AN1560
APPLICATION NOTE

Design Guide
µPSD3200 Family

MPCONTENTS

The µPSD3200 family is a series of 8051-class microcontrollers (MCUs) containing an 8032 core with a large dual-bank Flash memory, a large SRAM, many peripherals, programmable logic, and JTAG In-System Programming (ISP). This document shows the steps to create a design using the DK3200 development board, the software development tool PSDsoft Express, and uVision2 8051 Integrated Development Environment (IDE) from Keil Software.

µPSD3200 FAMILY OVERVIEW

The µPSD3200 family is a standard 12-clock per instruction 8032 MCU capable of being clocked up to 40MHz at 5.0V and 24MHz at 3.3V at industrial operating temperature range. Currently there are seven family members that are different combinations of Flash memory size, operating voltage, peripheral set, and packaging (see datasheet). The fullest featured part, µPSD3234A-40U6, is used in this Application Note. The term µPSD is used throughout the remainder of the document for brevity. See µPSD block diagram in Figure 1.

The µPSD has a unique memory structure that includes two independent Flash memory arrays (main and secondary) capable of read-while-write operation. This is ideal for In-Application Programming (IAP) because the 8032 can fetch instructions from one Flash memory 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 µPSD 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 64K byte addresses. The 8032 may write to the Page Register at runtime.

For more complex designs, the µPSD 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 memory 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.

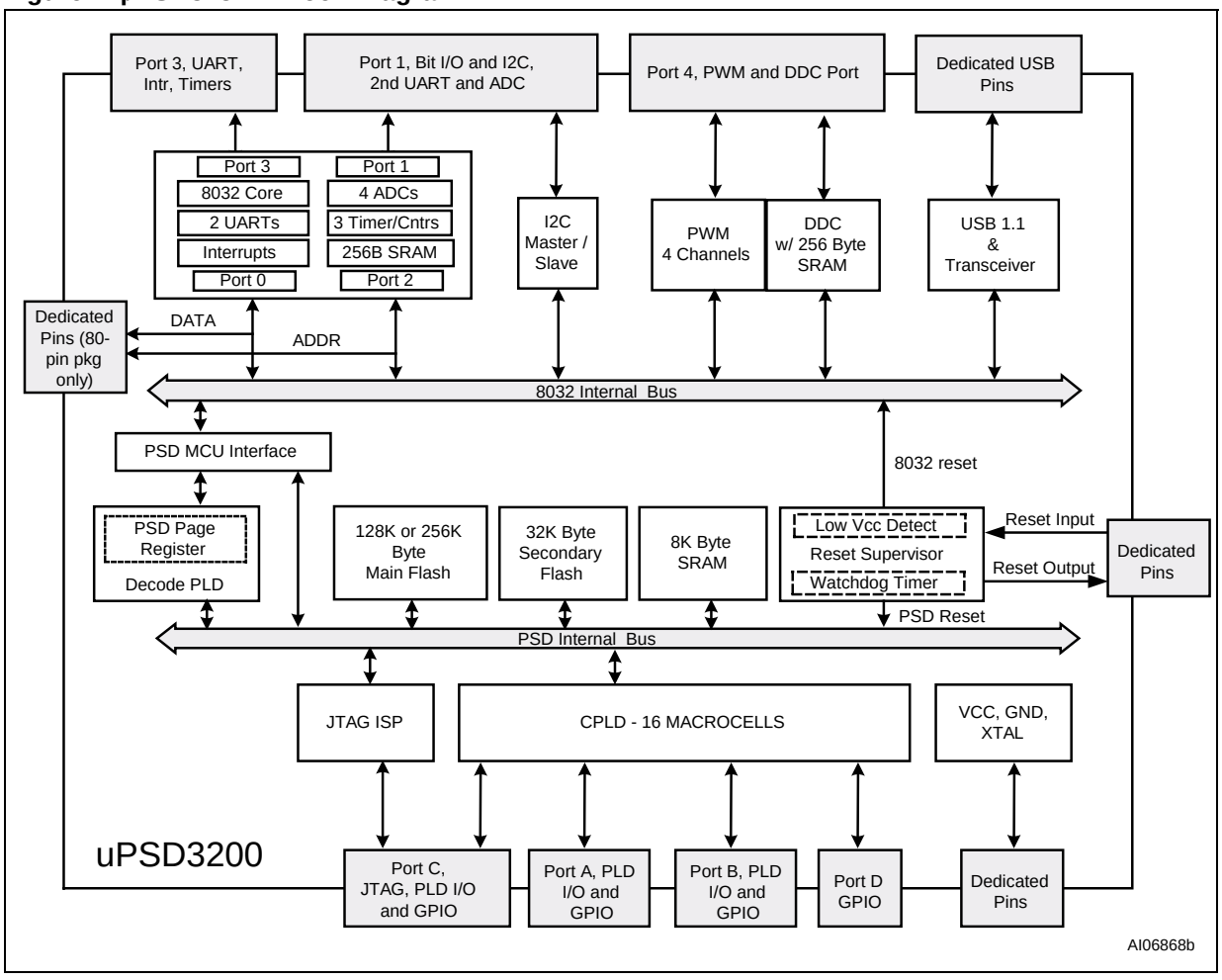
Many peripherals are available in this µPSD, including: USB

v1.1 (low speed), two UART channels, four PWM channels, one I2C channel, four 8-bit ADC channels, DDC (Data Display Channel for LCD monitors and projectors), a watchdog timer, low-Vcc detection with reset-out, a general purpose PLD, and many GPIO.

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: One, by a block of xdata memory mapped control registers, whose base address (*csiop*) can be mapped anywhere using the DPLD; Two, by the programmable logic.

The JTAG ISP channel on Port C 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.

Figure 1. µPSD3234A Block Diagram

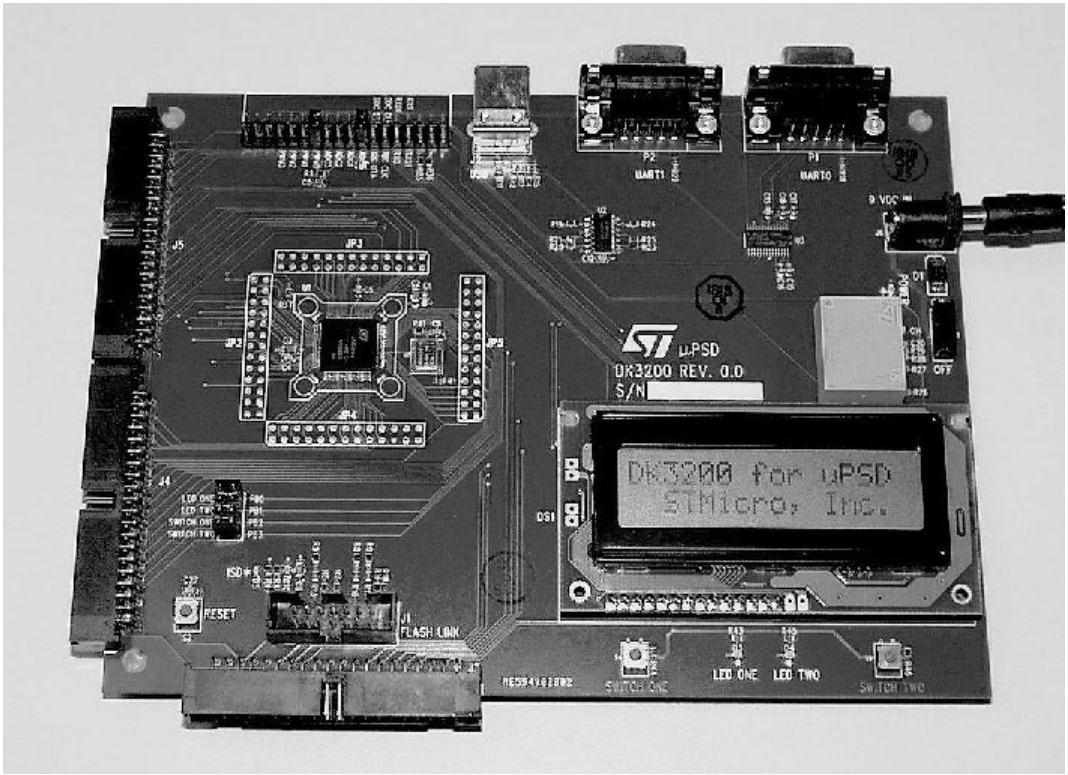


DK3200 OVERVIEW

A picture of the DK3200 board is shown in Figure 2. Board layout and schematics are in the Appendix. Connectors JP1, J3, J4, J5 provide easy access to all µPSD I/O signals for expansion or testing. JP1 accepts jumper shunts to wrap µPSD outputs back into µPSD inputs for testing. J3, J4, J5 can connect directly to standard Agilent (HP) Logic analyzer pods. UARTs are available on P1 and P2. A USB host can connect to the µPSD as a peripheral via J2. The FlashLINK JTAG ISP cable connects at J1. Connectors

JP2, JP3, JP4, JP5 allow direct connection of the In-Circuit Emulator from Nohau Corp, EMUL-µPSD3200-PC. JP6 accepts jumpers to connect the switches (SW1, SW2) and the LEDs (LED1, LED2) to PSD port B. LED D5 indicates JTAG ISP Programming. The DK3200 also has a 2-line 16 character LCD interface and a full featured real-time clock with SNAPHAT snap-on battery/crystal pack.

Figure 2. DK3200 Development Board



DESIGN EXAMPLE BLOCK DIAGRAM

This simple design example is represented by the block diagram of Figure 3, and the memory map of Figure 4. All 16 macrocells of the PLD are used, Flash memory is paged, and few of the 8032 interfaces (ADC, PWM, UART) are configured and used. The idea is to touch several aspects of the µPSD that may be unfamiliar to a typical 8051 user and to give you an idea of how to use the design tools and become familiar with µPSD architecture.

Figure 3. Design Block Diagram

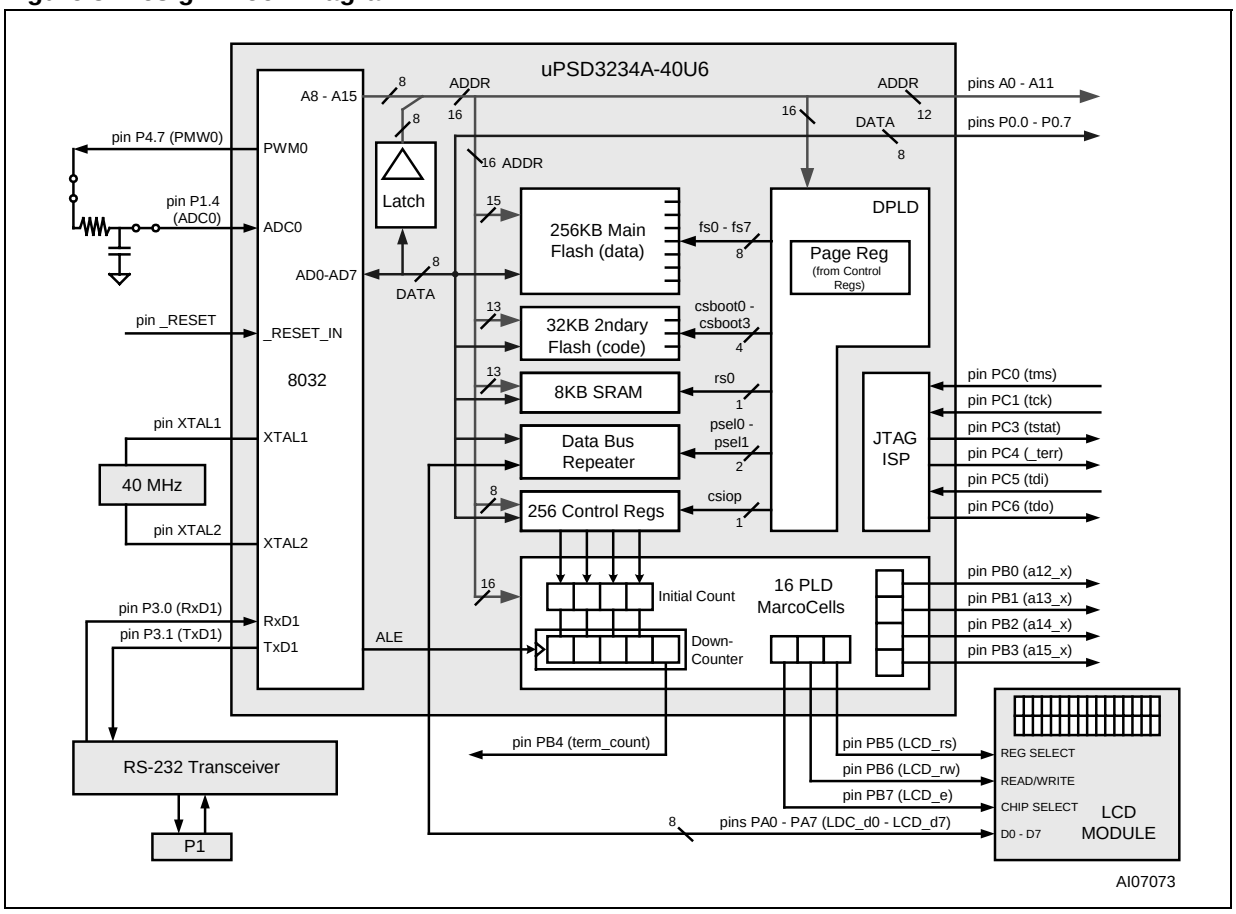


Figure 3 shows the design implemented in this application note. Major elements are the µPSD, an LCD module, and an RS-232 transceiver chip.

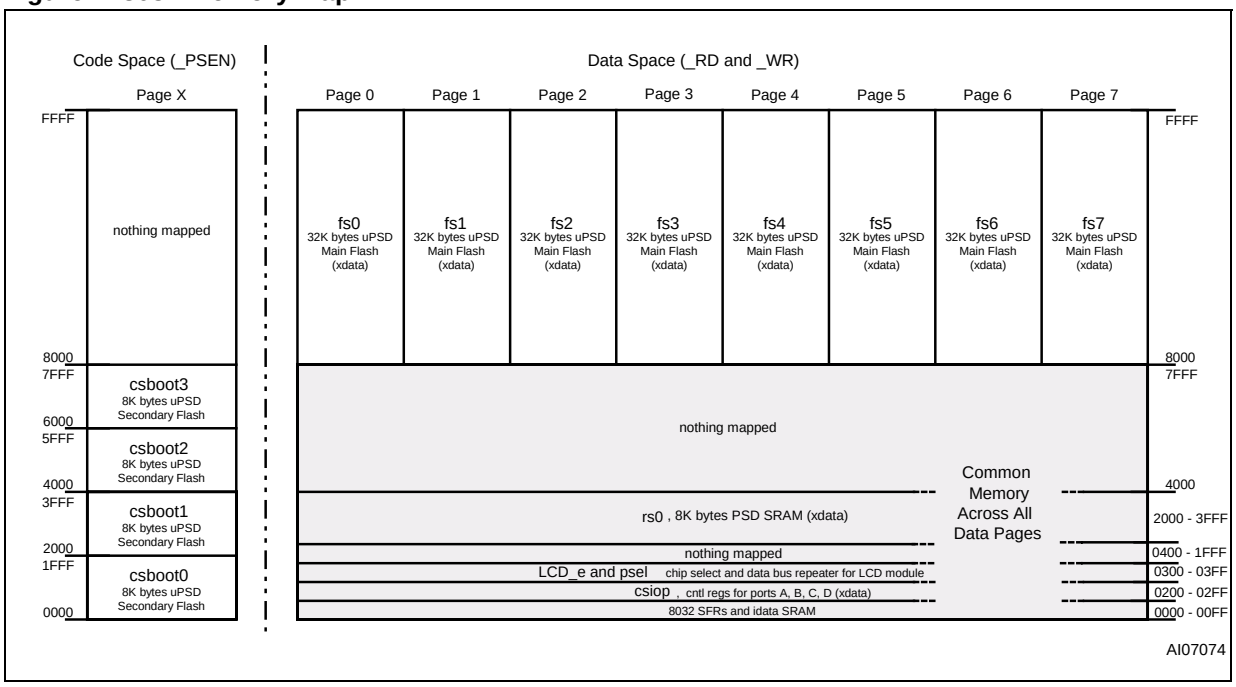
The 8032 outputs a repetitive PWM pulse train with a slowly varying pulse width to an RC network which converts the pulse train into a slowly sweeping DC voltage (0 to 5V). This DC signal is looped back into an ADC input. The 8032 will write the resulting Hexadecimal ADC conversion value to the LCD so you can watch the results. The RC network and loop-back is implemented with two jumpers on the DK3200 board.

