DK900
USER MANUAL

DK900 Development Kit
For PSD9xxF Family of Flash PSDs

CONTENTS

- (Please see next pages)

January 2002

DK900
DEVELOPMENT KIT
For PSD9xxF Family of Flash PSDs
Rev 1.10

Contents:

- PSDsoft Express - Point and Click Windows based Development Software(from web)
- PSD9xxF Sample
- DK900 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 (from web)
- Serial UART cable for PSDload
- CDROM - Data Book, Software and Videos
- 110V or 220V Power supply

DK900 Development Kit

Introduction

Congratulations on purchasing ST DK900 Development kit. The DK900 (110V or 220 Volt version) is a low cost kit for evaluating the PSD9xx family of FLASH Programmable System Devices. The kit is extremely versatile, and can be used in several different modes. In it's simplest mode, it can be used to demonstrate the PSD9xx's capability of JTAG In-System Programmability (ISP). After ISP is accomplished, the DK900 can be set-up to update the program while the MCU is running, called In-Application Programming (IAP). And lastly, 8051 family users can utilize the DK900 as an evaluation platform for code development.

Regardless of how much development work is done on the DK900, 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.

What's Included

Hardware

- PSD9xx FLASH PSD (Programmable System Device) - see www.st.com/psm 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 8032 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

What? No Software?

- To assure latest version, download from our website (note item 3 contains 3 components zipped together):
  1. PSDsoft Express - Point and Click Windows programming development software. This will install to it's 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 it's own directory.
     - In-Application Programming
     - Performs erase, fill, read, write, upload and download of PSD
     - All functions performed through MCU's UART channel.
  3. U809_10x.zip contains the following archived (zipped) components. Please place them in the indicated directories under \PSDExpress. Create PSDExpress\DK900 directory.
     - U809c10x.zip Demo ISP executable program for the eval board. \DK900\uart80_c
     - U809p10x.zip psd file for above. \DK900\uart80_p
     - Uart1_c.zip Demo IAP executable program for the eval board. \DK900\uart1_c
     - Uart2_c.zip - Another demo program. \DK900\uart2_c

At this point, you should have the following files in the PSDExpress\DK900 directory.

Iap_80Ex.mmf
Iap_80Ex.obj
Iap_80Ex.psd
Uart1.hex
Uart2.hex

These are all the files that are needed for the demo's in this manual. The remaining archives are source information from which these files were constructed.

Detailed Descriptions

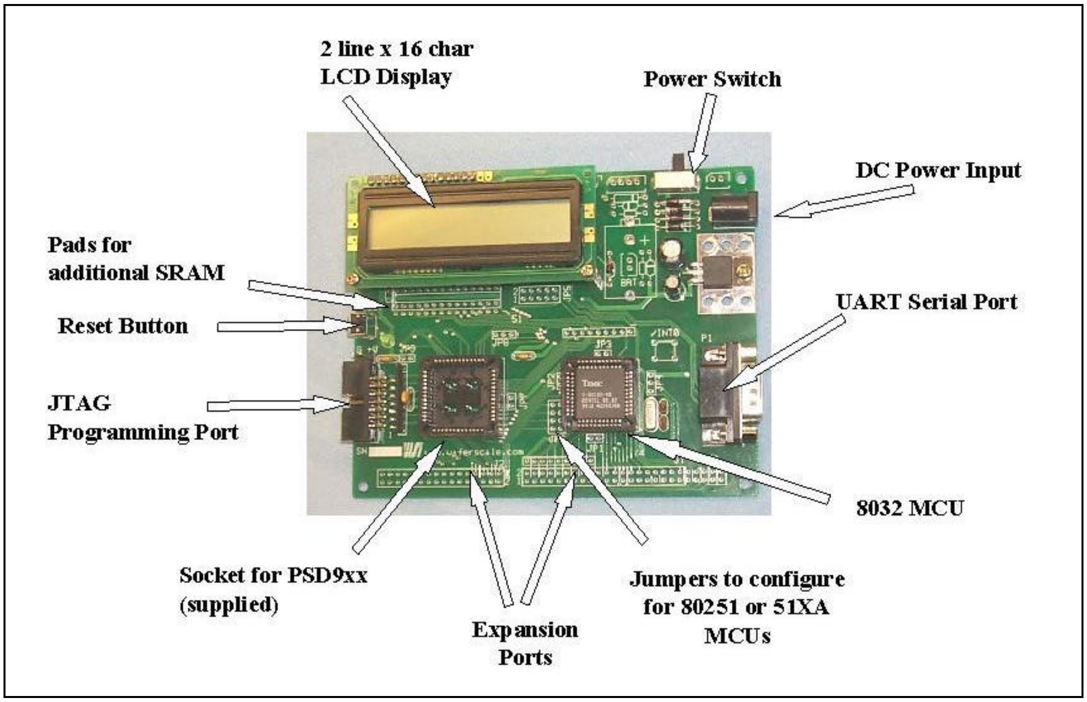


Figure 1 DK900 Development Board

- **Display -** A two line by 16 character LCD display is included on the Development Board.
- **Power switch (left position is on)**
- **UART Serial Port(male)** - Connected to MCU serial port; used for In-Application Programming (IAP)
- **8032 MCU** - Low cost MCU (80251 or P51XA can be substituted, see Appendix A), 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
- **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:

a) Go to ST website (www.st.com/psm) and select "Development Tools" from the top menu, then go to the DK900 Kit

b) Download both PSDload and PSDsoft Express (note: you will be asked to fill out a form for PSDsoft Express so that a password can be immediately emailed to you)

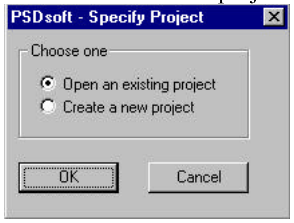
c) Install both programs on your PC running Windows 95/98/NT

d) Plug the blank PSD9XX device into the Eval board socket

e) Plug the FlashLINK Programmer into your PCs parallel port and plug in the ribbon cable to the JTAG port on the eval board (for help see the Appendix C, FlashLINK manual). **Note that the serial cable should not be plugged in during this ISP exercise.**

f) Plug in power supply and turn on power. Notice that the LCD display is blank because the PSD is blank

g) Run PSDsoft Express. Here is the initial screen if no project is open.


PSDsoft - Specify Project

Choose one

● Open an existing project
○ Create a new project

Figure 2 Ope ning 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 as a development platform”, a further tutorial is given on using PSDsoft Express with the Eval Board for development.

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

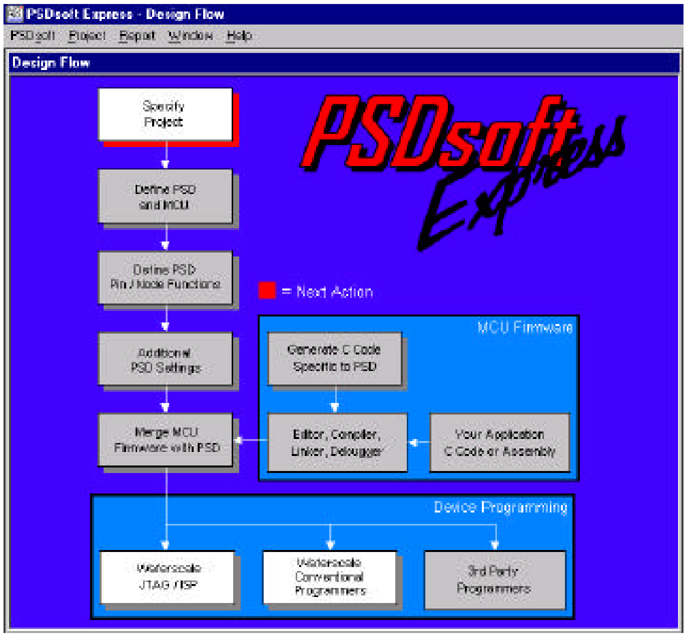
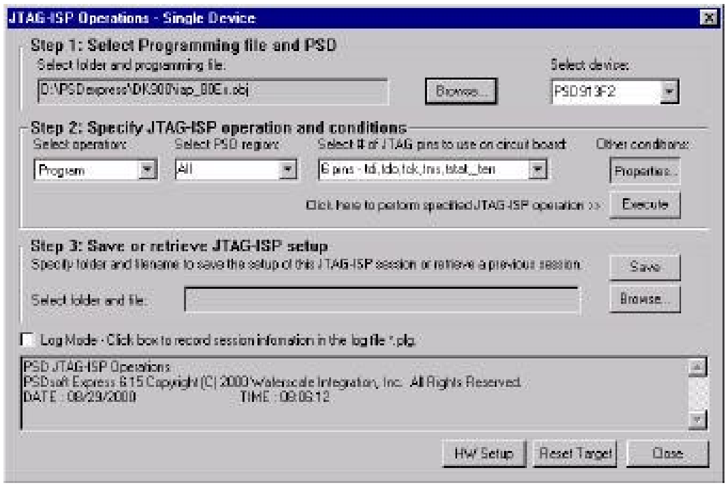

PSDsoft Express - Design Flow
PSDsoft
Project
Report
Window
Help
Design Flow
Specify Project
Define PSD and MCU
Define PSD Pin / Node Functions
Additional PSD Settings
Merge MCU Firmware with PSD
PSDsoft Express
🟥 = Next Action
MCU Firmware
Generate C Code Specific to PSD
Editor, Compiler, Linker, Debugger
Your Application C Code or Assembly
Device Programming
Waferscale JTAG / ISP
Waferscale Conventional Programmers
3rd Party Programmers

Figure 3 PSDsoft Express flow

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



JTAG-ISP Operations - Single Device

Step 1: Select Programming file and PSD

Select folder and programming file: D:\PSDexpress\DK900\iap_80Ex.obj

Select device: PSD913F2

Step 2: Specify JTAG-ISP operation and conditions

Select operation: Program

Select PSD region: All

Select # of JTAG pins to use on circuit board: 6 pins - tdi,tdo,tck,tms,tstat,_terr

Click here to perform specified JTAG-ISP operation >>

Step 3: Save or retrieve JTAG-ISP setup

Specify folder and filename to save the setup of this JTAG-ISP session or retrieve a previous session.

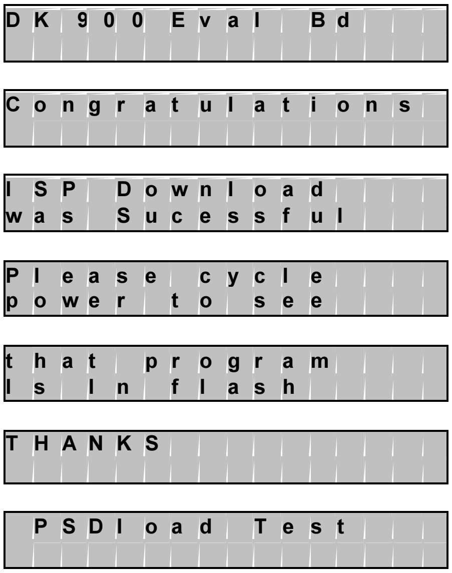
Select folder and file:

□ Log Mode - Click box to record session information in the log file *.plg.

```
PSD JTAG-ISP Operations
PSDsoft Express 6.15 Copyright (C) 2000 Waferscale Integration, Inc.  All Rights Reserved.
DATE : 08/29/2000                    TIME : 09:06:12
```

Figure 4 PSDsoft Express, JTAG Operations dialog

i) In Step 1, “select folder and file” browse to find under …\DK900*.obj
j) The “select device” box should be filled in for you.
k) In Step 2, click Execute
l) Observe in the lower pane the JTAG activities that occur while programming your device.
m) Watch the board mounted display. 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. **Note that the serial cable should not be plugged in during this ISP exercise.**



DK 900 Eval Bd

Congratulations

ISP Download
was Sucessful

Please cycle
power to see

that program
Is In flash

THANKS

 PSDload Test

Figure 5 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 FLASH Memory.

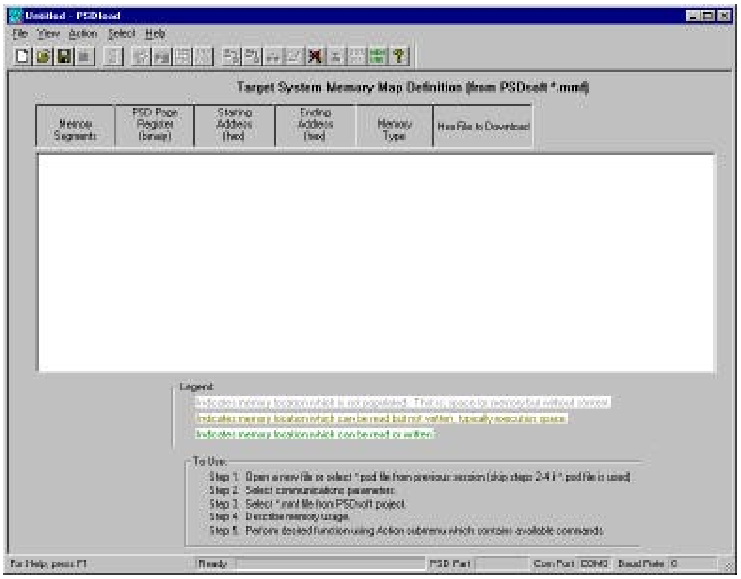
o) For better understanding of the program you may want to examine:
    1. System memory map in the memory map section later in this document.
    2. PSDsoft Express project
    3. The file source code (included) to see how the executing code was configured

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 eval board. Note that this cable is a null modem cable(F-F).

b) Once the eval 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 no communications as shown below.



