSD's(note that in circuit programming is not available due to the OTP families EPROM base).


### Definition of Terms

A few term definitions will ease the understandability of the document.

- a. *PSDLoad* is the windows interface running on the PC.
- b. *PSDStep* is the protocol used to communicate between the PC and the Evaluation board. (Simple Test and Evaluation Protocol).

### Serial Interface

The connection from the PC to the evaluation board is via a standard 9 pin null modem cable(F-F). The communications parameters are 8 data bits, 1 stop bit and no parity. The interface uses simple three wire (TX, Rx and GND) RS-232 with full-duplex operations. Flow control is accomplished via software handshaking incorporated into the protocol (this is not XON XOFF). The baud rate of PSDload is selectable from 4.8k to 56k but the HC11 board is presently restricted to 19.2kbaud. Software flow control is used in order to minimize the master/slave physical connections.

Each command sent from PSDload is intended to elicit a response from the Development Board. This handshake is used to verify a valid receipt of the transaction. Two methods exist to terminate this handshake if it should become disrupted for any reason; the first is a hot key inside PSDload,  and the second is a communications timeout parameter entered on comm invocation screen.

### PSD Architecture

The PSD contains several different blocks of memory which vary within each family and between the families. These encompass the following memory types; EPROM, FLASH, EEPROM, SRAM, and registers. Generically these memory blocks are termed a memory "**region**". The PSD913 contains 128kx8 FLASH, 32kx8 FLASH and 2kx8 sram.

PSDLoad must be aware of how these regions map into the system memory as all operations occur based on addresses associated with the system memory. The system memory map is determined using the development tool, PSDsoft Express. This information is provided in the form of a \*.mmf file automatically generated from PSDsoft Express and requested by PSDload at invocation. PSDload utilizes this information to portray the system memory map to the user and construct commands to send to the Eval Board. The \*.psd file, once constructed, contains the information in the \*.mmf file.

Since the system memory map is utilized to achieve the download, the PLD within the PSD must have been programmed prior to a serial download attempt. PLD programming is accomplished via
















either the JTAG interface or with a conventional parallel programmer, both of which are external to PSDstep/PSDload.

Note that the addressing scheme used by PSDload is a different addressing scheme than is used by PSDPro(parallel programmer) and/or FLASHlink. PSDload uses the system addresses; that is, the addresses generated by the microcontroller in the system and correlated by the linker. PSDsoft Express and FLASHlink use direct addresses (flat 24 bit memory space), that are independent of the PLD and the end system application.

The FLASH region is erased by sector or bulk(entire FLASH) and programmed byte by byte. The EEPROM region does not require erase and may be written by byte or by page. Which technology resides in the BOOT area depends on the device you have chosen. For example, the F1 has EEPROM in the BOOT area. An unambiguous method to determine the BOOT area technology is by reading the flash ID.

### Functions Available

Along with the standard windows controls of save , open , new , close  and help  and the serial port controls , the following are available. These functions are can be accessed either from a pull down menu (Action) or from the shown hot keys.

Function		Description
Erase		Erase FLASH(by segment or bulk)
Fill		Fill area
Download		Download new file to memory
Upload		Upload file from memory
Read		Read area(restricted to 160 bytes)
Write memory		Write area(restricted to 160 bytes)
Write display		Write to display (on dev board)
Reset board		Reset development board
User data		Encapsulate user specific commands
Source file entry		Enter source file to be downloaded
Describe memory usage		User interface aid

**Table 1 PSDload Commands**

## Memory Map

Before we really get started using PSDload, we should be familiar with the system memory map. Recall that all PSDload operations occur by using addresses in this map. The application is set up to take advantage of the entire memory space of the 9xx using paging techniques even though the MAIN FLASH is initially unpopulated(fs0..7). CSIOP is the base of the register band used to communicate with the PSD using the microcontroller.

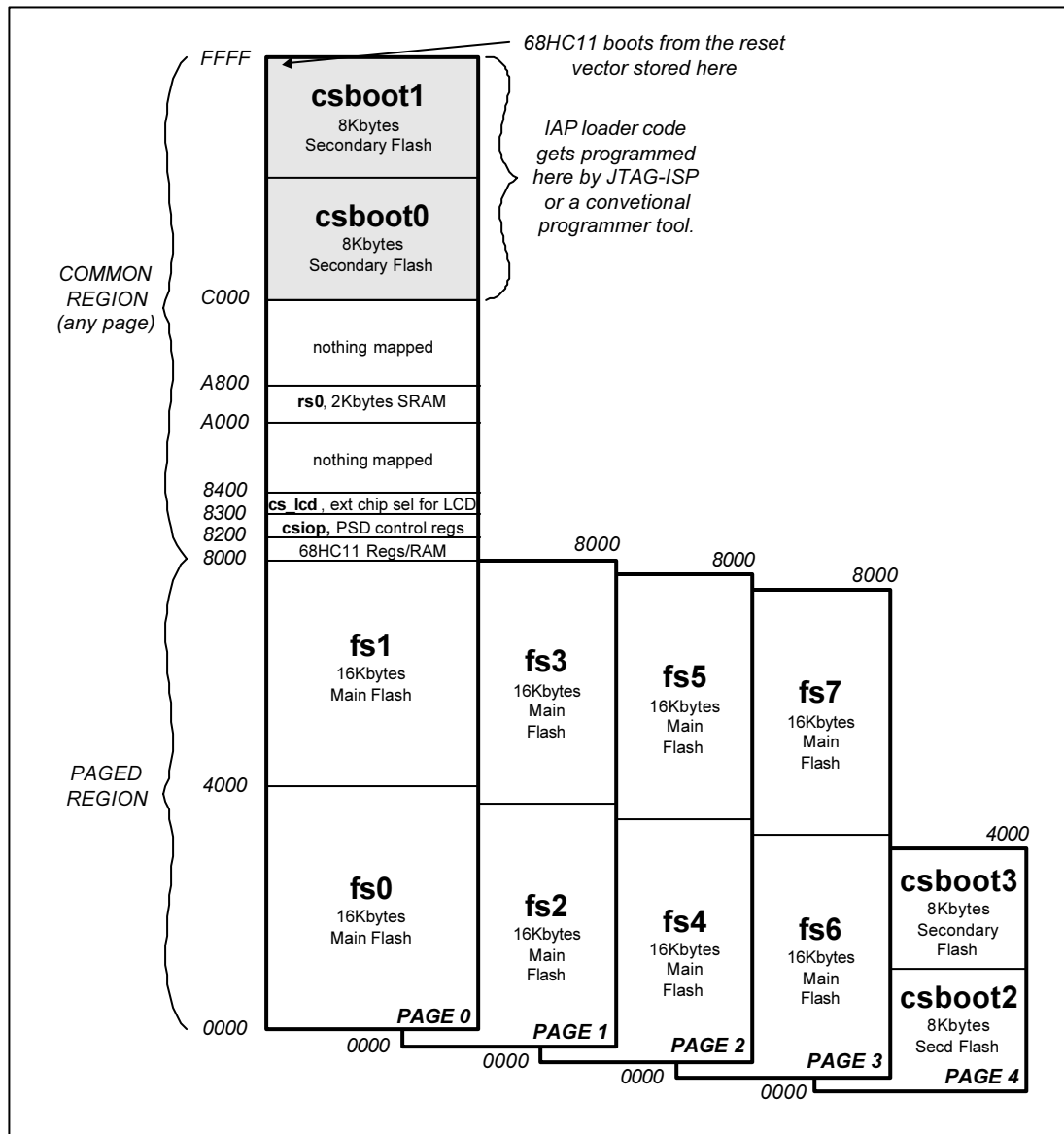


Figure 18 Memory Map of Eval Board

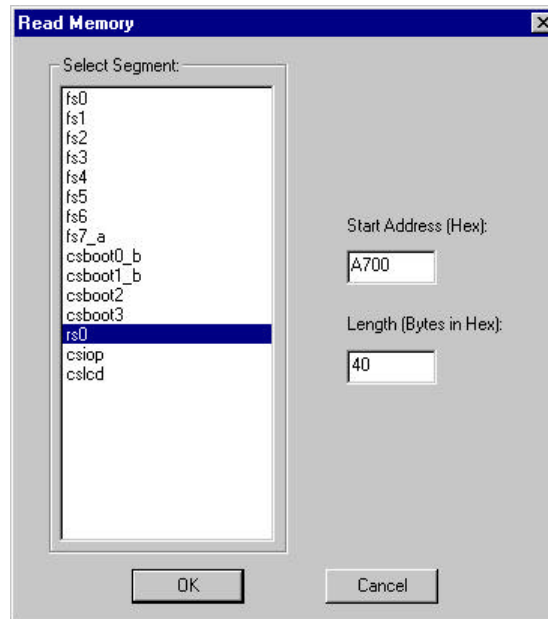
## Getting started with PSDload

Since you've done this before in the previous step by step demo section, we'll start with PSDload being active. To establish a baseline communications, write something to the display by selecting the Action submenu and then Write Display. A dialog will pop up allowing you to enter text. After you have completed the message, click on the Write button. PSDload will send out the message. After the message has been received, the development board responds by displaying the message and sending a response back to

PSDload. This response prompts PSDload to display an “operation completed” dialog to the user on the PC. All transactions between PSDload and the development board use this handshaking scheme to maintain continuity of the communications link.