Additionally and independently, a 4-bit auto-reloading down-counter is created using PLD macrocells. The 8032 directly loads the initial count value into four macrocells, and that count is automatically loaded into another four macrocells that create the 4-bit down-counter. Reloading occurs each time the counter reaches terminal count of zero. Terminal count is indicated externally by a pulse on a µPSD output pin. The down-counter is clocked by ALE signal (ALE was random choice, could be any signal). The 8032 may load a different initial count at anytime, creating a variable divider of the ALE signal.

Four more macrocells are used to output the high four 8032 address signals. The 80-pin µPSD only outputs the low twelve 8032 address signals on dedicated pins. If more address signals are needed externally, they have to be added this way using the PLD.

The LCD module is connected to the µPSD via a Port A for data and Port B for some glue logic and a chip-select signal. Port A is operating is an special data bus repeater mode this example, called Peripheral I/O mode. 8032 data will pass through port A only for a given address range specified in PSDsoft Express (illustrated later).

Figure 4. 8032 Memory Map



The memory map in Figure 4 shows that the 32K byte secondary Flash memory is used for 8032 code, and the 256K byte main Flash memory is used for 8032 data, banked over eight pages. The nomenclature *fsx, csbootx, rs0*, *csiop,* and *psel* in Figure 4 refer to the individual internal µPSD memory segments. The µPSD main Flash memory has a total of eight 32K byte segments (*fs0..fs7*). The µPSD secondary Flash memory has a total of four 8K byte segments (*csboot0-csboot3*). The µPSD 8K byte SRAM has a single segment (*rs0*). A group of µPSD control registers which control I/O ports A, B, C, and D lie in a 256-byte xdata address space whose base address is named *csiop*. The µPSD has a data bus repeater feature that is enabled over a given address range as specified by *psel*. Figure 4 also shows one external memory select signal, *LCD_e*, for the LCD module. This memory map is specified using the software tool PSDsoft Express. Each memory segment can be placed at virtually any address, which provides an infinite number of mapping schemes. This is just one example.

We'll keep things simple for this particular application note, meaning the 8032 will "boot" and run code contained completely within the 32K byte secondary Flash memory in code space and we'll treat the 256K byte main Flash memory as data only. However, this memory map may grow with the needs of your project. For example, if a large Flash memory is needed for code space and IAP is required, a slight variation of the map in Figure 4 can accomplish this. The 8032 can boot from secondary Flash memory (secondary Flash memory resides in code space from 0-7FFF as in Figure 4), then the 8032 can calculate a checksum on the main Flash memory and then program the main Flash memory if necessary (main Flash memory resides in data space from 8000-FFFF on eight pages as in Figure 4). After the contents of main Flash memory are verified, the 8032 can write to special register, called the VM register within the *csiop* register block, to "reclassify" the main Flash memory from data space to code space. After which, the 8032 will have access to 256K bytes of Flash memory for code in code space, paged across eight code pages in upper memory (8000-FFFF), and the 8032 will have access to 32K bytes of Flash memory for code in code space common to all pages in lower memory (0-7FFF). At that point no Flash memory will reside in data space. Upon reset, the memory map is reset to look like Figure 4 again. The VM register can be accessed by the 8032 at runtime to perform a variety of manipulations. PSDsoft is used to set the initial value of the VM register upon power-up. Future Application notes will illustrate various memory schemes.

ENTERING DESIGN IN PSDSOFT EXPRESS

Highlights of the design process will be given here. The steps are simple and navigation through PSDsoft Express is easy. Invoke PSDsoft Express and follow along if you wish. PSDsoft Express is included in the DK3200 CD, but you should check for latest updates. Updates are available from our website at www.st.com/psm, in the “Software Downloads” area.

Invoke PSDsoft Express and Create Project

* Install PDSsoft Express (from the web or the included CD)

* Start PSDsoft Express

* Create a new project

* Select your project folder and name the project (in this example, name the project “DK3200_1” in the folder PSDexpress\my_project).

Select MCU and Initial Placement of Flash in Code Space or Data Space

* Select an MCU. In this case it is STMicroelectronics, then µPSD32xx, then µPSD3234A.

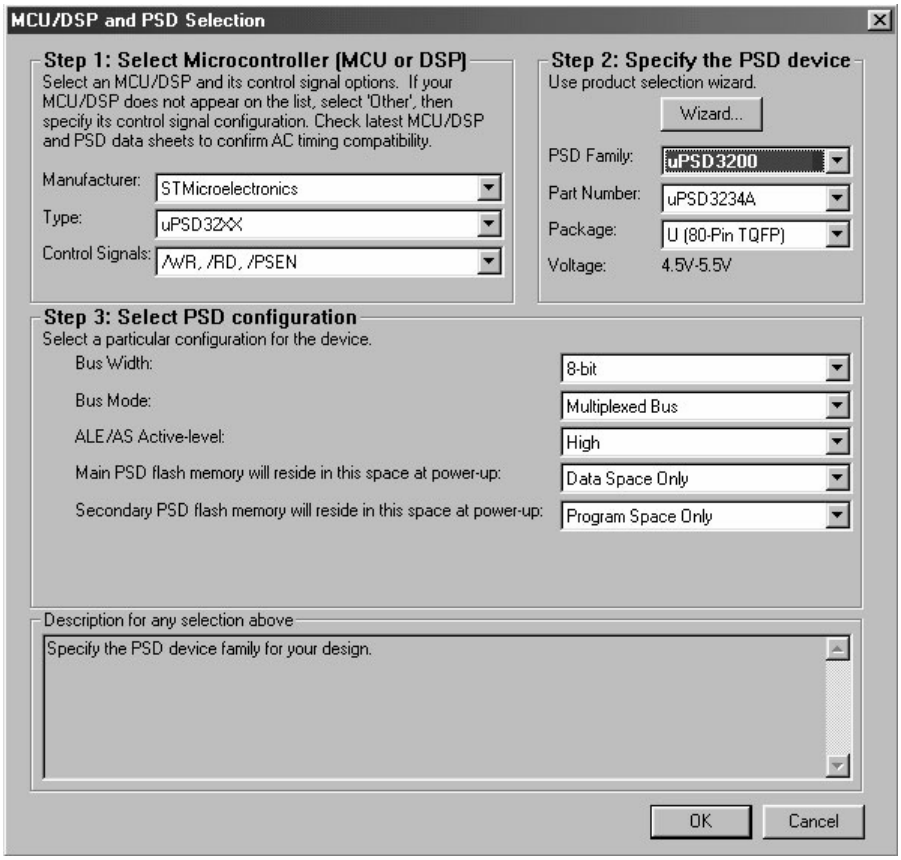
* Select the main Flash memory to reside in 8032 data space at power-up (means that the 8032 _RD and _WR signals are routed to the main Flash memory array)

* Select the secondary Flash memory to reside in 8032 code space at power-up (means that the 8032 _PSEN signal is routed to the secondary Flash memory array)

Note: At runtime, the 8032 can alter the initial settings of code and data space by writing to the VM register.

Figure 5 shows what the screen should look like after you’ve made the selections.

Figure 5. MCU Selection



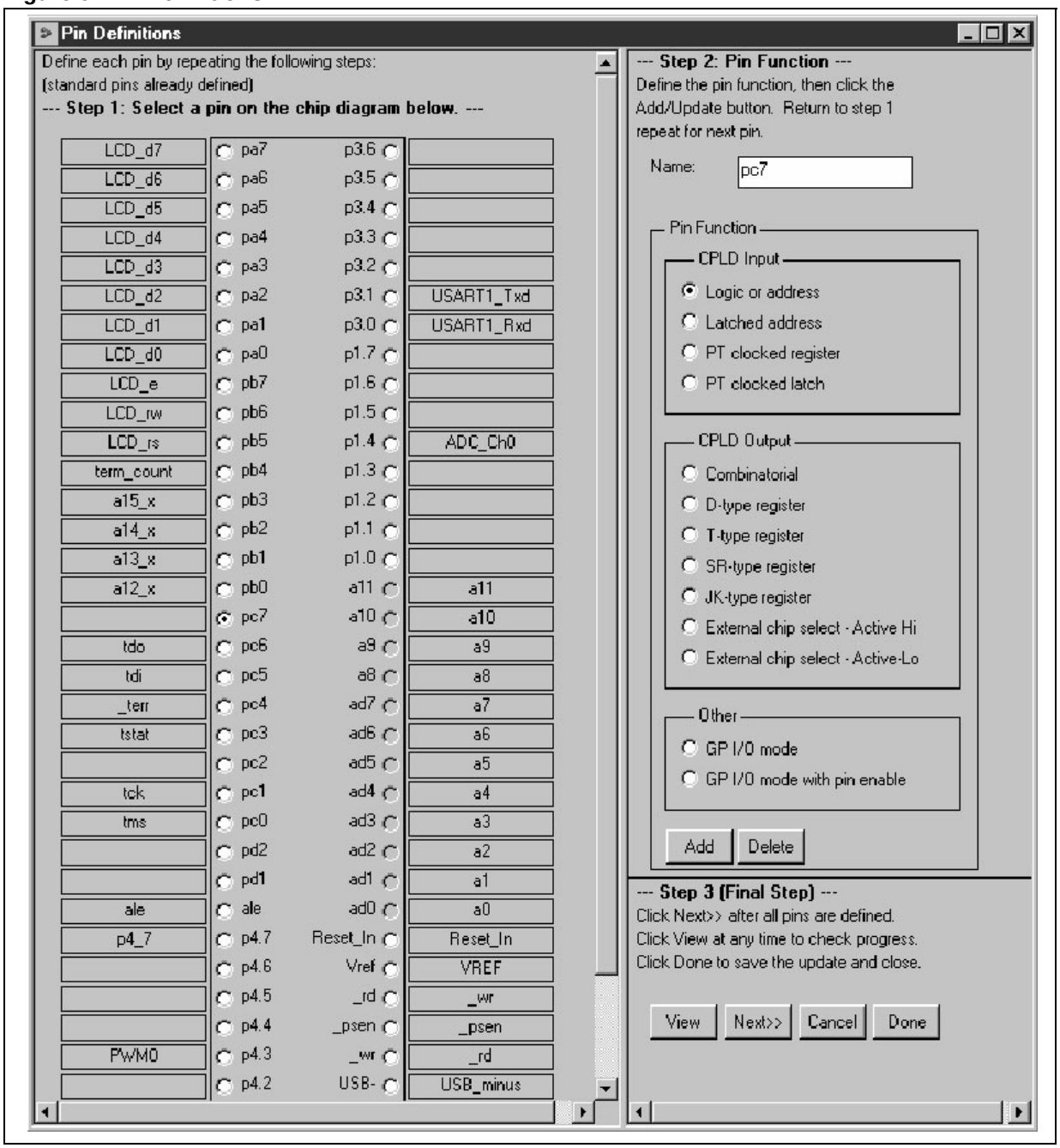
Click OK. Now you will be asked if you want to use the Design Assistant, Extended Design Assistant, or Example Template. Choose Example Template. This is a predefined design that matches this application note and it runs on the DK3200 board. Next choose the template for the DK3200 Kit when prompted.

Pin Definitions

You will see the Pin Definitions screen appear. All of the pin definitions shown in block diagram of Figure 3 are filled in. Click through the pins and see how they are configured and how they relate to Figure 3. You'll notice that you cannot change the definition of some pins because they have a fixed function.

A comment about JTAG pins. This example uses 6-pin JTAG which is up to 30% faster than the default standard 4-pin JTAG. The two extra pins in the 6-pin JTAG configuration are *_tstat* and *terr*.

Figure 6. Pin Definitions



Now click “Next” to move on to the Design Assistant for memory mapping and logic equations. You will see the Page Register definition screen.

Memory Map

Defining the memory map requires defining the address range of chip-selects for individual memory elements of the µPSD (memory external to the 8032 core). Definition of the use of the µPSD Page Register is also required.

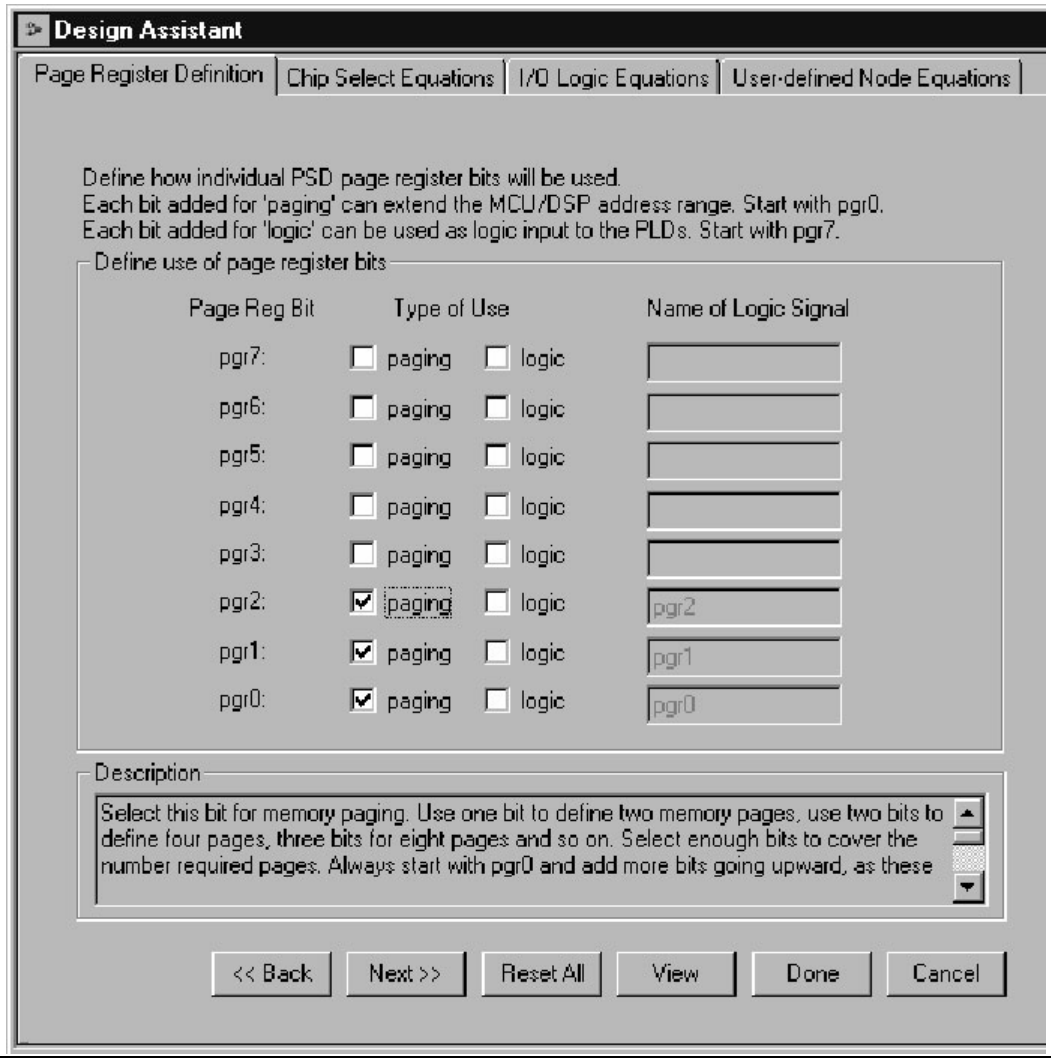
Four memory blocks (main Flash memory, secondary Flash memory, SRAM, and control registers) exter-

nal to the 8032 core are available and are individually selected segment-by-segment when 8032 addresses are presented to the Decode PLD (DPLD). Each of these memory segments has its own chip-select name (*fs3, csboot1, rs0, csiop*, etc.). Equations for these chip-selects, and for any external chip-selects, must be specified using PSDsoft Express. For this example, chip-selects are defined to match the memory map of Figure 4.