Target System Memory Map Definition (from PSDsoft *.mmf)

Memory Segments	PSD Page Register (binary)	Starting Address (hex)	Ending Address (hex)	Memory Type	Hex File to Download

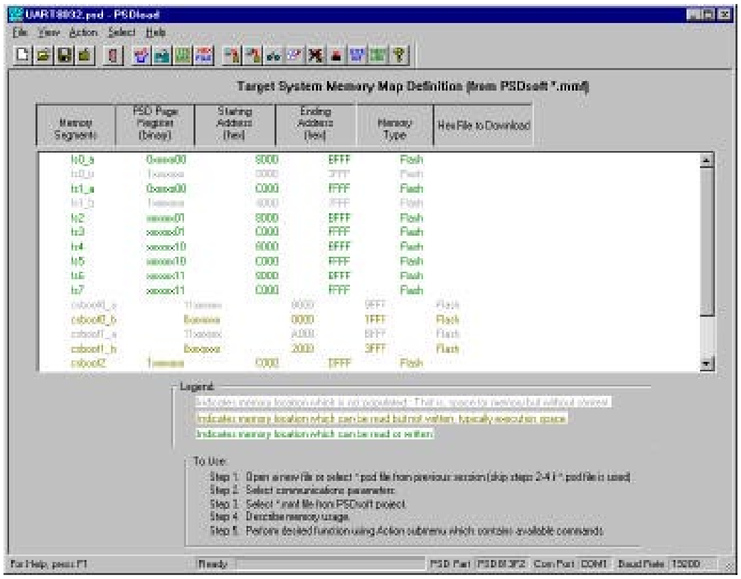
Legend:
Indicates memory location which is not populated. That is, space for memory but without device.
Indicates memory location which can be read but not written, typically execution space.
Indicates memory location which can be read or written.

To Use:
Step 1: Open a new file or select *.psd file from previous session (skip steps 2-4 if *.psd file is used).
Step 2: Select communications parameters.
Step 3: Select *.mmf file from PSDsoft project.
Step 4: Describe memory usage.
Step 5: Perform desired function using Action submenu which contains available commands.

For Help, press F1 | Ready | PSD Part | Com Port COM0 | Baud Rate 0

Figure 6 Initial PSDload invocations screen (no comm)

d) File, Open. Find the file as follows; \DK900*.psd. This is a configuration file for PSDload that's been constructed for this demo containing the particulars of the design. Observe the buttons become active(colorful) when this file is selected indicating the communications port is properly 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.



Target System Memory Map Definition (from PSDsoft *.mmf)

Memory Segments	PSD Page Register (binary)	Starting Address (hex)	Ending Address (hex)	Memory Type	Hex File to Download
fs0_a	0xxxxx00	8000	BFFF	Flash	
fs0_b	1xxxxxxx	8000	BFFF	Flash	
fs1_a	0xxxxx00	C000	FFFF	Flash	
fs1_b	1xxxxxxx	C000	FFFF	Flash	
fs2	xxxxxx01	8000	BFFF	Flash	
fs3	xxxxxx01	C000	FFFF	Flash	
fs4	xxxxxx10	8000	BFFF	Flash	
fs5	xxxxxx10	C000	FFFF	Flash	
fs6	xxxxxx11	8000	BFFF	Flash	
fs7	xxxxxx11	C000	FFFF	Flash	
csboot0_a	11xxxxxx	8000	9FFF	Flash	
csboot0_b	0xxxxxxx	0000	1FFF	Flash	
csboot1_a	11xxxxxx	A000	BFFF	Flash	
csboot1_b	0xxxxxxx	2000	3FFF	Flash	
csboot2	1xxxxxxx	C000	DFFF	Flash	

Legend:
Indicates memory location which is not populated. That is, space for memory but without device.
Indicates memory location which can be read but not written, typically execution space.
Indicates memory location which can be read or written.

To Use:
Step 1: Open a new file or select *.psd file from previous session (skip steps 2-4 if *.psd file is used).
Step 2: Select communications parameters.
Step 3: Select *.mmf file from PSDsoft project.
Step 4: Describe memory usage.
Step 5: Perform desired function using Action submenu which contains available commands.

For Help, press F1 | Ready | PSD Part PSD813F2 | Com Port COM1 | Baud Rate 19200

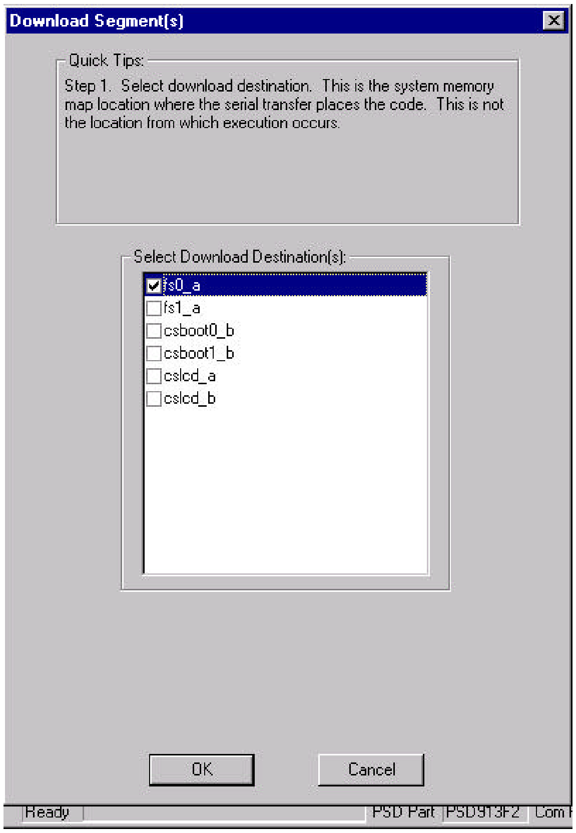
Figure 7 Initial PSDload invocations screen (with comm)

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 is entered in PSDsoft Express during the design phase of the project.

e) 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 eval board display. If it does, you've successfully established communications between the PC and eval board. If this doesn't work, check the following;

1. cable is plugged in
2. cable is of correct type(straight through)
3. correct comm port is selected

f) Select action, download observing the Download Segments dialog. The following screen will appear.



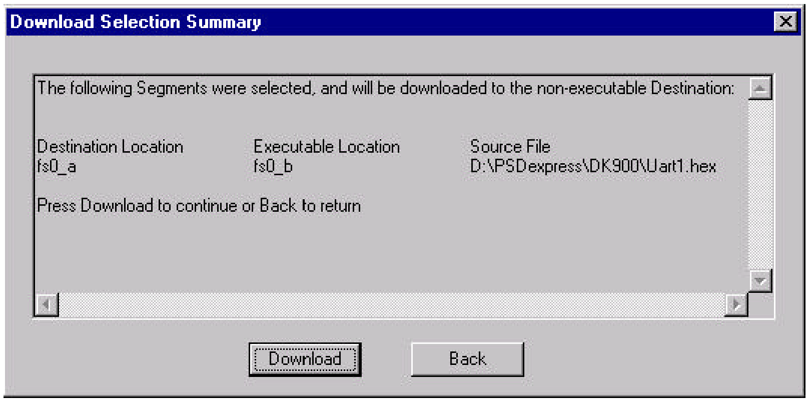
Step 1. Select download destination. This is the system memory map location where the serial transfer places the code. This is not the location from which execution occurs.

☑ fs0_a
□ fs1_a
□ csboot0_b
□ csboot1_b
□ cslcd_a
□ cslcd_b

Figure 8 Download Segments dialog, PSDload

Selecting the download destination (Step 1) to be fs0_a. Behind the scenes fs0_b will automatically be selected as the execution location. Click OK.

g) Now the Download Selection Summary screen, below, pops up. The intent is to validate the settings chosen in the last screen. You should see fs0_a as the download destination and fs0_b as the execution location. Click Download to start the process or back to change.



Download Selection Summary

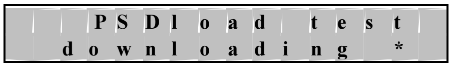
The following Segments were selected, and will be downloaded to the non-executable Destination:

Destination Location	Executable Location	Source File
fs0_a	fs0_b	D:\PSDexpress\DK900\Uart1.hex

Press Download to continue or Back to return

Figure 9 Download Summary screen

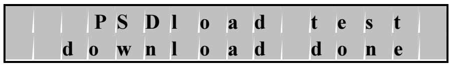
h) Observe the progress bar at the bottom of the PSDload window for activity. Also, observe the display on the eval board as follows.



```
PSDload test
downloading *
```

Figure 10 Eval Board display for download in process

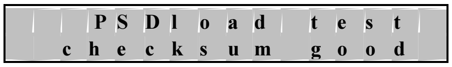
During the download, you'll observe the * character changing between the following -, \, |, and /. A change from one character to the next occurs with each new packet received by the eval board. When the download is complete you will see the following.



```
PSDload test
download done
```

Figure 11 Eval Board display for download complete

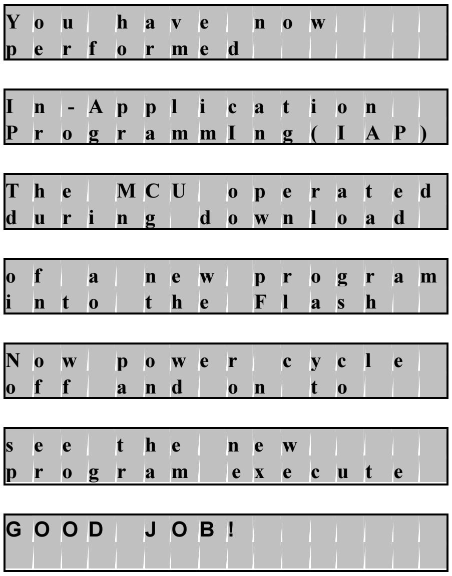
Next, observe the results of the checksum calculation covering the entire downloaded contents as shown below. Of course this was a successful download. This particular display does not persist, so watch the display intently.



```
PSDload test
checksum good
```

Figure 12 Eval Board display for checksum validation

i) Now click the reset button and observe the eval board display. The program you just downloaded will boot showing the displays listed below.

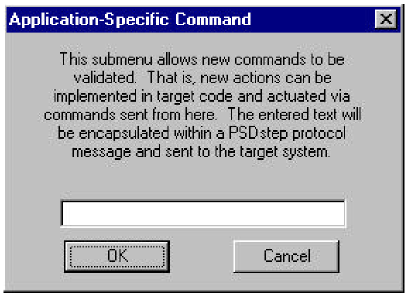


```
You have now
performed

In-Application
ProgrammIng(IAP)

The MCU operated
during download

of a new program
into the Flash

Now power cycle
off and on to

see the new
program execute

GOOD JOB!
```

Figure 13 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.

i) Now, let's reinvoke the original program that was runnign prior to IAP download. Using PSDload, press the User defined button USER DEF . A dialog will pop as below.



Application-Specific Command

This submenu allows new commands to be validated. That is, new actions can be implemented in target code and actuated via commands sent from here. The entered text will be encapsulated within a PSD step protocol message and sent to the target system.

Figure 14 Application Specific Command, PSDload (User Defined)

In the window type "RET" in upper case and press OK. Observe the display on the eval board now shows the "PSDload Test" banner on the upper line. This display indicates that the original CSBOOT resident code is running again. You can do some other functions from PSDload like Write to Display to further verify the original program is again active. For an explanation of the details that allow this to occur, see "Using DK900 as a Development Platform" later in this manual.

j) This concludes the IAP demonstration. These activities illustrate how a new code bundle can be downloaded over the serial channel and invoked remotely from PSDload with a serial command. Also, how another serial command can be used to return to the original code, pre IAP code. Good job!

Using DK900 as a Development Platform for 8051 MCU users:

Concept

The ST DK900 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 Development Board is specific to the 8051 microcontroller family. The board contains an empty socket for the PSD9xx, which can be populated with the included PSD9xx family component.

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 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 8032. This method allows only data and executable code to be downloaded over a PC serial link. This method is not restricted in destination to the PSD. The destination can be any resources on the Eval 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.

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.st.com/psm .

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.

ST provides a FlashLINK programmer to facilitate this JTAG programming operation. The FlashLINK programmer connects the PC parallel port to the Eval Board JTAG connector 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 ST 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 Eval 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. PSD*Load* is the windows interface running on the PC.
b. PSD*Step* 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. 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 8032 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 Evaluation 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 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.

Since the system memory map is utilized to achieve the download, the PLD within the PSD mu st 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.

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	LCD DISP	Write to display (on dev board)
Reset board		Reset development board
User data	USER DEF	Encapsulate user specific commands
Hex file entry	HEX FILE	Enter hex file to be downloaded
Describe memory usage	MEM USE	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.

