AN2396
Application note

uPSD3400 series design guide for DK3400 using RIDE and CAPS

Introduction

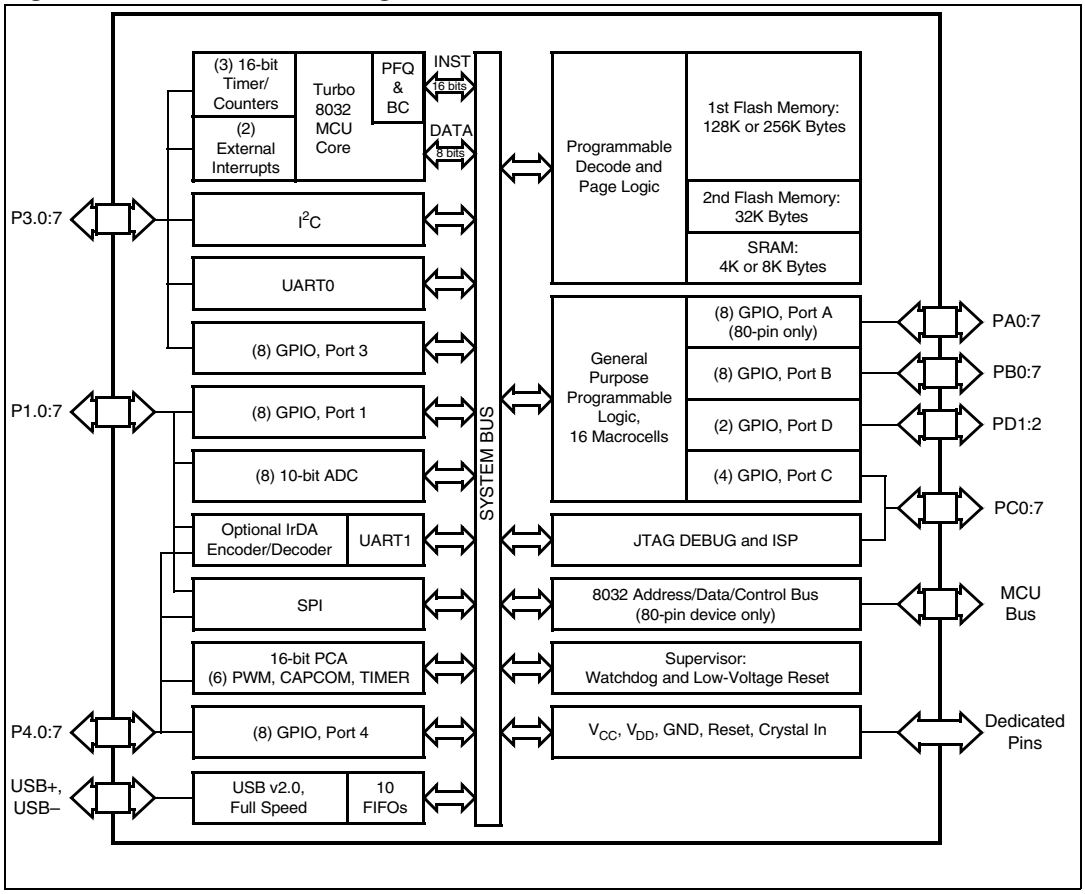
This application note provides guidelines for creating and developing applications for the Turbo+ uPSD Family of devices and shows a number of key steps to follow for creating a design based on the DK3400 Development Kit. The Kit includes code examples discussed in this document.

Here, the basic flow is provided for creating a project using the Raisonance Integrated Development Environment (RIDE) tools. A simple application included in the Kit is demonstrated using RIDE and shows the key features of RIDE. The key steps in designing an application are enumerated in this document. CAPS, a key tool in using Turbo+ uPSD, is explained in detail by illustrating the design used for the demonstration section. CAPS supports ST's FlashLINK and Raisonance's JTAG programmer (RLINK-ST).

As shown in *Figure 1*, the uPSD3400 family is a series of 8051-class microcontrollers (MCUs) containing a new fast Turbo+ 8032 core with a large dual-bank flash memory, a large SRAM, many peripherals, programmable logic, and JTAG In-System Programming (ISP). Please see the uPSD on-line resources page for latest documentation and other referenced User Guides and Application Notes at the following URL: *http://www.st.com/mcu*.

1 uPSD3400 family

Figure 1. General block diagram of the uPSD3400



1.1 uPSD3400 family overview

The uPSD3400 family is a Turbo+ 4-clock per instruction 8032 MCU capable of being clocked up to 40MHz at 3.3V or 5.0V at industrial operating temperature range. Currently there are sixteen family members that contain different combinations of flash memory size, operating voltage, and packaging (please see the full datasheet). In this Application Note, uPSD3434E-40U6 is used as the example. The term "Turbo+ uPSD" is used throughout the remainder of the document for brevity (see the Turbo+ uPSD3434 block diagram shown in *Figure 2*).

The Turbo+ uPSD family has a unique memory structure that includes two independent flash memory arrays (Main and Secondary) capable of read-while-write operations. This is ideal for In-Application Programming (IAP) because the 8032 can fetch instructions from one flash array while erasing/writing the other array. Individual sectors of each flash memory array can be mapped to virtually any 8032 address by the Decode PLD (DPLD) for total flexibility. The Turbo+ uPSD also contains a Page Register whose outputs feed the inputs of the DPLD. This allows paging (or banking) of flash memory to break the 8032's inherent limit of 64 Kbyte addresses. The 8032 may write to the Page Register at runtime.

For more complex designs, the Turbo+ uPSD is capable of placing each of the flash memory arrays (Main or Secondary) into 8032 code address space, into 8032 data space, or into both code and data space on the fly. Mapping flexibility like this supports IAP because either flash array may be temporarily placed into data space while the firmware is updated, then moved back into code space when finished, all under control of the 8032.

Many peripherals are available in this Turbo+ uPSD, including: two UART channels, one IrDA channel, one SPI channel, one I2C channel, six PWM channels, eight 10-bit ADC channels, nine Timer/Counters, a watchdog timer, low-VCC detection with reset-out, a general purpose PLD, many GPIO and a USB-JTAG Debugger.

All of the peripherals on Ports 1, 3, and 4 are controlled using 8032 Special Function Registers (SFRs).

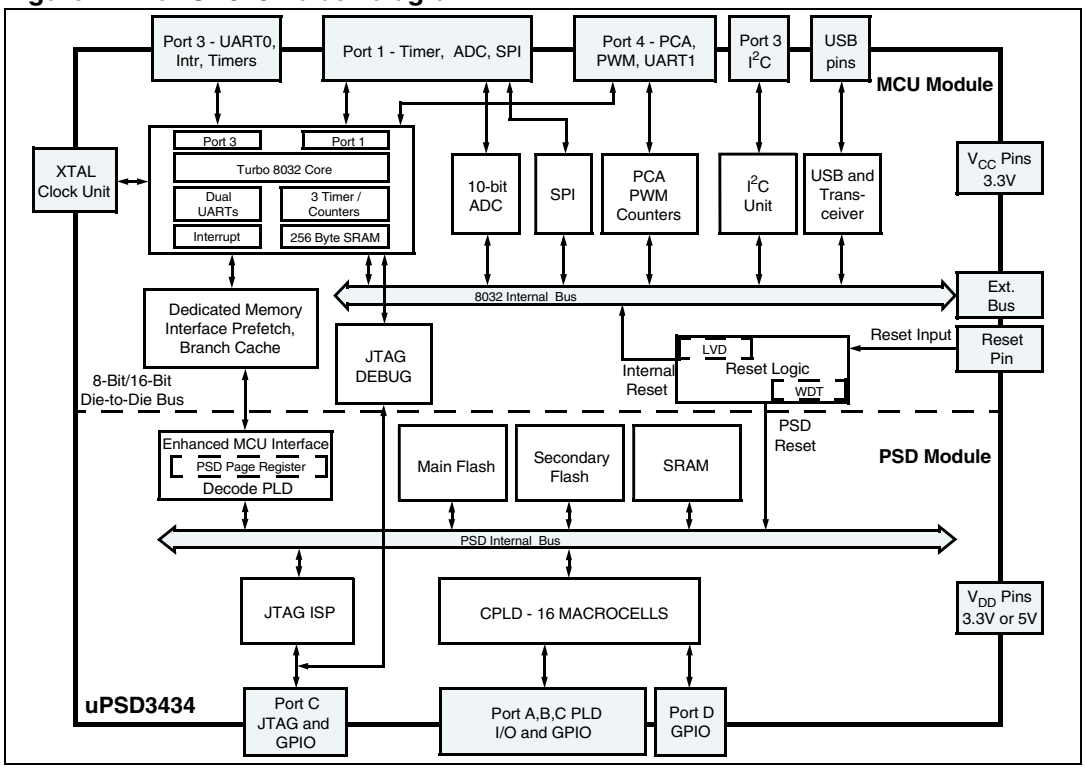
I/O Signals on ports A, B, C, and D are controlled one of two ways:

1. by a block of xdata memory mapped control registers, whose base address (csiop) can be mapped anywhere using the DPLD; and
2. by the programmable logic

In addition, Turbo+ uPSD offers a Cross-Bar I/O, which means that Peripheral functions on Port 1 are also available on Port 4 (cross-bar switch), providing more flexibility. There is no need to sacrifice one peripheral function when two functions are available on a single pin, just use the other port.

The JTAG channel on Port C is used for in-system programming (ISP) and debug of the 8032 MCU core. ISP is ideal for rapid code iterations during firmware development and for Just-In-Time inventory management during manufacturing. JTAG ISP eliminates the need for sockets and pre-programmed devices, and requires no participation of the 8032. JTAG debug eliminates the need for expensive and intrusive hardware In-Circuit Emulator (ICE).

Figure 2. uPSD3434 block diagram



2 DK3400 development kit

2.1 Overview

A picture of the DK3400 board is shown in *Figure 3*. A list of jumpers JP0 - JP12 and their functions can be found on the DK3400 board's silk screen. For more detailed information on these jumpers, please refer to *Appendix A: DK3400 jumpers selection and defaults* or the DK3400 User's Manual (UM0131, Turbo Plus uPSD DK3400 Development Kit). Board layout and schematics are also available in the User's Manual. Connectors CON1, CON2, and CON3 provide easy access to all Turbo+ uPSD signals for expansion or testing. One UART is accessible on the connector marked CN6. The FlashLINK/ RLINK-ST/ ULINK JTAG ISP cable connects at the connector, JTAG. The DK3400 includes a graphical LCD, real-time clock, serial EEPROM, IrDA transceiver, serial flash, NAND flash, and an embedded RLINK.

The sample design example code used for this application note is a RIDE based project which blinks the daughter board LED. The purpose of using this simple design project is to illustrate and demonstrate the use of Raisonance RIDE software and tools with the RLINK-ST adapter on a uPSD development board. The RIDE tools provide many features for editing, compiling, programming, and debugging a uPSD3400 MCU Series from STMicroelectronics. In the following sections, some of the main features are described to give you a feel for the simplicity and capabilities of the tools used for this sample design. A brief overview to the methods involved in importing applications developed with the Keil compiler and debugging on DK3400 using RIDE Debugger is also provided in the appendices of this document.

Figure 3. DK3400 motherboard

Figure 4. DK3400 daughter-board

2.2 Contents of DK3400 kit

STMicroelectronics provides a DK3400 Development Kit which is shipped with the following contents:

- uPSD DK3400 daughter board- with a uPSD3434E-40U6 MCU
- Motherboard with Enhanced Graphic LCD
- RLINK-ST embedded on the DK3400 daughter board, a USB-based JTAG adapter from Raisonance for debugging with Raisonance Integrated Development Environment (RIDE)
- ULINK, a USB-based JTAG adapter from Keil for debugging with Keil's uVision Tools
- USB Cables and RS232 Cables
- 110/220V Universal Power Supply Adapter
- DK3400 CD from STMicroelectronics contains:
  - STMicroelectronics Datasheets, Tools, Software, uPSD3400 sample projects
  - User Manual and Application Notes
  - Keil uVision3 Software and support Tools (Demo Version) for uPSD - (Limited to 2 Kbytes code size)
- RKit Development Suite from Raisonance contains:
  - Trial version of RIDE C-Compiler and Assembler (limited to 4 Kbytes code size)
  - RIDE Debugger Utility (no code size limit)
- ST's Configuration and Programming Software (CAPS) for configuring the Programmable Logic inside the uPSD3400

3 Project creation and sample design development process

The sections below introduced the process of using RIDE for creating application Code using the Development board and the associated tools supplied with the Kit. The key steps for creating a new project with RIDE are described. This is followed by section that uses the RIDE environment and DK3400 board to demonstrate the sample design. The main features of RIDE and its usage are then shown by loading and debugging the sample application.

The Configuration and Programming Software (CAPS) is a unique tool required in project development for the uPSD3400. It is used to design and configure the programmable logic in the uPSD3400 as well as specifying the content that is programmed into the various Flash sectors. It's covered in detail in a separate application note, but the sections that follow explain how to use it along with RIDE for project setup.