Page Register

Since eight memory pages (or banks) are needed as shown in Figure 4, three paging bits ($2^3 = 8$) are specified as shown in Figure 7. The µPSD supports up to 4 paging bits (pg0, pg1, pg2, pg3) for a total of 16 pages. Unused paging bits including pg4, pg5, pg6 and pg7 may be used for other functions. Note that the paging bits used must be the LSB bits in the paging register. Click “Next”.

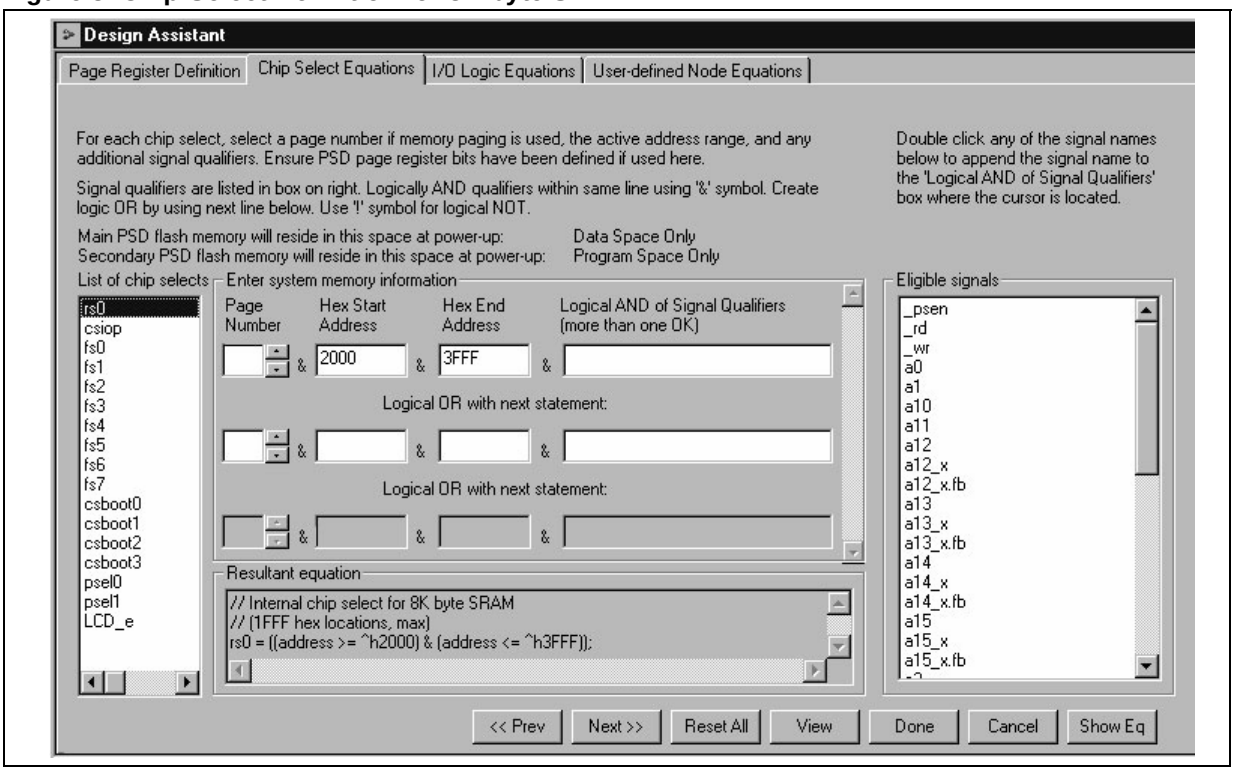
Figure 7. Page Register Definition



Chip-Select Equations

Now you will see the Chip-Select definition screen. Click the chip-select signal *rs0* for the 8K byte xdata SRAM, and see that its definition matches the memory map of Figure 4.

Figure 8. Chip-Select Definition for 8K byte SRAM



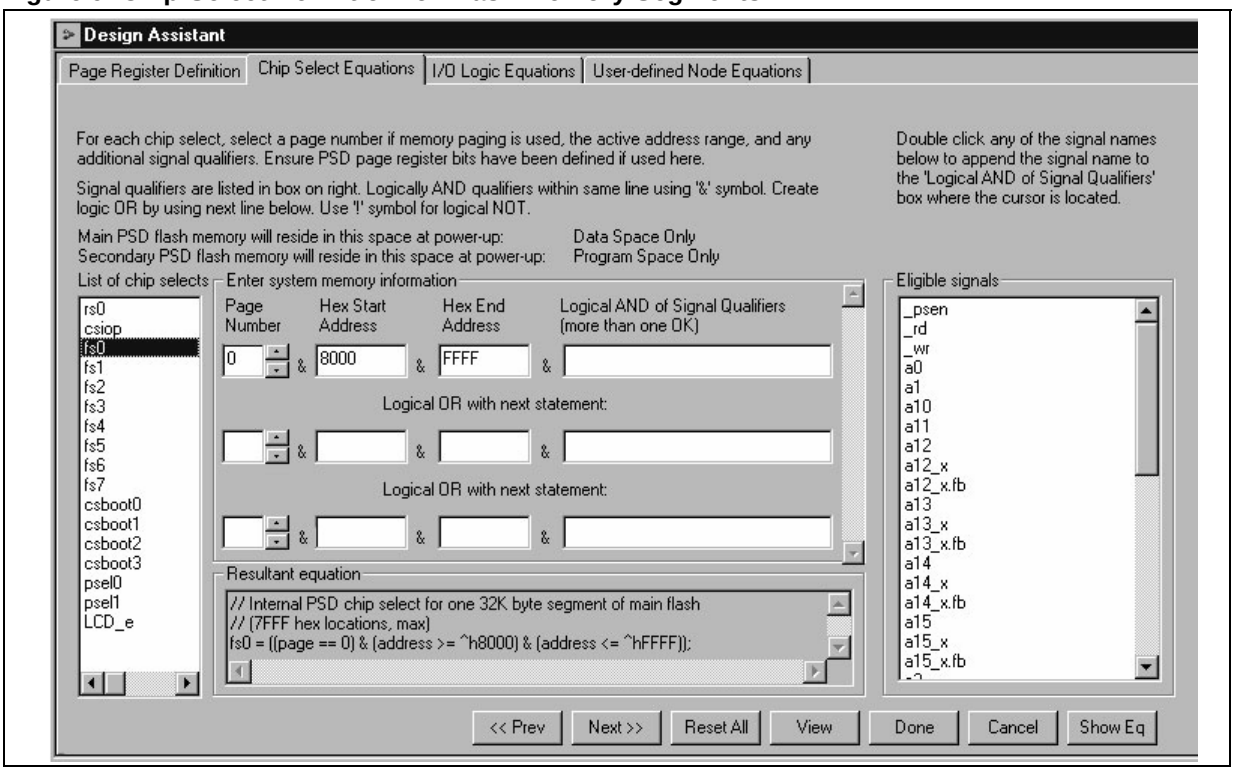
Notice that no page number is specified for *rs0* since the SRAM is common to all pages (page independent). Additional signal qualifiers (8032 control signals *_rd, _wr, _psen, ale*) are NOT needed for internal µPSD chip-selects as this is taken care of in silicon. The SRAM always defaults to 8032 data space.

At any time, you can click the “View” button to see how you are doing. A summary will appear.

Now click on the chip-select *csiop* (Chip Select I/O Port). This is a band of 256 xdata registers used to control µPSD ports A, B, C, D, the Page Register, power management, and other functions. 40 of the 256 registers are used, see µPSD data sheet for register definitions and their address offset from the *csiop* base address. There is no need to specify additional signal qualifiers for *csiop*, and it is not allowed to place *csiop* on a particular memory page. The *csiop* must be xdata address space.

Next click on *fs0*. *fs0 .. fs7* are chip-selects for the eight 32K byte segments of µPSD main Flash memory. Notice the page number is 0 for *fs0*, and the address range is 8000 - FFFF as shown in memory map of Figure 4. Click on remaining chip-selects for main Flash memory and notice the page number assignments. No additional signal qualifiers are needed.

Figure 9. Chip-Select Definition for Flash Memory Segments

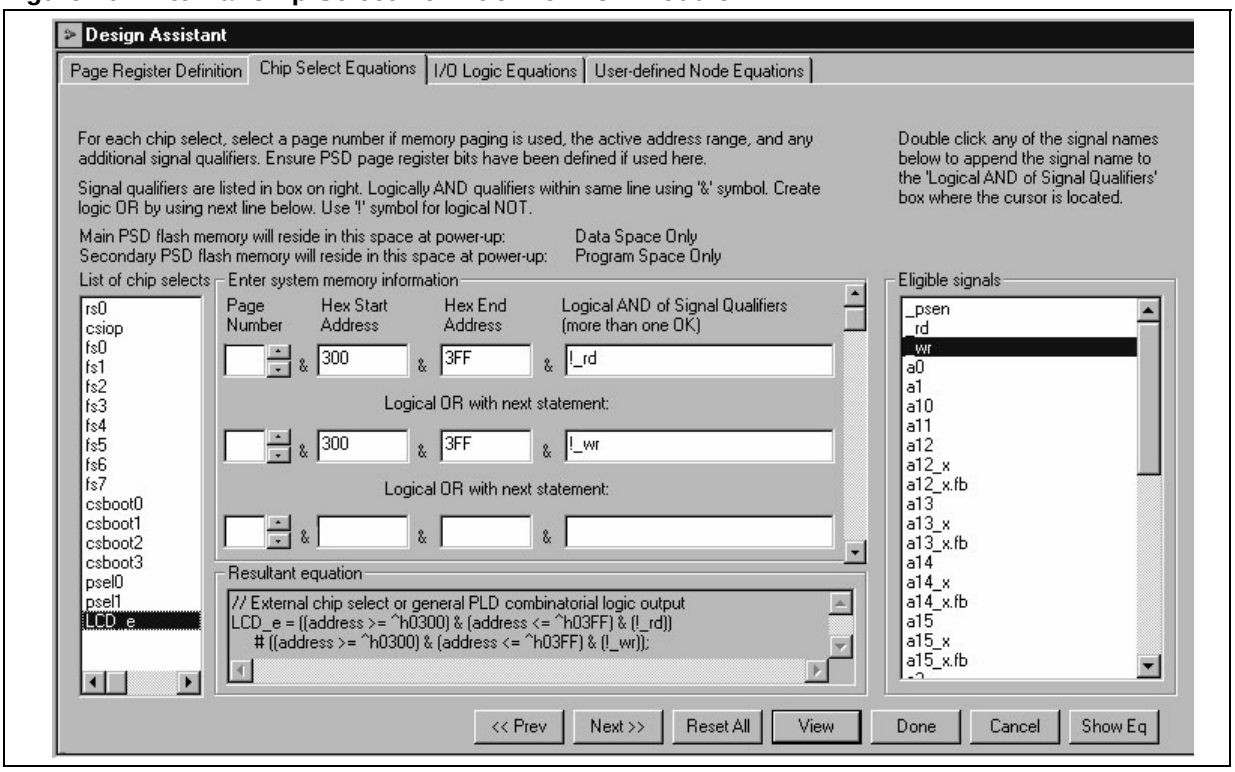


Now click on *csboot0*. *csboot0 .. csboot3* are chip-selects for the four 8K byte segments of µPSD secondary Flash memory. Check the address assignments for each of these chip-selects and notice there are no page numbers assigned. The secondary Flash memory is common to all pages.

Next click on *psel0*. This address range specifies when Port A pins will behave like a data bus repeater in Peripheral I/O Mode to drive the LCD module. Port A pins were earlier specified a “Peripheral I/O Mode” which acts like a ‘245 bus transceiver chip connecting the 8032 data bus to external peripherals over a given address range specified by the label *psel0* or *psel1*. The direction of this transceiver function is controlled automatically in silicon by the 8032 *_rd* and *_wr* signals. See µPSD data sheet for details. So all we have to do is click on *psel0* and enter the address range 300 to 3FF to enable this feature for that address range as shown in Figure 4, with no Page Number assignment. *psel1* is not needed because the Peripheral I/O feature is active for the logical OR of *psel0* or *psel1*.

And finally, click on *LCD_e*. This is an external chip-select for the LCD module. Since this is an **external** chip-select, we must include signal qualifiers *_rd* and *_wr*. In this design, *LCD_e* is true (active hi) only when the 8032 presents an address in the range of 300 to 3FF AND when either 8032 control signal *_rd* is true, OR when 8032 control signal *_wr* is true. To create this logic, information is entered as shown in Figure 10. Since both signals *_rd* and *_wr* are active low, the logical NOT operator (!) is used when they are specified as qualifiers. Signal qualifiers may be added by setting the cursor where you want the signal name to go, then just double click on the signal name in the list of eligible qualifiers.

Figure 10. External Chip-Select Definition for LCD Module



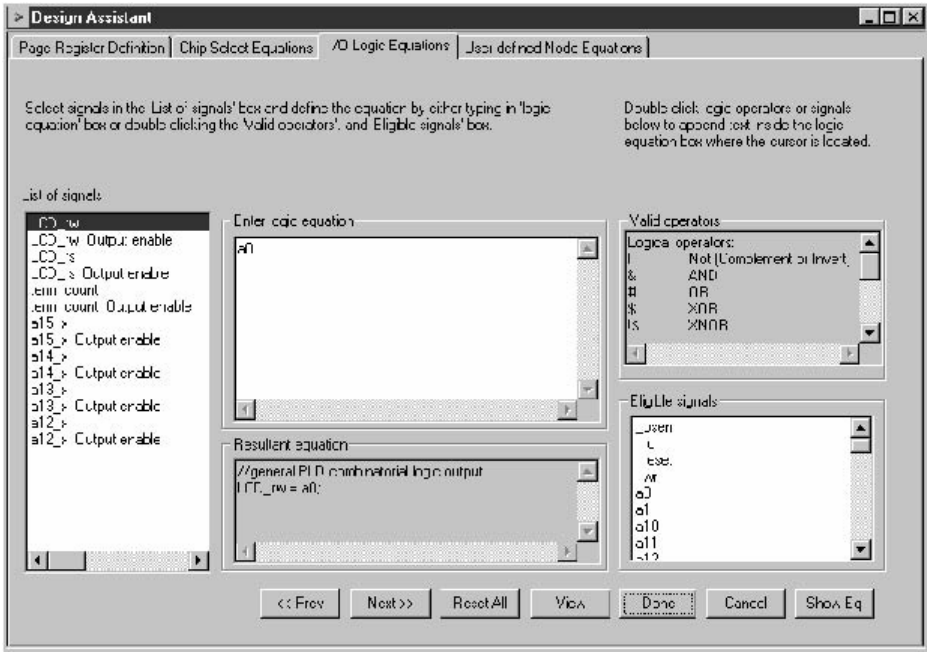
Click “next” to move on to logic definitions.

I/O Logic Equations

Defined here are equations for PLD outputs for the LCD interface signals, the additional 8032 address outputs, and the terminal count output signal from the down-counter. The Design Assistant (DA) will create HDL logic statements using the ABEL language in the background after you enter logic in this point-and-click design entry environment. The DA will also create all the declaration statements in ABEL. This saves much typing and reduces the chance of error. For more complicated logic PSDsoft allows you to edit the ABEL statements directly. In this example you’ll see simple logic entered point-and-click style, and you’ll see the 4-bit down-counter entered by editing the ABEL file directly.

Click on “*LCD_rw*” as shown in Figure 11, and notice that the internal signal *a0* is assigned to drive the output signal “*LCD_rw*”. Although this was a very simple logic equation, AND, OR, XOR, NOT, and other logic operators are also available for general purpose logic.