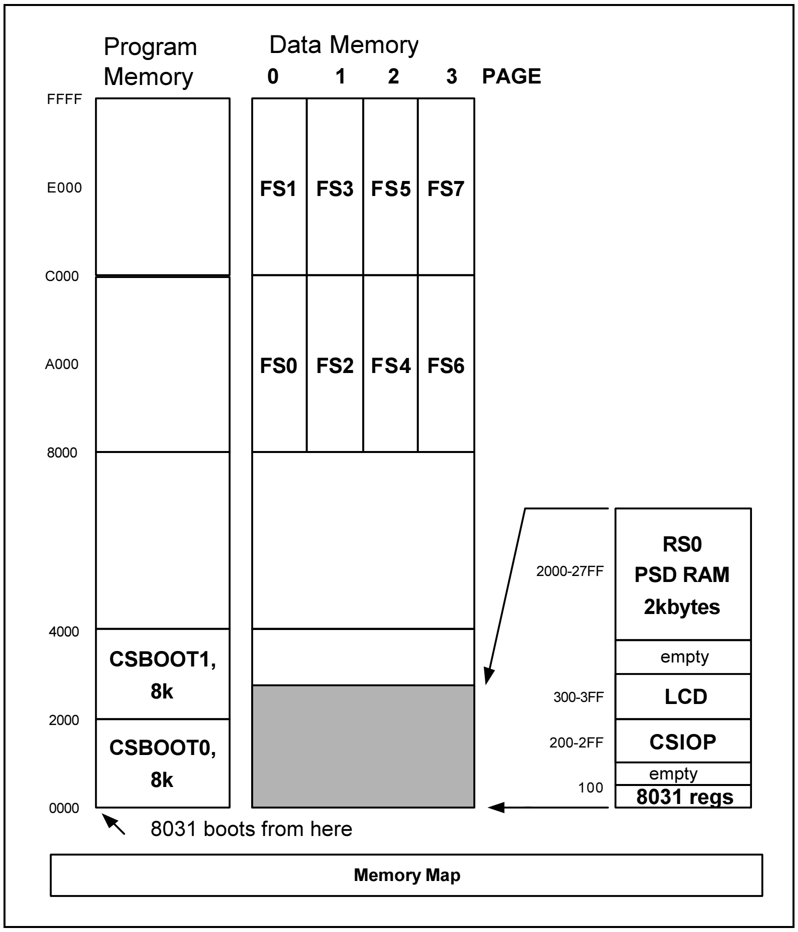


Figure 15 Memory Map of Eval Board

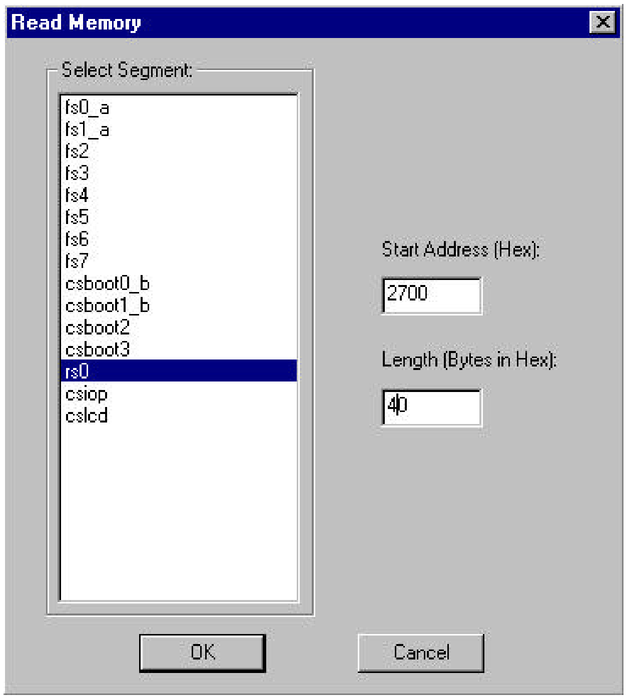
Getting started with a PSDload

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 at location 2700h with a length of 40h by selecting the command from the pulldown menu.

You will observe the following dialog box. First click on RS0 in the left hand box. The start address (in hex) will automatically be populated for you by the application. Now, enter the length and click OK.



Read Memory

Select Segment:

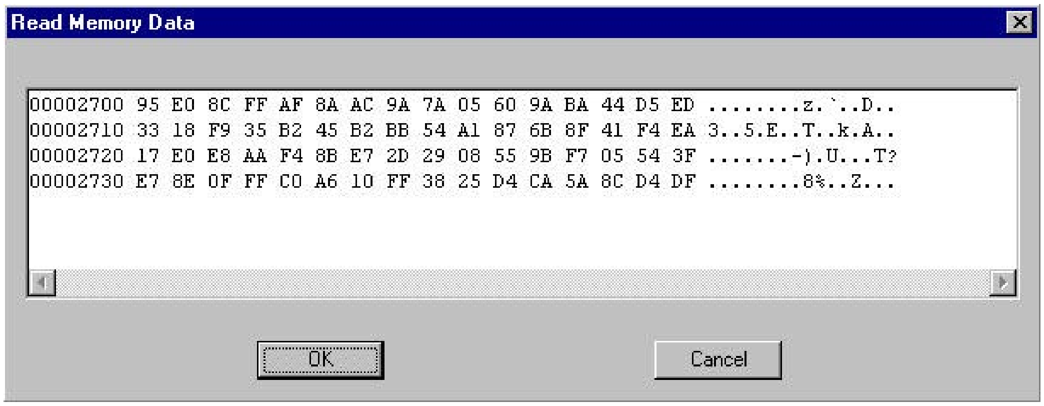
- fs0_a
- fs1_a
- fs2
- fs3
- fs4
- fs5
- fs6
- fs7
- csboot0_b
- csboot1_b
- csboot2
- csboot3
- rs0
- csiop
- cslcd

Start Address (Hex): 2700

Length (Bytes in Hex): 40

Figure 16 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.

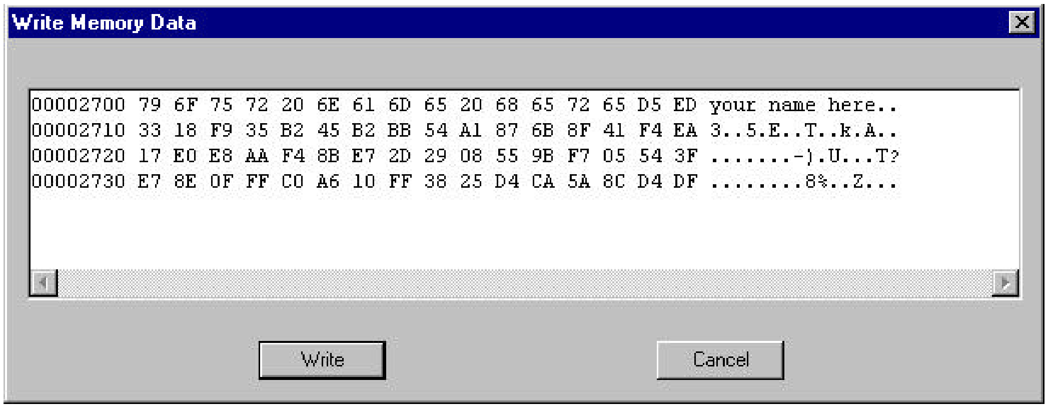


Read Memory Data

```
00002700 95 E0 8C FF AF 8A AC 9A 7A 05 60 9A BA 44 D5 ED ........z.`..D..
00002710 33 18 F9 35 B2 45 B2 BB 54 A1 87 6B 8F 41 F4 EA 3..5.E..T..k.A..
00002720 17 E0 E8 AA F4 8B E7 2D 29 08 55 9B F7 05 54 3F .......-).U...T?
00002730 E7 8E 0F FF C0 A6 10 FF 38 25 D4 CA 5A 8C D4 DF ........8%..Z...
```

Figure 17 Read Memory Data in PSDload

The contents appear random as this is volatile memory and has not been cleared. Now do a write of the same locations. You'll see the same box 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 happens automatically as shown below. Try typing your name or something into the ASC field. You will notice the hex bytes changing as you type.



Write Memory Data

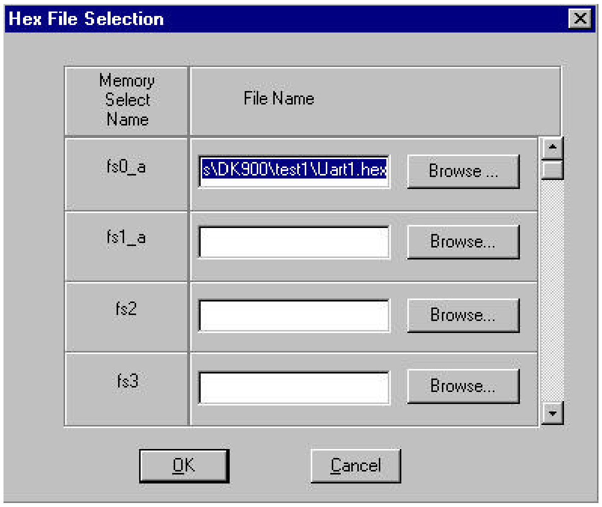
```
00002700 79 6F 75 72 20 6E 61 6D 65 20 68 65 72 65 D5 ED your name here..
00002710 33 18 F9 35 B2 45 B2 BB 54 A1 87 6B 8F 41 F4 EA 3..5.E..T..k.A..
00002720 17 E0 E8 AA F4 8B E7 2D 29 08 55 9B F7 05 54 3F .......-).U...T?
00002730 E7 8E 0F FF C0 A6 10 FF 38 25 D4 CA 5A 8C D4 DF ........8%..Z...
```

Figure 18 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. The fact that the information is now gone confirms that the area was SRAM.

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 the FLASH memory is not used in the application yet, no harm will be done. Select Write Memory and, in the write dialog, select fs7, which stars at 0xC000. Read 40h bytes of the area. You will notice that instead of the random 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. Two methods exist to erase FLASH, by sector or by bulk (the whole thing). In this case the bulk erase is used. 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 hex file. This screen is 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.



Hex File Selection

Memory Select Name	File Name
fs0_a	s\DK900\test1\Uart1.hex
fs1_a	
fs2	
fs3	

Figure 19 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. Internal PSD resources (PAGE and VM registers)are 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).

The VM(virtual memory) register is specific to 8031 family devices and allows the PSD memory resources to be controlled between program space and data space. This register, located at csiop+0xE2, can be set to a non-volatile initial value using PSDsoft Express and thereafter can be read or written by the microcontroller. This register is volatile.

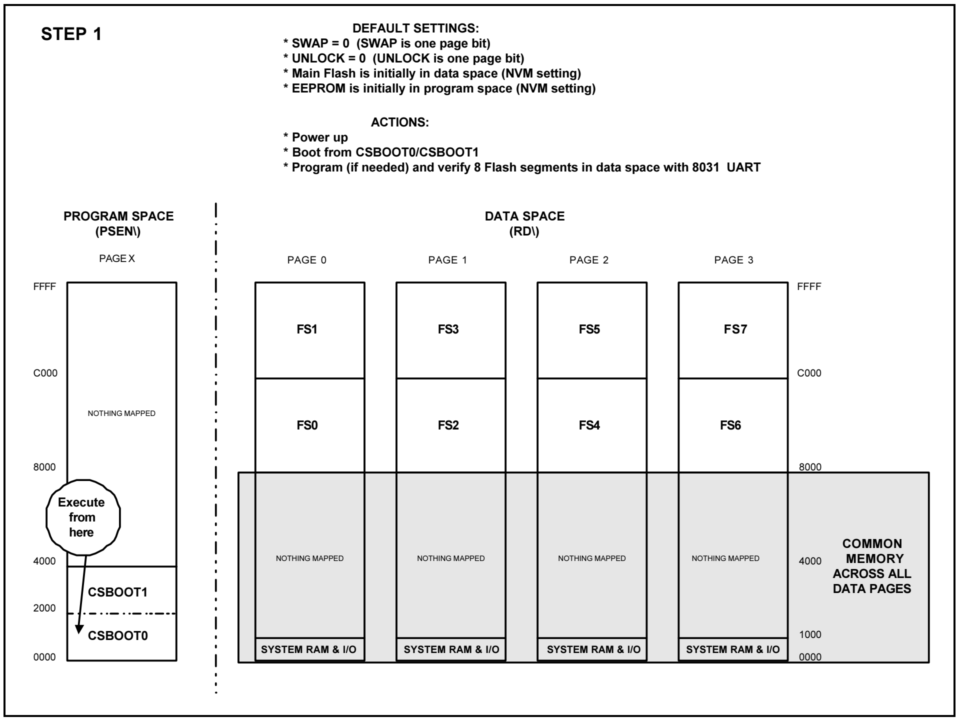
The PAGE register (csiop+0xE0, 8 bits) is traditionally used to control memory paging, but we also use it to control memory addresses, as presented to the microcontroller, using 1 or more bits. This register can be read or written by the microcontroller. The initial value of the PAGE register is 0 at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

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 for any system containing program and data space.

1. Power up system with default memory map. swap=0(PAGE register msb), VM=0x12
2. Write VM register =0x06.
3. Write swap=1(PAGE register msb)
4. Write VM=0x0C

These steps are further depicted graphically in the following 4 figures.

Here's the memory map at power up. Note that we are executing from CSBOOT0/1 and that MAIN FLASH is in data space. During the download, the complete new executable, including the vector table, is copied into FS0. During this time the swap bit in the PAGE register is 0 and the VM register is 0x12.



STEP 1

DEFAULT SETTINGS:
* SWAP = 0 (SWAP is one page bit)
* UNLOCK = 0 (UNLOCK is one page bit)
* Main Flash is initially in data space (NVM setting)
* EEPROM is initially in program space (NVM setting)

ACTIONS:
* Power up
* Boot from CSBOOT0/CSBOOT1
* Program (if needed) and verify 8 Flash segments in data space with 8031 UART

Figure 20 Memory map at power up, NVswap=0

Now, let's set a flag (NVswap) to indicate we want to run the code in FS0 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 detect the state of Nvswap and to write that value into the PAGE register(msb) at power up. If it's 0, the code bundle residing in CSBOOT0/1 continues to run. If it's 1, we perform the memory manipulations depicted in the next three figures.

For purposes of this example, let's assume NVswap = 1 indicating the desire to execute from the MAIN FLASH memory. First we write to the VM (virtual memory) register in the PSD a value of 0x06. This action moves the MAIN FLASH area (FS0..FS7) into program space as shown in the following figure. At this point, the code residing in CSBOOT0/1 is still running.