3.1 Key design development steps

Design and development of applications using Turbo+ uPSD Family of products require use of both Development Boards from STMicroelectronics or hardware developed by the user in conjunction with Software and Tools that support uPSD Devices. It is important to follow some simple steps and guidelines for successful implementation of the project. STMicroelectronics provides full support with Hardware Development Kits and Software Tools, utilities and support through the Support Website. The key design development steps for using RIDE tools are as follows:

- Identify and select the right development Kits and Tools
- Design a Block Diagram of your Application in relation to the Turbo+ uPSD
- Design the Logic and connections to be used for the PLD available in uPSD
- Create Memory Maps and inputs for programming devices using CAPS tools
- Develop your application Code for the chosen Compiler (the Raisonance 8051 C Compiler is used here)
- Verify the project needs and match with the device used
- Compile and create the firmware and applications Code
- Enter data from the Block diagram and memory maps using CAPS Design flow
- Merge hex files(s) generated by RIDE's linker and the PLD programming info (hex format) to create a combined file with the name .OBJ.
- Upload code and data to the development board using one of the supported tool paths (e.g., RLINK-ST / Flashlink).)
- Debug, make changes, reprogram and finalize the project
- Test and qualify the design

This application note provides guidelines for design by showing the key steps as mentioned above. The document has been divided into sections that cover various areas. It is expected that the reader has previous experience of programming and applications development including the use of compilers.

In previous sections, you were introduced to the Turbo+ uPSD. The family basic block diagram and features were introduced, followed by an overview of the DK3400 Development Kit (Board). The sections that follow cover the installation of the kit and project creation. A

simple example is used for demonstration and explained in detail to provide an understanding of the RIDE tools and the DK3400 Development Kit. It is hoped that with this information and other supporting documents available from STMicroelectronics, you can design and develop your application/project using Turbo+ uPSD.

3.2 Requirements

In order to follow the examples and processes described here you will need:

- A Windows host system with USB support (Win98SE, Win2000, ME, XP);
- A DK3400 Development Kit.

The DK3400 Development Kit, as described in Section 2.2, includes all hardware and software needed for the examples covered here.

Note: The examples here assume that CAPS has been installed and is used to create the hardware configuration files. A current version of CAPS is included in the development kit, along with a "quickstart" sheet for its installation and use. For more information on how to use all the features of RIDE, see Ride.pdf (available in the RIDE installation directory Ride\Doc or by selecting Help | PDF | Ride General | Ride in the RIDE program menu).

3.3 Software installation and connections

3.3.1 Software installation

- Insert the DK3400 CD in the drive.
- The auto-run brings up the home page or the main menu page. Select **Install ST and 3rd Party Tools**.
- First install CAPS, taking all the default choices.
- Next install RIDE, taking all the default choices.
- Go back to the home page.
- Select **Copy Device Drivers and Demo Code**.
- Unzip the files to the folder of choice on the hard drive.

3.3.2 Physical connections

There are several ways to establish a communications path between the host PC and the uPSD device for uploading and debugging applications. All of them ultimately make use of the JTAG port interface on the uPSD3400 package, but they use different means to access that port.

One method supported by the Raisonance tools employs a small bridge device, referred to elsewhere as the "RLINK dongle". On one side of the bridge device is a USB device socket, for connecting to the host PC by a standard USB cable. On the other side is a short ribbon cable that plugs in to a JTAG socket on the development board. This method is appropriate when using the RIDE tools to upload application code and data to production boards with minimal provision for external interfacing.

For the DK3400 development board, use of the RLINK dongle is possible but not required. The board includes a USB device socket and support circuits that emulate the function of the RLINK dongle. That enables the DK3400 board to be linked directly to the host PC via a

standard USB cable. That is the easiest method of connection, and is assumed in the remainder of this application note. The next steps for brining up the development environment are then as follows:

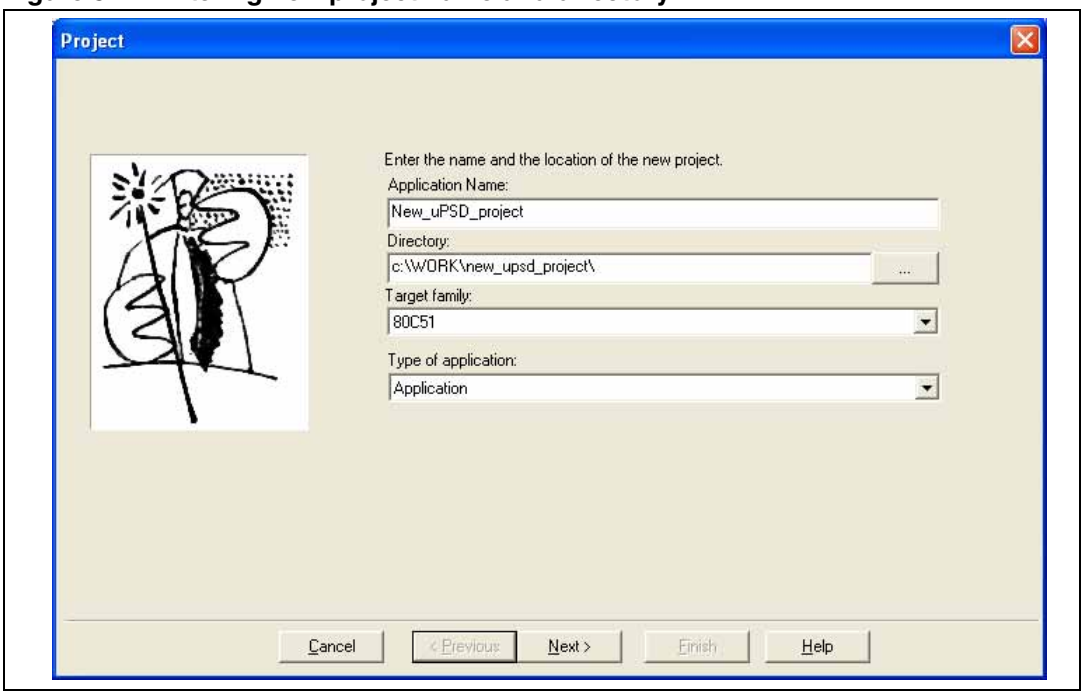
- Connect DK3400 to your PC/Laptop using the supplied USB cable and let the USB driver install on Windows.
- Make sure that the board is powered up using the Universal Adapter supplied with the kit. The LCD displays various text messages to indicate the board is functioning.
- Make sure that the Jumpers are set correctly. (Refer to *Appendix A: DK3400 jumpers selection and defaults* at the end of the document for Jumper settings).

4 Using RIDE and RLINK-ST for creating a new project

In this section the key steps for using RIDE to create a new project are shown.

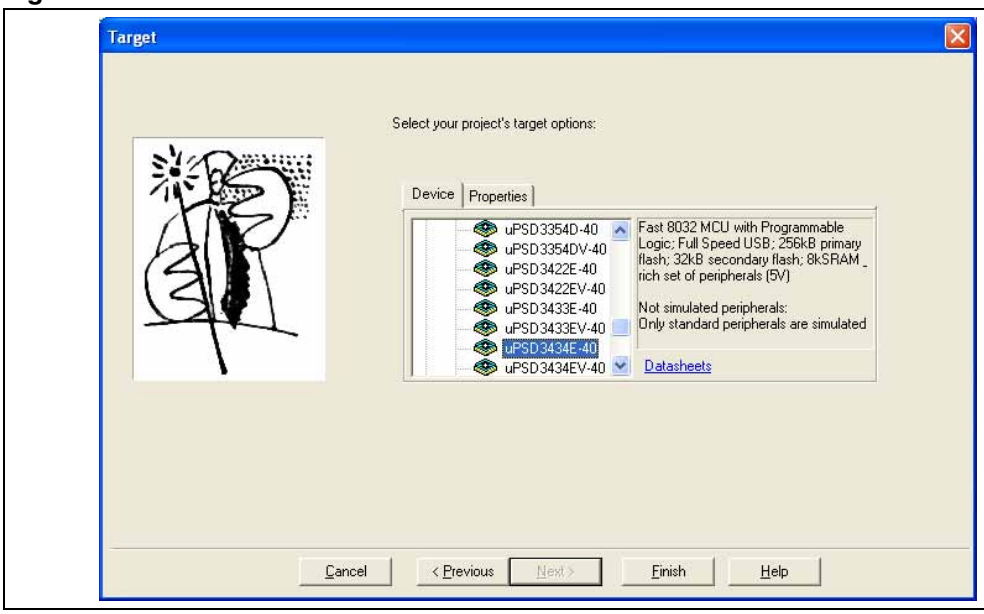
1. As shown in the section *Section 3.1*, there are a number of recommended steps involved in creating a new uPSD project using the RIDE 8051 software development tool from Raisonance. Please also refer to the general Users Guide for RIDE (included on the RIDE CD).
2. With RIDE already running, select **New** from the **Project** pulldown menu and enter the project name, path, and family as shown below (*Figure 5*) In this example, the name "New_uPSD_project" is used. The directory path is C:\WORK\new_upsd_project. The directory need not be an existing one; when necessary, RIDE creates the required directories if they don't already exist.

Figure 5. Entering new project name and directory



3. Click **Next** and the uPSD device selection dialog appears (*Figure 6*). Select the correct uPSD device from within the ST folder. (Here, uPSD3434E-40 is selected as these are used in the ST Development board's DK3400)

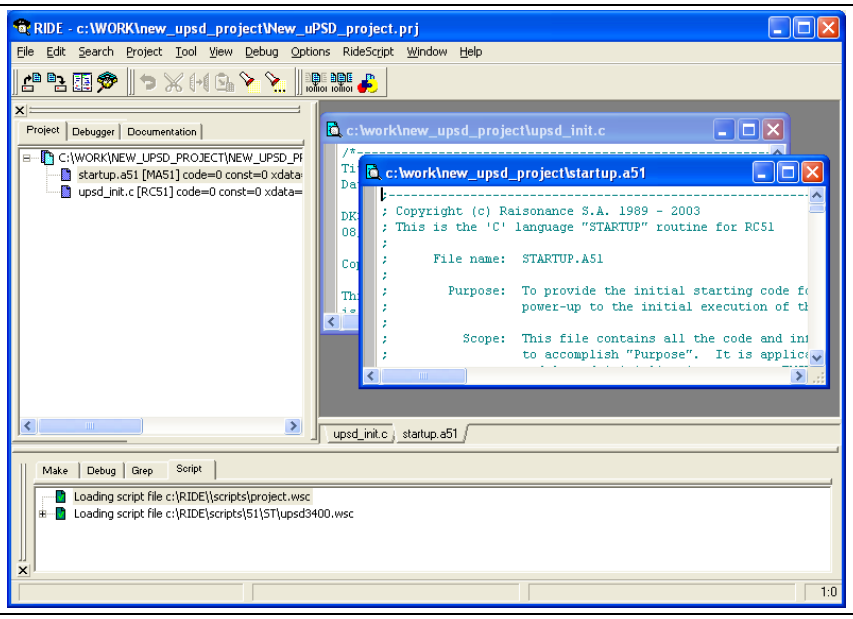
Figure 6. Turbo+ uPSD device selection



4. Click **Finish** and RIDE creates the new project copying the needed uPSD files into your project directory. RIDE also automatically adds the "*startup.a51*" and "*uPSD_Init.c*" files to your RIDE project folder. These files comprise the firmware that is executed by the uPSD3400 MCU upon a power-up or a reset event. It also creates a CAPS directory and copy a default project file into that directory.

   This is reflected in the RIDE environment as shown in *Figure 7*. The created files and start up files are shown in the Project Window.

Figure 7. RIDE environment



RIDE generates a default CAPS project by creating a folder and the CAPS project file. The project file is named *project.upj* and resides in the folder named "CAPS". This new folder is placed in your RIDE project folder. In this example, the path to the generated CAPS project is C:\Work\new_upsd_project\CAPS. Certain settings in RIDE depend on this structure so do not change the name of the generated CAPS project or its path. This project is examined later in this document.

5. At this point, we are ready to start building the source code for the application. If we were building a new application from scratch, we could use RIDE's **File | New** command to open a new edit window, and enter our code there. Or, we could use our favorite code editing program to create the files outside of RIDE, and move them into the RIDE project directory. It doesn't really matter how the files are created. In this case, however, we are just going to copy three existing files from the RIDE installation directory to our new project directory. The source path for the copies is:
   C:\RIDE\EXAMPLES\8051\DERIVATIVES\ST_UPSD\DK3400\UPSD3400\LED_BLINK
   The three files are *led_blink.c*, *upsd3400_timer.c*, and *upsd3400_timer.h*. The first C file contains our main program, and the second contains a support function that the main program uses. Copy the three files into the folder C:\work\new_upsd_project.
6. Next we need to inform RIDE that our project depends on the files that we just copied into our project folder. Right click on the root target in the project window (i.e., on C:\WORK\NEW_UPSD_PROJECT\NEW_UPSD_PROJECT.AOF) and select the **Add node Source/Application** command (*Figure 8*). This brings up a standard file browser sub-window in the project directory (*Figure 9*). Select *upsd3400_timer.c* and click **Open**. Repeat for *led_blink.c*. The result is as shown in *Figure 10*.