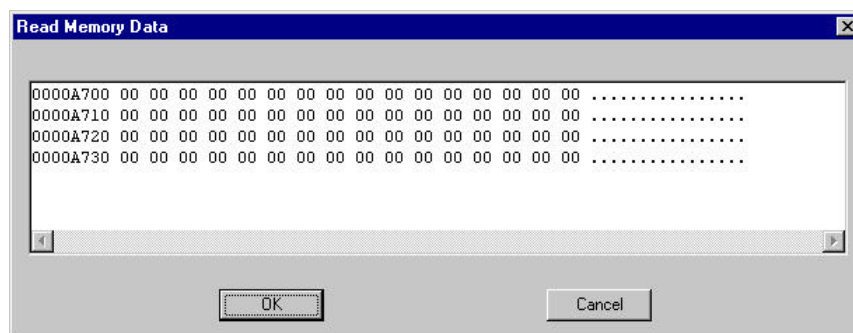
A few reads and writes

Now let's do a few read/write operations. We want to be careful in the selection of the address that we're writing to, so we won't interfere with the execution of the present application. Do a read memory of RS0 by selecting RS0 in the Select Segment field. When you select RS0, the start address of 0xA000 is populated in the start address field. Modify this field to 0xA700 for the purposes of this test and enter a length of 40h in the Length field. The following figure shows the dialog prior to clicking OK. Click OK.



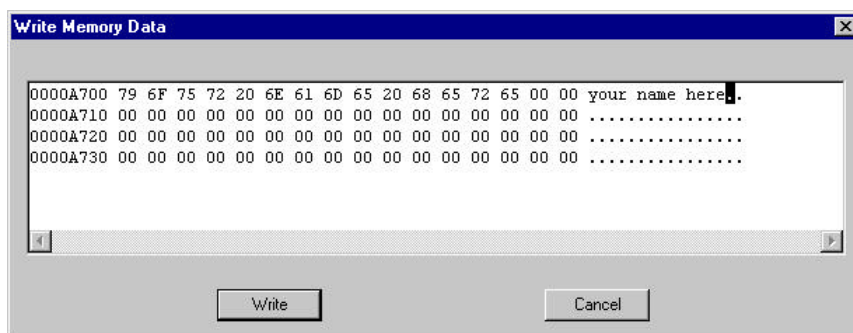
**Figure 19 Read Memory dialog in PSDload**

A dialog will pop up with the contents of the memory in both hex (left side) and asc formats(right side) as shown below.



**Figure 20 Read Memory Data in PSDload**

The contents appear as zeros as this is initialized volatile memory. Now, do a write of the same locations. You'll see the same box (read memory data) come up as PSDload always does a read prior to a write, but now the box is editable. You can edit in either the hex display or the asc display and the conversion to hex happens automatically as shown below. Try typing your name or something identifiable into the ASC field. You will notice the hex bytes changing as you type.

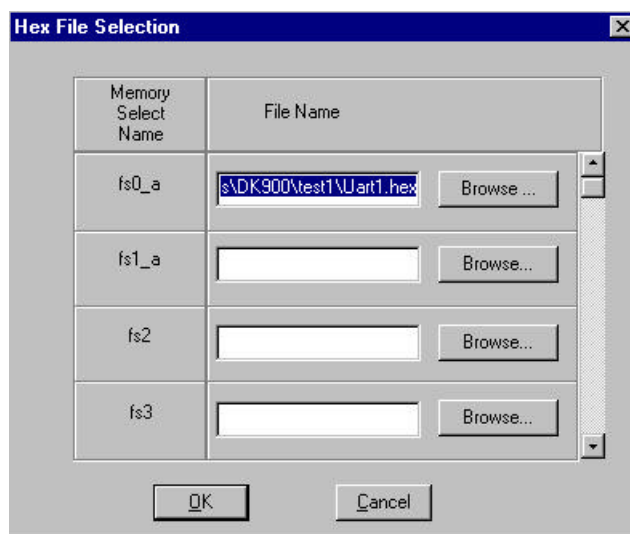


**Figure 21 Write Memory Data dialog in PSDload**

Click Write. After the response, read it again to see if it's really there. Cycle power and reread. You should observe the data you entered is no longer there, indicating the fact that the information was stored in volatile SRAM which is volatile.

Now let's repeat these operation using FLASH. The dialogs are the same except for the FLASH selection so they won't be repeated. Since it's not used in the application yet, no harm will be done. Select Write Memory and, in the write dialog, select fs7 which starts at 0x0000. Read 40h bytes of the area. You will notice that instead of the characters you observed in the above example using SRAM, you now get 0xff in all locations. This is because the FLASH is blank. Type in something and click write. Now do a read to see if it's there. Type in something else of lesser length than above and read it back again. You will notice that the entire first message is gone. This is because the FLASH was erased prior to the last write. Also, FLASH is erased by sector; that is, the entire sector must be erased before you can rewrite the locations of interest. You can also cycle power on the target to see that the information is held in non volatile form. Also try ERASE which only works on the non volatile areas.

When you're ready to do a download, one of the operations that's needed is the selection of the source file. This screen available from the Action submenu or the **HEX FILE** button. After exiting this screen, the selected hex file shows up in the main mmf display. The same file and path are stored in the \*.psd file when it's saved.



**Figure 22 Hex File Selection screen, PSDload**

Download

You've already done this in the earlier demo portion of this document so lets dig a bit deeper to see what makes it all work. See the following section.

### ***How does this swapping stuff work anyway?***

#### **Macro level**

First, let's take a look at how the memory map changes during the transitional operations from one executable code bundle to the other. The internal PSD resource of the PAGE register is used to affect this change in addition to the PLD equations described. We will also use a non volatile resource to carry through a power off condition. This resource will be called NVswap and can consist of any of the following (spare non volatile segment in the PSD, board level switch, etc). In our case, Nvswap will be the board mounted DIP switch.

The PAGE register (csiop+0xE0, 8 bits) is traditionally used to control memory paging, but we also use it to control memory addresses, as presented to the microcontroller, using 1 or more bits. This register can be read or written by the microcontroller. The initial value of the PAGE register is 0 at power up and is the register is volatile. The swap bit is the msb of the PAGE register.

Following is a step by step procedure to boot from one code and change, on the fly, to another. Certainly, there is more setup detail involved (described later under Micro level), but this is the essential procedure.

1. Power up system with default memory map. swap=0 (PAGE register msb)
2. Write swap=1 (PAGE register msb)

These steps are further depicted graphically in the following figures.

Here's the memory map at power up. Note that we are executing from CSBOOT0/1. During the IAP download, the complete new executable, including the vector table, is copied into FS7. During this time the swap bit in the PAGE register is 0.