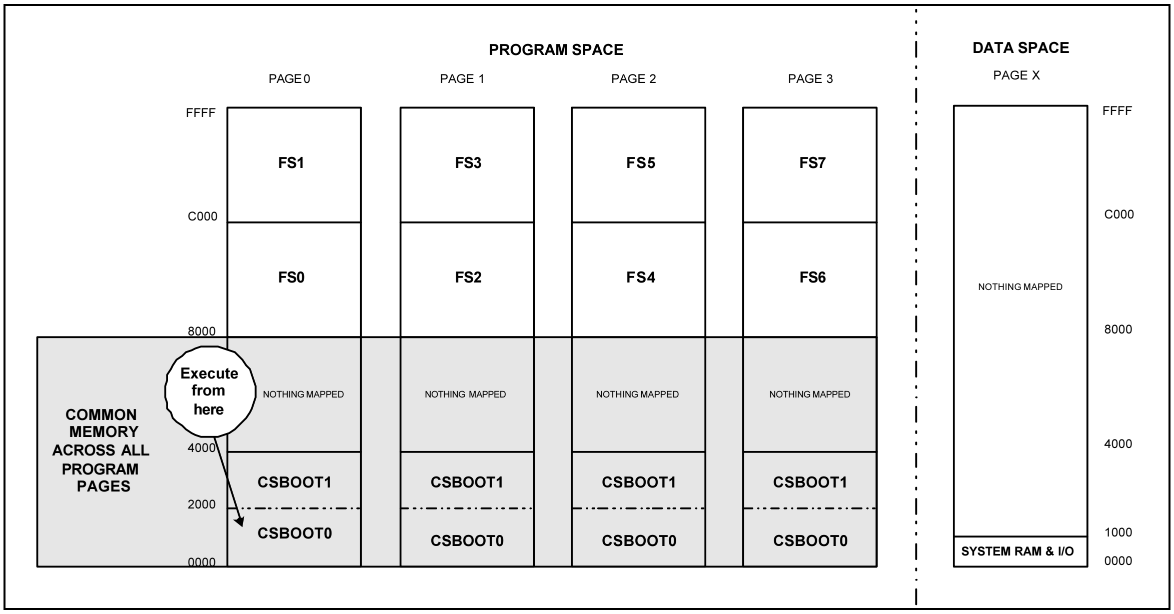


Figure 21 Memory positions after step 2 of memory swap

Next, we write to the PAGE register, to the swap bit location a value of 1. This action changes the system location where the code appears to the microcontroller moving FS0 to 0x0000 and CSBOOT to 0x8000 as shown below. After this write operation is complete, the very next instruction is fetched from FS0. Execution continues from FS0 until the next time the system is powered down.

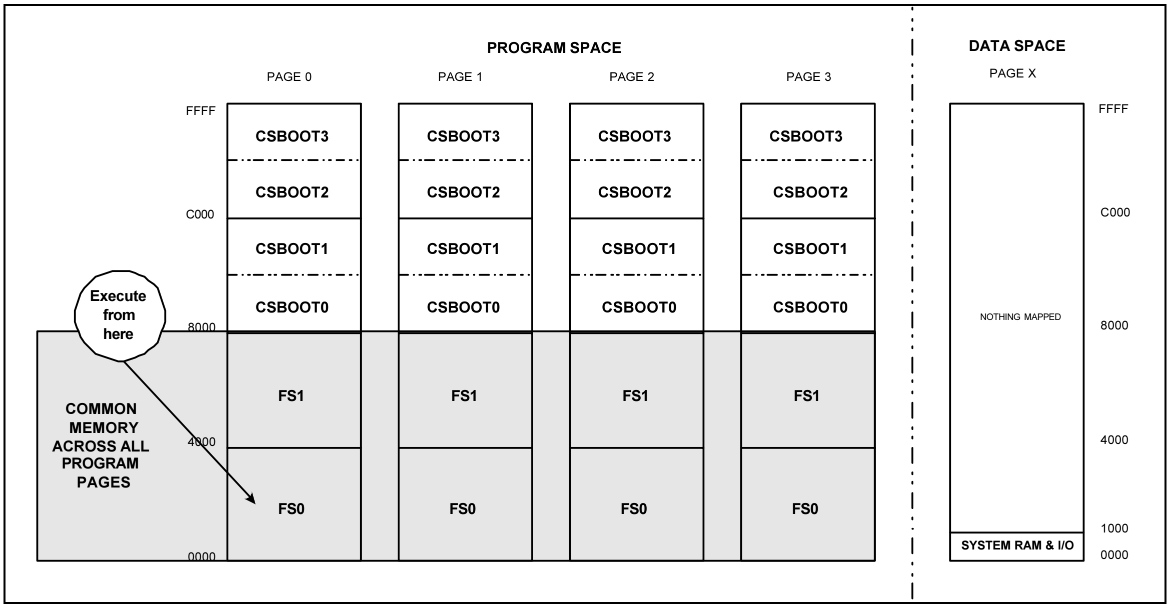


Figure 22 Memory locations after step 3 of memory swap

As a final step, the CSBOOT area is moved to data space so it can be written. This is accomplished by another write to the VM register of a value of 0x0C.

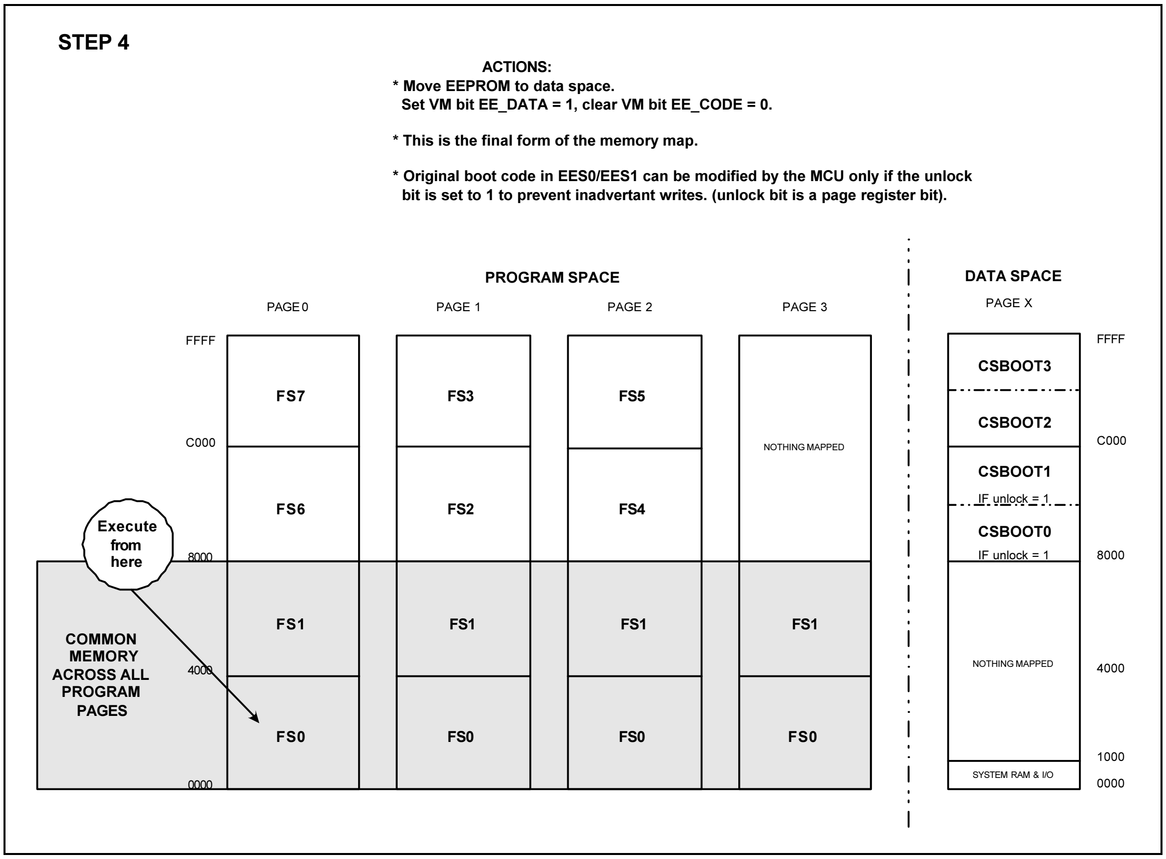

STEP 4
ACTIONS:
* Move EEPROM to data space.
Set VM bit EE_DATA = 1, clear VM bit EE_CODE = 0.
* This is the final form of the memory map.
* Original boot code in EES0/EES1 can be modified by the MCU only if the unlock bit is set to 1 to prevent inadvertant writes. (unlock bit is a page register bit).
PROGRAM SPACE
PAGE 0
PAGE 1
PAGE 2
PAGE 3
DATA SPACE
PAGE X
FFFF
C000
8000
4000
0000
FS7
FS6
FS1
FS0
FS3
FS2
FS1
FS0
FS5
FS4
FS1
FS0
NOTHING MAPPED
FS1
FS0
Execute from here
COMMON MEMORY ACROSS ALL PROGRAM PAGES
CSBOOT3
CSBOOT2
CSBOOT1
IF unlock = 1
CSBOOT0
IF unlock = 1
NOTHING MAPPED
SYSTEM RAM & I/O
FFFF
C000
8000
4000
1000
0000

Figure 23 Memory locations after final step of memory swap

With the NVswap bit set, 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 changed that memory to FS0. The new memory contents contained a completely different set of code that picked up immediately. It sounds like a stretch, but really isn't.

PSDload address translation

When a download occurs, the downloaded hexfile contains addresses appropriate for execution that, in this case is 0x0000-0x3fff for fs0. We download this data to 0x8000 –BFFF. If the addresses are in low memory how does the data get in high memory? PSDload does an address translation on every data byte in the hexfile; that is, it changes the addresses according to the download destination of 0x8000-BFFF using the following equation.

Destination address = hex file address + destination base – execution base.

For this 8031 family example, code exe(hex file) is 0x0123, dest base = 0x8000, exe base = 0x0000
Download destination = 123 + 8000 – 0 = 0x8123

While this equation may look like overkill for this example, it allows transparent PSDload operation to an MCU that boots to high memory.

Now that we've described this level of operation, lets take a bit closer look at the detailed sequence that occurs between steps 2 and 3; that is, as the memory is 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 fs0 and CSBOOT0/1. Certainly the same techniques apply with paging when using the remaining FLASH segments.

```
CSBOOT0 = ((address >= ^h0000) & (address <= ^h1FFF) & !swap )
          #   ((address >= ^hC000) & (address >= ^hDFFF ) & swap );

CSBOOT1 = ((address >= ^h2000) & (address <= ^h3FFF) & !swap )
          #   ((address >= ^hE000) & (address >= ^hFFFF ) & swap );

FS0    =        ((address >= ^h8000) & (address <= ^hBFFF) & !swap)
           # ((address >= ^h0000) & (address <= ^h3FFF) & swap  );
```

The above equation tells us that fs0 can show up in either of two places; 0x0-0x3FFF or 0x8000-0xBFFF. The choice of which location is used is based on the variable **swap,** a single bit in the PAGE register (msb). 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 fs0 must appear at 8000-BFFF and CSBOOT0 is at 0-0x1fff and CSBOOT1 is at 0x2000-3FFF. Code executes from CSBOOT0 and CSBOOT1. This is the original memory map presented in Figure 20.

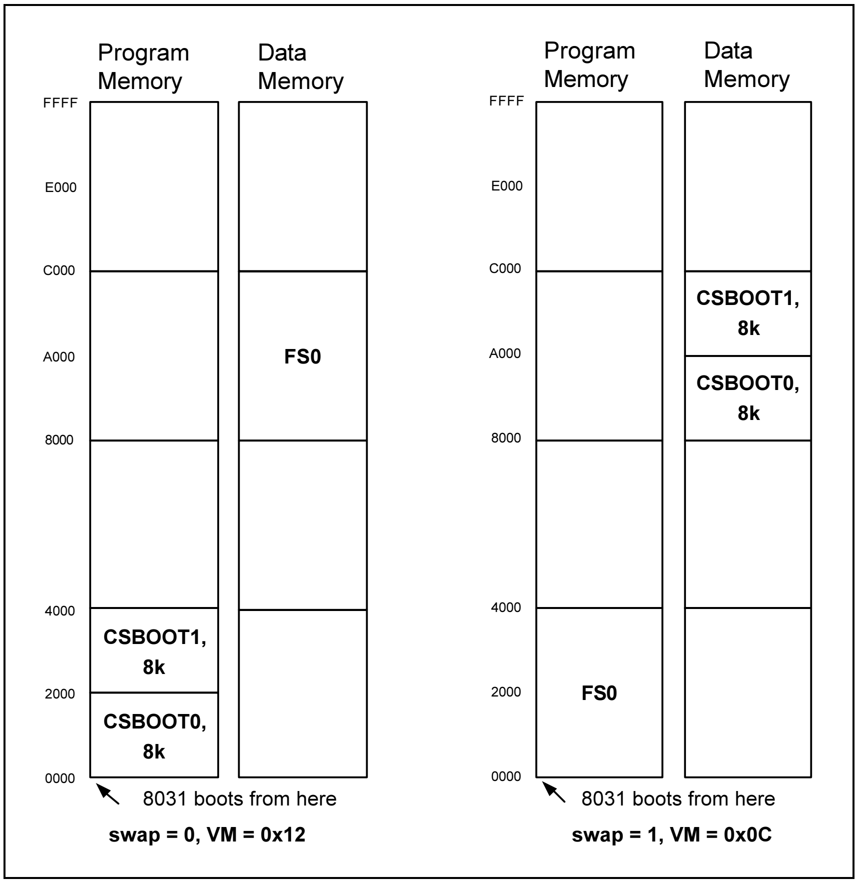


Figure 24 Segment positions with swap and VM values

After the memory contortions are completed swap=1 and VM=0C. We end up with the memory map on the right with fs0 in low program memory and CSBOOT in high data memory. Note the values for swap and VM that cause this to occur.

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.

There are a few more items involved in the seamless transition from one code bundle to the other using this method. These elements can be totally controlled by the linker and are listed below.

a. The location of certain code in the BOOT memory must be located identically to the same code in the MAIN FLASH memory. Due to this constraint the code that does the swap and VM writes is located in the c51_startup routine and used in all code bundles. This is needed since the microcontroller doesn't know anything about the memory swap, it just keeps on generating addresses. After the instruction that writes to the PAGE register, the microcontroller generates the next sequential address. The code fetch from this next address in memory 2 must be the same as if it were occurring from memory 1. This results in the microcontroller executing seamlessly without knowledge of the swap.

b. The stack must be located in the identical locations in both code bundles.