Figure 8. Adding source files

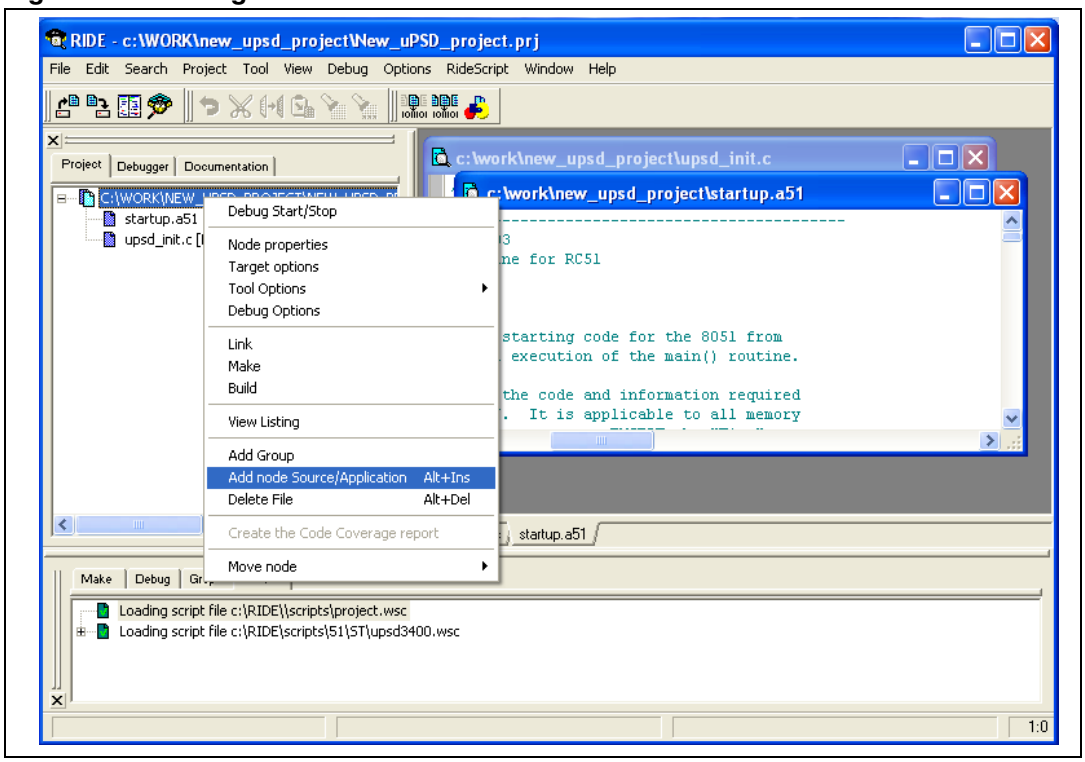


Figure 9. Add File browser sub-window

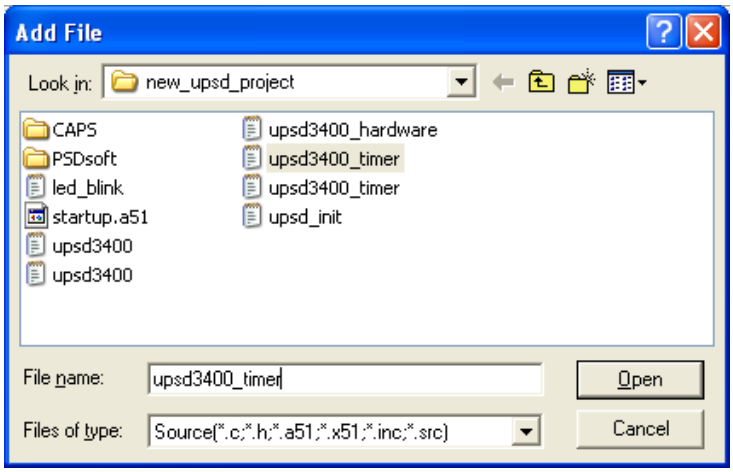
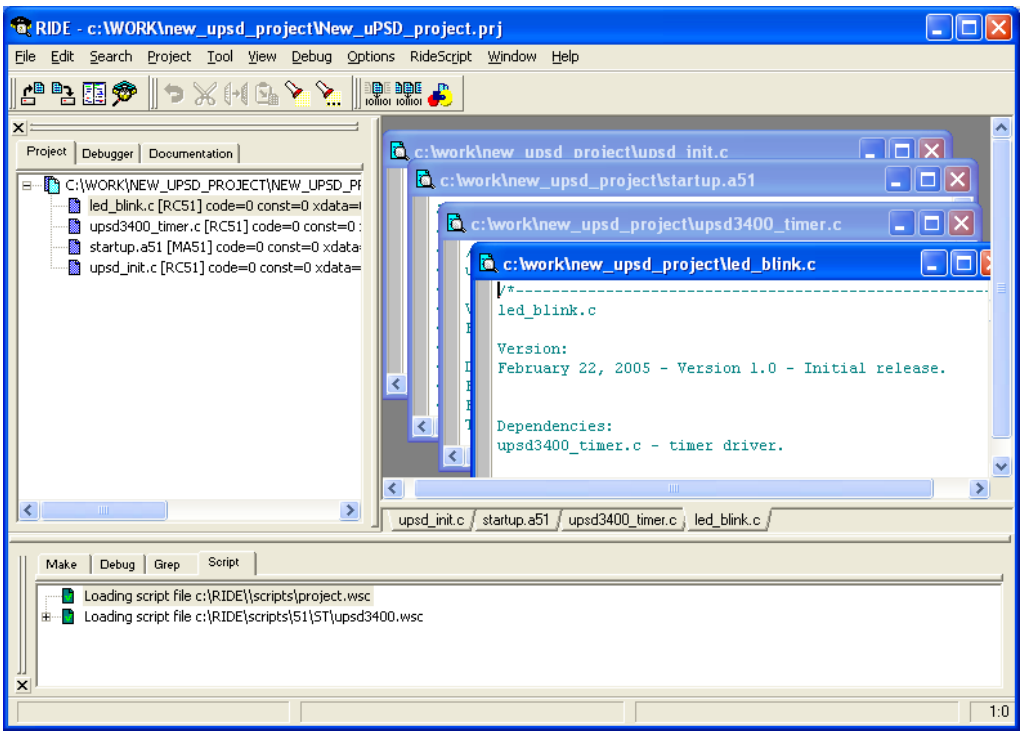


Figure 10. RIDE environment after adding source files



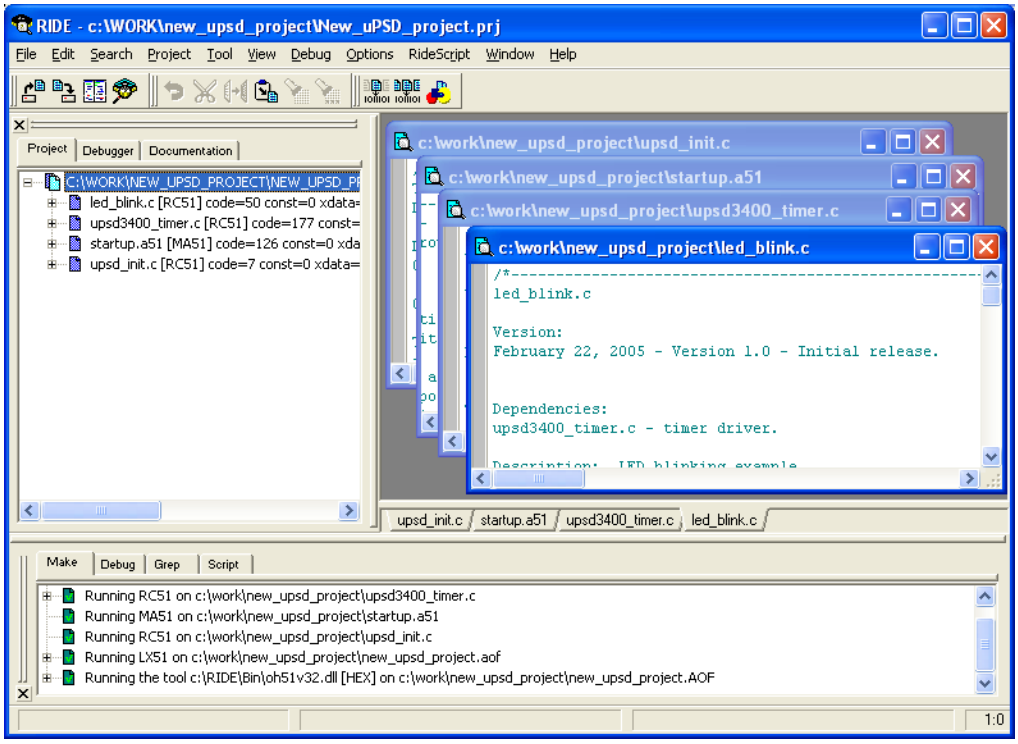
Note that it is not necessary to add the *.h* file *upsd3400.h*. The tree displayed in RIDE's Project window is actually a tree of make targets and dependencies. When it does a program build after any new source files have been added, RIDE first runs the compiler in a mode that identifies any file dependencies from #include statements. The included files are then added automatically to the project dependency tree. If an included file is subsequently changed, RIDE then knows that any dependent source file must be recompiled.

It is also worth noting that RIDE is not actually fussy about where the source files reside. The default location is in the project directory, and it was convenient to use that for this example. However, the browser sub-window opened by **Add New Source/**

Application command supports navigation to other locations. The full path name is added to RIDE's project database.

7. With all source files added, we are now ready to do the initial project build. In the Project pull-down menu, select **Build All**. RIDE runs its make utility, reporting status in the make sub-window at the bottom of its main window (*Figure 11*). Note that the *.c* files in the Project window are now prefixed with '+' boxes. Clicking on one of these expands the dependency tree to show the *.h* and *.inc* files that were found during the build.

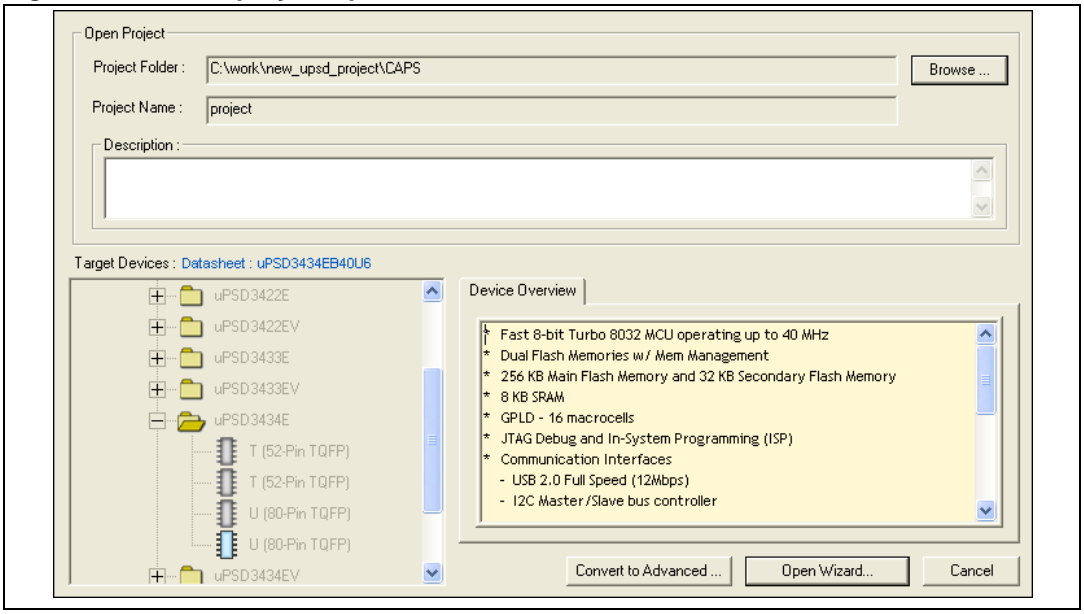
Figure 11. Result after Build All



Although the application source files have now all been compiled, and initial versions of the object and Intel Hex files have been built, the project is not yet ready to upload and test on the DK3400 board. The hardware configuration files are incomplete. These are the low-level files that program the PLD chip selects for the Flash memory sectors, SRAM, and CSIOP registers, the programmable logic, and also specify what hex files are programmed into each of the Flash sectors. The CAPS tool is used to create these files and merge them into the object file for uploading. So the next step is to run CAPS.