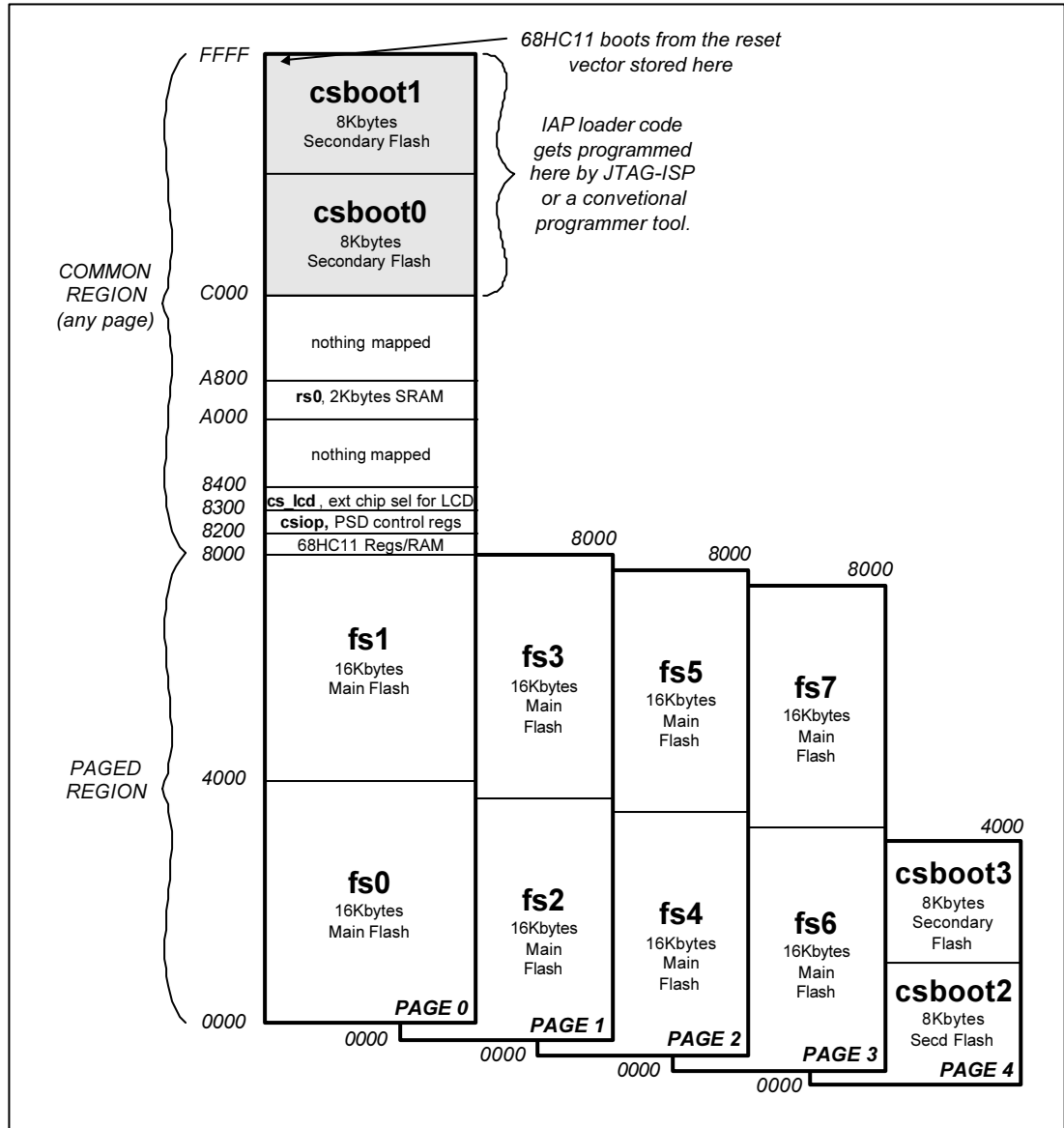


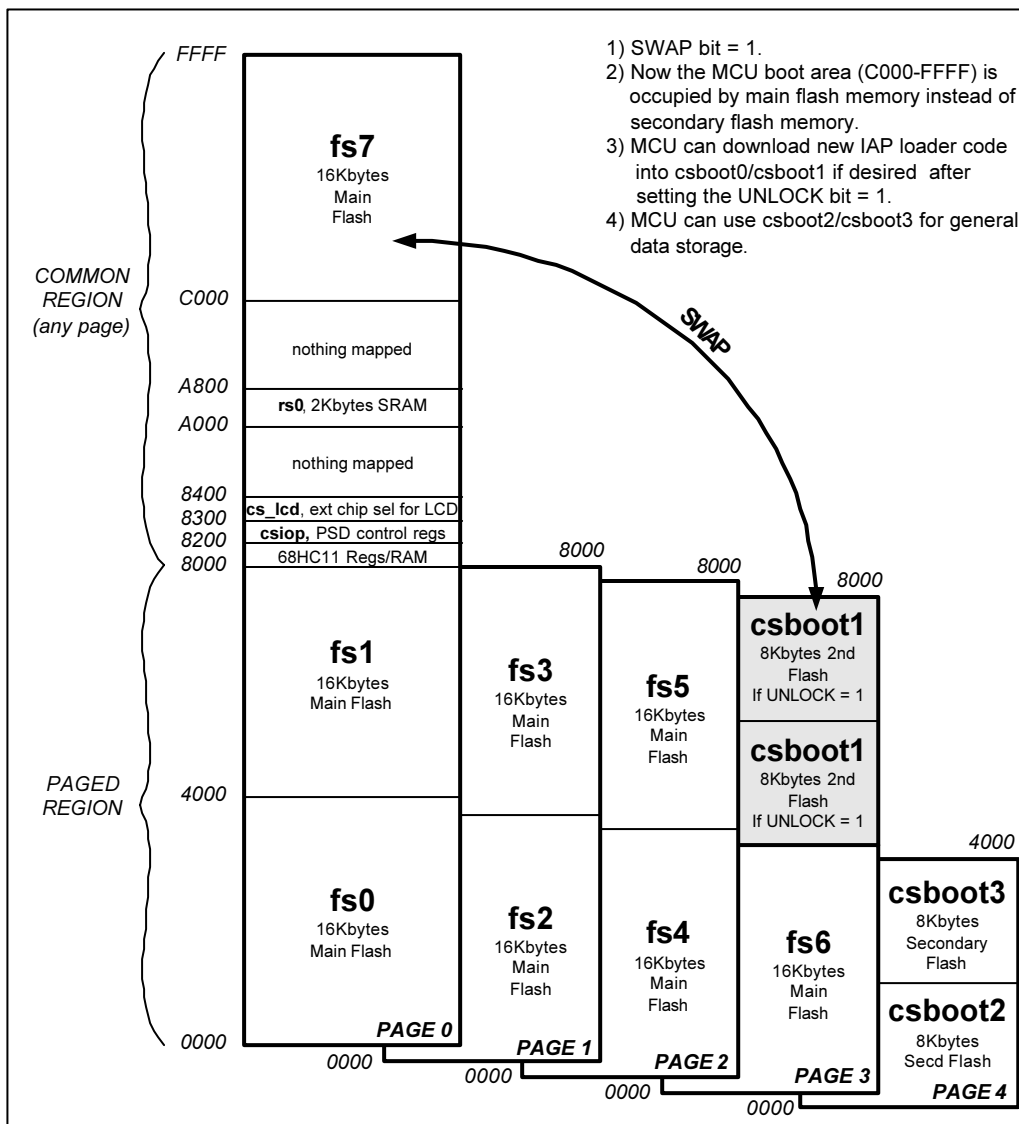
Figure 23 Memory map at power up, NVswap=0

Now, let's set a flag (NVswap) to indicate we want to run the code in FS7 the next time we power up. This flag is non volatile so that, if power is removed, the system knows how it's desired to power up.

Cycle power to the unit. We have embedded code running in the initialization routine to read the state of NVswap and to write that value into the PAGE register (msb, swap) at power up. If swap = 0, the code bundle residing in CSBOOT0/1 continues to run. If swap = 1, we perform the memory manipulations depicted in the next figure.

For purposes of this example, let's assume NVswap = 1 indicating the desire to execute from the MAIN FLASH memory. At this point, the code residing in CSBOOT0/1 is still running.

Next, we write to the PAGE register. This action changes the system location where the code appears to the microcontroller moving FS7 to 0xC000 and CSBOOT to 0x4000 as shown below.



**Figure 24 Memory locations after step 3 of memory swap**

After this write operation is complete, the very next instruction is fetched from FS7. Execution continues from FS7 until the next time the system is powered down. At the same time, the CSBOOT area is moved.

With the NVswap bit set ( SW-PB3 on, up), this sequence will occur every time power is applied.

As a short review, let's talk about what just transpired. We booted from one memory(CSBOOT), then, at full speed and without the awareness of the microcontroller, we swapped execution from that memory to FS7. The new memory contents contained a substantially different set of code that picked up immediately. It sounds like a stretch, but really isn't.

## PSDload address translation

If you look closely at the memory map, you will observe that the system addresses are not the same for fs7 and csboot0/1. However, when these respective code bundles execute, they must occupy the same address range. Else, the mcu could not find the reset vector, boot and execute the code.

More specifically, when a download occurs, the downloaded hexfile contains addresses appropriate for execution that, in this case is 0xC000-0xFFFF for FS7. However, we download this data to 0x4000 –7FFF. If the downloaded addresses of the hex file start at 0xC000, how does the data get to 0x4000? Then, after download is complete , how does the code get in high memory for execution? PSDload does an address translation on every data byte in the hexfile; that is, it changes the addresses according to the download destination of 0x4000-7FFF using the following equation.

$$\text{Destination address} = \text{hex file address} + \text{destination base} - \text{execution base.}$$

For this HC11 family example, code exe(hex file) is 0xC123, dest base = 0x4000, exe base = 0xC000  
Download destination = C123 + 4000 – C000 = 0x4123

While this equation may look like overkill for this example, it allows transparent PSDload operation regardless if the MCU boots from high memory(HC11) or low memory (8031).

Now that we've described this level of operation, lets take a bit closer look at the detailed sequence that occurs between steps 1 and 2; that is, as the memory is physically swapped.

### Micro level

You might ask how can this happen without knowledge of the microcontroller? You might be wondering how can this all happen with the microcontroller running full speed? It all happens due to the chip select decoding.

Here are the equations that control the memory map before, after and during the transition. For clarity we'll only consider the segments of interest for this application which are FS7 and CSBOOT0/1. Certainly the same techniques apply with paging when using the remaining FLASH segments.

```
CSBOOT0 = ((address >= ^hC000) & (address <= ^hDFFF) & !swap )
          # ((address >= ^h4000) & (address >= ^h5FFF) & swap );

CSBOOT1 = ((address >= ^hE000) & (address <= ^hFFFF) & !swap )
          # ((address >= ^h6000) & (address >= ^h7FFF) & swap );

FS7    =   ((address >= ^h4000) & (address <= ^h7FFF) & !swap)
          # ((address >= ^hC000) & (address <= ^hFFFF) & swap );
```

The above equation tells us that FS7 can show up in either of two places; 0x4000-0x7FFF or 0xC000-0xFFFF. The choice of which location is used is based on the variable **swap**, a single bit in the PAGE register. The swap bit is the most significant bit of the PAGE register (csiop+0xE0). The PAGE register is 0 at power up. So, if swap=0 at power up, then fs7 must appear at 4000-7FFF and CSBOOT0 is at C000-0xDFFF and CSBOOT1 is at 0xE000-FFFF. In this case, code executes from CSBOOT0 and CSBOOT1 as a default. See previous figure for a graphical representation.