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.

PSDload example code bundles

Following are the code bundles used with the DK900 Development Kit. This code is available from the Coded Example under the Tools submenu within PSDsoft Express. As mentioned before, to get the latest check the web at www.st.com/psm .

archive	description
U8c9_10x.zip	C level source code for UART8032.
U8p9_10x.zip	Psd code for UART8032.
U8c9a10x.zip	Sample app for uart download, uart1
U8c9b10x.zip	Sample app for uart download, uart2

Table 2 Software included with Development Board

- U8c9_10x
  This is the C level source code used for ISP download earlier in the document. This includes full uart functionality.
- U8p9_10x
  This is the psd design files that match with the above ISP code.
- U8c9a10x
  This is the C level source code used for IAP described earlier in this document(uart1).
- Uart2
  This is the C level source code used for IAP described earlier in this document(uart2).

A detailed look at the IAP example implementation

The previous example uses two code bundles; UART8032_C and UART1_C. Note that UART2_C is essentially the same as UART1_C for the purposes of this discussion. 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 code to see how it works.

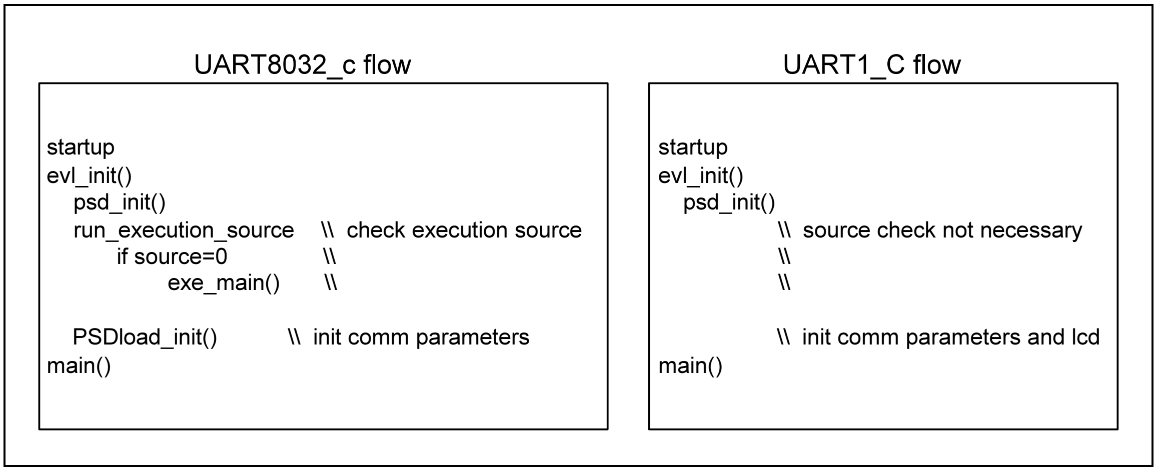
For purposes of this discussion, the code is broken into three components as listed below.

1. Top level flow charts for UART8032_C and UART1_C
2. Top level flow for return from main memory execution
3. Detailed flows for startup.a51 for the UART8032_C and UART1_c

Top level functional flow

Let's start with the top level flows. As an aside, the main action occurs in the startup.a51 file, but lets leave that till last. Notice the symmetry between UART8032_C and UART1_C. They are identical except the test and check for run_execution_source that is in UART8032_C but not in UART1_C.

Now let's see what keeps execution in UART8032 in the BOOT area. The value of the variable "source" (base+0x70) resides in the FS0 segment that, on a new part, is 0xff since it's erased. This value indicates to execute from BOOT FLASH. As you can see, the if statement in this case is false so execution continues from the BOOT area. That is, exe_main() is not executed. So, when UART8032_C boots, the if statement is not true and execution remains in the BOOT area. This is the state after the ISP download and before IAP download. If power is cycled, the code always does executes from the BOOT area.



UART8032_c flow

```
startup
evl_init()
   psd_init()
   run_execution_source    \\ check execution source
        if source=0         \\
              exe_main()    \\

   PSDload_init()        \\ init comm parameters
main()
```

UART1_C flow

```
startup
evl_init()
   psd_init()
              \\ source check not necessary
              \\
              \\

              \\ init comm parameters and lcd
main()
```

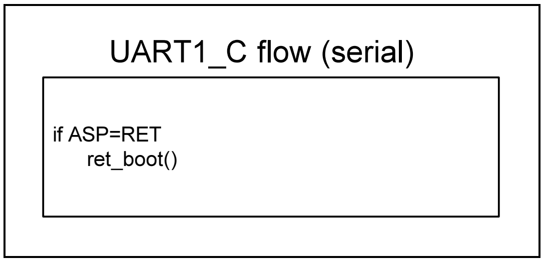
Figure 25 Top Level flows, UART8032_C and UART1_C

The next thing that occurs is the IAP download. UART1_C code is downloaded via the serial channel to FS0(MAIN FLASH area). At this point, no functional changes are observed on the display. The new code is resident in FS0 but not active. As a component of this download, the value of "source" has changed to 00. Now, when the Reset command is issued from PSDload, the if statement in UART8032_C flow is true and the routine exe_main() executes. This changes the values of the VM and PAGE register to enable the main area to execute and clears out the stack so that UART1_C can continue to run. These details are discussed later.

Note that in the UART1_C flow there is no source check. That is, run_execution_source() is not included. This is because the manipulations required for MAIN FLASH to run have already been taken care of since the variable "source"=00.

Now, the UART1_C code is running as evidenced by the display. Note that, in this code, only a subset of the serial functionality is included(ASP and RST). This limitation was pursued in order to make UART1_C able to be compiled with the evaluation version of the Keil compiler(2k exe).

Now, all we need to do is regain execution from the BOOT area code. Following is the top level flow that results in the return to BOOT memory execution; that is, the issue of the ASP instruction (User Defined) from PSDload with an argument of RET. The routine ret_boot causes the VM and PAGE register manipulations allowing the BOOT area to regain control.



```
if ASP=RET
     ret_boot()
```

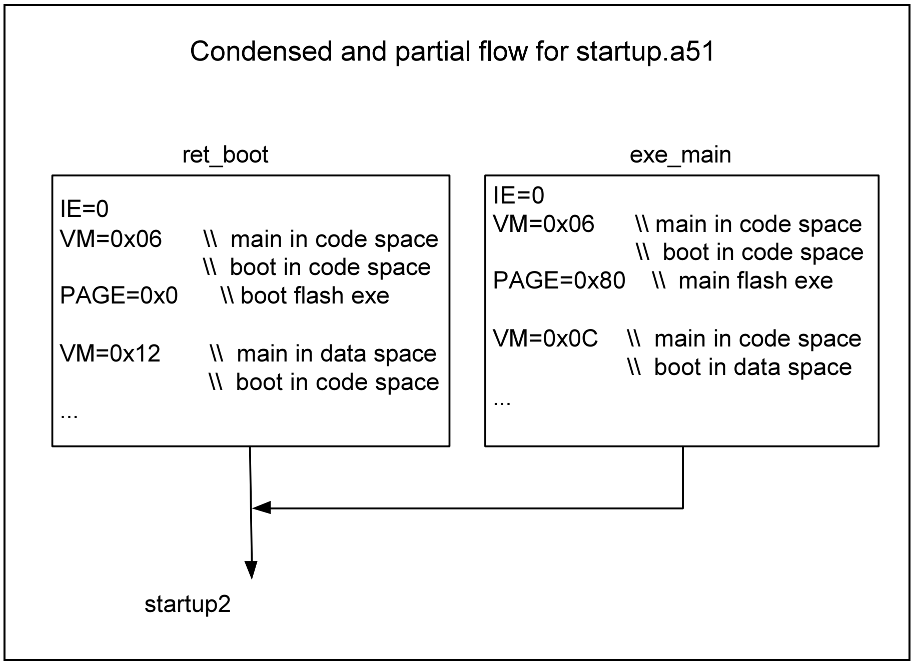
Figure 26 Flow to return to BOOT memory execution

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.

Detailed flow, startup.a51

Now that the top level flow has been discussed, lets turn out attention to the startup.a51 routine. This module contains the code that manipulates the VM and PAGE registers that allows the boot loader to relinquish and regain control. In addition, other necessary details are handled such as stack manipulation.

Appendix E and F include the source startup.a51 files from each of these bundles. The only differing element between these files is the byte EXECUTION_SOURCE. UART8032 declares this as storage only while UART1_C sets this byte to 0. At boot, the byte is read, and the following action is taken based on it's value; if source=0xFF, then the execution proceeds from BOOT area. If source=00, then execution proceeds from MAIN FLASH. These are depicted below.



```
IE=0
VM=0x06      \\  main in code space
             \\  boot in code space
PAGE=0x0      \\ boot flash exe

VM=0x12       \\  main in data space
              \\  boot in code space
...
```

```
IE=0
VM=0x06     \\ main in code space
            \\  boot in code space
PAGE=0x80    \\  main flash exe

VM=0x0C    \\  main in code space
           \\  boot in data space
...
```

Figure 27 Partial flow of startup.a51

You may notice, in the top level discussion, that the two above routines, ret_boot() and exe_main() are called from differing positions in the code. While this is common practice for a subroutine, we do not want to return from these routines as a subroutine normally would return. The desired operation at the invocation of either of these routines is to precipitate a system reset. This allows desired memory swap to occur bringing the new code into execution position.

How to create your own app for UART Download

Typically, getting a single application to run is relatively straightforward as the linker (and user) make sure 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 the linker. Typically no code changes are required.

First,a quick review of what we're trying to do. We are attempting to smoothly transition fromone 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
2. Flag indicating desire to jump from BOOT memory to main memory. The demo uses the variable "source" described in the previous section.
3. Method to tell system of desire to return from main memory to BOOT memory. The demo uses the User Defined command (ASP) with an argument of RET described in the previous section.

When using a PSD, we recommend the use of our startup.a51 routines or an equivalent included in the code bundles. The specifics of the VM and PAGE registers are already worked out for your convenience. The code placement issues are serviced in the file itself with conventional "CSEG at" statements resulting in no linker directives being needed.

For STEP 2 (flag indicating desire to jump from BOOT memory to main memory; "Nvswap" or "source"), the described flag should be set after the downloaded code is successfully transferred and validated. Then, after the system is rebooted, the new location is automatically delivered. Depending on your application, this element can be either volatile or non-volatile. The motivation to use a non-volatile method is that the desired boot source can be carried through a power outage. If a volatile medium is acceptable in your application, a convenient holder for this variable is the internal PSD ram.

For STEP 3 (method to tell the system of desire to return from main memory to BOOT memory), the method can be conveyed to the software by virtually any means from a simple mechanical switch or, for remote operation, via some communications medium. Once this is done, the ret_boot() is run, manipulating the VM and PAGE register to the desired states and rebooting the system.

The code content and positioning after the initialization code (startup.a51) 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 all the critical code placement is handled within a single file; startup.a51.

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 Turorial for 8032/PSD9XXF

Appendix

Appendix A - Jumper configuration on DK900 eval board

Setting of MCUs Power pins

JP1(PIN12), JP2(PIN1), JP3(PIN34), JP4 (PIN23)

Some 8031/32 series use pins 1, 12, 23 and 34 ofthe PLCC package as additional power input pins. You can set proper power to these pins with these jumpers.

Typical setting of several MCUs

	INTEL 80C31/32 *	WINBOND W78C3x	DALLAS DS803x0	PHILIPS 8051XA	SIEMENS 80C511/3	INTEL 80C251
JP1						Close (VCC)
JP2				Close (VSS)		Close (VSS)
JP3						Close (VSS)
JP4			1-2 (VSS)	2-3 (VCC)		1-2 (VSS)

* Default = 8031/32

Setting of MCU RESET polarity (JP8)

MCU RESET polarity can be chosen using this jumper.

1-2 Active HIGH reset
2-3 Active LOW reset

Typical setting of several MCUs.

JP8	**INTEL 80C31/32 ***	WINBOND W78C3x	DALLAS DS803x0	PHILIPS 8051XA	SIEMENS 80C511/3	INTEL 80C251
	1-2	1-2	1-2	2-3	1-2	1-2

* Default = 8031/32, Active HIGH reset (1-2).

Connection of PHILIPS 8051XA's Low addresses (A0-A3) (JP6) *

In the case of PHILIPS 8051X, low address (A0-A3) should be connected to Port A (PA0-PA3) of PSD9xxFx. (See PSD9xxF2 data sheet)
These low address bits can be connected to Port A of PSD9xxF2 through JP6.

PSD81Fx	8051XA
Port A0	A0
Port A1	A1
Port A2	A2
Port A3	A3

* Default = 8031/32, All of JP6 pins are not connected.

PSD9xxF2 Latched address out function for 8031 Families.