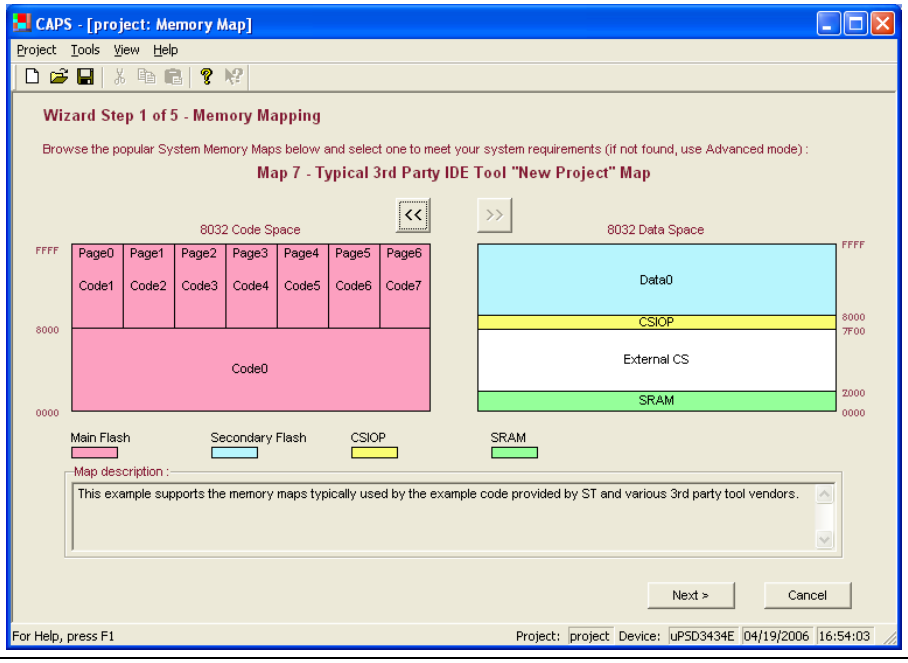
8. Bring up CAPS, click the **Open** icon on the menu bar, and browse to the CAPS project file that RIDE created in step 4. Select that file in the browser sub-window and click open. The result is shown in *Figure 12* below.

Figure 12. CAPS project opened



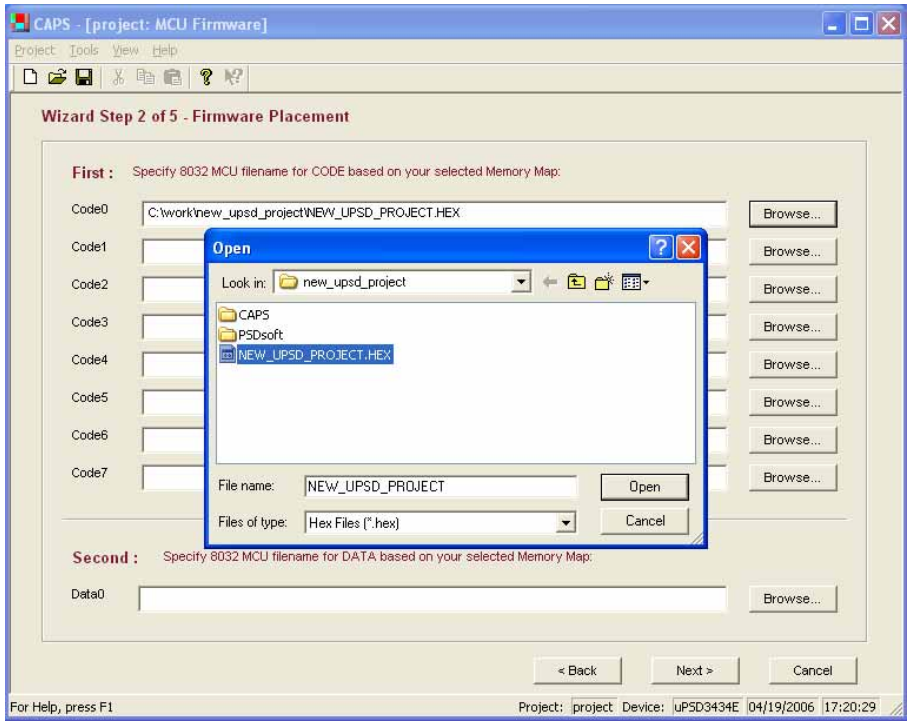
9. Click on the **Open Wizard** button in the lower right portion of the above window to bring up the CAPS project wizard. The resulting window is shown below. It is for the memory mapping step of the configuration process. CAPS is preprogrammed with seven standard memory maps. Map 7, the default shown here, is suitable for the example used in this note. Click **Next** to accept it.

Figure 13. CAPS Wizard memory mapping window



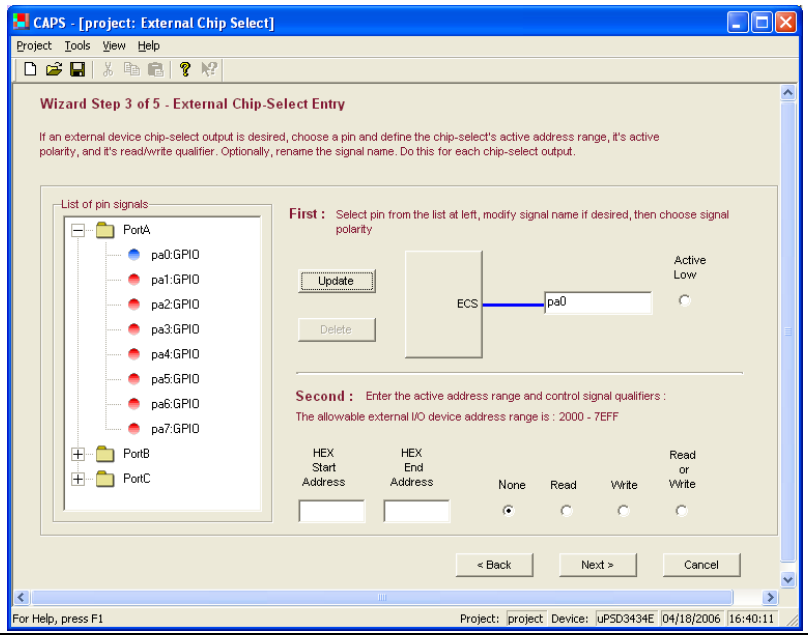
10. The next window for the CAPS configuration wizard is the Firmware Placement window shown in *Figure 14*. Its purpose is to assign names for the files that are to be used to program the code and data pages identified in the memory map. For the Code0 page, the only one that is used in this example, click the **Browse** button and go one directory up to find the Intel Hex file that was created in step 7. CAPS requires a valid .HEX file for merging its firmware files.

Figure 14. Firmware Placement step



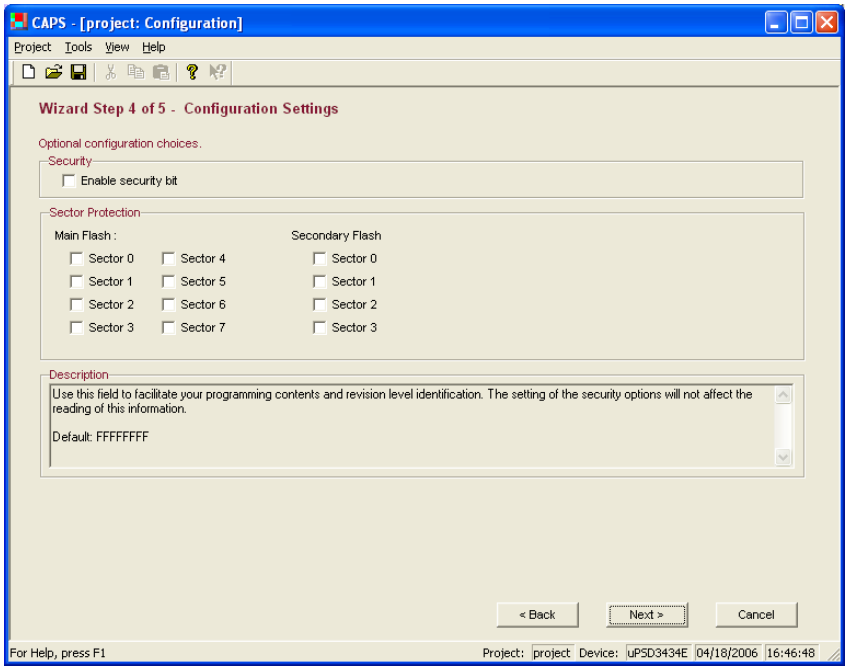
11. Click **Next**. This brings up the window shown in *Figure 15*, for setup of any external chip select outputs required. For this example, the default presented is all that's needed, so just click **Next** to continue.

Figure 15. External chip select setup



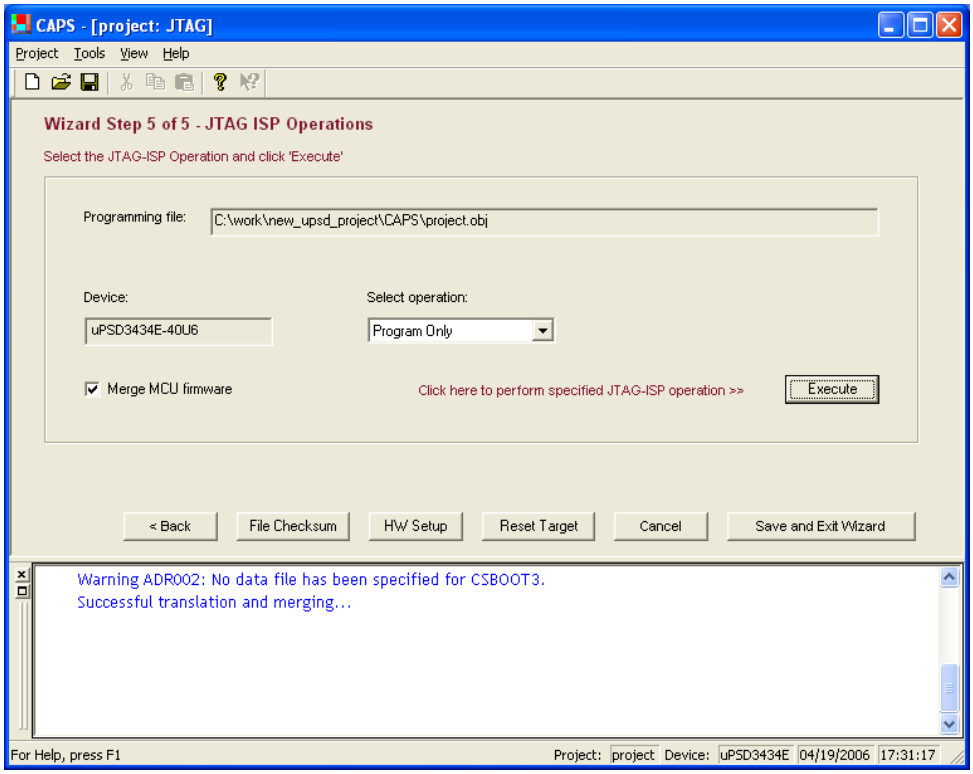
12. The next step is for configuring the security bit, and the sector protection bits. Selecting any of the sectors for protection prevents the selected sector from being altered, until the configuration is changed to remove that protection. During development, there is usually no reason to select any segment for protection. However, protection for individual segments can be removed by rerunning CAPS to reconfigure the chip. The Enable security bit, once programmed, prevents reading/writing of the Flash sectors via JTAG. The security feature is disabled when a full erase of the device is performed. Click **Next** to continue.

Figure 16. Security and Protection settings



13. The final step of the CAPS setup Wizard creates the MCU firmware image according to the configuration options selected and merges it into the object file. The result is reported in the window shown in *Figure 17*. (The warning message about no data file for the FS0:7 and the CSBOOT1:3 sectors can be ignored as this example does use one.) The window gives options to program the device through the JTAG port at this point, or to **Save and Exit Wizard**. Since we intend to debug the project in RIDE, which has its own upload capability, we pick the latter and close CAPS.

Figure 17. JTAG ISP operations step



This completes the project creation and setup phase. The next chapter describes how the RIDE debugger is used to upload applications to the DK3400 board and to run tests.

5 Uploading and debugging with RIDE

5.1 Purpose

The preceding chapter took you through the steps of using RIDE and CAPS to create a new project, *New_uPSD_project*, and to build a simple application with the main program led_blink. This chapter shows how RIDE, in conjunction with RLINK-ST, can be used to upload, test, and modify application code running on the DK3400 board.

This simple demonstration project illustrates the powerful software development tools based upon Raisonance RIDE software, and the RLINK-ST capabilities, which provides many features for editing, compiling, programming, and debugging a µPSD33/3400 MCU Series from STMicroelectronics. This demo quickly illustrates the specific features below to give you a feel for their simplicity and capability:

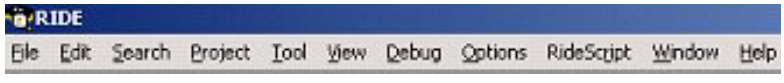
- Compile Project and Program Flash Memory
- Single-Step Execution and Source-Level Debugging
- Device-Specific Formatted Displays
- Breakpoints
- Symbolic Debugging and Variables Watch
- Code Iteration
- Instruction Tracing approaching Real-Time performance

The DK3400 Development Board with its embedded RLINK tool (or your own designed circuit board with a µPSD34xx MCU) is all that is needed to develop code. RIDE's debugger utility can be used to symbolically debug 8051 code generated by almost any 8051 compiler. You may choose to use your existing 8051 compiler with the RIDE debugger (no code size limit) or upgrade the evaluation version of the RIDE compiler to also compile with no code size limit. See http://www.raisonance.com for more information on RIDE and upgrades.