Figure 11. Logic Equation for signal LCD_rw

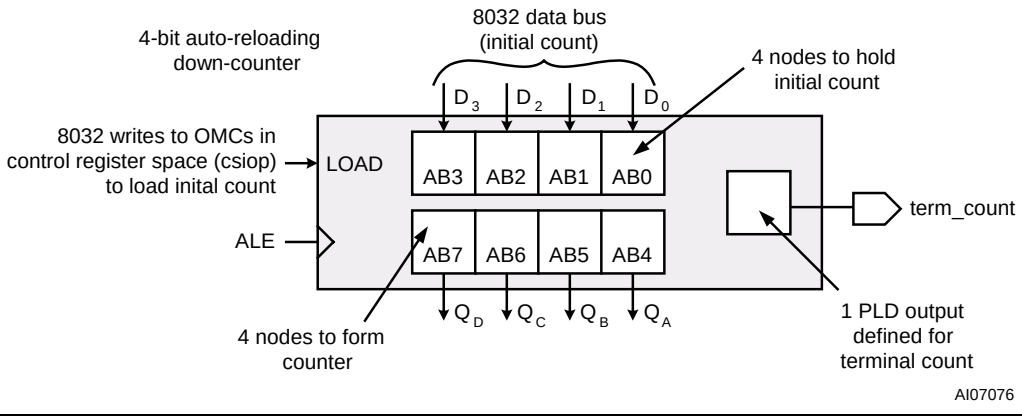


Click through the remaining signal names and observe the logic assigned. Notice there is no logic equation assigned to *term_count* because that assignment will be made by editing the ABEL file directly. Click “Next”.

User-Defined Node Equations

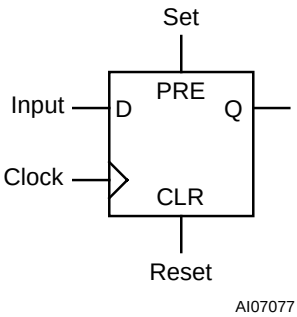
Here you will see how internal logic nodes are created. In this example there are four registers (or nodes) to hold the initial count of the 4-bit down-counter, and four additional registers to create the actual 4-bit down-counter. See Figure 12.

Figure 12. 4-bit Down-Counter with Automatic Reload of Initial Count



These nodes were created by clicking the “Def Node..” button, naming the node, and then selecting the type node (combinatorial, D-register, J-K register, etc.). In this example, all eight nodes are D-register type. When a register is created, you can specify it’s source of Input, Clock, Reset, as Set illustrated in Figure 13.

Figure 13. D-register Node



Click though the signal names and look at the assignments. Notice there are no definitions for inputs on any of the eight nodes. For the *down_count* nodes, the inputs are defined elsewhere (the ABEL file). For the init_count nodes, no logic input (or clock input) is specified because the 8032 will load the nodes directly by writing to the appropriate Output MacroCell register that resides the band of 256 registers of *csiop*.

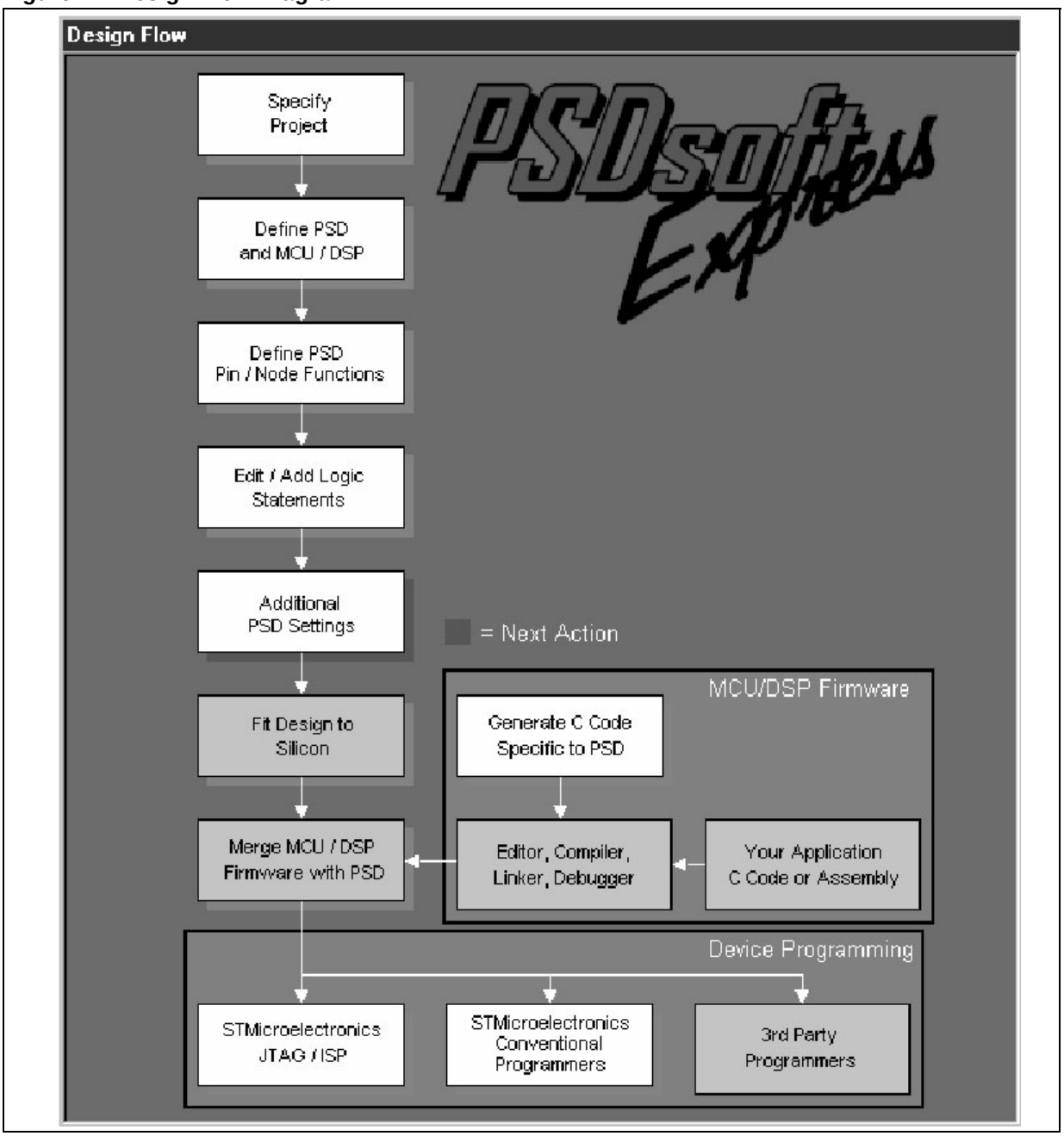
It may seem odd to divide the design entry this way (some point-and-click entry and some direct ABLE file editing), but many declaration statements are automatically created in the background by the point-and-click entry. You will see that when it is time to enter ABEL equations for the down-counter, there is very little typing involved.

Click “Done”. Now you will see the main PSDsoft flow diagram that will guide you through the remaining steps. You can see a summary report at this time by pulling down the “Report” selection in the main menu bar at the top of the screen, then select “Design Assistant Summary”. Your report will match the one in Appendix A.

Edit ABEL HDL Statements for PLD Design

If your PSDsoft flow diagram does not include the block “Edit/Add Logic Statements” as shown in Figure 14, then pull down the “Project” selection in the main menu bar at the top of the screen, then select “Preference”. Click the box that says “Enable ABEL Editing Capability”, then “OK”.

Figure 14. Design Flow Diagram



Click the “Edit/Add Logic Statements” box. You will see an “HDL Assistant” window pop up. Browse through this to see ABEL logic and syntax examples that you can cut and paste into future designs. Close the HDL Assistant and you will see the ABEL HDL source file. All the declarations and logic equations generated from the Design Assistant are there, and should match Appendix B.

There are only two regions in the ABEL file in which you can type statements, otherwise the DA will overwrite what you have typed next time you get into the DA.

The first safe region is for ABEL declarations and lies between the two statements: “// Begin user preserved declarations” and “// End user preserved declarations”.

The second safe region is for logic equations and lies between the two statements: “// Begin user pre-

served equations" and "// End user preserved equations".

Scroll down to the declaration region in the ABEL file, it should look like:

```
// Begin user preserved declarations (not affected by iterations of DA usage)
==================
WSIPSD PROPERTY 'DataBus_OMC D[7:4]:down_count[3:0] MCELLAB'; // This statement forces the
alignment
                          // of down_count bits [3..0] to the MCU data bus bit positions
[7..4].
                        // If this WSIPSD PROPERTY statement was not present, then PSDsoft
                      // would pick random MCU bit positions. The WSIPSD PROPERTY is needed
                         // only if the MCU will read or write to MicroCells and only if a
                           // particular MCU data bus position is required by the designer.

WSIPSD PROPERTY 'DataBus_OMC D[3:0]:init_count[3:0] MCELLAB'; // This statement forces the
alignment
                     // of init_count bits [3..0] to the MCU data bus bit positions [3..0].

DCOUNT = [down_count3..down_count0];  // 4-bit down counter
INIT = [init_count3..init_count0];// 4-bit initial count from MCU
//INIT = [0,1,0,0];

// End user preserved declarations (not affected by iterations of DA usage)
==================
```

Notice the WSIPSD PROPERTY statements. These are needed whenever you want to dictate the placement of certain macrocells of the PLD. If you do not enter any WSIPSD PROPERTY declarations statement, then the PSDsoft "fitter" process will place the macrocells in random order. This is not a problem for most designs. But in this example we want to load an initial count for the down-counter from the 8032 data bus so we must make sure the output macrocells holding the initial count are in the correct bit order and the correct position in the bank of eight output macrocells. The property statement:

WSIPSD PROPERTY 'DataBus_OMC D[7:4]:down_count[3:0] MCELLAB'

forces the order of the bits of the down-counter and places them on the upper half of the 8032 data bus.

The property statement:

WSIPSD PROPERTY 'DataBus_OMC D[3:0]:init_count[3:0] MCELLAB'

forces the order of the bits of the initial count and places them on the lower half of the 8032 data bus. Now when the 8032 writes to the OMCAB register at address csiop+0x20, the low four bits of the byte will get loaded into the initial count. There is also a OMCAB mask register at csiop+0x22 that is used to prevent the 8032 from disturbing the other bits in the OMCAB register while writing.

If the PROPERTY statements above ended with MCELLBC instead of MCELLAB, then the other bank of eight output macrocells would be used for the counter. See the µPSD data sheet and PSDsoft Express User's Guide for more details.

The next declaration statements DCOUNT and INIT create a shorthand notation for use in the logic equations.

Now scroll down in the ABEL file to the logic equations until you see:

```
// Begin user preserved equations (not affected by iterations of DA usage)
==================

term_count = (DCOUNT == 0);  // term_count true when count reaches zero

when term_count then DCOUNT := INIT;  // automatically reload counter with initial
                                      // value after a count of zero is reached
```

```
else DCOUNT := DCOUNT - 1;           // specify down count action

// End user preserved equations (not affected by iterations of DA usage)
==================
```

These three statements define the down-counter and the PLD output that appears on pin PB4 (term_count).

So you can see that very little typing is needed to implement logic designs. The same approach is used to create state machines, shifters, etc.

Close the ABEL file and you will see the PSDsoft flow diagram again.

Additional µPSD Configuration

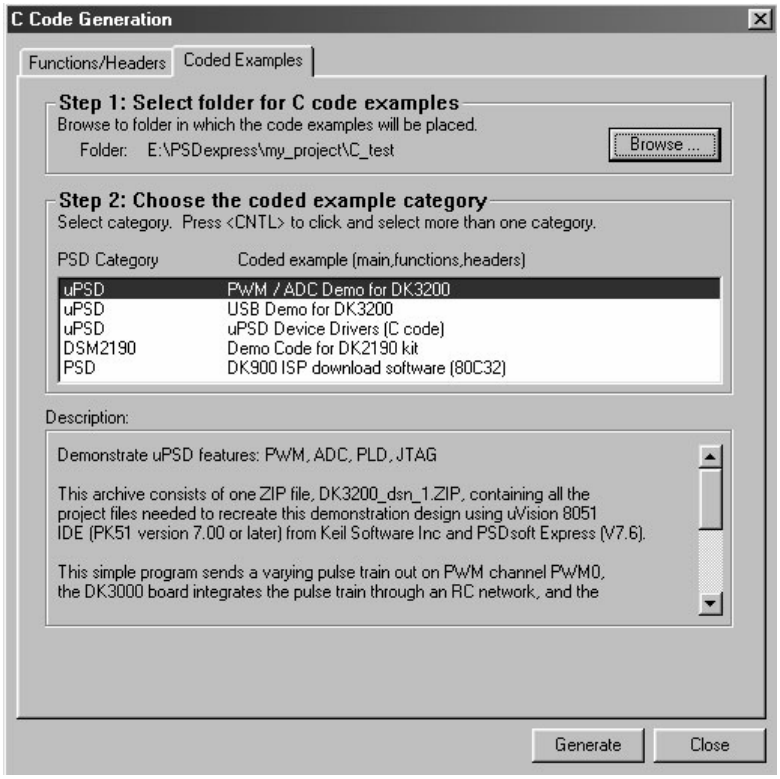
Click the box “Additional PSD Configuration”. This is where you can choose to set the security bit to prevent a device programmer from examining or copying the contents of the µPSD. The only way to defeat the security bit is to erase the entire µPSD, then it can be used again as a blank part. You can also click through the other sheets on this screen to set the JTAG USERCODE value and set sector protection on individual µPSD Non-Volatile memory segments. Just click “OK” for now.

C Code Generation

Click on the “Generate C Code” box in the main flow diagram.

The “Coded Example” section shown in Figure 15 will generate complete project files that you can use in the Keil uVision2 IDE.

Figure 15. Coded Example Generation



In this screen you can specify a folder in which the ZIPPED project files will be written, along with a readme file with instructions. The selection shown in Figure 15 is the complete project for this application note and

it runs on the DK3200 board. It contains all the Keil source and project files as well as all the PSDsoft Express project files. Now close the C Code Generation window.

Fitting Design

Click the next highlighted box in the design flow, “Fit Design to Silicon”. PSDsoft will compile all the configuration selections and present a report (also available in Appendix C). The fitter report documents how pins are configured and how the programmable logic is allocated. It also shows how many programmable logic product terms are used, which is needed to estimate power consumption.

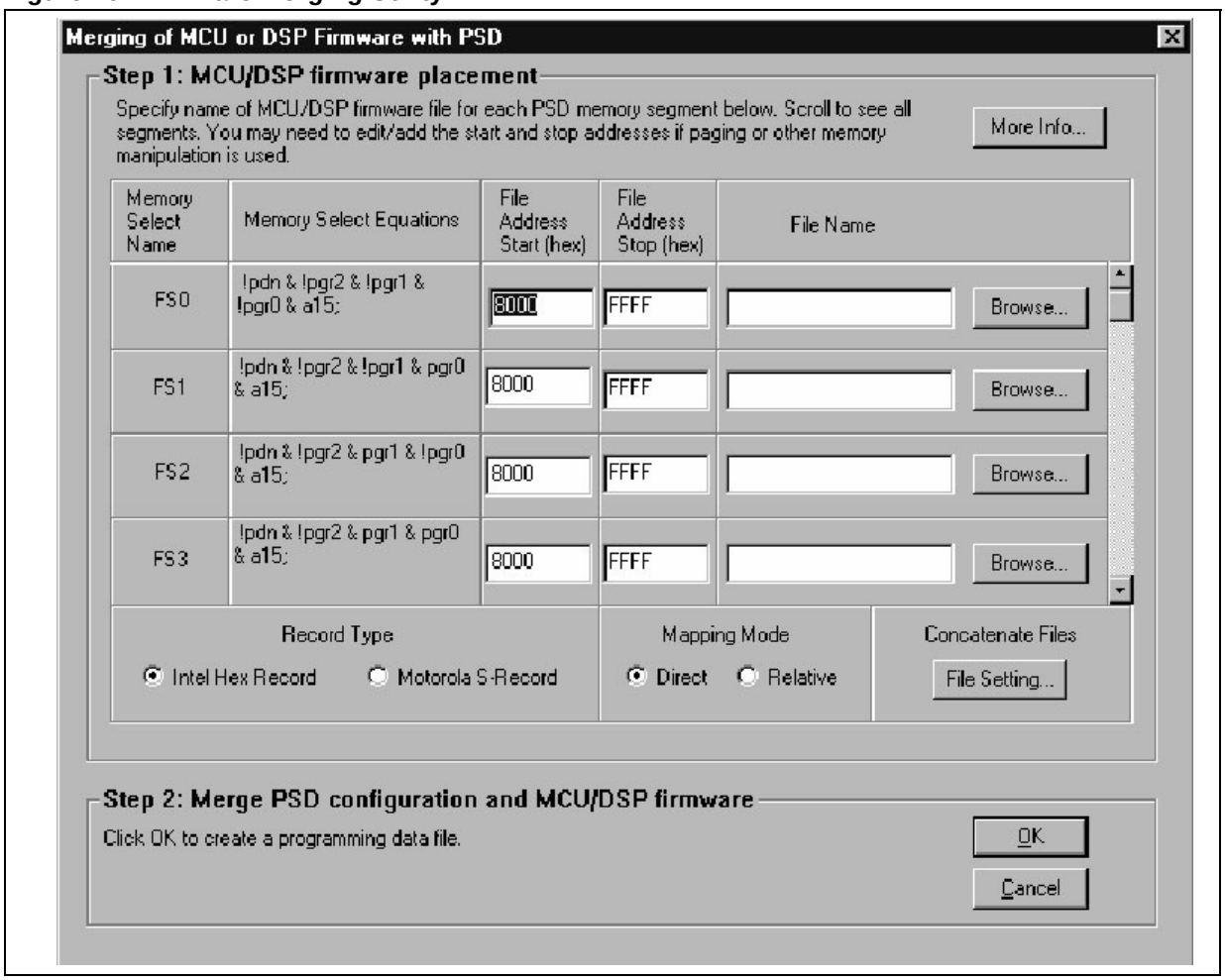
Merging 8032 Firmware with µPSD Configuration

Now that all µPSD pins and configuration settings have been defined, PSDsoft Express will create a single object file (*.obj) that is a composite of the 8032 firmware (*.hex) and the µPSD configuration. FlashLINK or third party programmer tools can use this object file to program a µPSD device. PSDsoft Express will create DK3200_1.obj for this design example.