After the memory contortions are completed swap=1. We end up with the memory map of Figure 24 with FS7 at 0xC000 (execution position) and CSBOOT at 0x4000.

The location where the vector table is located is generally referred to as the **execution location** in this document. That is, this is where code needs to reside so that the microcontroller can find it easily. This method of **hardware relocation** is very convenient due to the integrated components within the PSD. Alternative methods use software relocation to accomplish the same task.

As an overview, consider this. What the microcontroller needs from the memory is really pretty simple. The memory needs to provide the sequential instructions for the task at hand. The microcontroller generates the address and the memory provides the instruction. Then the microcontroller executes that instruction. This occurs over and over again. If a jump needs to occur, the microcontroller provides a new address to the memory. Same with a subroutine return, the microcontroller gets the return address from the stack.

#### What really happens

There is a subtlety involved in the transfer of execution described above. This subtlety is because the MCU really doesn't know the source of the instruction bytes; boot area or main FLASH. All the MCU knows is that valid instructions on valid address boundaries are presented on the bus when the MCU needs them. Then the MCU executes the instruction and generates the next address. The key element involved is the generation of the address by the MCU.

To understand this critical transfer of control, let's examine the instruction by instruction transition from one memory to the other. After the reset signal is deasserted, the MCU is executing from the csboot area normally. This continues until the swap bit is written, moving FS7 into the execution location (0xC000-0xFFFF). At this same time, csboot area is moved to 0x4000-7FFF. At this point, the MCU is generating the next address from the instruction received from the csboot area. However, the next instruction will come from the FS7 area. This next instruction fetch must be appropriate to maintain the program flow. That is, the next instruction must be received by the MCU on an instruction boundary and be appropriate for the program flow. In addition, any issues with the stack and stack pointer must be resolved so program flow can continue (subroutine return addresses, temporary variables, etc.).

The method we've used to ensure correct operation is to place identical code at identical locations in both applications through the point of the swap. After the point of the swap, the code bundles can diverge without problems.

## ***A detailed look at the IAP example implementation***

The previous example uses two code bundles; IAP\_6811 and UART. The discussion will take the same course as the previous demos and explain what occurs behind the scenes. Let's take a walk through the critical code to see how it works.

### Top level functional flow

Let's start with the top level flow. After the reset vector is fetched and executed the routine `evl_init.c` runs. This is where the main action occurs (`evl_init.c`) that resides in both IAP\_6811 and UART1 applications.

`Evl_init.c` contains a routine `Run_Execution_Source` that determines where execution resides. The flow of this routine is listed below:

```
Read_dip switch
If (dip_switch = up)
    Execution_Main
If (dip_switch = down)
    Execution_Boot
```

Of course, the execution from main flash will only occur properly if appropriate code is resident in main flash. For Main flash execution, the swap bit is written and execution continues in the main flash area.

Now, let's assume that we are executing from main flash (DIPSW-PB3 = up) and wish to revert to the original code for execution. All we need to do is place DIPSW3-PB3 in the down position and hit the reset button. The `csboot0/1` code starts out, then `evl_init.c` runs, leaving the swap =0 resulting in executing remaining in the boot area.

As you can observe from the above discussion, the manipulations at the top level to accomplish the traditional boot loader function using hardware techniques are straightforward.

## ***How to create your own app for UART Download***

Typically, getting a single application to run is relatively straightforward since the linker (and user) ensure all references are resolved when the executable file is created. Setting up your application for UART download takes only a little more coordination between the two executable files; specifically in the area of code placement and using the linker. Typically only minor code changes are required.

First, a quick review of what we're trying to do. We are attempting to smoothly transition from one running application to another. The microcontroller will initiate the action, but be substantially unaware of its occurrence. We are going to accomplish this by manipulation of the code memory presented to the microcontroller.

Certainly this will take some coordination between the two applications, but probably not as much as you might initially think. To make things easier, we'll do this critical transition just after a system reset as described in "A detailed look at the IAP example implementation" section earlier in this document. This reset can be initiated either through software or hardware means based on the method(s) available in your system.

You can tailor the scheme as described earlier in this document, or utilize the key generic elements listed below;

1. Startup routine placed identically in both applications(`csrtsi.s`)
2. Flag indicating desire to jump from BOOT memory to main memory. This is the variable (`NVswap= 1`)described earlier in this document.
3. Method to tell system of desire to return from main memory to BOOT memory. This is the variable (`Nvswap=0`) also.

When using a PSD, we recommend the use of our `crtsi.s` routine or an equivalent included in the code bundles. The code placement issues are serviced in the `*.lkf` file also included in the code bundles.

The code content and positioning **after** the initialization code ( crt1.s) need have no correlation between the two applications. That is, the linker can be allowed to handle post initialization code without ill effects to the desired swapping operation. This element eases the creation of compatible applications as the critical code placement is handled by this single file.

## **References**

IEEE Std 1149.1-1990 IEEE Test Access Port and Boundary Scan Architecture  
PSDSoft Express User Manual  
Flashlink User Manual

## **Application notes**

AN054 JTAG Information  
AN067 Design Tutorial for 8032/PSD9XX

## **Appendix**

## Appendix A - Jumper configuration on DK900-HC11 eval board

### 9. PSD's power consumption measurement point (JP7)

Two pins of this jumper are already connected using copper trace. To measure PSD's power consumption, connect DMM to these two pins after cutting pre-connected copper trace on PCB.

The measured PSD's current will be,

$$I_{cc} = \text{PSD } I_{cc} + \text{PSD } I_c (\text{I/O ports}) + \text{MCU Bus leakage } I_c$$

This measurement could be different from result of calculation according to formula in data sheet. To measure correct value, make sure all of other terms should be zero.

### 10. PC1 TCK input option (JP8)

Default setting of this jumper is non-buffered

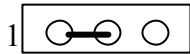


1-2 : direct connection to FlashLink TCK output

2-3 : buffered TCK output from HC14 on board

### 11. PSD SRAM Battery Vstby input to PC2 (JP9)

Default setting of this jumper is weakly pulled up (disabled Vstby input from battery)



1-2 : connect PC2 to battery on board

2-3 : PC2 is weakly pulled up through 100K ohm

## 12. SRAM (1M/256Kb) / ST's TimerKeeper SRAM Expansion

(a) 0.3" pitch 256Kb SRAM expansion site (28PIN)

A14	A14	VCC	VCC
A12	A12 /WE	PB5	(/WR)
PA7 (A7)	A7	A13	A13
PA6 (A6)	A6	A8	A8
PA5 (A5)	A5	A9	A9
PA4 (A4)	A4	A11	A11
PA3 (A3)	A3 /OE	PB4	(/RD)
PA2 (A2)	A2	A10	A10
PA1 (A1)	A1 /CS	PB6	(/RAM_CS)
PA0 (A0)	A0	D7	AD7
AD0	D0	D6	AD6
AD1	D1	D5	AD5
AD2	D2	D4	AD4
GND		D3	AD3

(b) 0.6" pitch 1Mb/256Kb SRAM or ST's TimeKeeper SRAM expansion site (32PIN)

NC		VCC	
PB1	A16	A15	PB0
JP3 (A14)	A14	CS2	JP1 (VCC)
A12	A12 /WE	PB5	(/WR)
PA7 (A7)	A7	A13	JP2 (A13)
PA6 (A6)	A6	A8	A8
PA5 (A5)	A5	A9	A9
PA4 (A4)	A4	A11	A11
PA3 (A3)	A3 /OE	PB4	(/RD)
PA2 (A2)	A2	A10	A10
PA1 (A1)	A1	CS1	PB6 (/RAM_CS)
PA0 (A0)	A0	D7	AD7
AD0	D0	D6	AD6
AD1	D1	D5	AD5
AD2	D2	D4	AD4
GND		D3	AD3

\*) PB0, 1 can be used for banked SRAM

(c) Jumper settings for 0.6" pitch devices

	256Kb SRAM	1Mb SRAM	ST M48T59	ST M48T129
JP1	ON ( VCC )	ON ( CS2-VCC )	ON ( VCC )	OFF ( /IRQ/FT )
JP2	ON ( A13 )	ON ( A13 )	OFF ( /IRQ/FT )	ON ( A13 )
JP3	ON ( A14 )	ON ( A14 )	OFF ( /RST )	ON ( A14 )

\*) Default : All JP1 -3 are OFF

### 13. System expansion connectors (J1,J2,J3)

J1 (68HC11Dx)		1	2
GND			
AD0			
AD1	E		
AD2	MODA		
AD3	MODB		
AD4	A8		
AD5	A9		
AD	A10		
AD7	A11		
/XIRQ	A12		
R W	A13		
AS	A14		
/RESET	A15		
/IRQ	PA0		
PD0	PA1		
PD1	PA2		
PD2	PA3		
PD3	PA5		
PD4	PA7		
PD5	JP4 (VCC)		

J2 (PSD8/9xx)		1	2
PA0	PA1		
PA2	PA3		
PA4	PA5		
PA6	PA7		
GND	GND		
PB0	PB1		
PB2	PB3		
PB4	PB5		
PB6	PB7		
GND	GND		
PC0	PC1		
PC2	PC3		
PC4	PC5		
PC6	PC7		
CNTL2	/JEN		
PD1	PD2		

\* /JEN is connected to FlashLink  
 \* JP4 is OPEN as default.