5.2 Upload project and program Flash memory

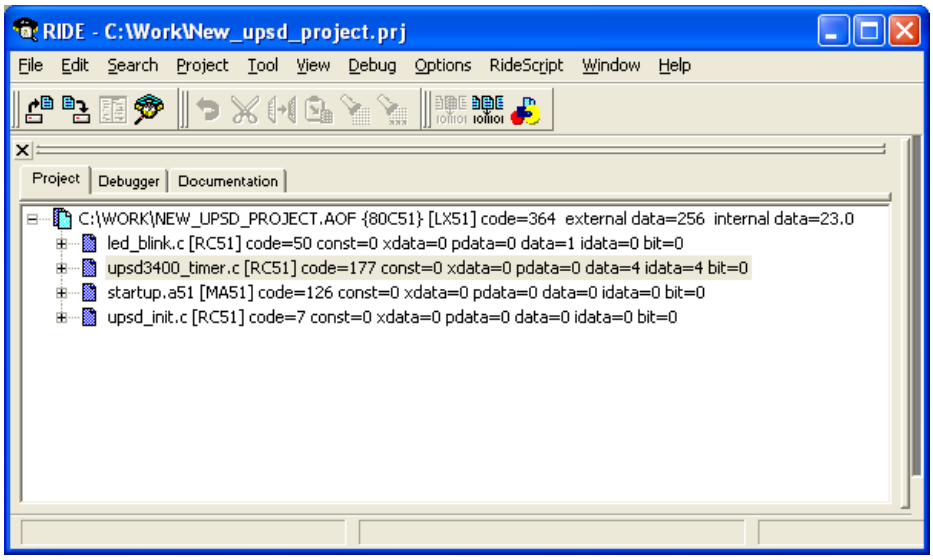
If it is not already running, launch RIDE from the Windows programs menu (Raisonance Kit 6.1) or by clicking the RIDE icon on the desktop. On opening, a blank work area appears with the RIDE title menu bar as shown in *Figure 18.*

Figure 18. RIDE Title Bar



- Open the demo project created in *Section 4*. In title menu bar click **Project**, then **Open**. Next double-click the project named *New_uPSD_project.prj*, from the folder C:\work\new_upsd_project. If you skipped *Section 4*, the same project can be found in the RIDE installation directory. The project name there is led_blink, rather than new_upsd_project. It is found in the folder:
  \RIDE\EXAMPLES\8051\DERIVATIVES\ST_UPSD\UPSD3400\DK3400\LED_BLINK
- The RIDE environment displays new content in the project windows. The left window shows the project files. Click on the "+" to expand the project component files and then double-click on *led_blink.c* to open the file (Refer to *Figure 19*).

Figure 19. Project window



- Click **Make All**. If you are starting from the 'led_blink' project under the EXAMPLES folder of the RIDE installation directory, this compiles and builds the project. If you are continuing with the 'New_uPSD_project' from *Section 4*, the project has already been compiled and built. In this case, clicking **Make All** effectively does nothing and is harmless.
- Click **Options | Debug** and the "Debug Options" window will appear (*Figure 32*). In this example, use RLINK-ST to debug your code under JTAG control. Select **Real Machine (Emulator or ROM-Monitor)** as the "Tool" and **RLINK-ST -uPSD** in the drop down box under "Tools." Click on the **Advanced Options** button and the "uPSD debugger options" window appears as shown in *Figure 33*. Make sure the settings are as shown in the figure. Of particular importance is the setting for "Merge Options." This setting specifies the location and name of the CAPS project file associated with the RIDE project. The path and filename must be correct for proper programming of the device. In this example a relative path is used to point to the CAPS file. From this window, various JTAG operations may be performed using the buttons near the bottom of the window in the "Instant Checks" area. Once the settings have been confirmed, click **OK** to close this window, then **OK** again to close the "Debug Options" window.
- Start the Debugger by clicking **Start**. This programs the Flash and refreshes the RIDE environment showing actions in the "Debug / Action/Status Window" (*Figure 25*).

The highlighted line indicates where MCU execution has stopped at the first line of executable code in the main program. The Debugger now waits for your command.

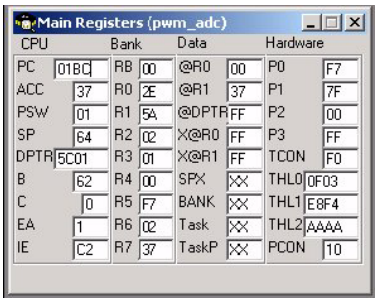
5.3 Single-step and source-level debugging

- Click **Go** to see the program run full speed with the D6 red LED blinking at a couple Hz frequency.
- Click **Reset** and the program returns to the first line of the main program. The blue line should be on blink_delay = SHORT.
- Click **Step-In** twice. The debugger is now in the called function, `timer0_init()`.
- Double click **Disassembly Code** in the left debugger window. This opens a tabbed window, “code (led_blink)” showing both C and Assembly code source instructions.
- Click **Step-Over** a few times to see that code execution can be stepped one assembly instruction at a time.
- Click **Reset** to return to the main program, *led_blink.c*.
- Click on **LED_BLINK.C** tab to return to the C code window.

5.4 Device-specific formatted displays

- Double-click **Main Registers** in the left debugger window to show the contents of the MCU core registers (Refer to *Figure 20*).
- Double-click **Port 1** in the left debugger window to show current value of pins on I/O port 1.
- Go back to file *led_blink.c* by clicking on the tab at the bottom of the main display window (Refer to *Figure 25*), and expand the window view back to full screen.

Figure 20. MCU registers



5.5 Breakpoints

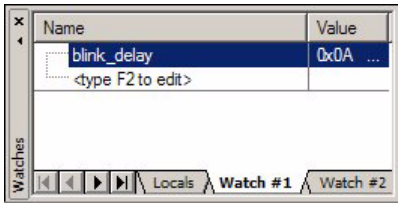
Four hardware breakpoints are available on uPSD3400.

- Set two breakpoints by clicking on each of the green dots on the left of the two lines of code `timer0_delay(blink_delay)`, in the `while(1)` loop. The green dot and the selected line both highlight in red.
- Click **Go**, the program runs until reaching a breakpoint. Notice the status of the red LED.
- Click **Go** repeatedly. Notice that the red LED toggles from ON to OFF by repeatedly clicking the **Go** icon.
- Now, remove the two breakpoints, by clicking in the margin on the two red icons. Click **Go** again to resume program execution. The blinking routine is now running without interruption.

5.6 Symbolic debugging and variables watch

- Click **Stop** to halt program execution. With the mouse, highlight (double-click) the entire variable name “blink_delay” at the beginning of the `main()` function. Then, right-click on the variable and select “Add Watch” to add this variable to the Watch Window, which appears at the bottom left of the screen. You may also press the **F6** key (refer to *Figure 25* and *Figure 21*).

Figure 21. Watch Window



- Remove any existing breakpoints.
- Click **Reset**.
- Click **Go** followed by **Stop** to see that the blink_delay value is updated in the watch window and that it reports a value of 0x0A.
- While the code is running (Click **Go** again), place the mouse above the Value field (which should read 0x0A), right-click, and select **Evaluate**. Then, enter 0x64 in New Value field, followed by clicking on **Modify**. The LED should then blink at a rate 10 times slower. (0x64 causes a 10x longer delay versus 0x0A). Notice that the debugger is active while the code is running.
- Click **Reset** followed by clicking **Go**. The LED blinks again at the faster rate.

5.7 Code iteration

- Halt the debugging session to make the blink delay interval code change permanent in Flash memory.
- Close the Debugger by clicking on the same icon that "starts" the Debugger.
- Now you are in the editor. Go to file *led_blink.c* by clicking on its file tab and change the C code statement from,

```
blink_delay = SHORT;
```

  to

```
blink_delay = LONG;
```

- Click **Make All** to recompile and rebuild the program.
- Start the Debugger by clicking **Start** to re-program this new code into Flash memory.
- Click **Go** and see that the LED is now permanently blinking at the slower rate. This code modification now resides in Flash memory.
- Click **Reset**.

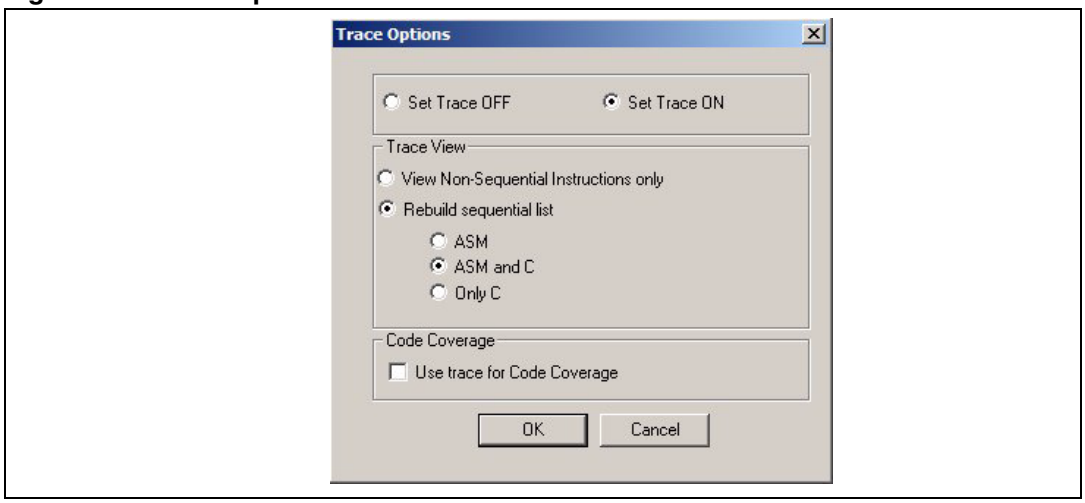
5.8 Instruction tracing, near real-time performance

The uPSD rapidly streams a record of all the MCU instruction steps out to the RLINK-ST adaptor. From this data, RIDE creates a formatted file to help you find even the most stubborn bugs, showing an MCU execution history depth of 500,000+ instruction steps.

Note: When trace mode is enabled, CPU performance can be expected to decrease about 20 to 30 percent.

- To enable Trace, select from the title bar **Debug** then **Trace**, and select trace **Options** as shown in *Figure 22*. Then click **OK**.

Figure 22. Trace Options



- Open the Trace Display. Select from the title bar **Debug**, then **Trace**, then **View**. A blank Trace window displays.

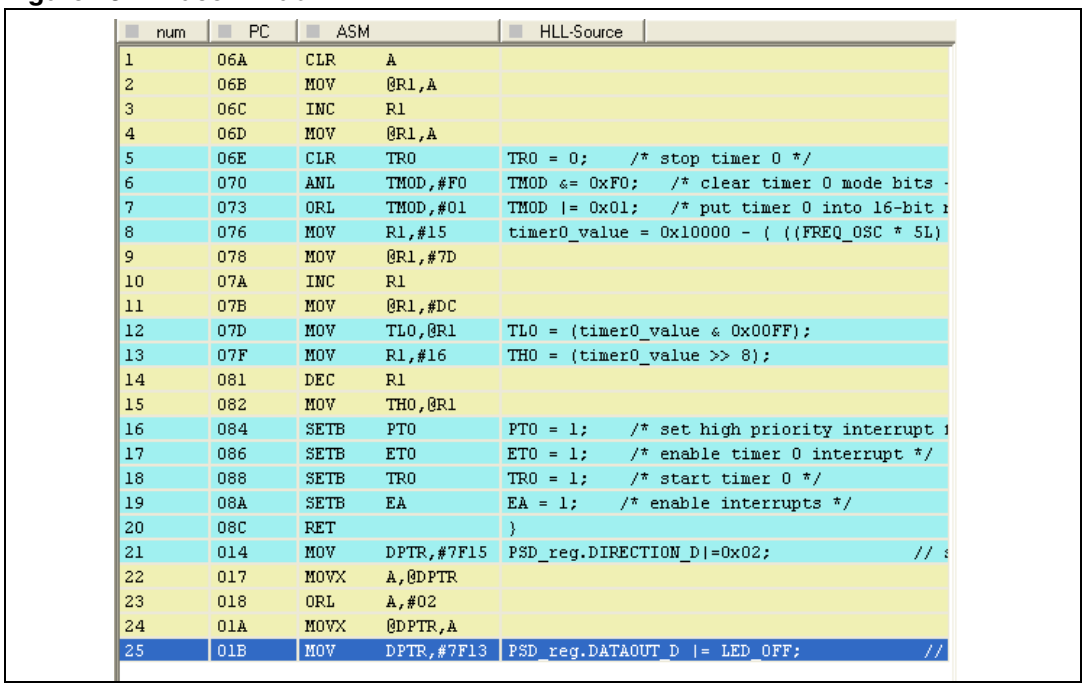
A Trace Display file can display program source code in both C and Assembly formats. Tracing runs in the background with little impact to real-time performance in this project.

- Return to the file *led_blink.c* by clicking on its file tab.
- Set one breakpoint at the line of code immediately before `while(1)`, at `PSD_reg.DATAOUT_D |= LED_OFF`, by clicking on the green dot to the left of the line of code. The breakpoint line highlights in red.
- Click **Go**, and the MCU runs until hitting the breakpoint, then a window opens showing the Assembly source code.

Note that the red line indicates where the breakpoint is set, the blue line indicates the next instruction to execute, and a pink line indicates where a breakpoint occurred.

- Now, open the Trace Display window by clicking on the file tab “Trace (led_blink)” (*Figure 23*). At the bottom of the Trace display is the last instruction that was executed: MCU Program Counter at 01B. Above this line is the history of all instructions executed before hitting the breakpoint. There should be 13 records.