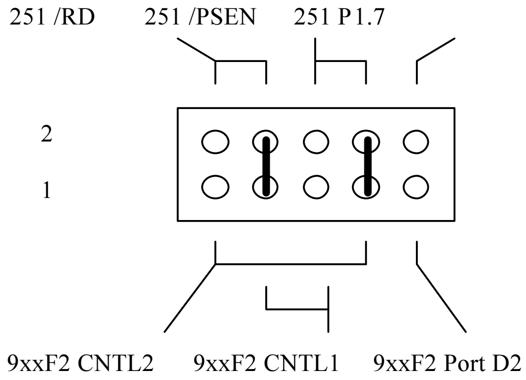
	Port A		Port B	
	Port A (3:0)	Port A (7:4)	Port B (3:0)	Port B (7:4)
8051XA		Address [7:4]	Address [11:8]	
251 (page mode)			Address [11:8]	Address [15:12]
others	Address [3:0]	Address [7:4]	Address [3:0]	Address [7: 4]

For more details, see FLASH PSD9xxF2 data sheet
(Reference: 9.3 Microcontrollers Bus Interface, 9.4 I/O Ports)

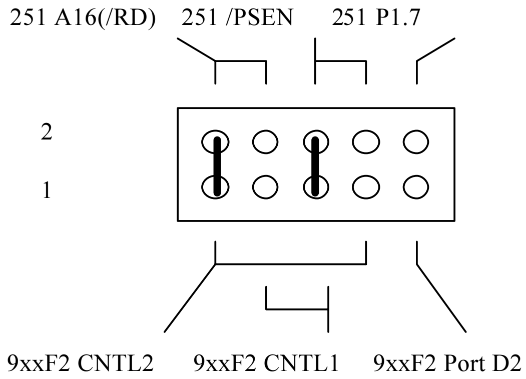
Connection of 80251's control signals for each mode (/PSEN, /RD, /WR) (JP5)

(Reference: 9.3 Microcontrollers Bus Interface)

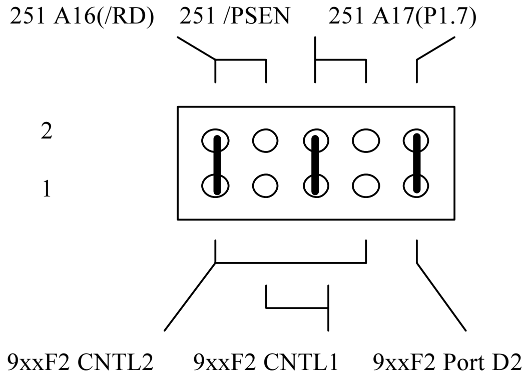
(a) 16 bits address mode (8031 compatible mode)
Default setting, no need to change jumpers.



(b) 17 bits address mode
/PSEN and /WR will be used as control signals, /RD pin will out A16.



(c) 18bit address mode
/PSEN and /WR will be used as control signal, /RD and P1.7 will out A16 and A17respectively.
CNTL2 and PD2 of PSD9xxF2 will be used as general PLD input to decode internal/external resources.



PSD SRAM Battery Backup Enable/Disable (JP9)

Default setting of this jumper is ON (close), but a battery should be connected to use this function and FLASH PSD9xxF2 should be re-programmed with a new configuration that PC2 configured to Vstby input in PSDsoft Express/Device Config/Other.
To program a new configuration, download PSDsoft Express design file and 8031 sources from ST web site (www.st.com) and modify them.

PSD's power consumption measurement point (JP7)

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

The measured PSD's current will be,
Icc = PSD Icc + PSD Ic (I/O ports) + MCU Bus leakage Ic

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.

32Kbyte SRAM Expansion (62256 / 68257)

(a) 8051XA mode (Use a upper location marked as XA)

A14	A14		VCC
A12	A12	/WE	/WR
PA7 (A7)	A7	A13	A13
PA6 (A6)	A6	A8	PB0 (A8)
PA5 (A5)	A5	A9	PB1 (A9)
PA4 (A4)	A4	A11	PB3 (A11)
A3	A3	OE	PB6 (/RAM_OE)
A2	A2	A10	PB2 (A10)
A1	A1	/CS	PB5 (/RAM_CS)
A0	A0	D7	A11/D7
A4/D0	D0	D6	A10/D6
A5/D1	D1	D5	A9/D5
A6/D2	D2	D4	A8/D4
	GND	D3	A7/D3

(b) 8031 / 80251 Non-page mode (Use lower location marked as 51)

A14	A14	VCC	
A12	A12	/WE	/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	PB6 (/RAM_OE)
PA2 (A2)	A2	A10	A10
PA1 (A1)	A1	/CS	PB5 (/RAM_CS)
PA0 (A0)	A0	D7	AD7
AD0	D0	D6	AD6
AD1	D1	D5	AD5
AD2	D2	D4	AD4
	GND	D3	AD3

System expansion connectors (J1,J2,J3)

J1 (8031)

1	2
P1.0	
P1.1	AD0
P1.2	AD1
P1.3	AD2
P1.4	AD3
P1.5	AD4
P1.6	AD5
P1.7	AD6
*RESET	AD7
P3.0	
P3.1	ALE
P3.2	/PSEN
P3.3	A15
P3.4	A14
P3.5	A13
/WR	A12
/RD	A11
	A10
	A9
GND	A8

J2 (PSD9xxF2)

1	2
PA0	PA1
PA2	PA3
PA4	PA5
PA6	PA7
GND	GND
PB0	PB1
PB2	PB3
PB4	PB5
PB6	PB7
/RESET	GND
PC0	PC1
PC2	PC3
PC4	PC5
PC6	PC7
GND	GND
PD1	PD2

*Polarity of /RESET pin of J1 could be chosen by setting of JP8.

JP3 (78C33)

1	2
P4.0	P4.2
P4.1	P4.3

Others

(a) Battery power connector and re-charging circuit

When using re-chargeable battery as power source, you can use the prepared 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 to Manganese, Lithium or NiMH 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)

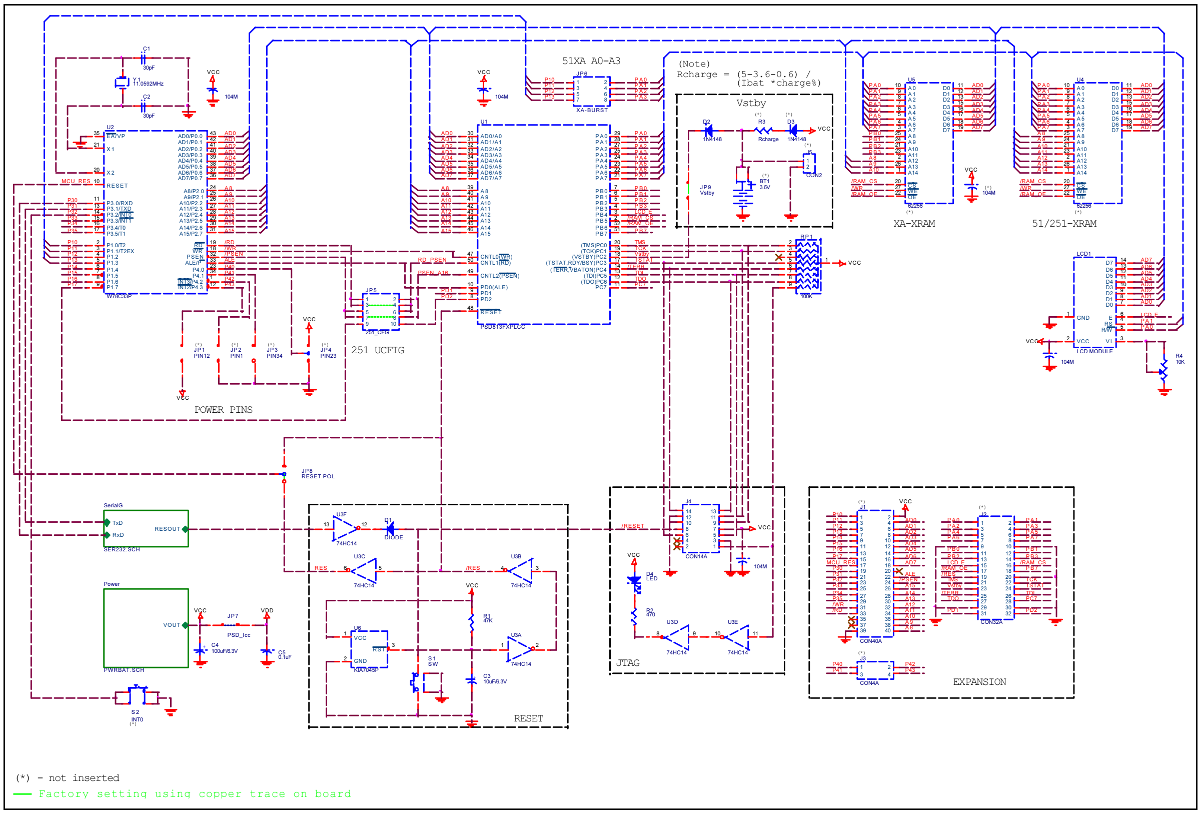
*) Do not use charging circuit to Manganese, Lithium or NiMH batteries.

(d) Connection between this Eval kit with PC

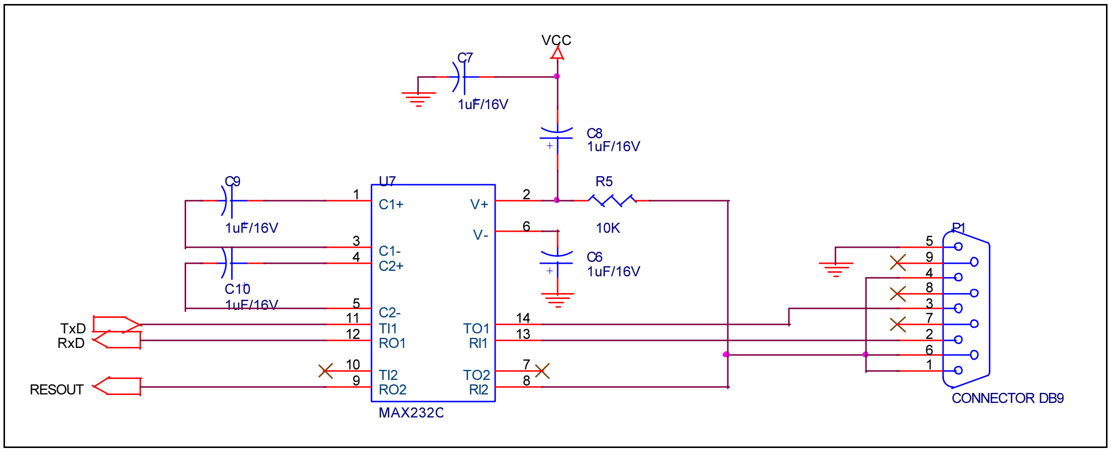
You need a null-modem serial cable, and use PSDload in Windows95/98/NT as host application. The kit baud rate is fixed at 19200bps.

Appendix B Development Board Schematic and parts list

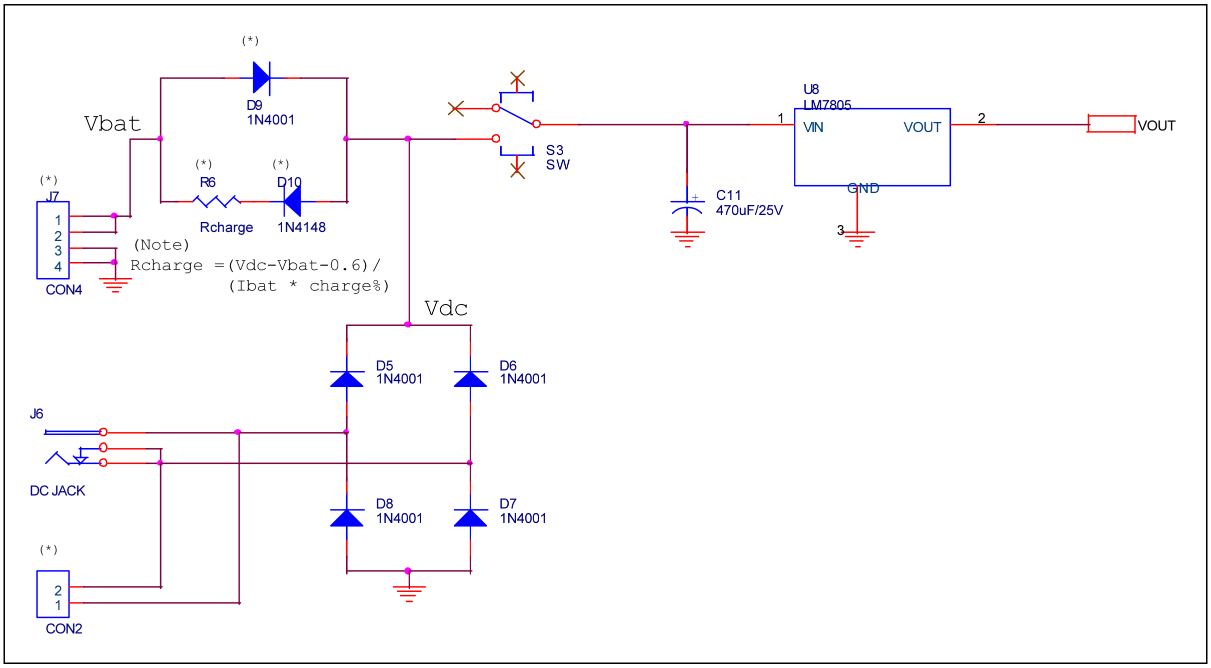
Main Schematic



Serial Port Schematic



Power Supply Schematic



Eval Board Parts List

No.	description	part number	Q'ty
1	8032 MCU (40MHz)	LGS90C32PL	1
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	11.0592MHz	1
9	block resister array	AR100K-09P	1
10	resister	47K 1/8W	1
11	resister	10K 1/8W	1
12	resister	470 1/8W	1
13	potentiometer	GF06S10K	1
14	diode (switching)	1N 4148RL	2
15	diode (rectifier)	1N 4002RL	4
16	electrolytic capacitor 1uF/50V	EC1U50V	5
17	electrolytic capacitor 10uF/16V	EC10U16V	1
18	electrolytic capacitor 470uF/16V	EC470U16V	1
19	electrolytic capacitor 100uF/6.3V	EC100U6.3V	1
20	ceramic capacitor 18pF	CC18	1
21	ceramic capacitor 39pF	CC39	1
22	monolithic capacitor 0.1uF/50V	M104	4
23	LED (green, 3mm)	BL-B2141-3D	1
24	power switch (slide 3P)		1
25	reset switch		1
26	SIP 2 pin header		1
27	SIP 14 pin header (LCD side)		1
28	SIP 14 pin connector (PCB side)		1
29	DB-9 connector	DB-9SR	1
30	DC-JACK		1
31	7x2 pin ribbon cable w/ male con. (150mm)		1
32	7x2 pin connector (angle)		1
33	standoffs (3 mm x 10 mm) for LCD		2
34	bolt, nut (2.6 mm x 16mm) for LCD		2
35	anti-static bag (170 mm x 300 mm)		1
36	box (110 mm x 150 mm x 24 mm)		1
37	LCD module		1
38	standoffs for PCB board		4
39	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.