J3		1	2
	PA4		
	PA5		

### 14. Others

#### (a) Battery power connector and re-charging circuit

When using re-chargeable battery as power source, you can use prepared normal charging circuit in this kit. To use this charging circuit, assemble a diode with register that has proper value.  
 (Recommended battery is NiCD 10.8V)

\*) Do not use charging circuit for Manganese, Lithium or Hydrargyrum batteries.

#### (b) Other power source input connector

To use other power sources (SMPS, Transformer, ...), a connector is prepared in this kit.

(Recommended power source is AC/DC adapter, over 9V, output can be AC or DC)

#### (c) Re-charging circuit for Vstby Battery

When using re-chargeable battery as Vstby source, you can use prepared normal charging circuit in this kit. To use this charging circuit, assemble a diode with register that has proper value.

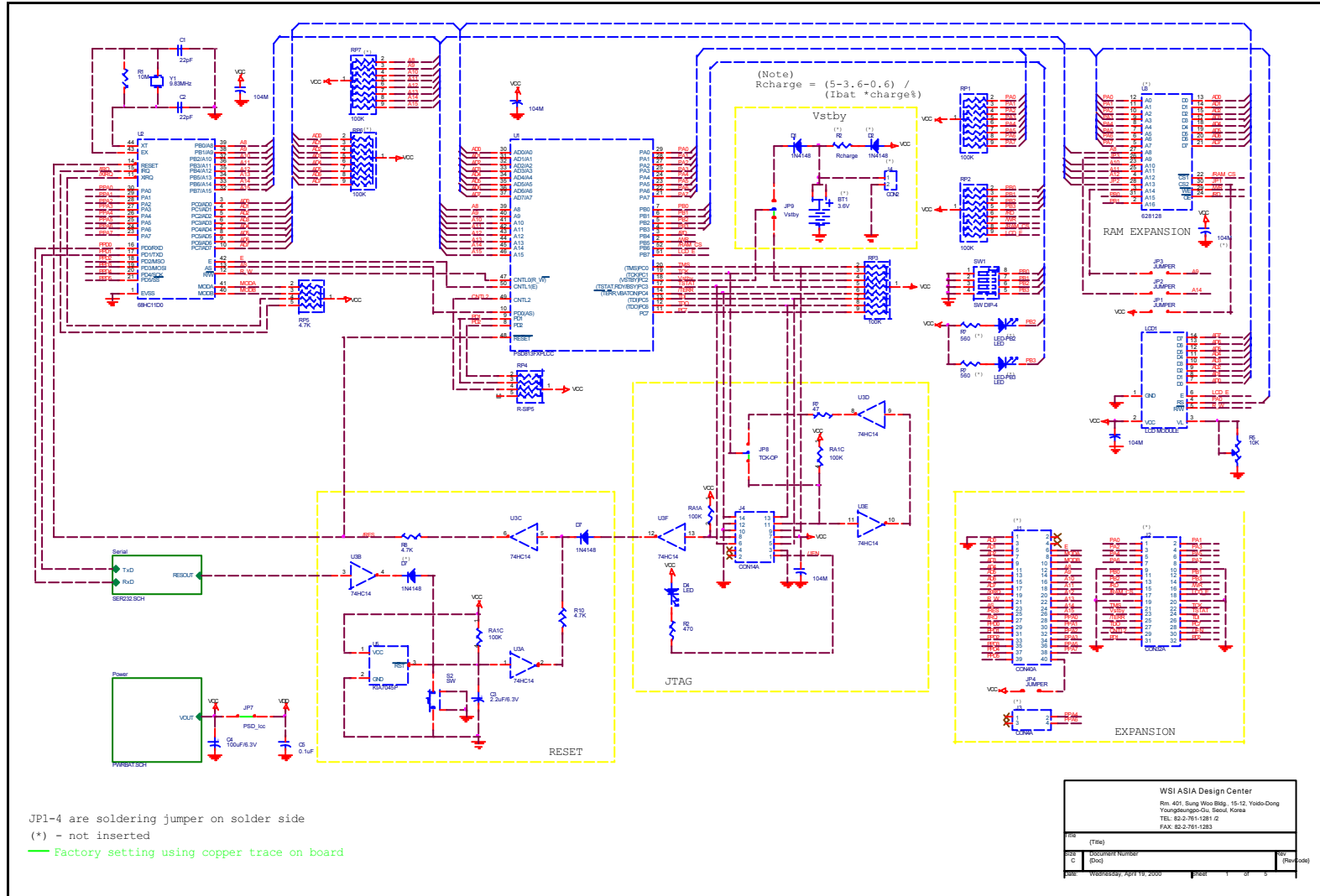
(Recommended battery is NiCD 3.6V)

\*) Do not use charging circuit for Manganese, Lithium or Hydrargyrum batteries.