Figure 23. Trace window



num	PC	ASM		HLL-Source
1	06A	CLR	A	
2	06B	MOV	@R1,A	
3	06C	INC	R1	
4	06D	MOV	@R1,A	
5	06E	CLR	TR0	TR0 = 0; /* stop timer 0 */
6	070	ANL	TMOD,#F0	TMOD &= 0xF0; /* clear timer 0 mode bits
7	073	ORL	TMOD,#01	TMOD |= 0x01; /* put timer 0 into 16-bit
8	076	MOV	R1,#15	timer0_value = 0x10000 - (((FREQ_OSC * 5L)
9	078	MOV	@R1,#7D	
10	07A	INC	R1	
11	07B	MOV	@R1,#DC	
12	07D	MOV	TL0,@R1	TL0 = (timer0_value & 0x00FF);
13	07F	MOV	R1,#16	TH0 = (timer0_value >> 8);
14	081	DEC	R1	
15	082	MOV	TH0,@R1	
16	084	SETB	PT0	PT0 = 1; /* set high priority interrupt
17	086	SETB	ET0	ET0 = 1; /* enable timer 0 interrupt */
18	088	SETB	TR0	TR0 = 1; /* start timer 0 */
19	08A	SETB	EA	EA = 1; /* enable interrupts */
20	08C	RET		}
21	014	MOV	DPTR,#7F15	PSD_reg.DIRECTION_D|=0x02; //
22	017	MOVX	A,@DPTR	
23	018	ORL	A,#02	
24	01A	MOVX	@DPTR,A	
25	01B	MOV	DPTR,#7F13	PSD_reg.DATAOUT_D |= LED_OFF; //

- If you right-click on the trace windows and select **Options**, it is possible to list both C and ASM code by selecting **ASM and C sequential list**.
- Return to file led_blink.c and click **Go**. Notice that the LED blinks normally and in real-time.
- After about 10 seconds, click **Stop**. Notice the messages in the window that records the actions (*Figure 24*). This window shows the number of non-sequential instructions traced. Thousands of instructions are now showing in the Trace window.

Figure 24. Message window

6 Conclusion

Congratulations! You have seen the majority of steps to implement a Turbo+ uPSD design on the DK3400 board. This design guide showed the basic steps to pre-configure the memories with CAPS, compile, program in Flash and debug with RIDE Tools. The process flow diagram steps were described so that the method for creating a new project from scratch was shown and a detailed design and process, based upon the blink LED demo, has also been described in detail with all the tools required.

There is additional documentation about the uPSD Turbo+ architecture on the DK3400 CD ROM. There is also further documentation available through the website links provided earlier. The trial version of the RIDE C compiler and tools supplied with the DK3400 limit the Code size to 4KB. Any application larger than 4KB would require purchase of the full tools from Raisonance.

The example code and the steps clearly demonstrate the powerful firmware development and debugging capabilities of the RIDE environment with RLINK-ST for uPSD DK3400-Development Board.

For more information, please refer to:

- *Datasheet* of the uPSD34xx MCU
- *Getting Started with RIDE and µPSD* (Application Note AN48-uPSD)
- Schematic for the DK3400 circuit board in the *User Manual UM0131*

Please see the ST web site for these documents and for the latest information on uPSD products, tools, application notes, and other documentation: *http://www.st.com/mcu*

Appendix A DK3400 jumpers selection and defaults

The following Table describes the DK3400 Jumpers. See the Schematic and DK3400 User manual for more information regarding the jumpers.

Table 1. DK3400 jumpers

Jumper No.	Description	Default Setting	Comments
JP1	Enable SPI interface Flash M25P80.	Closed	M25P80 is enabled when JP1 is closed.
JP2	Reserved	Open	Please keep this jumper on open.
JP3	Enable USB auto-disconnect function.	Closed	USB auto-disconnect function is enabled when JP3 is closed.
JP4	select a power source for JTAG port.	JP4.1 connected to JP4.2	Keep JP4 on following status when ED3K4 works on Mode1, 2, 4 and 5: JP4.1 connected to JP4.2. Keep JP4 on open when ED3K4 works on Mode 3.
JP5	Select which power source to be used as USB power input of power management circuit, power from E-RLINK USB cable or power from uPSD USB cable.	JP5.1 connected to JP5.2	Keep JP5 on following status when ED3K4 powered from RLINK USB cable: JP5.2 connected to JP5.3. Keep JP5 on following status when ED3K4 powered from uPSD USB cable: JP5.1 connected to JP5.2.
JP6	Provide a boot option for ED3K4 board.	Open	ED3K4 boot from internal main flash when JP6 is closed. ED3K4 boot from internal boot flash when JP6 is open.
JP7	Select clock generation source, external clock or internal clock.	JP7.1 connected to JP7.2	ED3K4 works with internal clock when JP7 is set as following: JP7.1 connected to JP7.2. ED3K4 works on external clock mode when JP4 is set as following: JP7.2 connected to JP7.3.
JP8	Enable NAND Flash.	Closed	128Mbit NAND flash is enabled when JP8 is closed.
JP9	Select JTAG circuit operation mode along with JP10 depending on operation mode of ED3K4.	Closed	Keeps JP9 on closed when ED3K4 works on mode 1, 2, 3 and 5. Keeps JP9 on open when ED3K4 works on mode 4.
JP10	Select JTAG circuit operation mode along with JP9 depending on operation mode of ED3K4.	Closed	Keeps JP10 on closed when ED3K4 works on mode 1, 2, 4 and 5. Keeps JP10 on open when ED3K4 works on mode 3.

Jumper No.	Description	Default Setting	Comments
JP11	Enable DEBUG signal.	Open	DEBUG signal is enabled when JP11 is closed.
JP12	Select which transceiver to be connected to UART1 port, RS232 transceiver or IrDA transceiver.	JP12.2 connected to JP12.4 and JP12.1 connected to JP12.3.	UART1 is connected to RS232 transceiver when JP12 is set as following: JP12.1 connected to JP12.3 and JP12.1 connected to JP12.4. UART1 is connected to IrDA transceiver when JP12 is set as following: JP12.3 connected to JP12.5 and JP12.4 connected to JP12.6.

Appendix B Interface display windows and code view

Figure 25. RIDE interface display windows

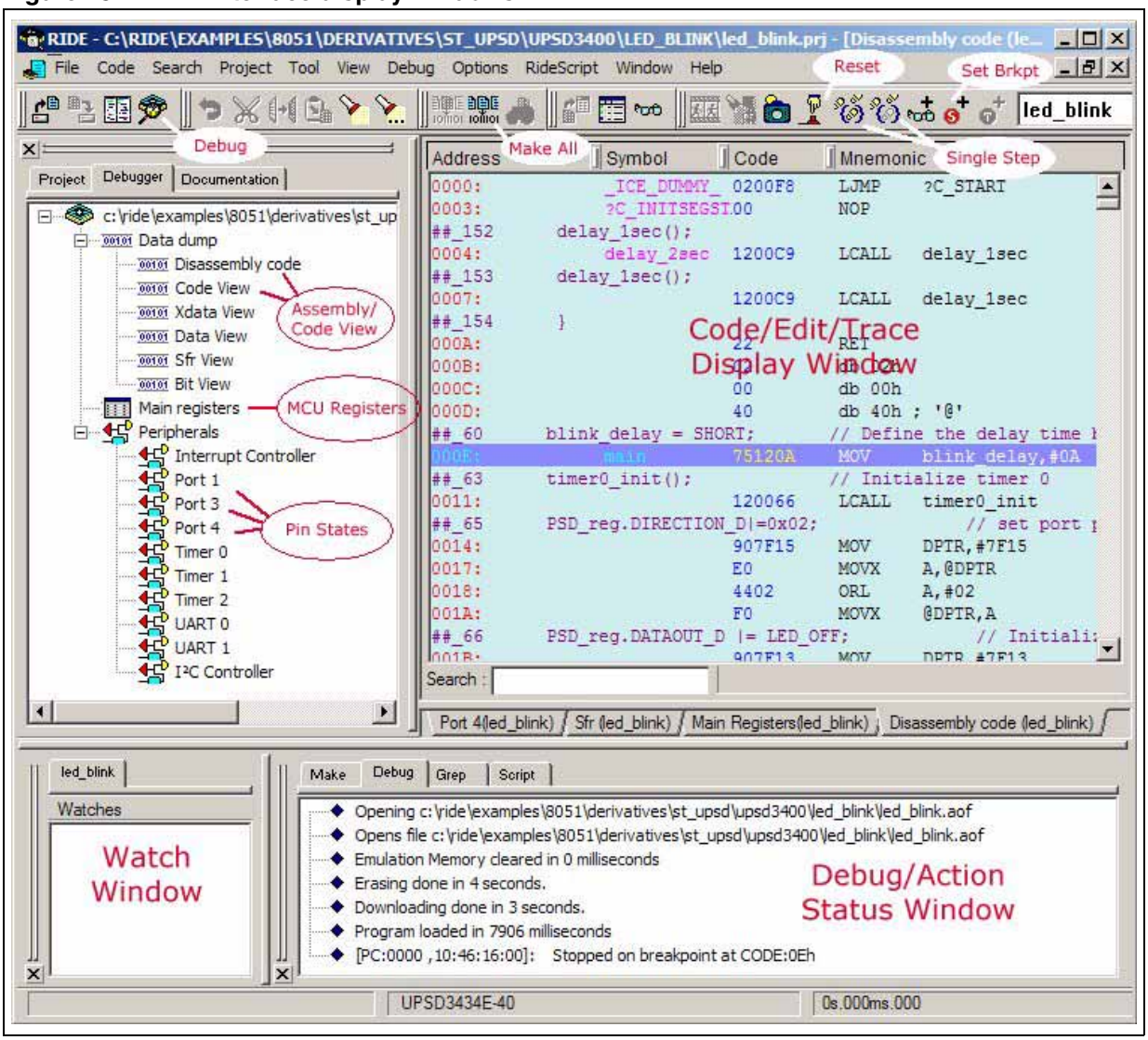
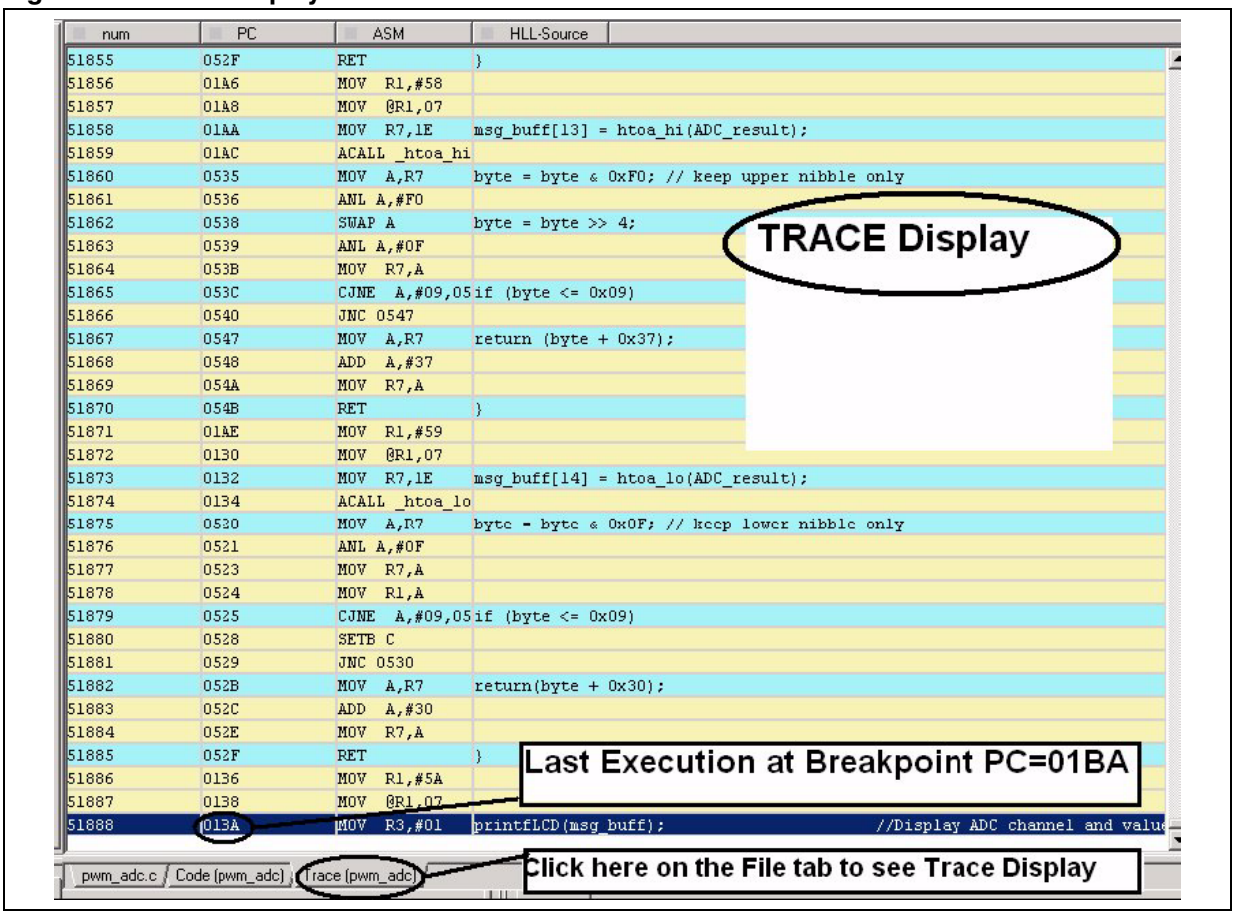