During this merging process, PSDsoft Express will input firmware files from the 8032 compiler/linker in S-record or Intel HEX format. It will map the content of these files into the physical memory segments of the µPSD according to the choices that were made in the ‘Chip Select Equations’ screen. This mapping process translates the absolute system addresses inside 8032 firmware files into physical internal µPSD addresses that are used by a programmer device to program the µPSD. This address translation process is transparent. All you need to do is type (or browse) the file name that was generated from the 8032 linker into the appropriate boxes and PSDsoft Express does the rest. You can specify a single file name for more than one µPSD chip-select, or a different file name for each µPSD chip-select. It depends on how the 8032 linker has created the firmware file(s). For each µPSD chip-select in which you have specified a firmware file name, PSDsoft Express will extract firmware from that file only between the specified start and stop addresses, and ignore firmware outside of the start and stop addresses.

Click on 'Merge MCU Firmware' in the main flow diagram. You will see an information window pop up to remind you to be sure you have configured the firmware compiler and linker to support a paged memory mapping scheme. “OK” and you'll see this screen:

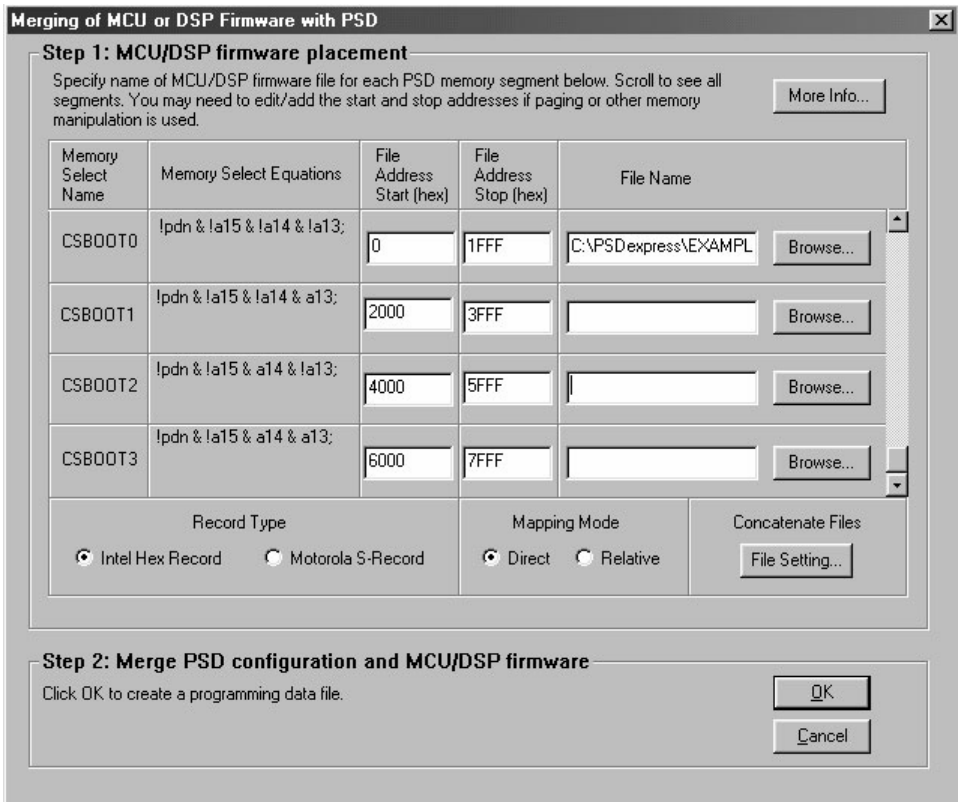
Figure 16. Firmware Merging Utility



In the left column are µPSD memory segment chip-selects (FS0, FS1, etc.). The next column shows the logic equations for selection of each µPSD memory segment. These equations reflect the choices that were made while defining µPSD internal chip-select equations in an earlier step. In the middle of the screen are hexadecimal start and stop addresses that PSDsoft Express has filled in based on the chip-select equations. On the right are fields to enter (browse) the 8032 firmware files.

Select 'Intel Hex Record' for 'Record Type' as shown. Now slide the bar on the right side all the way down to the bottom until you see CSBOOT0. Use the 'Browse' button and select the firmware file for CSBOOT0, \\PSDexpress\examples\DK3200_1.hex. This is a small example program that exercises the PWM and ADC channels of the µPSD on the DK3200 board, and this code fits completely within the 8K byte Flash memory segment CSBOOT0. The screen should look like this:

Figure 17. Merging the Example Firmware

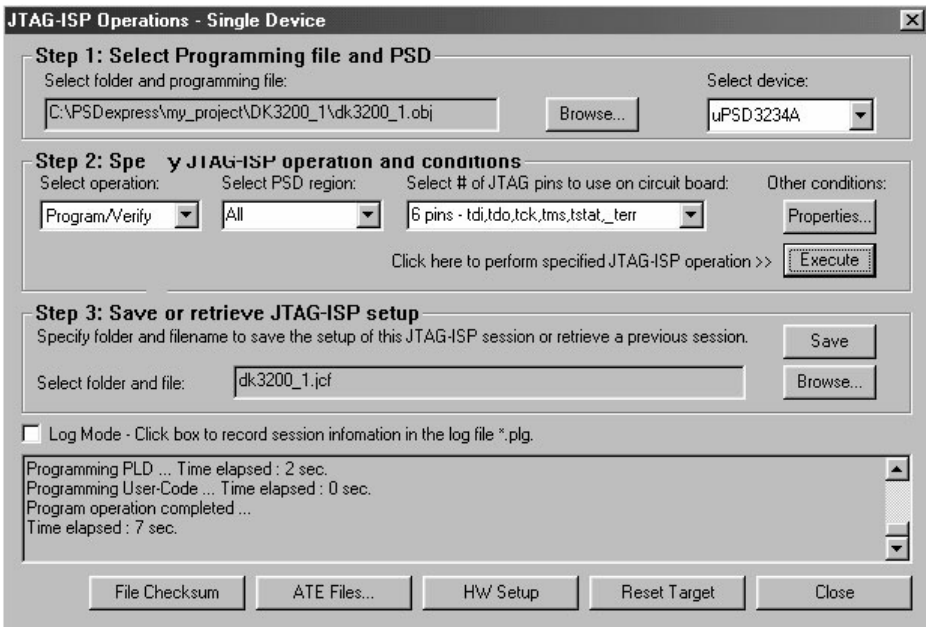


This specification places firmware in secondary PSD Flash memory segment csboot0. PSDsoft Express will extract any firmware that lies inside the file DK3200_1.hex between MCU addresses 0000 and 1FFF and place it in PSD memory segment csboot0. Click OK to generate the composite object file, DK3200_1.obj.

JTAG Programming

Now click the “STMicroelectronics JTAG/ISP” box to program the µPSD. You’ll be asked how many JTAG devices are on the target circuit board, choose “Only One”. You’ll see the screen shown in Figure 18.

Figure 18. Programming with FlashLINK JTAG Cable



This window enables you to perform JTAG-ISP operations and also offers a loop back test for your FlashLINK cable. If this is your first use, test your FlashLINK cable and PC parallel port by clicking the 'HW Setup' button, then click 'LoopTest' button and follow the directions.

Now let's define our JTAG-ISP environment. For this example project, PSDsoft Express should have filled in the folder and filename of the object file to program, the PSD device, and the JTAG-ISP operation, as shown in the screen above. For this design example, we have chosen to use all six JTAG-ISP pins (instead of four), so the screen should indicates 6-pin JTAG is being used.

To begin programming, connect the JTAG cable to the target system, power-up the target system, and click 'Execute' on the JTAG screen. The Log window at the bottom of the JTAG screen shows the progress. Programming should just take a few seconds, the ISP LED at D5 on the DK3200 will light during programming.

There are optional choices available when the 'Properties.." button is clicked. One choice includes setting the state of all pins on port A, B, C, or D during JTAG-ISP operations (make them inputs or outputs). The default state of these pins is "input", which is fine for this design example. The other choice allows you to specify a USERCODE value to compare before any JTAG-ISP operation starts. This is typically used in a manufacturing environment (see on-screen description for details).

After JTAG-ISP operations are complete, you can save the JTAG setup for this programming session to a file for later use. To do so, click on the 'Save' button. To restore the setup of a different previous session, click the 'Browse..' button.

WATCH IT RUN ON DK3200

After JTAG programming completes in just a few seconds, you should see a message appear on the LCD:

DK3200 for µPSD

PWM to ADC DEMO

Then you'll see the Hexadecimal value of the ADC conversion sweep up and down between 0x00 and 0xFF as the PWM pulse width changes. If you do not see the ADC value change, make sure there are two jumpers installed on the DK3200 board. On JP1, install one jumper across the two opposite rows of pins next to the word "PWM0", and the other jumper across the opposite rows of pins next to the word "ADC0".

Remove the jumper next the word “ADC0” and watch the ADC value on the LCD drop to 00h.

USING UVISION2 AND ISD51 DEBUGGER FROM KEIL SOFTWARE, INC.

This next section will briefly highlight the features of the Keil uVision2 IDE (Integrated Development Environment). Keil’s evaluation software was used for this example. This software is supplied on the DK3200 CD and can be installed by double clicking on ek501701.exe. Please refer to Keil documentation for more details.

Loading a Keil uVision2 Project

The file DK3200_dsn_1.ZIP is available which contains all the source and project files needed to build this design in Keil’s uVision2. To get this file, click on the “Generate C Code” box on the PSDsoft flow diagram. Then chose the “Coded Examples” tab and choose the selection for the DK3200 board. Specify a folder where you want the ZIP file written, and click “Generate”. The ZIP file contains two folders, DK3200_1_c and DK3200_1_p. DK3200_1_c has all the Keil files, DK3200_1_p has the PSDsoft Express project files for this application note.

Copy the folder DK3200_1_c and all of its contents to your Keil folders as follows, ..\Keil\C51\DK3200_1_c. Invoke Keil uVision2, pull down the “Project” menu, then select “Open Project”. Now open the uVision2 project that you just got from the ZIP file at ..\keil\c51\DK3200_1_C\DK3200_1.uv2. Everything should be ready to go.

Building the Project and Programming the µPSD

You can build the project for this application note which will create a new Intel HEX-80 file, DK3200_1.hex. Invoke PSDsoft Express and open the project DK3200_1, go to the “Merge MCU Firmware” section, select this new HEX file at \keil\c51\DK3200_1_C\\DK3200_1.hex for the Flash memory segment *csboot0* and merge, then program the DK3200 board with FlashLINK cable just as before. The LCD should display the PWM/ADC demo information.

Running the Keil ISD51 UART Debugger

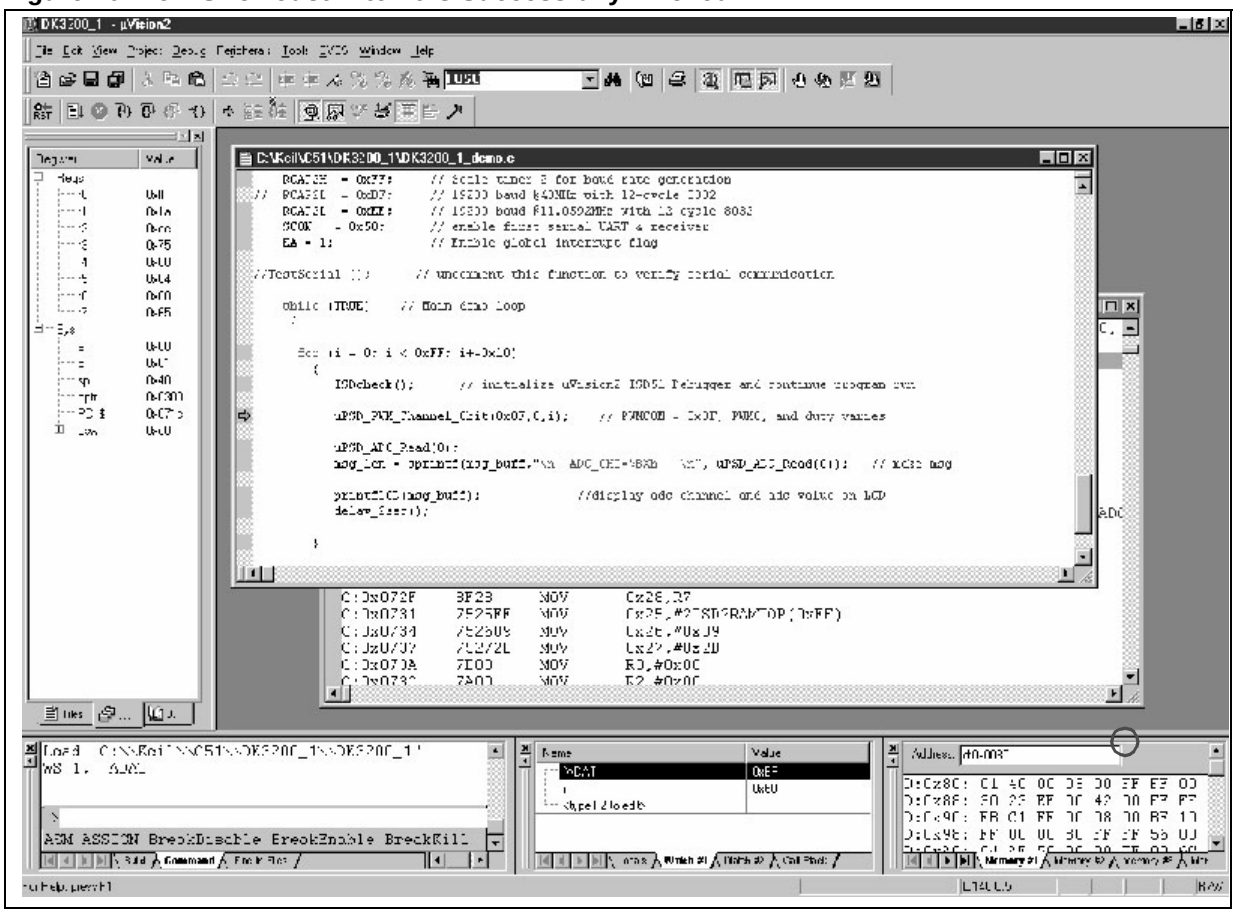
This simple demo program has the ISD51 UART debugger linked into the code. This is a new debug tool from Keil that only consumes 700 bytes of code space and is royalty-free so it can stay in your end product all the time. Unlike the older UART debugger, MON51, this debugger does not require you to debug code in small sections at a time from xdata SRAM. And unlike MON51, this debugger does not require you to combine your code and data space (tie _PSEN and _RD together). See Keil documentation for details.