# Appendix B Development Board Schematic and parts list

## Main Schematic

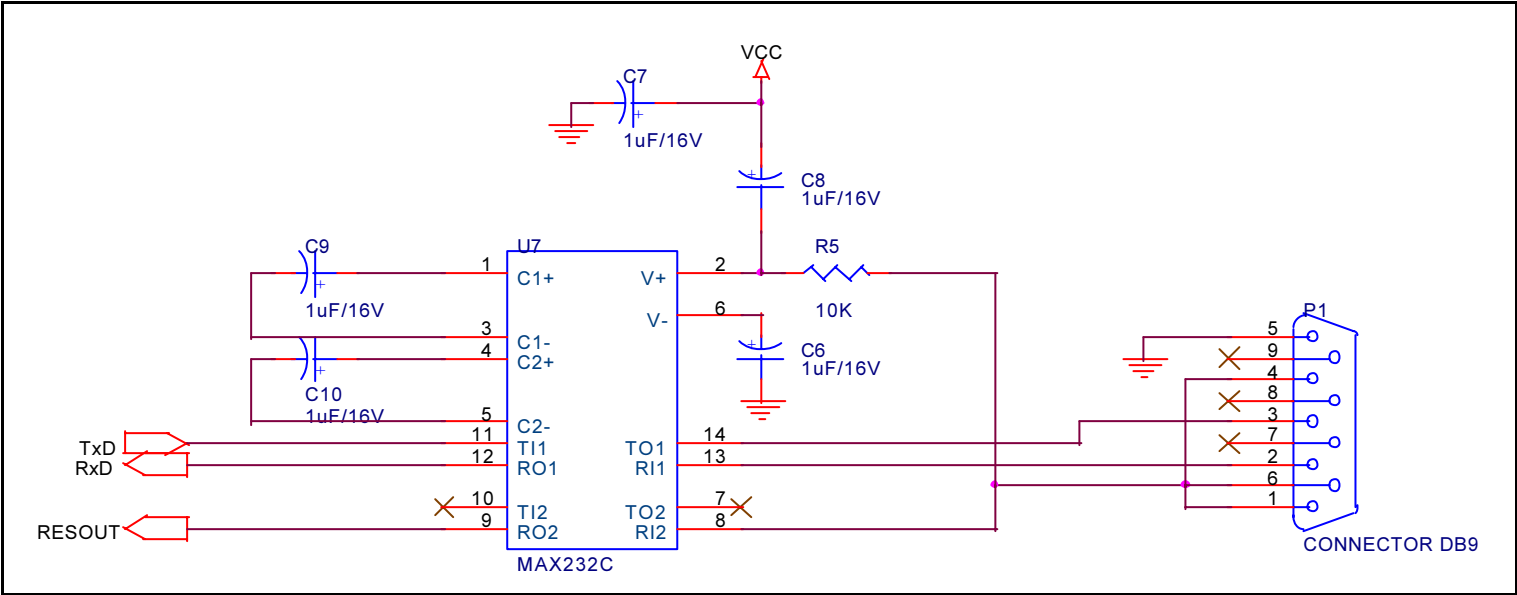


JP1-4 are soldering jumper on solder side

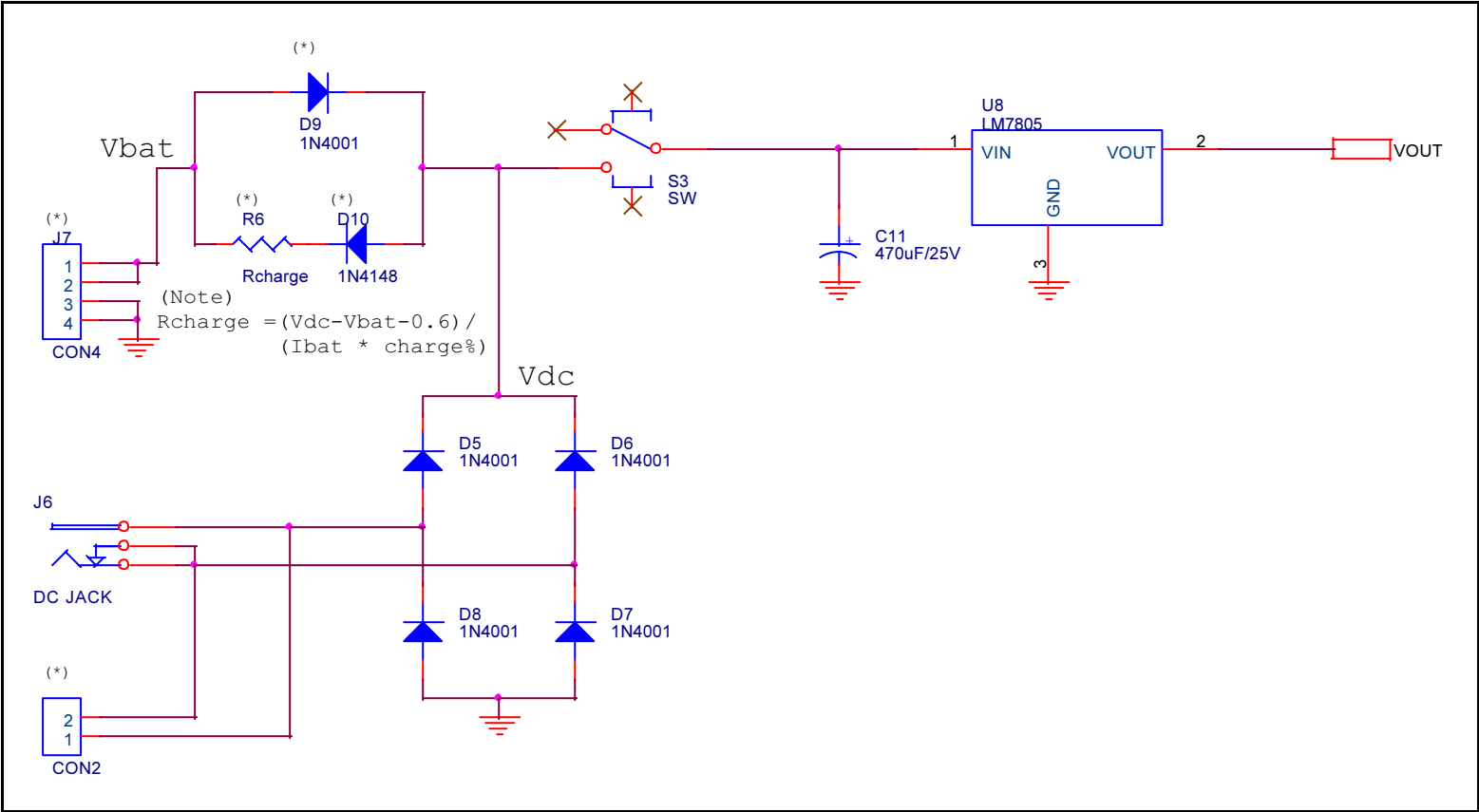
(\*) - not inserted

— Factory setting using copper trace on board

# Serial Port Schematic



# Power Supply Schematic



## Eval Board Parts List

No.	description	part number	Q'ty
1	MCU	68HC11D0	0
2	PLCC socket	44P-PLCC	1
3	PLCC socket	52P-PLCC	1
4	5V regulator	KIA7805P	1
5	Reset comparator	KIA7045P	1
6	TTL	MC74HC14AN	1
7	232 Driver	ICL232CPE	1
8	Crystal	9.8304MHz	1
9	block resistor array	AR100K-09P	3
10	block resistor array	AR100K-05P	2
11	block resistor array	AR4K7-05P	1
12	resistor	10M 1/8W	1
13	resistor	10K 1/8W	1
14	resistor	4.7K 1/8W	2
15	resistor	560 1/8W	3
16	resistor	47 1/8W	1
17	potentiometer	GF06S10K	1
18	diode (switching)	1N 4148RL	1
19	diode (rectifier)	1N 4002RL	4
20	electrolytic capacitor 1uF/50V	EC1U50V	5
21	electrolytic capacitor 2.2uF/16V	EC2.2U16V	1
22	electrolytic capacitor 470uF/16V	EC470U16V	1
23	electrolytic capacitor 100uF/6.3V	EC100U6.3V	1
24	ceramic capacitor 22pF	CC22	2
25	monolytic capacitor 0.1uF/50V	M104	5
26	LED (green, 3mm)	BL-B2141-3D	3
27	4 position dip switch	KSD04H	1
28	power switch (slide 3P)		1
29	reset siwtch		1
30	SIP 2 pin header		1
31	SIP 14 pin header (LCD side)		1
32	SIP 14 pin connector (PCB side)		1
33	DB-9 connector	DB-9SR	1
34	DC-JACK		1
35	7x2 pin ribbon cable w/ male con. (150mm)		1
36	7x2 pin connector (angle)		1
37	standoffs (3 mm x 10 mm) for LCD		2
38	bolt,nut (2.6 mm x 16mm) for LCD		2
39	anti-static bag (170 mm x 300 mm)		1
40	box (110 mm x 150 mm x 24 mm)		1
41	LCD module		1
42	standoffs for PCB board		4
43	PCB board		

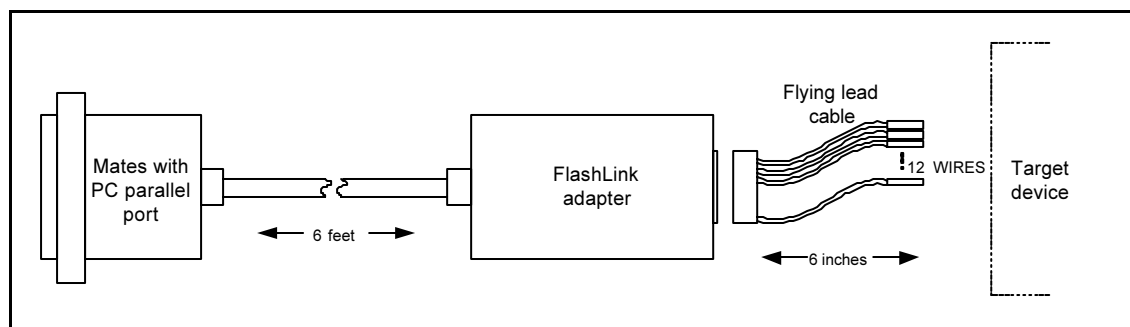
## Appendix C: FlashLINK Users Manual

### Features

- Allows PC parallel port to communicate with PSD9xx via PSDsoft Express
- Provides interface medium for JTAG communications
- Supports basic IEEE 1149.1 JTAG signals (TCK, TMS, TDI, TDO)
- Supports additional signals to enhance download speed (!TERR, TSTAT)
- Can be used for programming and/or testing
- Wide power supply range of 2.7 to 5.5v
- Pinout independent with target side flying leads
- Convenient desktop packaging allows varying applications(desk, lab or production)
- Synchronous JTAG interface allows speeds as fast as pc can drive

### Overview

Flashlink is a hardware interface from a standard PC parallel port to one or more PSD9xx devices located within a target PC board as shown below. This interface cable allows the PSD to be exercised for purposes of programming and/or testing. PSDsoft Express is the source for driving FlashLINK.



**Figure 25 Typical FLASHlink application**

### Operating considerations

Operating power for FlashLINK is derived from the target system in the range of 2.7 to 5.5 v. Compatibility over this voltage range is ensured by the design of FlashLINK. No settings are involved.

On a cautionary note, it is recommended that the target system be powered with a well regulated and stable source of power which is energized at the final value of Vcc. It is not recommended that the input voltage be varied using the vernier on a regulated power supply, as this may cause the internal FlashLINK IC's (74VHC240) to misoperate toward the lower end of the supply range.

Each FLASHLink is packaged with a six-inch "flying lead" cable for maximum adaptability (a ribbon cable requires the use a certain connector on the target assembly). This flying lead cable mates to the FlashLink adapter on one end and has loose sockets on the other end to slide onto 0.025 square posts on the target assembly.

PIN #	SIGNAL NAME	DESCRIPTION JTAG = IEEE 1149.1 EJTAG = WSi ENHANCED JTAG	Type	Flashlink is Signal
1	JEN\	Enables JTAG pins on PSD8XXF (optional)	OC,100K	Source
2	TRST\ *	JTAG reset on target (optional per 1149.1)	OC,10K	Source
3	<b>GND</b>	Signal ground		
4	CNTL *	Generic control signal, (optional)	OC,100K	Source
5	<b>TDI</b>	JTAG serial data input		Source
6	TSTAT	EJTAG programming status (optional)		Destination
7	<b>Vcc</b>	VDC Source from target (2.7 - 5.5 VDC)		
8	RST\	Target system reset (recommended)	OC,10K	Source
9	<b>TMS</b>	JTAG mode select		Source
10	GND	Signal ground		
11	<b>TCK</b>	JTAG clock		Source
12	GND	Signal ground		
13	<b>TDO</b>	JTAG serial data output		Destination
14	TERR\	EJTAG programming error (optional)		Destination
Notes				
1. <b>Bold</b> signals are required connections				
2. all signal grounds are connected inside FlashLink adapter				
3. OC = open collector, pulled-up to Vcc inside FlashLink adapter				
4. * = Not supported initially by PSDsoft.				
5. The target device must supply Vcc to the FlashLink Adapter (2.7 to 5.5 VDC, 15mA max @ 5.5V).				