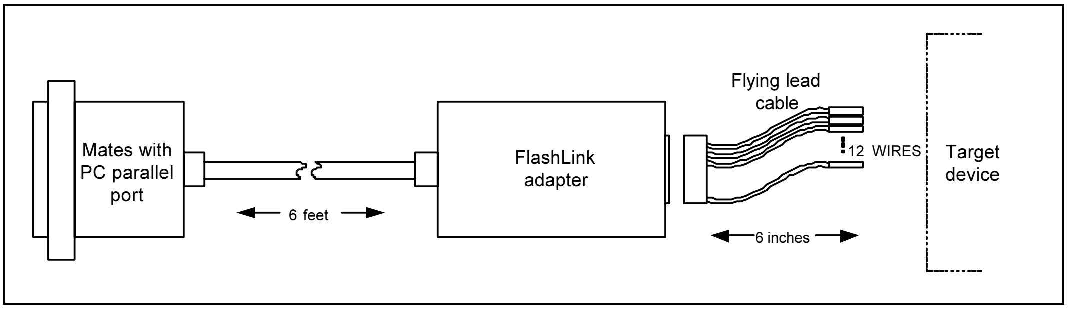


Figure 28 Typical FLASHlink application

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 verneer 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 = ST EHANCED 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	**GND**	Signal ground		
4	CNTL *	Generic control signal, (optional)	OC,100K	Source
5	**TDI**	JTAG serial data input		Source
6	TSTAT	EJTAG programming status (optional)		Destination
7	**Vcc**	VDC Source from target (2.7 - 5.5 VDC)		
8	RST\	Target system reset (recommended)	OC,10K	Source
9	**TMS**	JTAG mode select		Source
10	GND	Signal ground		
11	**TCK**	JTAG clock		Source
12	GND	Signal ground		
13	**TDO**	JTAG serial data output		Destination
14	TERR\	EJTAG programming error (optional)		Destination

Notes

1. **Bold** 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 29 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

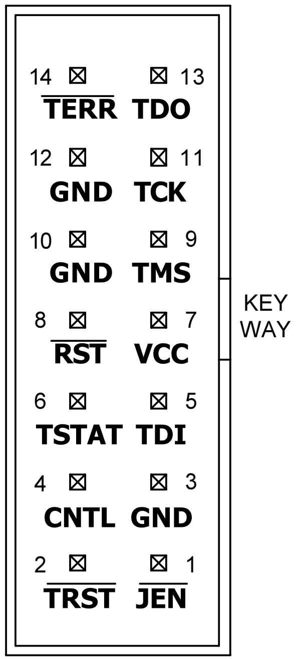
(2) Two additional ground lines to help reduce EMI if a ribbon cable is used. These ground lines "sandwich" the TCK signal in the ribbon cable. These lines are not needed for use with the flying lead cable, that is why the flying lead cable has only 12 of 14 wires populated.

FLASHlink pinouts

There is no "standard" JTAG connector. Each manufacturer differs. ST 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.

ST ENHANCED JTAG ISP CONNECTOR DEFINITION


14 ☒ ☒ 13
$\overline{\text{TERR}}$ TDO
12 ☒ ☒ 11
GND TCK
10 ☒ ☒ 9
GND TMS
8 ☒ ☒ 7
$\overline{\text{RST}}$ VCC
6 ☒ ☒ 5
TSTAT TDI
4 ☒ ☒ 3
CNTL GND
2 ☒ ☒ 1
$\overline{\text{TRST}}$ $\overline{\text{JEN}}$
KEY
WAY

VIEW: LOOKING INTO FACE OF SHROUDED MALE CONNECTOR. 0.025" POSTS ON 0.1" CENTERS.

Connector reference: Molex 70247-1401

Recommended ribbon cable for quick connection of FlashLink adapter to end product:
Samtec: HCSD-07-D-06.00-01-S-N
or
Digikey: M3CCK-14065-ND

Note:
TDI is a signal source on the Flashlink and a signal destination on the target board.

TDO is a signal destination on the FlashLink and a signal source on the target board.

Figure 30 Pinout for FlashLink Adapter and Target System

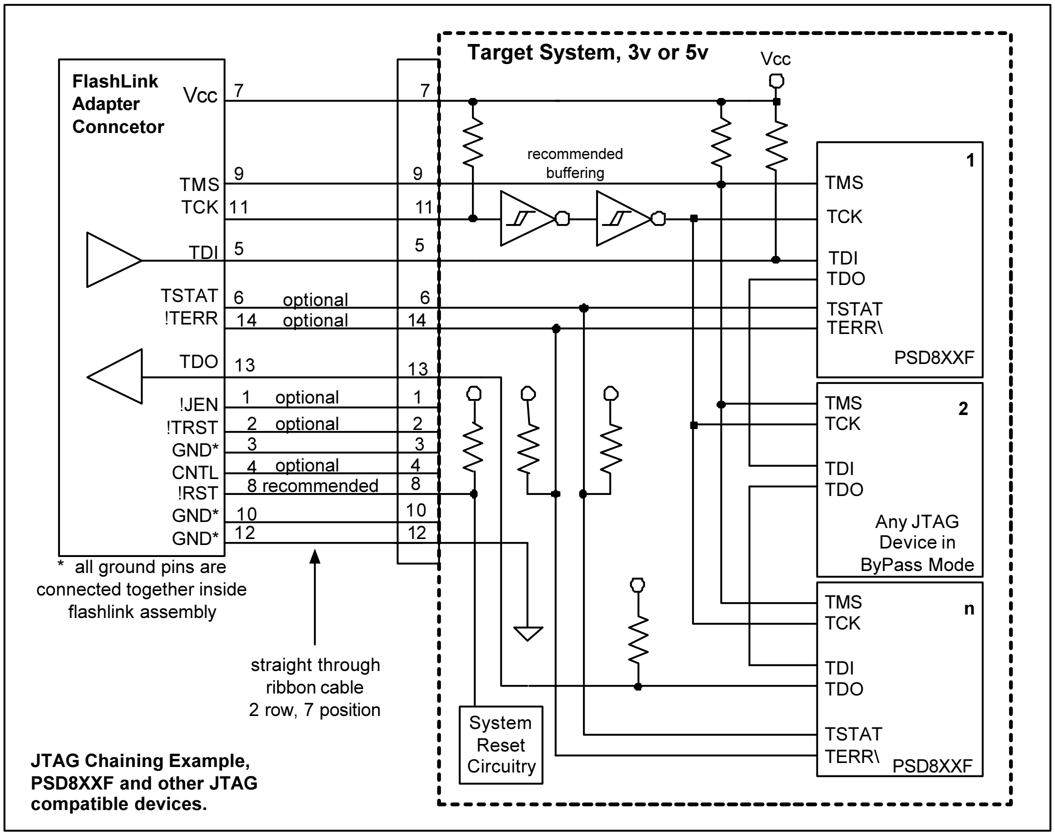
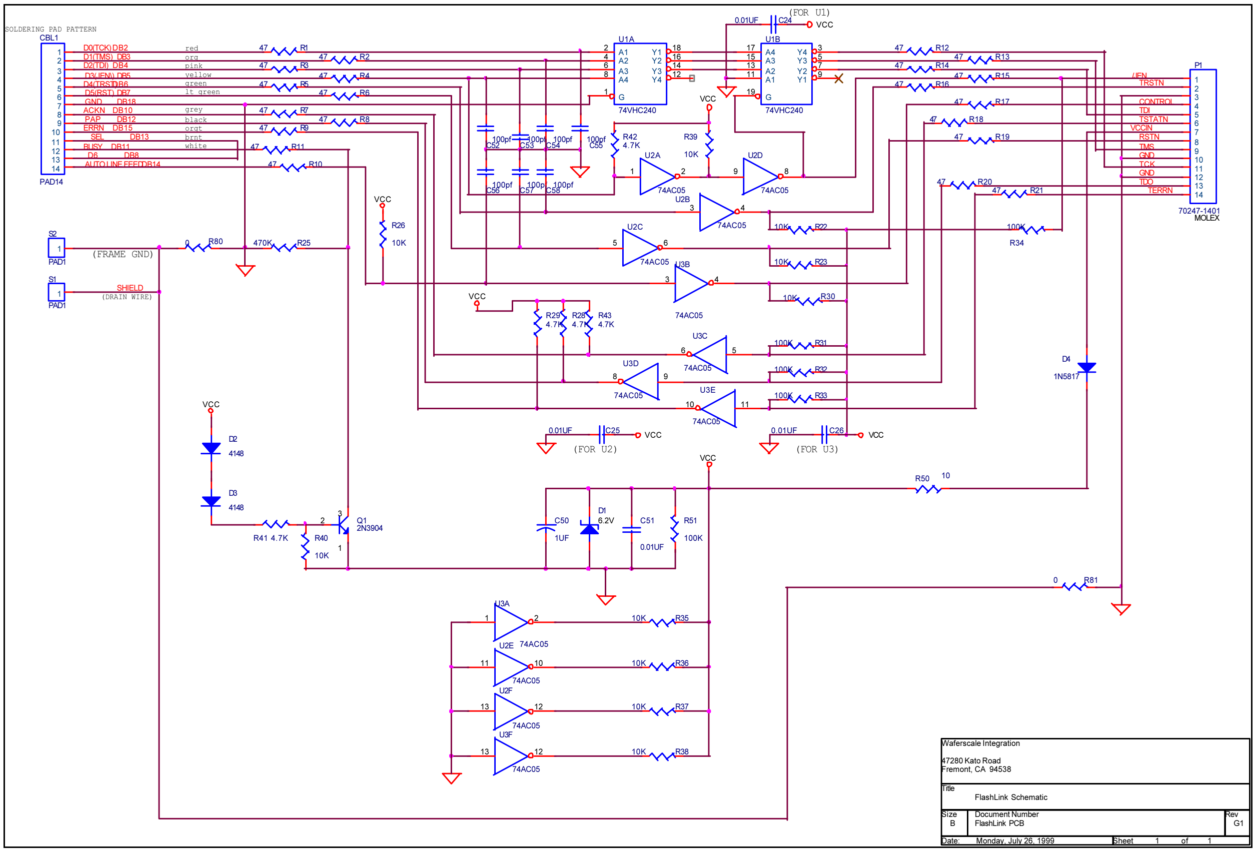


Figure 31 JTAG Chaining Example

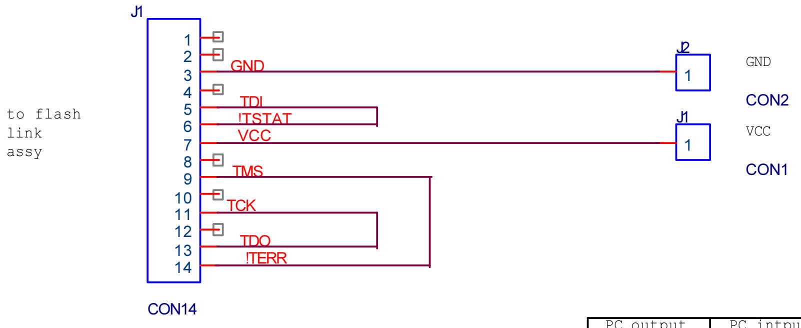


Waferscale Integration

47280 Kato Road
Fremont, CA 94538

Title		
	FlashLink Schematic	
Size B	Document Number FlashLink PCB	Rev G1
Date:	Monday, July 26, 1999	Sheet 1 of 1

Loop back connector schematic



14 pin dual row 0.025 sq
receptacle(polarized, same as cable 5)

PC output signal	PC intput signal	PC connector line
TDI	!TSTAT	ACKN (8)
TMS	!TERR	ERRN (10)
TCK	!TDO	PAP (9)