This project, DK3200_1.Uv2 has ISD51 already selected for the debugger tool, connected through PC serial COM1 port at 19.2 kbaud with no hardware handshaking. It also assumes there is a 40MHz crystal on the DK3200 board. Connect a DB-9 (nine-pin) male-female straight-through (pins 2 and 3 are not swapped) serial cable to COM1 port1 on your PC and to the UART0 (P1) connector on the DK3200 board.

Click the Debug icon shown in Figure 19, the debugger will start and it will compare contents of the Flash memory in the µPSD Flash memory with the source files, then program execution will begin running to the C source line until just after the function, ISD_check(), then it will stop and wait for your debug command. The screen should look like Figure 20.

Figure 19. Debug icon

Figure 20. Keil ISD51 Just After it is Successfully Invoked



Now you can set breakpoints, single-step, view 8032 internal registers and SFRs, view blocks of memory, etc. For example, in the memory window in the lower-right corner of the IDE screen, the byte of memory at Hexadecimal address 0x96 is the SFR named “ADAT”, which is the resulting 8-bit value from the ADC channel from the last voltage conversion before the 8032 stopped. The value at address 0x96 is 0xBF in Figure 20. If you set a breakpoint on the function uPSD_ADC_Read(0), then run the program, you will see the data byte at address 0x96 change value in the memory watch window, and that same data byte will be showing on the LCD. Each time to run until the breakpoint, you should see a new value appearing in the memory watch window at address 0x96 and the same value on the LCD. Click the debug icon again to exit the debugger ISD51.

CONCLUSION

Congratulations! You have seen the majority of steps to implement a µPSD design on the DK3200 board. Now you have a basis to understand more detail as you read the µPSD data sheet and the documentation from Keil Software Inc.

APPENDIX A. PSDSOFT EXPRESS PROJECT SUMMARY FILE, DK3200_1.SUM

```
*************************************************************************
                        PSDsoft Express Version 7.51
                        Summary of Design Assistant
*************************************************************************
PROJECT    : DK3200_1                      DATE : 07/08/2002
DEVICE     : uPSD3234A                     TIME : 19:11:13
MCU/DSP    : uPSD3xxx
*************************************************************************

Initial setting for Program and Data Space:
===========================================

  Main PSD Flash memory will reside in this space at power-up:      Data Space Only
  Secondary PSD Flash memory will reside in this space at power-up: Program Space Only

Pin Definitions:
================

Pin          Signal                    Pin
Name         Name                      Type
------------ ------------------------- ------------
pa7          LCD_d7                    Peripheral I/O mode
pa6          LCD_d6                    Peripheral I/O mode
pa5          LCD_d5                    Peripheral I/O mode
pa4          LCD_d4                    Peripheral I/O mode
pa3          LCD_d3                    Peripheral I/O mode
pa2          LCD_d2                    Peripheral I/O mode
pa1          LCD_d1                    Peripheral I/O mode
pa0          LCD_d0                    Peripheral I/O mode
pb7          LCD_e                     External chip select - Active Hi
pb6          LCD_rw                    Combinatorial
pb5          LCD_rs                    Combinatorial
pb4          term_count                Combinatorial
pb3          a15_x                     Combinatorial
pb2          a14_x                     Combinatorial
pb1          a13_x                     Combinatorial
pb0          a12_x                     Combinatorial
pc6          tdo                       Dedicated JTAG - TDO
pc5          tdi                       Dedicated JTAG - TDI
pc4          _terr                     Dedicated JTAG - /TERR
pc3          tstat                     Dedicated JTAG - TSTAT
pc1          tck                       Dedicated JTAG - TCK
pc0          tms                       Dedicated JTAG - TMS
ale          ale                       ALE output
p4.7         p4_7                      GP I/O mode
p4.3         PWM0                      PWM0 Output
p3.1         USART1_Txd                USART1 Txd
p3.0         USART1_Rxd                USART1 Rxd
p1.4         ADC_Ch0                   ADC channel0 input
a11          a11                       Address line
a10          a10                       Address line
a9           a9                        Address line
a8           a8                        Address line
ad7          a7                        Data/Address line
ad6          a6                        Data/Address line
ad5          a5                        Data/Address line
ad4          a4                        Data/Address line
```

```
ad3          a3                      Data/Address line
ad2          a2                      Data/Address line
ad1          a1                      Data/Address line
ad0          a0                      Data/Address line
Reset_In     Reset_In                Reset In
Vref         VREF                    VREF input
_rd          _wr                     Bus control output
_psen        _psen                   Bus control output
_wr          _rd                     Bus control output
USB-         USB_minus               USB- bus
USB+         USB_plus                USB+ bus
Xtal1        Xtal1                   Xtal1
Xtal2        Xtal2                   Xtal2

User defined nodes:
===================

Node         Node
Name         Type
------------ ------------
down_count0  D-type register
down_count1  D-type register
down_count2  D-type register
down_count3  D-type register
init_count0  D-type register
init_count1  D-type register
init_count2  D-type register
init_count3  D-type register

Page Register settings:
=======================

pgr0 is used for paging
pgr1 is used for paging
pgr2 is used for paging
pgr3 is not used
pgr4 is not used
pgr5 is not used
pgr6 is not used
pgr7 is not used

Equations:
==========

rs0 = ((address >= ^h2000) & (address <= ^h3FFF));
csiop = ((address >= ^h0200) & (address <= ^h02FF));
fs0 = ((page == 0) & (address >= ^h8000) & (address <= ^hFFFF));
fs1 = ((page == 1) & (address >= ^h8000) & (address <= ^hFFFF));
fs2 = ((page == 2) & (address >= ^h8000) & (address <= ^hFFFF));
fs3 = ((page == 3) & (address >= ^h8000) & (address <= ^hFFFF));
fs4 = ((page == 4) & (address >= ^h8000) & (address <= ^hFFFF));
fs5 = ((page == 5) & (address >= ^h8000) & (address <= ^hFFFF));
fs6 = ((page == 6) & (address >= ^h8000) & (address <= ^hFFFF));
fs7 = ((page == 7) & (address >= ^h8000) & (address <= ^hFFFF));
csboot0 = ((address >= ^h0000) & (address <= ^h1FFF));
csboot1 = ((address >= ^h2000) & (address <= ^h3FFF));
csboot2 = ((address >= ^h4000) & (address <= ^h5FFF));
csboot3 = ((address >= ^h6000) & (address <= ^h7FFF));
psel0 = ((address >= ^h0300) & (address <= ^h03FF));
```

```
LCD_e = ((address >= ^h0300) & (address <= ^h03FF) & (!_rd))
     # ((address >= ^h0300) & (address <= ^h03FF) & (!_wr));
LCD_rw = a0;
LCD_rw.oe = Vcc;
LCD_rs = a1;
LCD_rs.oe = Vcc;
a15_x = a15;
a15_x.oe = Vcc;
a14_x = a14;
a14_x.oe = Vcc;
a13_x = a13;
a13_x.oe = Vcc;
a12_x = a12;
a12_x.oe = Vcc;
down_count0.ck = ale;
down_count0.re = !_reset;
down_count0.pr = Gnd;
down_count1.ck = ale;
down_count1.re = !_reset;
down_count1.pr = Gnd;
down_count2.ck = ale;
down_count2.re = !_reset;
down_count2.pr = Gnd;
down_count3.ck = ale;
down_count3.re = !_reset;
down_count3.pr = Gnd;
init_count0.ck = Gnd;
init_count0.re = !_reset;
init_count0.pr = Gnd ;
init_count1.ck = Gnd;
init_count1.re = !_reset;
init_count1.pr = Gnd;
init_count2.ck = Gnd;
init_count2.re = !_reset;
init_count2.pr = Gnd;
init_count3.ck = Gnd;
init_count3.re = !_reset;
init_count3.pr = Gnd;
```

APPENDIX B. PSDSOFT EXPRESS ABEL HDL FILE DK3200_1.ABL

```
module DK3200_1
LCD_d7 PIN 21;  "Reserved for Peripheral I/O mode
LCD_d6 PIN 22;  "Reserved for Peripheral I/O mode
LCD_d5 PIN 24;  "Reserved for Peripheral I/O mode
LCD_d4 PIN 26;  "Reserved for Peripheral I/O mode
LCD_d3 PIN 28;  "Reserved for Peripheral I/O mode
LCD_d2 PIN 32;  "Reserved for Peripheral I/O mode
LCD_d1 PIN 34;  "Reserved for Peripheral I/O mode
LCD_d0 PIN 35;  "Reserved for Peripheral I/O mode
LCD_e PIN 66;
LCD_rw PIN 67;
LCD_rs PIN 72;
term_count PIN 73;
a15_x PIN 74;
a14_x PIN 76;
a13_x PIN 78;
a12_x PIN 80;
tdo PIN 6; "TDO
tdi PIN 7; "TDI
_terr PIN 9; "/TERR
tstat PIN 14; "TSTAT
tck PIN 16; "TCK
tms PIN 20;  "TMS
ale PIN 4; "ALE output
p4_7 PIN 18; "GP I/O
PWM0 PIN 27; "PWM0 Output
USART1_Txd PIN 77; "USART1 Txd
USART1_Rxd PIN 75; "USART1 Rxd
ADC_Ch0 PIN 59; "ADC channel0 input
a11 PIN 57; "Address line
a10 PIN 55; "Address line
a9 PIN 53; "Address line
a8 PIN 51; "Address line
a7 PIN 47;  "Data/address bus line
a6 PIN 45;  "Data/address bus line
a5 PIN 43;  "Data/address bus line
a4 PIN 41;  "Data/address bus line
a3 PIN 39;  "Data/address bus line
a2 PIN 38;  "Data/address bus line
a1 PIN 37;  "Data/address bus line
a0 PIN 36;  "Data/address bus line
Reset_In PIN 68;
VREF PIN 70; "VREF input
_wr PIN 62;
_psen PIN 63;
_rd PIN 65;
USB_minus PIN 8; "USB- bus
USB_plus PIN 10; "USB+ bus
Xtal1 PIN 48; "Xtal1
Xtal2 PIN 49; "Xtal2
psel0 node;
rs0 node;
csiop node;
fs0 node;
fs1 node;
fs2 node;
fs3 node;
```

```
fs4 node;
fs5 node;
fs6 node;
fs7 node;
csboot0 node;
csboot1 node;
csboot2 node;
csboot3 node;
_reset node 543;
a12 node 512;
a13 node 513;
a14 node 514;
a15 node 515;
pgr2..pgr0 node;
down_count0 NODE istype 'reg_D';
down_count1 NODE istype 'reg_D';
down_count2 NODE istype 'reg_D';
down_count3 NODE istype 'reg_D';
init_count0 NODE istype 'reg_D';
init_count1 NODE istype 'reg_D';
init_count2 NODE istype 'reg_D';
init_count3 NODE istype 'reg_D';

X = .x.;
address = [a15..a0];
page = [pgr2..pgr0];
Vcc = 1;
Gnd = 0;

// Begin user preserved declarations (not affected by iterations of DA usage)
==================

WSIPSD PROPERTY 'DataBus_OMC D[7:4]:down_count[3:0] MCELLAB';
          // This statement forces the alignment
                          // of down_count bits [3..0] to the MCU data bus bit positions
[7..4].
                        // If this WSIPSD PROPERTY statement was not present, then PSDsoft
                      // would pick random MCU bit positions. The WSIPSD PROPERTY is needed
                         // only if the MCU will read or write to MicroCells and only if a
                           // particular MCU data bus position is required by the designer.

WSIPSD PROPERTY 'DataBus_OMC D[3:0]:init_count[3:0] MCELLAB';
// This statement forces the alignment
                     // of init_count bits [3..0] to the MCU data bus bit positions [3..0].

DCOUNT = [down_count3..down_count0];  // 4-bit down counter
INIT = [init_count3..init_count0];// 4-bit initial count from MCU
//INIT = [0,1,0,0];

// End user preserved declarations (not affected by iterations of DA usage)
==================


equations

rs0 = ((address >= ^h2000) & (address <= ^h3FFF));
csiop = ((address >= ^h0200) & (address <= ^h02FF));
fs0 = ((page == 0) & (address >= ^h8000) & (address <= ^hFFFF));
fs1 = ((page == 1) & (address >= ^h8000) & (address <= ^hFFFF));
```

```
fs2 = ((page == 2) & (address >= ^h8000) & (address <= ^hFFFF));
fs3 = ((page == 3) & (address >= ^h8000) & (address <= ^hFFFF));
fs4 = ((page == 4) & (address >= ^h8000) & (address <= ^hFFFF));
fs5 = ((page == 5) & (address >= ^h8000) & (address <= ^hFFFF));
fs6 = ((page == 6) & (address >= ^h8000) & (address <= ^hFFFF));
fs7 = ((page == 7) & (address >= ^h8000) & (address <= ^hFFFF));
csboot0 = ((address >= ^h0000) & (address <= ^h1FFF));
csboot1 = ((address >= ^h2000) & (address <= ^h3FFF));
csboot2 = ((address >= ^h4000) & (address <= ^h5FFF));
csboot3 = ((address >= ^h6000) & (address <= ^h7FFF));
psel0 = ((address >= ^h0300) & (address <= ^h03FF));
LCD_e = ((address >= ^h0300) & (address <= ^h03FF) & (!_rd))
     # ((address >= ^h0300) & (address <= ^h03FF) & (!_wr));
LCD_rw = a0;
LCD_rw.oe = Vcc;
LCD_rs = a1;
LCD_rs.oe = Vcc;
a15_x = a15;
a15_x.oe = Vcc;
a14_x = a14;
a14_x.oe = Vcc;
a13_x = a13;
a13_x.oe = Vcc;
a12_x = a12;
a12_x.oe = Vcc;
down_count0.ck = ale;
down_count0.re = !_reset;
down_count0.pr = Gnd;
down_count1.ck = ale;
down_count1.re = !_reset;
down_count1.pr = Gnd;
down_count2.ck = ale;
down_count2.re = !_reset;
down_count2.pr = Gnd;
down_count3.ck = ale;
down_count3.re = !_reset;
down_count3.pr = Gnd;
init_count0.ck = Gnd;
init_count0.re = !_reset;
init_count0.pr = Gnd ;
init_count1.ck = Gnd;
init_count1.re = !_reset;
init_count1.pr = Gnd;
init_count2.ck = Gnd;
init_count2.re = !_reset;
init_count2.pr = Gnd;
init_count3.ck = Gnd;
init_count3.re = !_reset;
init_count3.pr = Gnd;


// Begin user preserved equations (not affected by iterations of DA usage)
==================

term_count = (DCOUNT == 0);  // term_count true when count reaches zero

when term_count then DCOUNT := INIT;  // automatically reload counter with initial
                                      // value after a count of zero is reached
```

```
else DCOUNT := DCOUNT - 1;           // specify down count action

// End user preserved equations (not affected by iterations of DA usage)
==================

end DK3200_1
```