Figure 26. Code view (disassembly)

Figure 27. Trace display

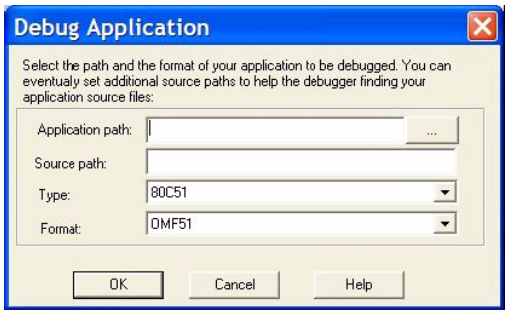


Appendix C Importing an external application into RIDE

C.1 Overview

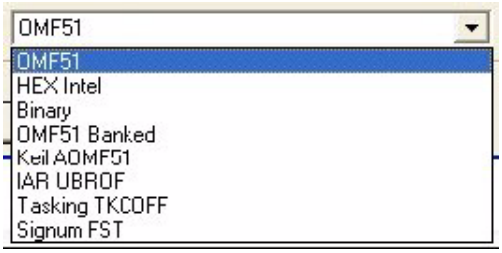
The RIDE IDE allows you to combine the building of the project and the debugging of the built application. However, you could wish to simply debug an application that has been written and compiled out of RIDE. In such a situation, choose **Debug** I **Load**. The following window appears:

Figure 28. Debug application window



You can then select your application. You also need to specify the format of this application. Take care that some of the listed formats does not contain any debug information (such as HEX or Binary):

Figure 29. Format drop-down list



Debugging an external application and a built-in-RIDE application will be then exactly the same.

If none of these formats matches with the format of your application, it is recommend that you check if the tools you are using allow conversion from the original format into one from this selection.

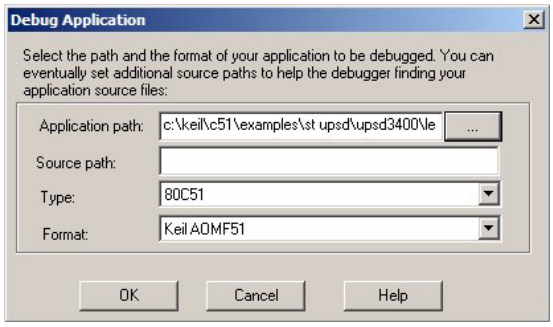
C.2 Importing a Keil project into RIDE for debugging

Here, the process is shown for importing the same application example as developed using Keil Compiler. This is available in the Keil folder of your installed Keil software tools. Browse through the application path and then select the correct file. (See note below). The Keil AOM5F51 format required by RIDE has no extension and in this case it is the following path and file name: right-click on the Format drop-down box and ensure that you select Keil AOMF51.

*Note: For Keil projects, the file name has no extension and is the same as the project name used for developing the application. uPSD projects for Keil have the extension *.uv2.*

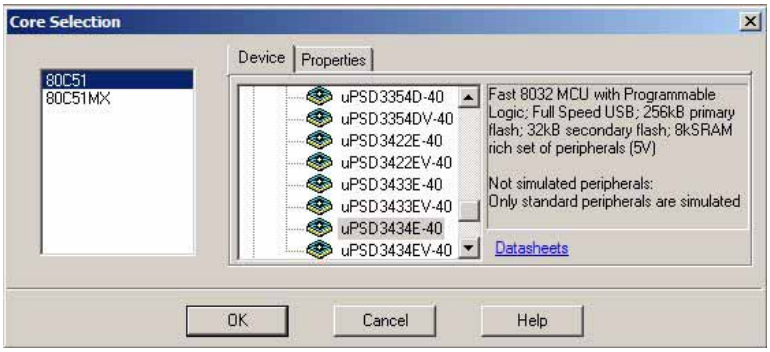
Below, the RIDE screens are shown for importing the same blink LED project from Keil and then loading in DK3400 and using RIDE tools to Debug. You may use this process to import large codes into the Eval and demonstration version of RIDE for Debugging only.

Figure 30. Keil application path and format



RIDE generates the next screen shown below. Select the correct device.

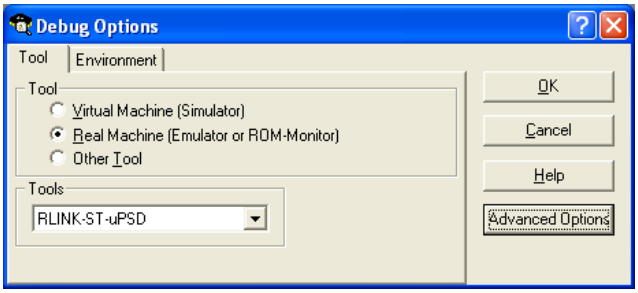
Figure 31. Core selection



C.3 Running the application on the target hardware

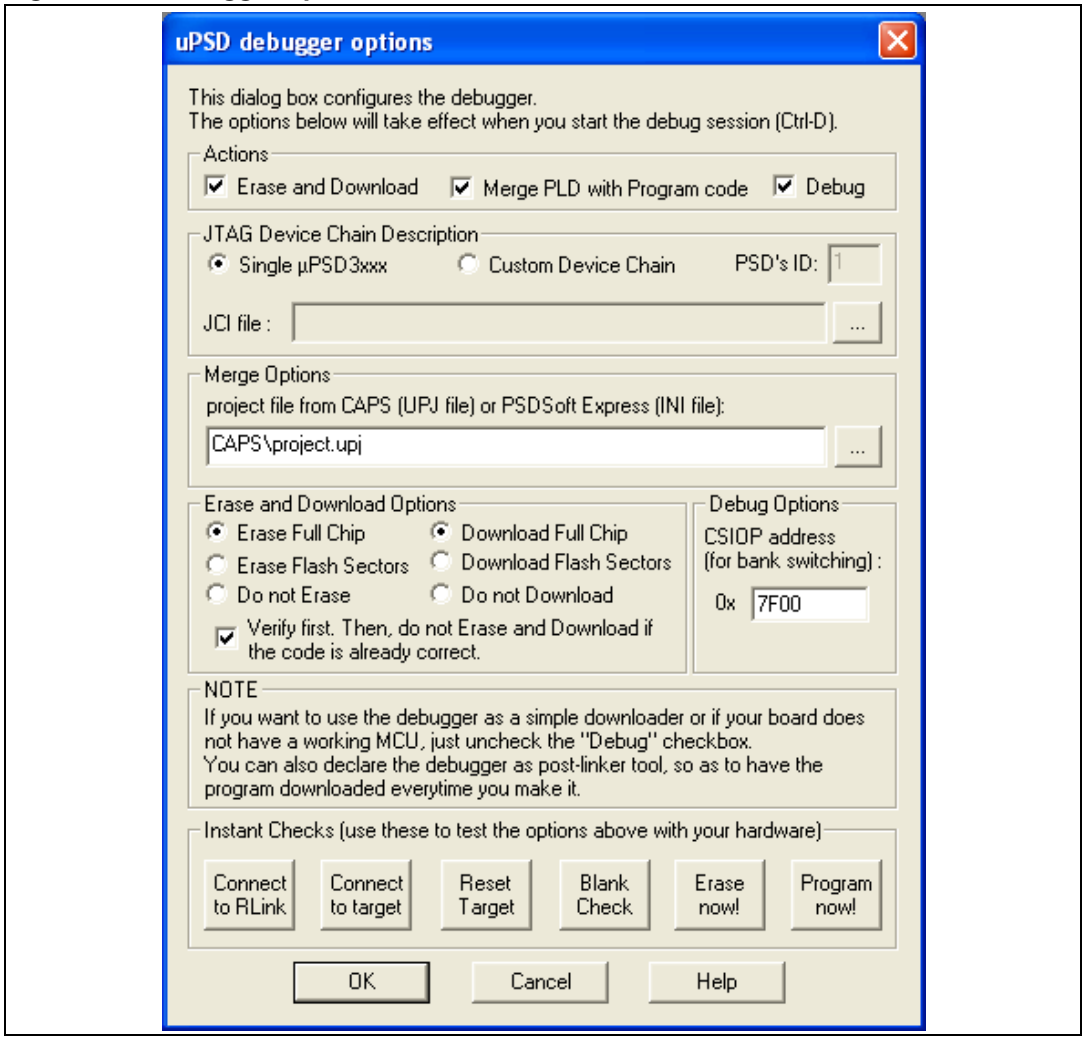
To load and to debug your application using the RLINK-ST dongle, you have first to configure the "Options | Debug" window as follows:

Figure 32. Debug options



Select the Real Machine option, Select in the "Tools" list, RLINK-ST-uPSD. If you then click on **Advanced Options**, the following window is presented:

Figure 33. Debugger options screen



This window allows you:

- To specify the CAPS project file (UPJ).
- To specify the JTAG chain description file (if any) when the uPSD part is included into a multiple-device JTAG chain.
- To execute simple commands such as Erase, Program and Blank-Check.
- To specify the CSIOP address.

C.4 Specifying the CAPS UPJ file information

In the above window, the file "*project.upj*" is a project you set up using CAPS, wherein all associated information pertaining to this project resides.

Merging is the action of creating an OSF file, using an UPJ file generated by CAPS. An OSF file is a file containing the code to be loaded in all the sectors of the part. This is the only format supported by the loader.

It is strongly recommended to always keep the "Merge" option checked, unless you plan to use the debugger as a simple downloader for programming a large number of boards, with a program that you have already tested and validated.

Note that for merging, you MUST have CAPS properly installed on your computer. Indeed, RIDE calls some ST utilities (present in the CAPS directory) to merge the PLD and the flash. These utilities are the following:

UMERGE.EXE

UOBJOSF.EXE

If you have issues, please check that these files exist into your CAPS folder. Make sure that you give the correct Keil folder path and get the CAPS files from the folder to ensure correct code loading.

C.5 Executing simple commands such as Erase, Program and Blank Check

It is recommended first to check that both the RLINK-ST dongle and the target board are properly connected and powered. The communication can be checked by clicking onto:

1. **Connect to Rlink**" to check that the USB dongle answers,
2. **Connect to target**" to check that the uPSD answers to the dongle.

Then, the first command available is **Erase**.

1. **Erase Full Chip** allows to erase both the PLD and the FLASH.
2. **Erase Flash Sectors** allows to erase only the FLASH, keeping intact the contents of the PLD.
3. **Do not Erase** makes sense only when the debug session is started and that the only selected options are executed at the loading time.

Once Erased (which is done by clicking on **Erase Now!)**, a blank check can be performed by clicking on the **Blank-Check** button. Then, programming can be done with the exact same options as **Erasing.**

Warning: **Note that the settings of "Erase" and "Program" are used when launching a debug session. You need, before clicking on "OK" to keep the settings required for debugging. In most cases, it is recommend to set either "Erase Flash Sectors" and "Program Flash Sectors" if you don't need to update the PLD (but keep the "Merge" option checked), or "Erase Full Chip" and "Program Full Chip" when you are still working on the design of the PLD.**

C.6 Specifying the CSIOP address

This information is mandatory when the application is larger than 64KB and uses the bank-switching technique. In this case, the RIDE debugger needs to read the PAGE register to calculate the current PC. This PAGE register is found within the CSIOP segment (that can be relocated anywhere in the XDATA segment).

C.7 Debugging the application on the target hardware using RIDE

Refer to *Section 3* of this document and also to RIDE documentation.

C.8 Main features

- Hardware breakpoints: the embedded debug module provides four hardware breakpoints that can be used either as standard breakpoints in the program, or as data breakpoints (See RIDE documentation for how to set breakpoints). Note that the RIDE debugger needs also to set temporary breakpoints to perform most of the HLL commands (step over/into/ out…). Therefore, it is highly recommended to disable the breakpoints when they are not used.

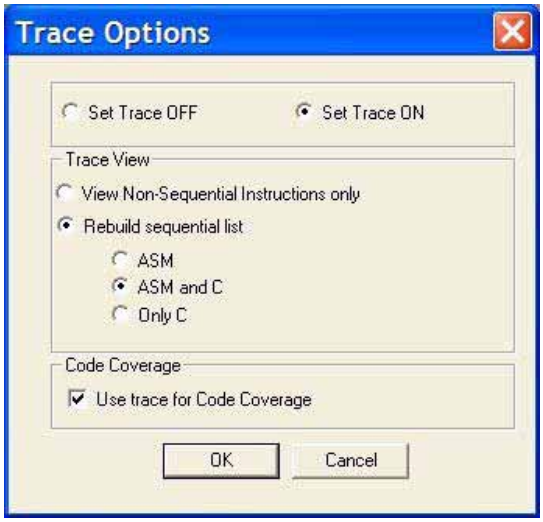
Note: When the four breakpoints are already set, the debugger displays a message to report this situation.

- Execution control (Step into/over/…),
- Data/SFR visualization,
- Trace mode (see next paragraph).