Figure 32 Loop Back Tester, Passive, FLASHlink

Appendix D Source code for C51_startup, UART8032

Modified from original Keil source code for memory swapping.

```
;--------------------------------------------------------------------------
;  This file is part of the C51 Compiler package
;  Copyright (c) 1988-1997 Keil Elektronik GmbH and Keil Software, Inc.
;--------------------------------------------------------------------------
;  STARTUP.A51:  This code is executed after processor reset.
;
;  To translate this file use A51 with the following invocation:
;
;     A51 STARTUP.A51
;
;  To link the modified STARTUP.OBJ file to your application use the
following
;  BL51 invocation:
;
;     BL51 <your object file list>, STARTUP.OBJ <controls>
;
;--------------------------------------------------------------------------
;
;  User-defined Power-On Initialization of Memory
;
;  With the following EQU statements the initialization of memory
;  at processor reset can be defined:
;
;     ; the absolute start-address of IDATA memory is always 0
IDATALEN EQU   0H ; the length of IDATA memory in bytes.
;
XDATASTART  EQU   0H ; the absolute start-address of XDATA memory
XDATALEN EQU   0H ; the length of XDATA memory in bytes.
;
PDATASTART  EQU   0H ; the absolute start-address of PDATA memory
PDATALEN EQU   0H ; the length of PDATA memory in bytes.
;
;  Notes:  The IDATA space overlaps physically the DATA and BIT areas of
the
;          8051 CPU. At minimum the memory space occupied from the C51
;          run-time routines must be set to zero.
;--------------------------------------------------------------------------
;
;  Reentrant Stack Initilization
;
;  The following EQU statements define the stack pointer for reentrant
;  functions and initialized it:
;
;  Stack Space for reentrant functions in the SMALL model.
IBPSTACK EQU   0  ; set to 1 if small reentrant is used.
IBPSTACKTOP EQU   0FFH+1   ; set top of stack to highest location+1.
;
;  Stack Space for reentrant functions in the LARGE model.
XBPSTACK EQU   0  ; set to 1 if large reentrant is used.
XBPSTACKTOP EQU   0FFFFH+1; set top of stack to highest location+1.
;
;  Stack Space for reentrant functions in the COMPACT model.
PBPSTACK EQU   0  ; set to 1 if compact reentrant is used.
PBPSTACKTOP EQU   0FFFFH+1; set top of stack to highest location+1.
;
;--------------------------------------------------------------------------
```

```
;
;  Page Definition for Using the Compact Model with 64 KByte xdata RAM
;
;  The following EQU statements define the xdata page used for pdata
;  variables. The EQU PPAGE must conform with the PPAGE control used
;  in the linker invocation.
;
PPAGEENABLE EQU    0  ; set to 1 if pdata object are used.
PPAGE    EQU    0  ; define PPAGE number.
;
;--------------------------------------------------------------------------

      NAME  ?C_STARTUP


?C_C51STARTUP  SEGMENT   CODE
?STACK      SEGMENT   IDATA

      RSEG  ?STACK
      DS  1

      EXTRN CODE (?C_START)
      PUBLIC   ?C_STARTUP

      CSEG  AT 0
?C_STARTUP:
      LJMP  STARTUP1
;
; INT VECTORS AT HERE
;
;*************************************************************************
; Followings are some routines for SWAP and EXECUTE
;
      EXTRN XDATA(PSD8XX_reg)
      EXTRN CODE(PSDload_init, PSDload)

      CSEG  AT 33h
      PUBLIC   RET_BOOT, EXE_MAIN

; Return from main flash to boot flash
;
RET_BOOT:
      MOV      IE,#0                ; diable all interrupts

      MOV      DPTR,#PSD8XX_reg+0E2h      ; VM register
      MOV      A,#06h                  ; both MAIN and BOOT=CODE SPACE
      MOVX  @DPTR,A

         MOV      DPTR,#PSD8XX_reg+0E0h      ; PAGE register
      MOV      A,#00h                  ; SWAP=0, UNLOCK=0, PAGE=0
      MOVX  @DPTR,A

   ;****** now works in boot flash ********

         MOV      DPTR,#PSD8xx_reg+0E2h      ; VM register
         MOV      A,#12h                  ; BOOT = CODE, MAIN = DATA space
      MOVX  @DPTR,A

      MOV      SP,#?STACK-1
      LCALL PSDload_init
      LCALL PSDload

      LJMP  STARTUP2             ; execute Cstartup of BOOT

; Set SWAP and EXECUTE main flash
;
EXE_MAIN:
      MOV      IE,#0                 ; diable all interrupts

      MOV      DPTR,#PSD8XX_reg+0E2h      ; VM register
      MOV      A,#06h                  ; both MAIN and BOOT=CODE SPACE
      MOVX  @DPTR,A

         MOV      DPTR,#PSD8XX_reg+0E0h      ; PAGE register
      MOV      A,#80h                  ; SWAP=1, UNLOCK=0, PAGE=0
      MOVX  @DPTR,A

   ;****** now works in main flash ********

      MOV      DPTR,#PSD8xx_reg+0E2h      ; VM register
         MOV      A,#0Ch                        ; BOOT = DATA, MAIN = CODE space
         MOVX  @DPTR,A

      LJMP  STARTUP2             ; execute Cstartup of MAIN
;
; This location will hold execution source
; EXECUTE_SOURCE: 0xFF, execute boot flash
;               : 0x00, execute main flash
;
      CSEG  AT 70h
      PUBLIC   EXECUTE_SOURCE

EXECUTE_SOURCE:
      DS    1
;
;**************************************************************************
;
      EXTRN CODE(psd_init)
      RSEG  ?C_C51STARTUP

STARTUP1:
;
;**************************************************************************
; When port A is used to generate latched address out for external data
memories,
; port A must be initialize before entering variable initialztion as
followings.
;
;     LCALL psd_init
;
;*************************************************************************
;
STARTUP2:

IF IDATALEN <> 0
      MOV   R0,#IDATALEN - 1
      CLR   A
IDATALOOP:  MOV   @R0,A
```

```
      DJNZ  R0,IDATALOOP
ENDIF

IF XDATALEN <> 0
      MOV   DPTR,#XDATASTART
      MOV   R7,#LOW (XDATALEN)
  IF (LOW (XDATALEN)) <> 0
      MOV   R6,#(HIGH XDATALEN) +1
  ELSE
      MOV   R6,#HIGH (XDATALEN)
  ENDIF
      CLR   A
XDATALOOP:  MOVX  @DPTR,A
      INC   DPTR
      DJNZ  R7,XDATALOOP
      DJNZ  R6,XDATALOOP
ENDIF

IF PPAGEENABLE <> 0
      MOV   P2,#PPAGE
ENDIF

IF PDATALEN <> 0
      MOV   R0,#PDATASTART
      MOV   R7,#LOW (PDATALEN)
      CLR   A
PDATALOOP:  MOVX  @R0,A
      INC   R0
      DJNZ  R7,PDATALOOP
ENDIF

IF IBPSTACK <> 0
EXTRN DATA (?C_IBP)

      MOV   ?C_IBP,#LOW IBPSTACKTOP
ENDIF

IF XBPSTACK <> 0
EXTRN DATA (?C_XBP)

      MOV   ?C_XBP,#HIGH XBPSTACKTOP
      MOV   ?C_XBP+1,#LOW XBPSTACKTOP
ENDIF

IF PBPSTACK <> 0
EXTRN DATA (?C_PBP)
      MOV   ?C_PBP,#LOW PBPSTACKTOP
ENDIF

      MOV      SP,#?STACK-1
      LJMP  ?C_START

      END
```

Appendix E Source code for C51_startup, UART1

Modified from original Keil source code for memory swapping

```
;------------------------------------------------------------------------------
;  This file is part of the C51 Compiler package
;  Copyright (c) 1988-1997 Keil Elektronik GmbH and Keil Software, Inc.
;------------------------------------------------------------------------------
;  STARTUP.A51:  This code is executed after processor reset.
;
;  To translate this file use A51 with the following invocation:
;
;     A51 STARTUP.A51
;
;  To link the modified STARTUP.OBJ file to your application use the
following
;  BL51 invocation:
;
;     BL51 <your object file list>, STARTUP.OBJ <controls>
;
;------------------------------------------------------------------------------
;
;  User-defined Power-On Initialization of Memory
;
;  With the following EQU statements the initialization of memory
;  at processor reset can be defined:
;
;     ; the absolute start-address of IDATA memory is always 0
IDATALEN EQU   0H ; the length of IDATA memory in bytes.
;
XDATASTART  EQU   0H ; the absolute start-address of XDATA memory
XDATALEN EQU   0H ; the length of XDATA memory in bytes.
;
PDATASTART  EQU   0H ; the absolute start-address of PDATA memory
PDATALEN EQU   0H ; the length of PDATA memory in bytes.
;
;  Notes:  The IDATA space overlaps physically the DATA and BIT areas of
the
;          8051 CPU. At minimum the memory space occupied from the C51
;          run-time routines must be set to zero.
;------------------------------------------------------------------------------
;
;  Reentrant Stack Initilization
;
;  The following EQU statements define the stack pointer for reentrant
;  functions and initialized it:
;
;  Stack Space for reentrant functions in the SMALL model.
IBPSTACK EQU   0  ; set to 1 if small reentrant is used.
IBPSTACKTOP EQU   0FFH+1   ; set top of stack to highest location+1.
;
;  Stack Space for reentrant functions in the LARGE model.
XBPSTACK EQU   0  ; set to 1 if large reentrant is used.
XBPSTACKTOP EQU   0FFFFH+1; set top of stack to highest location+1.
;
;  Stack Space for reentrant functions in the COMPACT model.
PBPSTACK EQU   0  ; set to 1 if compact reentrant is used.
PBPSTACKTOP EQU   0FFFFH+1; set top of stack to highest location+1.
;
;------------------------------------------------------------------------------
```

```
;
;  Page Definition for Using the Compact Model with 64 KByte xdata RAM
;
;  The following EQU statements define the xdata page used for pdata
;  variables. The EQU PPAGE must conform with the PPAGE control used
;  in the linker invocation.
;
PPAGEENABLE EQU   0  ; set to 1 if pdata object are used.
PPAGE    EQU   0  ; define PPAGE number.
;
;------------------------------------------------------------------------
      NAME  ?C_STARTUP


?C_C51STARTUP  SEGMENT   CODE
?STACK      SEGMENT   IDATA

      RSEG  ?STACK
      DS 1

      EXTRN CODE (?C_START)
      PUBLIC   ?C_STARTUP

      CSEG  AT 0
?C_STARTUP:
      LJMP  STARTUP1
;
; INT VECTORS AT HERE
;

;
;***********************************************************************
; Followings are some routines for SWAP and EXECUTE
;
      EXTRN XDATA(PSD8XX_reg)
      EXTRN CODE(PSDload_init, PSDload)

      CSEG  AT 33h
      PUBLIC   RET_BOOT, EXE_MAIN

;
; Return from main flash to boot flash
;
RET_BOOT:
      MOV      IE,#0                ; diable all interrupts

      MOV      DPTR,#PSD8XX_reg+0E2h      ; VM register
      MOV      A,#06h                  ; both MAIN and BOOT=CODE SPACE
      MOVX  @DPTR,A

         MOV      DPTR,#PSD8XX_reg+0E0h      ; PAGE register
      MOV      A,#00h                  ; SWAP=0, UNLOCK=0, PAGE=0
      MOVX  @DPTR,A

   ;****** now works in boot flash ********

         MOV      DPTR,#PSD8xx_reg+0E2h      ; VM register
         MOV      A,#12h                  ; BOOT = CODE, MAIN = DATA space
      MOVX  @DPTR,A
      MOV      SP,#?STACK-1
      LCALL PSDload_init
      LCALL PSDload

      LJMP  STARTUP2             ; execute Cstartup of BOOT

;
; Set SWAP and EXECUTE main flash
;
EXE_MAIN:
      MOV      IE,#0                ; diable all interrupts

      MOV      DPTR,#PSD8XX_reg+0E2h      ; VM register
      MOV      A,#06h                  ; both MAIN and BOOT=CODE SPACE
      MOVX  @DPTR,A

         MOV      DPTR,#PSD8XX_reg+0E0h      ; PAGE register
      MOV      A,#80h                  ; SWAP=1, UNLOCK=0, PAGE=0
      MOVX  @DPTR,A

   ;****** now works in main flash ********

      MOV      DPTR,#PSD8xx_reg+0E2h      ; VM register
         MOV      A,#0Ch                  ; BOOT = DATA, MAIN = CODE space
         MOVX  @DPTR,A

      LJMP  STARTUP2             ; execute Cstartup of MAIN
;
; This location will hold execution source
; EXECUTE_SOURCE: 0xFF, execute boot flash
;               : 0x00, execute main flash
;
      CSEG  AT 70h
      PUBLIC   EXECUTE_SOURCE

EXECUTE_SOURCE:
      DB    0                       ; SOURCE=0, autorun in next startup
;
;**************************************************************************
;
      EXTRN CODE(psd_init)
      RSEG  ?C_C51STARTUP

STARTUP1:
;
;**************************************************************************
; When port A is used to generate latched address out for external data
memories,
; port A must be initialize before entering variable initialztion as
followings.
;
;     LCALL psd_init
;
;**************************************************************************
;
STARTUP2:

IF IDATALEN <> 0
```

```
      MOV   R0,#IDATALEN - 1
      CLR   A
IDATALOOP:  MOV   @R0,A
      DJNZ  R0,IDATALOOP
ENDIF

IF XDATALEN <> 0
      MOV   DPTR,#XDATASTART
      MOV   R7,#LOW (XDATALEN)
  IF (LOW (XDATALEN)) <> 0
      MOV   R6,#(HIGH XDATALEN) +1
  ELSE
      MOV   R6,#HIGH (XDATALEN)
  ENDIF
      CLR   A
XDATALOOP:  MOVX  @DPTR,A
      INC   DPTR
      DJNZ  R7,XDATALOOP
      DJNZ  R6,XDATALOOP
ENDIF

IF PPAGEENABLE <> 0
      MOV   P2,#PPAGE
ENDIF

IF PDATALEN <> 0
      MOV   R0,#PDATASTART
      MOV   R7,#LOW (PDATALEN)
      CLR   A
PDATALOOP:  MOVX  @R0,A
      INC   R0
      DJNZ  R7,PDATALOOP
ENDIF

IF IBPSTACK <> 0
EXTRN DATA (?C_IBP)

      MOV   ?C_IBP,#LOW IBPSTACKTOP
ENDIF

IF XBPSTACK <> 0
EXTRN DATA (?C_XBP)

      MOV   ?C_XBP,#HIGH XBPSTACKTOP
      MOV   ?C_XBP+1,#LOW XBPSTACKTOP
ENDIF

IF PBPSTACK <> 0
EXTRN DATA (?C_PBP)
      MOV   ?C_PBP,#LOW PBPSTACKTOP
ENDIF

      MOV      SP,#?STACK-1
      LJMP  ?C_START

      END
```

Table 1. Document Revision History

Date	Rev.	Description of Revision
	1.0	Document written in the WSI format
30-Jan-2002	1.1	DK900: DK900 Development Kit For PSD9xxF Family of Flash PSDs Front page, and back two pages, in ST format, added to the PDF file Any 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.

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

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