APPENDIX C. PSDSOFT EXPRESS FITTER REPORT DK3200_1.FRP

```
*******************************************************************************
                              PSDsoft Express Version 7.51
                                  Output of PSD Fitter
*******************************************************************************
  PROJECT    : DK3200_1                          DATE : 07/08/2002
  DEVICE     : uPSD3234A                         TIME : 17:55:01
  FIT OPTION : Keep Current
  DESCRIPTION: Example design for uPSD3234A in Application Note AN1560.
               Simple memory map with 32K secondary Flash memory in code space, and
               256K main Flash memory paged in data space. Down-Counter built in
               PLD.
*******************************************************************************
     ==== Pin Layout for U (80-Pin TQFP) Package Type ====
               -----------------------------
              |                             |
              |1 ] pd2            adio4 [41| Address Bus A4/Data Port D4, a4
              |2 ] p3_3            p3_5 [42|
              |3 ] pd1            adio5 [43| Address Bus A5/Data Port D5, a5
          ale |4 ] pd0            p3_6 [44|
              |5 ] pc7            adio6 [45| Address Bus A6/Data Port D6, a6
    tdo, TDO |6 ] pc6/TDO         p3_7 [46|
    tdi, TDI |7 ] pc5/TDI         adio7 [47| Address Bus A7/Data Port D7, a7
   USB_minus |8 ] USBm            Xtal1 [48| Xtal1
 _terr, TERR |9 ] pc4/TERR        Xtal2 [49| Xtal2
    USB_plus |10] USBp              VCC [50|
              |11] N/C            adio8 [51| Address Bus A8, a8
              |12] VCC             p1_0 [52|
              |13] GND            adio9 [53| Address Bus A9, a9
tstat, TSTAT |14] pc3/TSTAT        p1_1 [54|
              |15] pc2           adio10 [55| Address Bus A10, a10
    tck, TCK |16] pc1/TCK          p1_2 [56|
              |17] N/C           adio11 [57| Address Bus A11, a11
        p4_7 |18] p4_7            p1_3 [58|
              |19] p4_6            p1_4 [59| ADC_Ch0
    tms, TMS |20] pc0/TMS          p1_5 [60| LCD_d7 ,Peripheral I/O Mode
              |21]                 p1_6 [61| LCD_d6 ,Peripheral I/O Mode
              |22] pa6            cntl1 [62| _wr
              |23] p4_5           cntl2 [63| _psen, LCD_d5 ,Peripheral I/O Mode
              |24] pa5             p1_7 [64|
              |25] p4_4           cntl0 [65| _rd, LCD_d4 ,Peripheral I/O Mode
              |26] pa4              pb7 [66| LCD_e
        PWM0 |27] p4_3              pb6 [67| LCD_rw, LCD_d3 ,Peripheral I/O Mode
              |28] pa3          Reset_In [68| Reset_In
              |29] GND               GND [69|
              |30] p4_2              Vref [70| VREF
              |31] p4_1               N/C [71| LCD_d2 ,Peripheral I/O Mode
              |32] pa2               pb5 [72| LCD_rs
              |33] p4_0               pb4 [73| term_count, LCD_d1 ,Peripheral I/O Mode
              |34] pa1               pb3 [74| a15_x, LCD_d0 ,Peripheral I/O Mode
              |35] pa0              p3_0 [75| USART1_Rxd, a0, Address Bus A0/Data Port D0
              |36] adio0             pb2 [76| a14_x, a1, Address Bus A1/Data Port D1
              |37] adio1            p3_1 [77| USART1_Txd, a2, Address Bus A2/Data Port D2
              |38] adio2             pb1 [78| a13_x, a3, Address Bus A3/Data Port D3
              |39] adio3            p3_2 [79|
              |40] p4_4               pb0 [80| a12_x
              |                             |
               -----------------------------
```

```
==== Global Configuration ====

Data Bus                                                              : 8-Bit
Address/Data Mode                                                     : Multiplexed
ALE/AS Signal                                                         : Active High
Control Signals                                                       : /WR, /RD, /PSEN
Main PSD Flash memory will reside in this space at power-up           : Data space
Secondary PSD Flash memory will reside in this space at power-up      : Program space
Enable Chip-Select Input(/CSI)                                        : OFF
Standby Voltage Input (PC2)                                           : OFF
Standby-on Indicator (PC4)                                            : OFF
RDY/Busy function (PC3)                                               : OFF
Load Micro-Cell on                                                    : edge
Security Protection                                                   : OFF

     ==== DataBus_IMC access information ====

                 CSIOP
Location     Address Offset     Register Name           Signals
-----------------------------------------------------------

     ===== Resource Usage Summary =====

Total Product Terms Used:  71

Device Resources                    used / total
-------------------------------------------------
Port A: (pins 35 34 32 28 26 24 22 21)
I/O Pins :                            8   /  8
   GP I/O or Address Out        :    0
   Peripheral I/O               :    8
   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                   :    8   /  8
     Micro-Cells AB :
      Buried Microcells         :    8
      Output Microcells         :    0
   Product Terms                :    15  /  24
   Control Product Terms        :    24  /  34

Port B: (pins 80 78 76 74 73 72 67 66)
I/O Pins :                            8   /  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        :    8
   Registered Outputs           :    0
Other Information
   Microcells                   :    8   /  8
     Micro-Cells AB :
      Buried Microcells         :    0
```