**Figure 26 Pin descriptions for FlashLink adapter assembly**

All 14 signals may not be needed for a given application. Here's how they break down:

- (6) Core signals that must be connected: TDI, TDO, TMS, TCK, Vcc, GND
- (2) Optional signals for enhanced ISP (Option 3 flow control): TSTAT, TERR\
- (1) Optional signal to control multiplexing of the JTAG signals: JEN\
- (1) Recommended signal to allow FlashLink to reset target system during and after ISP: RST\
- (1) Optional IEEE-1149.1 signal for JTAG chain reset: TRST\
- (1) Optional generic control signal from FlashLink to target system: CNTL
- (2) Two additional ground lines to help reduce EMI if a ribbon cable is used. These ground lines "sandwich" the TCK signal in the ribbon cable. These lines are not needed for use with the flying lead cable, that is why the flying lead cable has only 12 of 14 wires populated.

## FLASHlink pinouts

There is no "standard" JTAG connector. Each manufacturer differs. WSi has a specific connector and pinout for the FlashLink programmer adapter. The connector scheme on the FlashLink adapter can accept a standard 14 pin ribbon connector (2 rows of 7 pins on 0.1" centers, standard keying) or any other user specific connector that can slide onto 0.025" square posts. The pinout for the FlashLink adapter connector is shown in figure 4.

A standard ribbon cable is good way to quickly connect to the target circuit board. If a ribbon cable is used, then the receiving connector on the target system should be the same connector type with the same pinout as the FlashLink adapter shown in Figure 4. Keep in mind that the JTAG signal TDI is sourced from the FlashLink adapter and should be routed on the target circuit card so that it connects to the TDI input pin of the PSD device. Although the name "TDI" infers "Data In" by convention, it is an output from FlashLink and an input to the PSD device. Also keep in mind that the JTAG signal TDO is an input received by the FlashLink adapter and is sourced by the PSD device on the TDO output pin. Use Figures 1, 2, 3, and 6 as a guide.

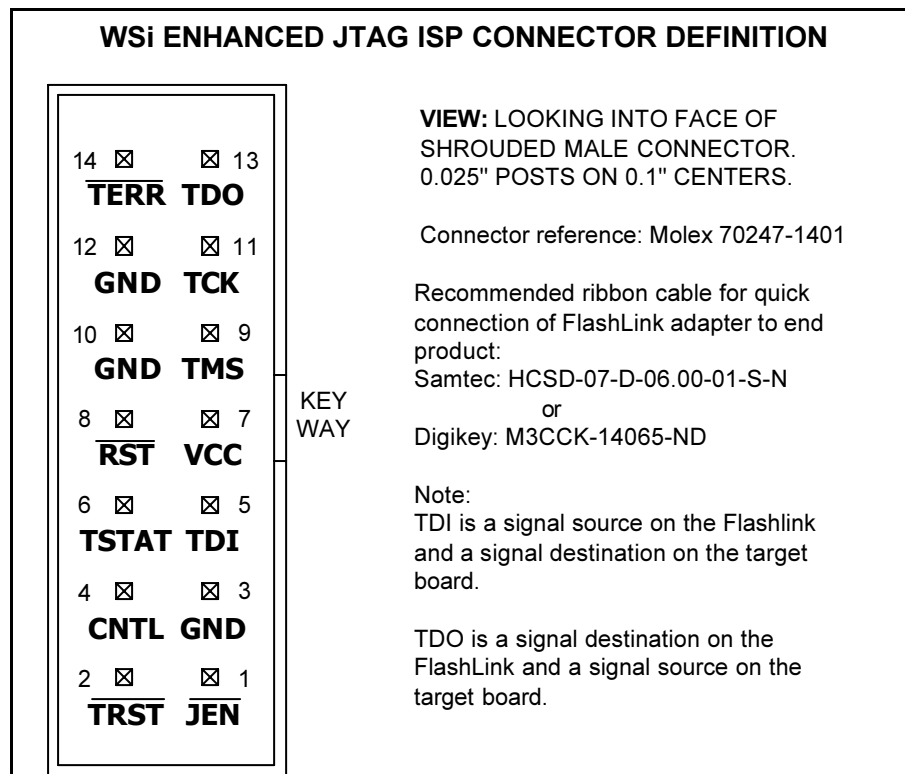


Figure 27 Pinout for FlashLink Adapter and Target System

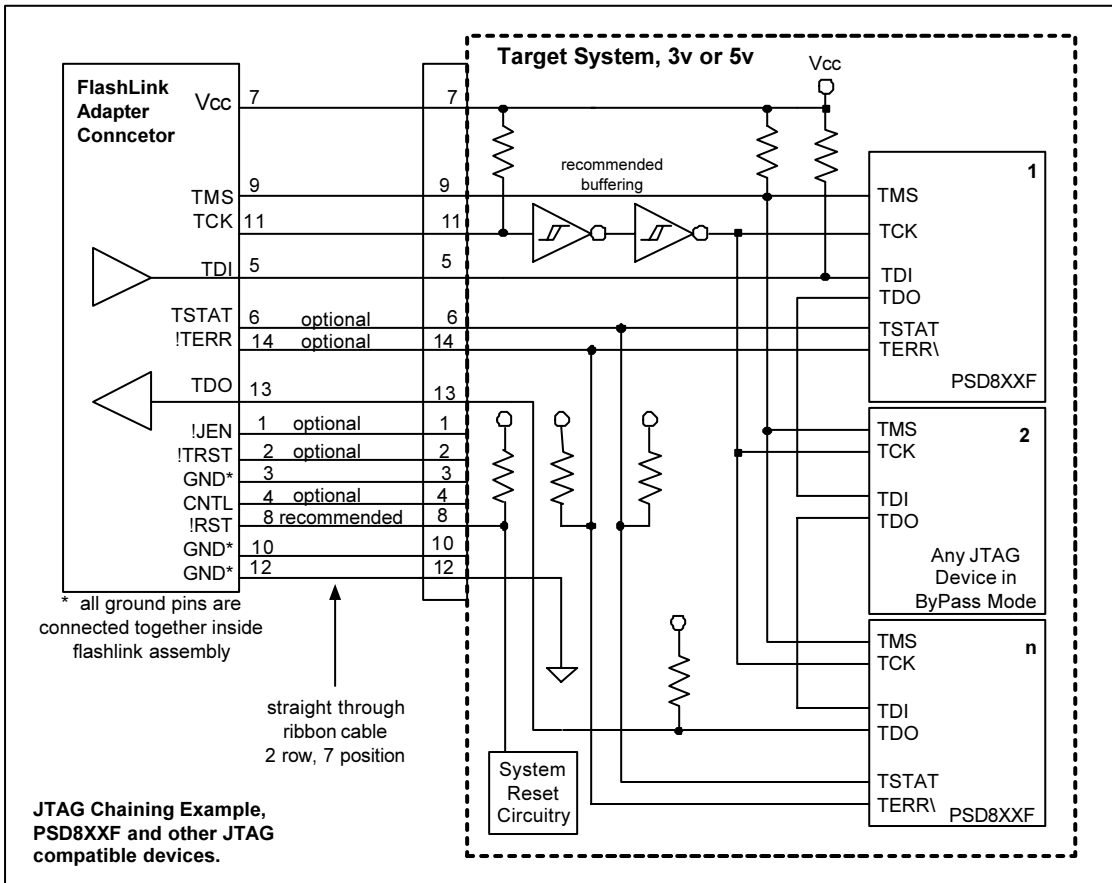
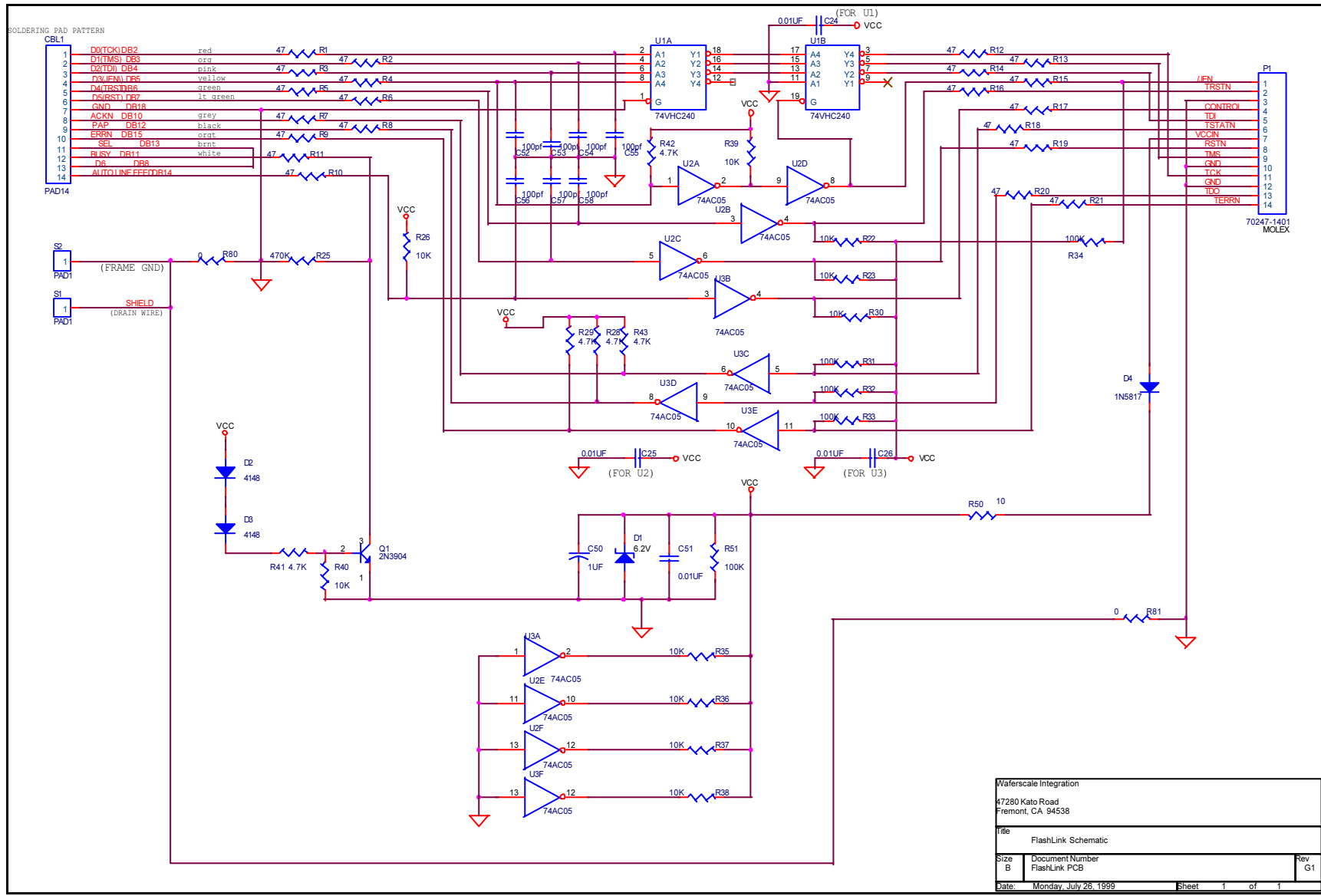
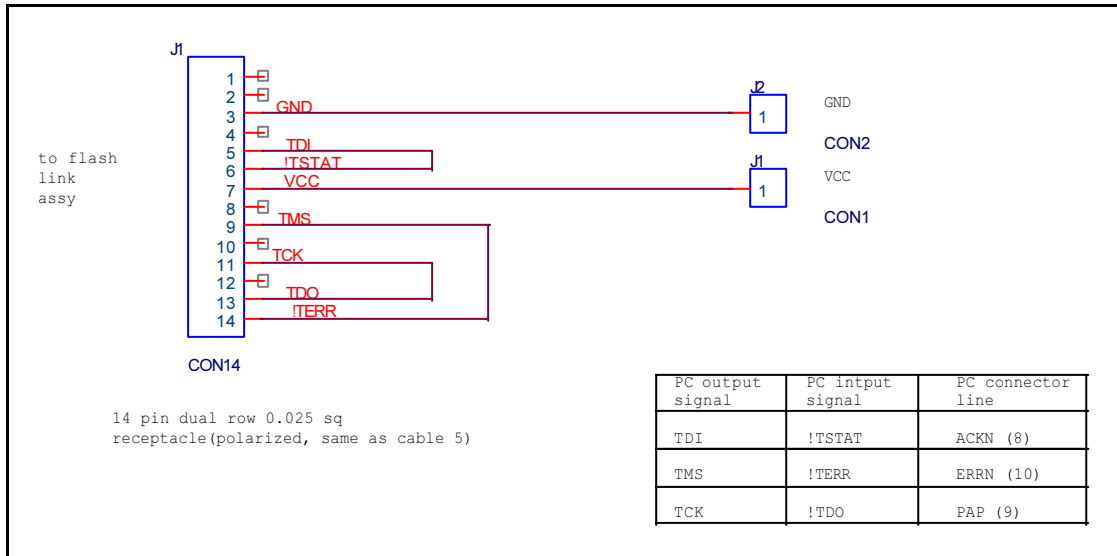