C.9 Trace mode

The on-chip debug system of the Turbo+ uPSD core features a powerful trace mode. To either enable or disable this mode, choose **Debug | Trace | Options** and the dialog shown in *Figure 34* appears.

Figure 34. Trace options window



When enabled, the CPU transfers the destination address at every non-sequential instruction (e.g. JMP, CALL, RET…) into a JTAG buffer that is read by the RLINK-ST dongle. In the case where two non-sequential instructions are executed almost consecutively, the bit-rate on the JTAG communication is not sufficient to read the previous destination address, and the execution is paused automatically (and released as soon as the JTAG buffer is empty). Therefore, setting the TRACE mode could slow down the overall execution.

Moreover, the standard breakpoint mechanism is no longer available when the trace mode is enabled. The breakpoints can be set, but they only trigger an interrupt instead of freezing the execution.

Table 2 summarizes the restrictions that are present when the trace is enabled.

Table 2. Execution performance with trace ON/OFF

	Trace OFF	Trace ON
Transparency	HW breakpoints stop the execution (the CPU clock is disabled)	HW breakpoints trigger an interrupt EA must be kept set to allow breakpoints Breakpoint interrupt vector () must be reserved
Real-time	Full-speed	Wait states are added (depending on the program) when non-sequential instructions are too frequent.

- When the trace mode is set, the breakpoints behave differently. Executing an instruction with a breakpoint sets the breakpoint interrupt flag. Therefore, the execution is stopped only one or two instructions later.
- The execution is stopped ONLY if the interrupt is currently enabled. When the execution is launched, RIDE enables the breakpoint interrupt. However, your program must avoid disabling the global interrupts or the debug breakpoint interrupt.

C.10 Reliability of the trace/code coverage information

Due to the dynamic mechanism used for tracing, Trace and Code Coverage has some limitations which needs to be noted. Some known issues are listed below:

1. When several conditional jumps branch to the same address, it's not possible to detect the effective branch.
2. When an interrupt occurs, the current instruction (when non-sequential) is unknown. The following instructions are listed in the trace buffer until encountering the next-nonsequential instruction. But a correction is done in the code coverage to avoid counting twice these instructions.

Appendix D CAPS reports

D.1 Project.rpt

This report is generated by CAPS after the Fit design to silicon step. The report for the LED BLINK example is listed here.

```
********************************************************************************
* Project file generated by CAPS Version 1.00 - 4/25/2006 16:52:01
* Project Name           : project
* Project Folder         : C:\Work\new_upsd_project\CAPS
* Project Description    :
* Target Device          : uPSD3434E-40U6
* Design Entry Mode      : Wizard mode
********************************************************************************
--------------------------------------------------------------------------------
System Memory Map
=================
  Main Flash memory will reside in this space at power-up      : Program Space Only
  Secondary Flash memory will reside in this space at power-up : Data Space Only

  fs0 = (address >= ^h0000) & (address <= ^h7FFF);
  fs1 = (page == 0) & (address >= ^h8000) & (address <= ^hFFFF);
  fs2 = (page == 1) & (address >= ^h8000) & (address <= ^hFFFF);
  fs3 = (page == 2) & (address >= ^h8000) & (address <= ^hFFFF);
  fs4 = (page == 3) & (address >= ^h8000) & (address <= ^hFFFF);
  fs5 = (page == 4) & (address >= ^h8000) & (address <= ^hFFFF);
  fs6 = (page == 5) & (address >= ^h8000) & (address <= ^hFFFF);
  fs7 = (page == 6) & (address >= ^h8000) & (address <= ^hFFFF);
  csboot0 = (address >= ^h8000) & (address <= ^h9FFF);
  csboot1 = (address >= ^hA000) & (address <= ^hBFFF);
  csboot2 = (address >= ^hC000) & (address <= ^hDFFF);
  csboot3 = (address >= ^hE000) & (address <= ^hFFFF);
  rs0 = (address >= ^h0000) & (address <= ^h1FFF);
  csiop = (address >= ^h7F00) & (address <= ^h7FFF);

  Assigned Firmware Files
  -----------------------
  Mapping mode : Direct

              File     File
  Memory BlockStart AddressEnd AddressFirmware File
  ---------------------------------------------------
  fs0           0000           7FFF
C:\Work\new_upsd_project\NEW_UPSD_PROJECT.HEX
  fs1           8000           FFFF
  fs2           8000           FFFF
  fs3           8000           FFFF
  fs4           8000           FFFF
  fs5           8000           FFFF
  fs6           8000           FFFF
  fs7           8000           FFFF
  csboot08000          9FFF
  csboot1A000          BFFF
  csboot2C000          DFFF
  csboot3E000          FFFF
--------------------------------------------------------------------------------
External Chip-Select Equations
==============================
```

```
  N/A
--------------------------------------------------------------------------------
Additional Setting
==================

  Device Security Protection     : Off

  Sector Protection :
        Main Flash           Protection Status
        ----------           -----------------
        Sector 0             unprotected
        Sector 1             unprotected
        Sector 2             unprotected
        Sector 3             unprotected
        Sector 4             unprotected
        Sector 5             unprotected
        Sector 6             unprotected
        Sector 7             unprotected

        2nd  Flash           Protection Status
        ----------           -----------------
        Sector 0             unprotected
        Sector 1             unprotected
        Sector 2             unprotected
        Sector 3             unprotected
--------------------------------------------------------------------------------
I/O Pin Assignment
==================
  Pin Function                  Signal Name           Pin Number
  ------------                  -----------           ----------
  ALE output                    ale                   4
  Dedicated JTAG - TDO          tdo                   6
  Dedicated JTAG - TDI          tdi                   7
  JTAG debug pin                JTAG_debug_pin        8
  USB+ bus                      USB_plus              11
  USB- bus                      USB_minus             14
  Dedicated JTAG - TCK          tck                   17
  Dedicated JTAG - TMS          tms                   20
  Data/Address line             a0                    36
  Data/Address line             a1                    37
  Data/Address line             a2                    38
  Data/Address line             a3                    39
  Data/Address line             a4                    41
  Data/Address line             a5                    43
  Data/Address line             a6                    45
  Data/Address line             a7                    47
  Xtal1                         Xtal1                 48
  Xtal2                         Xtal2                 49
  Bus control output            _wr                   62
  Bus control output            _psen                 63
  Bus control output            _rd                   65
  Reset In                      _Reset_In             68
  VREF input                    VREF                  70

--------------------------------------------------------------------------------
Fitting Result
==============
                                                  ----------------------------
                                                 |                            |
```

```
                                          |1 ] pd2            adio4 [41| Address
Bus a4/Data Port d4, ad4
                                           |2 ] p3_3            p3_5 [42|
                                          |3 ] pd1            adio5 [43| Address
Bus a5/Data Port d5, ad5
                                     ale |4 ] pd0            p3_6 [44|
                                          |5 ] pc7            adio6 [45| Address
Bus a6/Data Port d6, ad6
                                 tdo, TDO |6 ] pc6/TDO         p3_7 [46|
                                tdi, TDI |7 ] pc5/TDI        adio7 [47| Address
Bus a7/Data Port d7, ad7
                           JTAG_debug_pin |8 ] debug          Xtal1 [48| Xtal1
                                          |9 ] pc4/TERR       Xtal2 [49| Xtal2
                                          |10] 3.3V VCC    5.0V VCC [50|
                                USB_plus |11] USBp             N/C [51|
                                          |12] 5.0V VCC       p1_0 [52|
                                          |13] GND            N/C [53|
                               USB_minus |14] USBm            p1_1 [54|
                                          |15] pc3/TSTAT      N/C [55|
                                          |16] pc2            p1_2 [56|
                                 tck, TCK |17] pc1/TCK          N/C [57|
                                          |18] p4_7           p1_3 [58|
                                          |19] p4_6           p1_4 [59|
                                 tms, TMS |20] pc0/TMS         p1_5 [60|
                                          |21] pa7            p1_6 [61|
                                          |22] pa6           cnt10 [62| _wr
                                          |23] p4_5           cnt12 [63| _psen
                                          |24] pa5            p1_7 [64|
                                          |25] p4_4           cnt11 [65| _rd
                                          |26] pa4             pb7 [66|
                                          |27] p4_3             pb6 [67|
                                         |28] pa3         Reset_In [68| _Reset_In
                                          |29] GND             GND [69|
                                          |30] p4_2            Vref [70| VREF
                                          |31] p4_1             pb5 [71|
                                          |32] pa2            AVcc [72|
                                          |33] p4_0             pb4 [73|
                                          |34] pa1              pb3 [74|
                                          |35] pa0            p3_0 [75|
              ad0, Address Bus a0/Data Port d0 |36] adio0         pb2 [76|
              ad1, Address Bus a1/Data Port d1 |37] adio1         p3_1 [77|
              ad2, Address Bus a2/Data Port d2 |38] adio2          pb1 [78|
              ad3, Address Bus a3/Data Port d3 |39] adio3         p3_2 [79|
                                          |40] p4_4             pb0 [80|
                                          |                         |
                                           -----------------------------
     ===== Resource Usage Summary =====

Total Product Terms Used:  15

Device Resources                    used / total
-------------------------------------------------
Port A: (pins 35 34 32 28 26 24 22 21)
I/O Pins :                           0   /  8
   GP I/O or Address Out        :    0
   Peripheral I/O               :    0
   Logic Inputs                 :    0
   Address Latch Inputs         :    0
   PT Dependent Latch Inputs    :    0
   PT Dependent Register Inputs :    0
```

```
   Combinatorial Outputs        :    0
   Registered Outputs           :    0
Other Information
   Microcells                   :    0   /  8
     Micro-Cells AB :
      Buried Microcells         :    0
      Output Microcells         :    0
   Product Terms                :    0   /  24
   Control Product Terms        :    0   /  34

Port B: (pins 80 78 76 74 73 71 67 66)
I/O Pins :                           0   /  8
   GP I/O or Address Out        :    0
   Logic Inputs                 :    0
   Address Latch Inputs         :    0
   PT Dependent Latch Inputs    :    0
   PT Dependent Register Inputs :    0
   Combinatorial Outputs        :    0
   Registered Outputs           :    0
Other Information
   Microcells                   :    0   /  8
     Micro-Cells AB :
      Buried Microcells         :    0
      Output Microcells         :    0
     Micro-Cells BC :
      Buried Microcells         :    0
      Output Microcells         :    0
   Product Terms                :    0   /  24
   Control Product Terms        :    0   /  34

Port C: (pins 20 17 16 15 9 7 6 5)
I/O Pins :                           4   /  8
   GP I/O or Address Out        :    0
   Logic Inputs                 :    0
   Address Latch Inputs         :    0
   PT Dependent Latch Inputs    :    0
   PT Dependent Register Inputs :    0
   JTAG signals                 :    4
   Standby Voltage Input        :    0
   Rdy/Bsy signal               :    0
   Standby On Indicator         :    0
   Combinatorial Outputs        :    0
   Registered Outputs           :    0
Other Information
   Microcells                   :    0   /  8
     Micro-Cells BC :
      Buried Microcells         :    0
      Output Microcells         :    0
   Product Terms                :    0   /  32
   Control Product Terms        :    0   /  34

Port D: (pins 4 3 1)
I/O Pins :                           1   /  3
   GP I/O or Address Out        :    0
   Logic Inputs                 :    0
   Chip-Select Input            :    0
   Clock Input                  :    0
   Control Signal Input         :    1
   Fast Decoding Outputs        :    0
Other Information
```

```
   Product Terms                :    0   /  3
   Control Product Terms        :    0   /  3


     ==== OMC Resource Assignment ====

  Resources           PT             User
  Used                Allocation     Name
 ------------------------------------------------------------
Micro-Cell AB :

Micro-Cell BC :

External Chip Select :


     ========= Equations =========

DPLD          EQUATIONS :
=======================
     fs0 = !pdn & !a15;

     fs1 = !pdn & !pgr2 & !pgr1 & !pgr0 & a15;

     fs2 = !pdn & !pgr2 & !pgr1 & pgr0 & a15;

     fs3 = !pdn & !pgr2 & pgr1 & !pgr0 & a15;

     fs4 = !pdn & !pgr2 & pgr1 & pgr0 & a15;

     fs5 = !pdn & pgr2 & !pgr1 & !pgr0 & a15;

     fs6 = !pdn & pgr2 & !pgr1 & pgr0 & a15;

     fs7 = !pdn & pgr2 & pgr1 & !pgr0 & a15;

     csboot0 = !pdn & a15 & !a14 & !a13;

     csboot1 = !pdn & a15 & !a14 & a13;

     csboot2 = !pdn & a15 & a14 & !a13;

     csboot3 = !pdn & a15 & a14 & a13;

     csiop = !pdn & !a15 & a14 & a13 & a12 & a11 & a10 & a9 & a8;

     rs0 = !pdn & !a15 & !a14 & !a13;

     jtagsel = !_reset;

PORTA         EQUATIONS :
=======================
PORTB         EQUATIONS :
=======================
PORTC         EQUATIONS :
=======================
PORTD         EQUATIONS :
=======================
                                      ---  End  ---
```

7 Revision history

Table 3. Document revision history

Date	Revision	Changes
29-Mar-2007	1	Initial release.