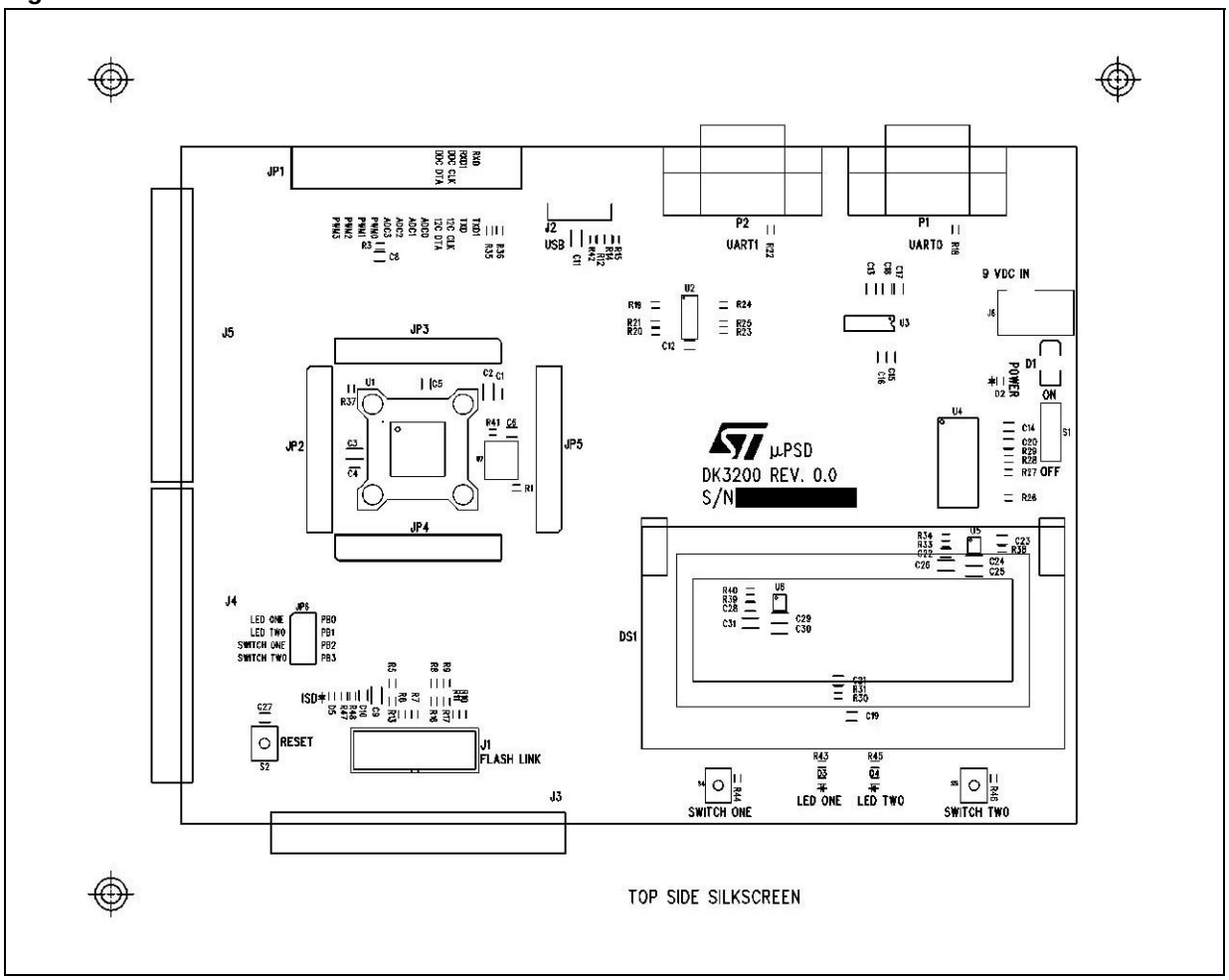
```
      Output Microcells         :    0
     Micro-Cells BC :
      Buried Microcells         :    0
      Output Microcells         :    8
   Product Terms                :    9   /  32
   Control Product Terms        :    8   /  34

Port C: (pins 20 16 15 14 9 7 6 5)
I/O Pins :                           6   /  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                 :    6
   Standby Voltage Input        :    0
   Rdy/Bsy signal               :    0
   Standby On Indicator         :    0
   Combinatorial Outputs        :    0
   Registered Outputs           :    0
Other Information
   Microcells                   :    8   /  8
     Micro-Cells BC :
      Buried Microcells         :    8
      Output Microcells         :    0
   Product Terms                :    9   /  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-Cells 0        -            init_count0 => Register
  Micro-Cells 1        -            init_count1 => Register
  Micro-Cells 2        -            init_count2 => Register
  Micro-Cells 3        -            init_count3 => Register
  Micro-Cells 4        -            down_count0 => Register
  Micro-Cells 5        -            down_count1 => Register
  Micro-Cells 6        -            down_count2 => Register
  Micro-Cells 7        -            down_count3 => Register

Micro-Cell BC :
  Micro-Cells 0        -            a12_x (mcellbc0)  => Combinatorial
  Micro-Cells 1        -            a13_x (mcellbc1)  => Combinatorial
```

```
  Micro-Cells 2        -            a14_x (mcellbc2)  => Combinatorial
  Micro-Cells 3        -            a15_x (mcellbc3)  => Combinatorial
  Micro-Cells 4        -            term_count (mcellbc4)  => Combinatorial
  Micro-Cells 5        -            LCD_rs (mcellbc5)  => Combinatorial
  Micro-Cells 6        -            LCD_rw (mcellbc6)  => Combinatorial
  Micro-Cells 7        -            LCD_e (mcellbc7)  => Combinatorial

External Chip Select :

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

DPLD          EQUATIONS :
=======================
fs0 = !pdn & !pgr2 & !pgr1 & !pgr0 & 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;
psel0 = !pdn & !a15 & !a14 & !a13 & !a12 & !a11 & !a10 & a9 & a8;

PORTA         EQUATIONS :
=======================
     init_count0.D := 0;
     init_count0.PR = 0;
     init_count0.RE = !_reset;
     init_count0.C = 0;

     init_count1.D := 0;
     init_count1.PR = 0;
     init_count1.RE = !_reset;
     init_count1.C = 0;

     init_count2.D := 0;
     init_count2.PR = 0;
     init_count2.RE = !_reset;
     init_count2.C = 0;

     init_count3.D := 0;
     init_count3.PR = 0;
     init_count3.RE = !_reset;
     init_count3.C = 0;

     down_count0.D := (!down_count0.Q & !term_count.PIN)
          # (init_count0 & term_count.PIN);
     down_count0.PR = 0;
     down_count0.RE = !_reset;
     down_count0.C = ale;

     down_count1.D := (down_count1.Q & down_count0.Q & !term_count.PIN)
          # (!down_count1.Q & !down_count0.Q & !term_count.PIN)
```

```
          # (init_count1 & term_count.PIN);
     down_count1.PR = 0;
     down_count1.RE = !_reset;
     down_count1.C = ale;

     down_count2.T := (!down_count1.Q & !down_count0.Q & !term_count.PIN)
          # (!down_count2.Q & init_count2 & term_count.PIN)
          # (down_count2.Q & !init_count2 & term_count.PIN);
     down_count2.PR = 0;
     down_count2.RE = !_reset;
     down_count2.C = ale;

     down_count3.T := (!down_count3.Q & init_count3 & term_count.PIN)
          # (down_count3.Q & !init_count3 & term_count.PIN)
          # (!down_count2.Q & !down_count1.Q & !down_count0.Q & !term_count.PIN);
     down_count3.PR = 0;
     down_count3.RE = !_reset;
     down_count3.C = ale;

PORTB         EQUATIONS :
=======================
     a12_x = a12;
     a12_x.OE = 1;

     a13_x = a13;
     a13_x.OE = 1;

     a14_x = a14;
     a14_x.OE = 1;

     a15_x = a15;
     a15_x.OE = 1;

     term_count = !down_count3.Q & !down_count2.Q & !down_count1.Q & !down_count0.Q;
     term_count.OE = 1;
     term_count.LE = 1;

     LCD_rs = a1;
     LCD_rs.OE = 1;

     LCD_rw = a0;
     LCD_rw.OE = 1;

     LCD_e = (!_rd & !a15 & !a14 & !a13 & !a12 & !a11 & !a10 & a9 & a8)
          # (!_wr & !a15 & !a14 & !a13 & !a12 & !a11 & !a10 & a9 & a8);
     LCD_e.OE = 1;

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

APPENDIX D. DK3200 BOARD LAYOUT

Figure 21. DK3200 BOARD LAYOUT



APPENDIX E. DK3200 SCHEMATICS

Figure 22. DK3200 SCHEMATICS (1)

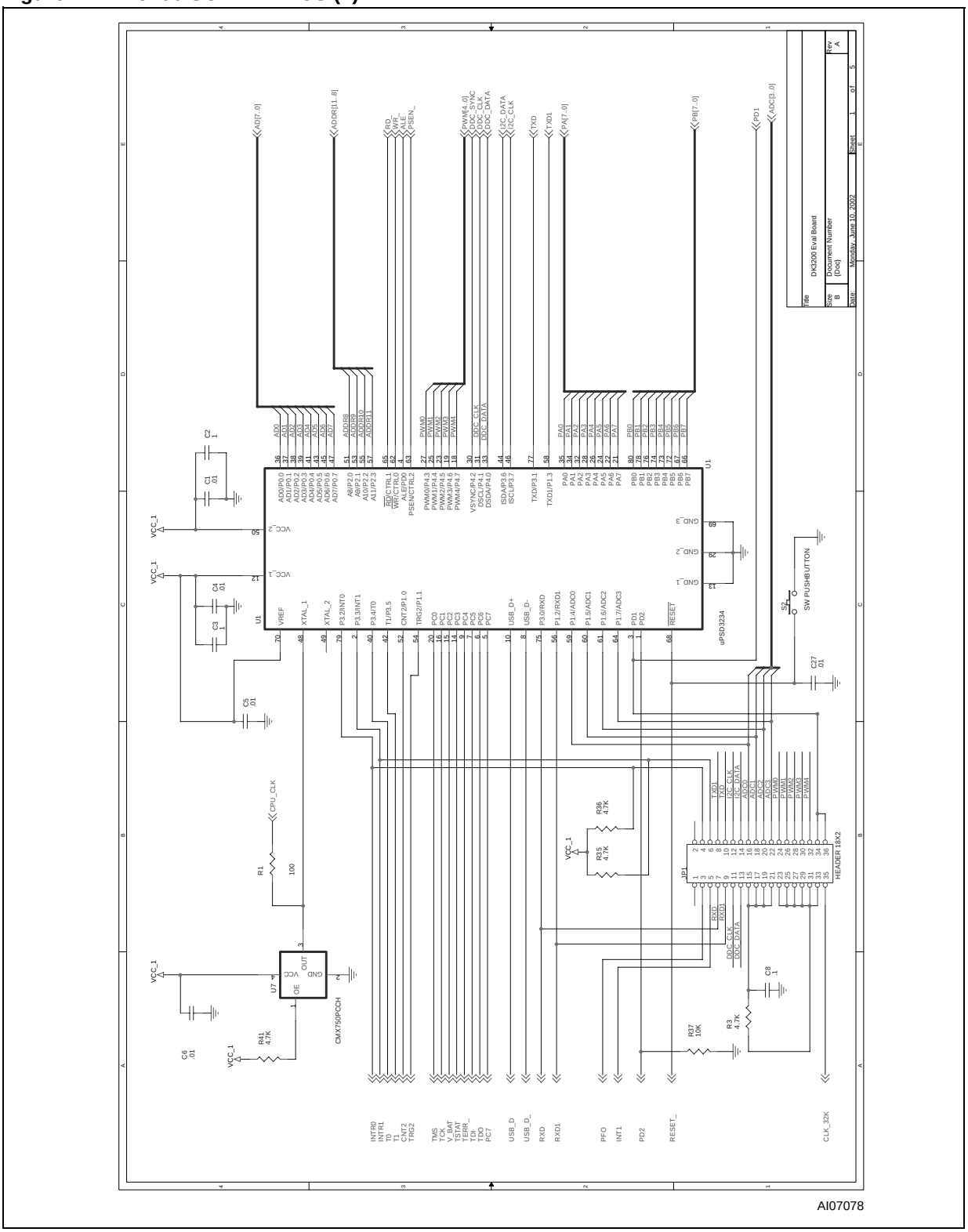


Figure 23. DK3200 SCHEMATICS (2)

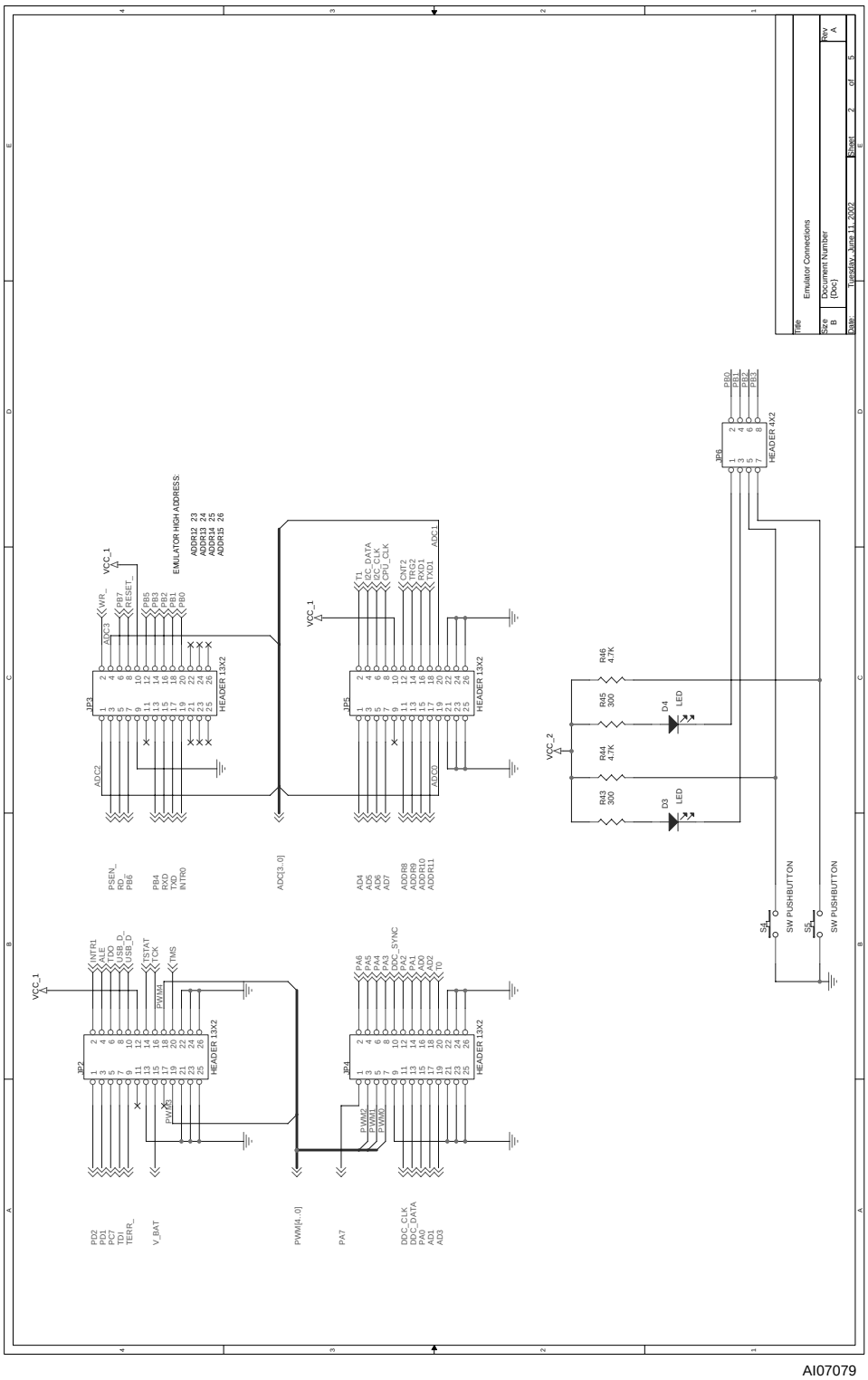


Figure 24. DK3200 SCHEMATICS (3)

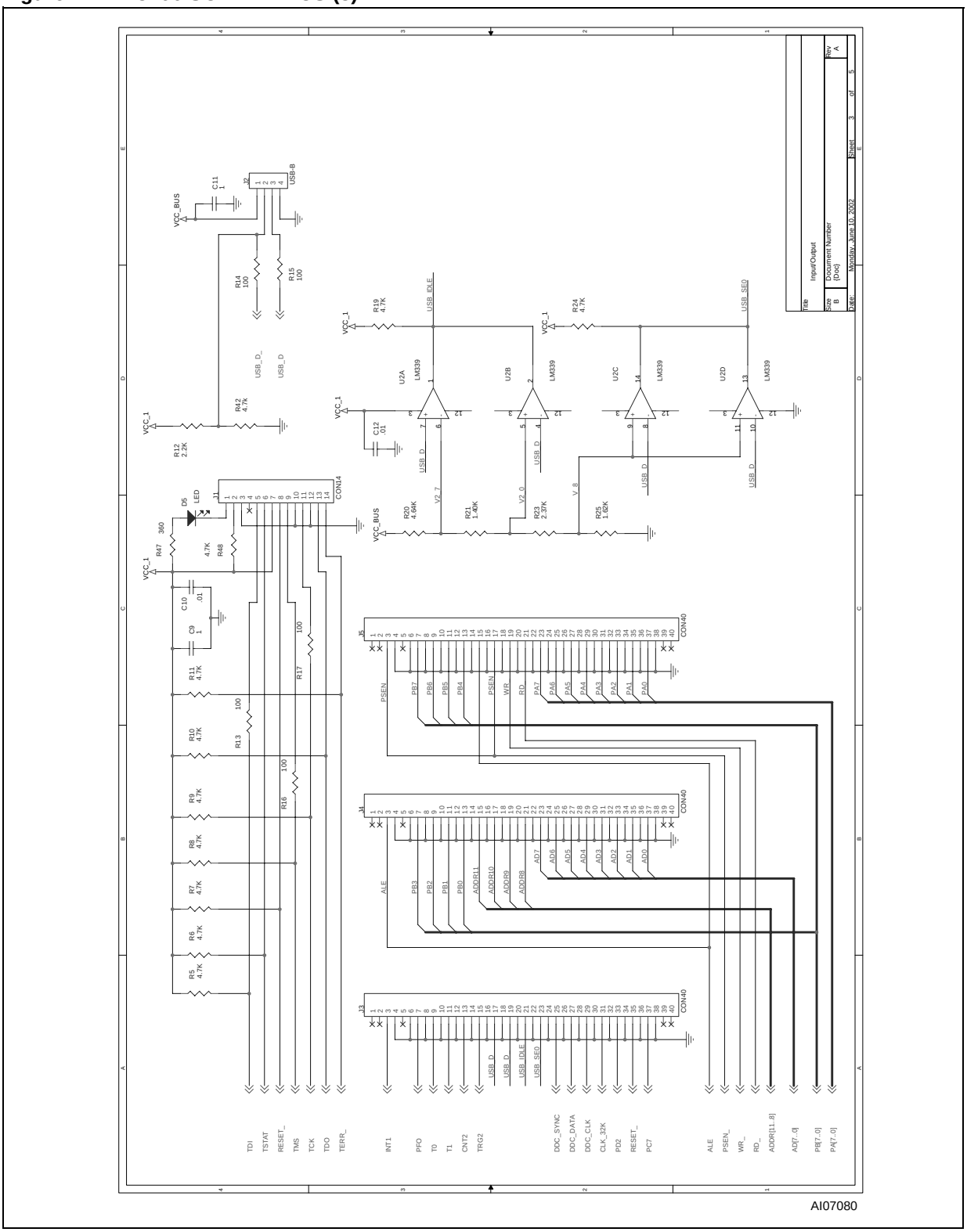


Figure 25. DK3200 SCHEMATICS (4)

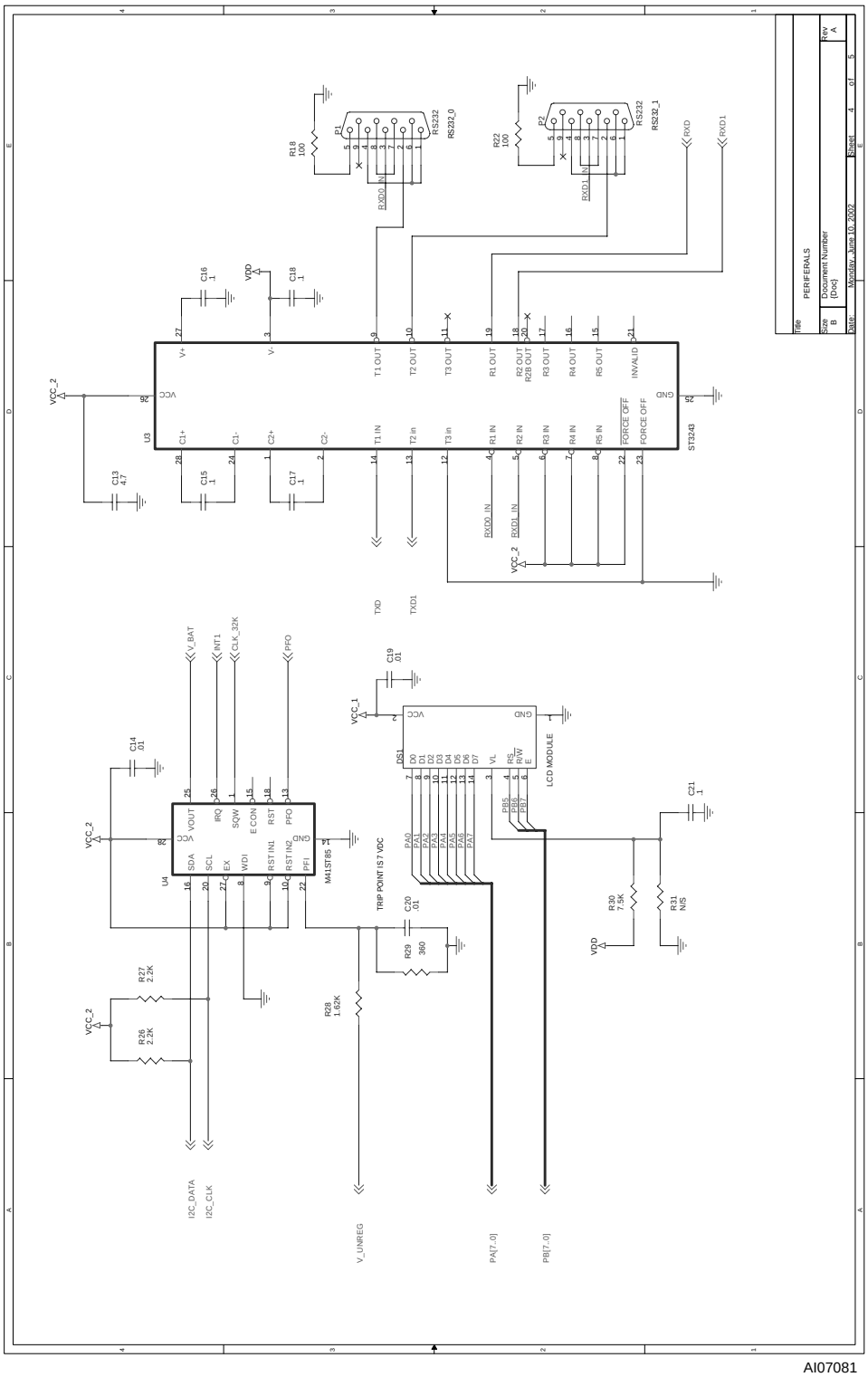


Figure 26. DK3200 SCHEMATICS (5)

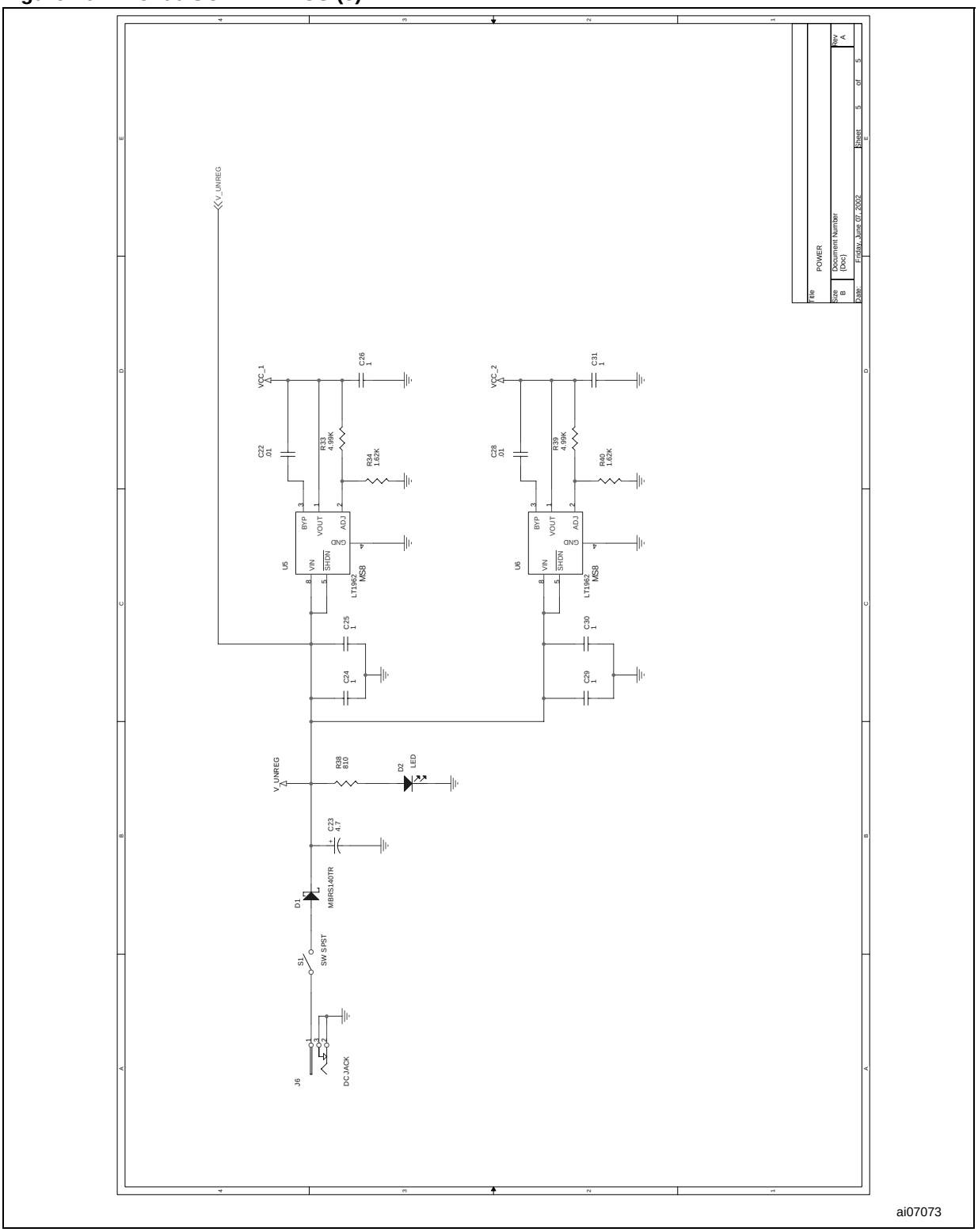


Table 1. Document Revision History

Date	Rev.	Description of Revision
29-Jul-2002	1.0	Document written
26-Aug-2002	1.1	Document updated: DK3200 replaces DK3000 development tool, Figures 1, 2, 3, 5, 16, 19, 20, 23 and 24 modified, Screen captures enlarged, Figure 15 (in previous document) removed together with related paragraph. Figure numbering shifted by 1 from Figure 15 on. Details added to paging bit description.

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

If you have specific technical inquiries, questions, or suggestions concerning matters raised in this document, please click on “Technical Support” on the *www.st.com/psm* web page.