Figure 28 JTAG Chaining Example





## Loop back connector schematic



**Figure 29 Loop Back Tester, Passive, FLASHlink**

## Appendix D *crtsi.s* routine

```
; crtsi.s
; C STARTUP FOR MC68HC11
; WITH AUTOMATIC DATA INITIALISATION
; Copyright (c) 1995 by COSMIC Software
;
xdef _exit, __stext
xref _main, __memory, __idesc__, __stack
xref _evl_optn
;
switch .bss
__sbss:
svx:
dc.w 0
sve:
dc.w 0
;
switch .text
__stext:
ldx #__idesc__ ; descriptor address
ldy 0,x ; start address of prom data
inx ; skip address
inx
ibcl:
ldaa 0,x ; test flag byte
beq zbss ; no more segment
bpl nobk ; skip bank
inx ; info
inx ; if any
nobk:
stx svx ; save pointer
ldd 3,x ; end address
std sve ; in memory
ldx 1,x ; destination address
dbcl:
ldab 0,y ; copy from prom
stab 0,x ; to ram
inx ; next byte
iny
cpy sve ; last one ?
bne dbcl ; no, loop again
ldx svx ; reload pointer to desc
ldab #5 ; size of one entry
abx ; point to next entry
bra ibcl ; and loop
zbss:
ldx #__sbss ; start of bss
bra loop ; start loop
zbcl:
staa 0,x ; clear byte
inx ; next byte
loop:
cpx #__memory ; up to the end
bne zbcl ; and loop
lds #__stack; re-initialize stack pointer
jsr _main ; execute main
_exit:
```

```

    bra _exit      ; and stay here
;
; Special routines for non-stop concurrnet swap & execute
; both boot and main code must have identical following routines
;
    xdef _EXECUTE_SOURCE
    xdef _Execute_Main, _Execute_Boot, _Return_Boot
    xref _PSD8xx_reg
    xref _PSDload_init, _PSDload
;
    switch .text
;
; swap to main flash and execute main code
;
_Execute_Main:
    cli          ; disable all interrupts
    ldaa#$80
    ldy #_PSD8xx_reg
    oraa$E0,y    ; set SWAP in PAGE to 1
    staa $E0,y   ; now in main flash
    jmp __stext  ; jump to Cstarup of main
                    ; hc11 option setting is not allowed
;
; swap to boot flash and execute boot code
;
_Execute_Boot:
    cli          ; disable all interrupts
    ldaa#$7F
    ldy #_PSD8xx_reg
    anda $E0,y   ; clear SWAP in PAGE to 0
    staa $E0,y   ; now in boot flash
    jmp __stext  ; jump to Cstarup of boot
                    ; hc11 option setting is not allowed
;
; swap to boot flash and execute PSDload with message in boot
;
_Return_Boot:
    cli          ; disable all interrupts
    ldaa#$7F
    ldy #_PSD8xx_reg
    anda $E0,y   ; clear SWAP in PAGE to 0
    staa $E0,y   ; now in boot flash
    lds #__stack ; initialize stack pointer before
                    ; calling PSDload subroutine(s) written in C
    jsr _PSDload_init
    jsr _PSDload  ; loop forever in C
;
_EXECUTE_SOURCE:
    dc.b 'PSDload'
    dc.b 'V1.0',0

end

```

## Appendix E evl\_init.c routine

```
void evl_init(void)
{
    psd_init();           // initialize PSD
                        // if any PSD port(s) is/are used for
                        // latched address output for external data memory,
                        // the PSD port must be initialized in "evl_optn.c"
                        // to guarantee initialization of data segment.

    Run_Execution_Source(); // check & run execution source
                        // EXECUTE_SOURCE[] contains this information

    PSDload_init();      // initialize other I/Os for PSDload function
}

// This is for checking setting of execution source and running
extern void Execute_Main(void); // to ensure, same code in both BOOT and MAIN
extern void Return_Boot(void); // these functions are in Cstartup(crtsi.s)
extern void Execute_Boot(void);

extern const char EXECUTE_SOURCE[]; // this data storage also in Cstartup(crtsi.s)

void Run_Execution_Source(void)
{
    char *copy_loc;

    if (!read_dipsw(DIP_SW3)) {
        // current execution location is BOOT
        // && same Cstartup code may be in MAIN
        copy_loc = (char *)(((uint)EXECUTE_SOURCE - boot_mem_start_addr) + unswapped_Fseg_addr);
        page_set (unswapped_Fseg_page);
        if ( !page_get(SWAP) && !strcmp(EXECUTE_SOURCE, copy_loc) )
            Execute_Main();
    }
    else
        // current execution location is MAIN
        // && execution source setting is BOOT
        if ( page_get(SWAP) )
            Execute_Boot();
}

// This is initialization for only direct PSDload running
// this can be called from STARTUP.A51
void PSDload_init(void)
{
    com_initialize(); // initialize UART buffer.
    init_message_level(); // initialize message level variables
    timer_initialize(); // initialize real time interrupt,

    lcd_init(); // initialize LCD. 8 bits, 2 lines, 5x7 font,
                // no blink, cursor off, clear
}
}
```