

# **Using the LPC32XX Chip Support Package (CSP) and the Phytex 3250 Board Support Package (BSP)**

# Table of contents

<b>1</b>	<b>Introduction.....</b>	<b>3</b>
1.1	Installing the BSP.....	3
1.2	Selecting a tool chain to use.....	3
1.2.1	IAR toolchain path modifications.....	4
1.2.2	IAR and Keil include path additions.....	4
1.2.3	Pre-built IDE based projects.....	4
1.3	IMPORTANT INFORMATION.....	4
<b>2</b>	<b>BSP software .....</b>	<b>5</b>
2.1	Supported LPC32XX drivers.....	5
2.2	Example code .....	5
2.2.1	Specific toolchain examples .....	5
2.3	Startup code and examples.....	5
2.3.1	Kickstart loader.....	5
2.3.2	Stage 1 loader (S1L).....	6
2.4	Other applications.....	6
2.4.1	LPC3250 serial loader (LSL).....	6
<b>3</b>	<b>Building and deploying an application.....</b>	<b>6</b>
3.1	Application types.....	6
3.1.1	Applications using the stage 1 loader .....	6
3.1.2	Applications replacing the stage 1 loader .....	7
3.2	Application deployment methods .....	7
3.2.1	Using S1L.....	7
3.2.2	LPC3250 boot methods .....	7
<b>4</b>	<b>Building and deploying applications .....</b>	<b>7</b>
4.1	Building an application for use with S1L.....	8
4.1.1	Building the application.....	8
4.1.2	Deploying the application.....	10
<b>5</b>	<b>Miscellaneous .....</b>	<b>16</b>
5.1	Where to get software.....	16
5.2	Startup code, S1L, boot loaders, and deployment.....	16
5.3	Using the LPC3250 serial loader (LSL) tool .....	16
5.4	Issues.....	16
5.4.1	IAR toolchain .....	16

## 1 Introduction

The PHY3250 Board Support Package (BSP) provides a starting point for applications and software based on the LPC3250 MCU and the Phytex 3250 board. The BSP is based on the LPC32xx Chip Support Package (CSP) and includes examples on how to use the CSP drivers to control LPC32XX MCU and PHY3250 board functions. Sample startup code and applications are also provided that can be used as a basis for developing other applications for the board.

The examples included with the BSP show how to use the LPC32XX CSP drivers in an example application. The examples can be loaded and executed through the Stage 1 loader included on the PHY3250 board or through a debugger. Examples can also be binded to the startup code to make self-executing examples that are booted when the board is powered up.

This document explains how to build the examples and startup code and how to load and execute the examples with the Stage 1 loader (S1L).

### 1.1 *Installing the BSP*

To install the BSP, simply unzip the BSP in a work area on your storage device. It is recommended that the BSP be installed at the C:/NXPMCU directory to prevent possible problems with projects that require absolute path names.

After installation, the tree will look similar to the following:

```
C:/NXPMCU
C:/NXPMCU/SOFTWARE
C:/NXPMCU/SOFTWARE/MAKERULE
C:/NXPMCU/SOFTWARE/CSPS
C:/NXPMCU/SOFTWARE/TOOLS
C:/NXPMCU/SOFTWARE/CSPS/<CHIP>
...
...
...
...
```

### 1.2 *Selecting a tool chain to use*

To build code and examples from this BSP, a tool chain is needed. The tool chain consists of compilers, linkers, and other build tools necessary to make the libraries and executables from the source code. The PHY3250 BSP supports GNU (CodeSourcery), Keil (uVision4), and ARM Realview 3.1. Support for the IAR Embedded Workbench is only available in the examples.

At least 1 of these toolchains must be installed to build the CSP and BSP code. If a toolchain isn't currently available, a lite version of the GNU tools can be downloaded and used for free from [www.codesourcery.com](http://www.codesourcery.com).

### 1.2.1 IAR toolchain path modifications

If IAR Embedded Workbench is used to develop the applications and the make based build system is going to be used, then the paths to the IAR tool executables location must be added to the PATH environment variable. This can be done by editing the setenv.bat file, adding a small batch file to add the path when a CMD shell is opened, or by changing the default user PATH value in Windows.

### 1.2.2 IAR and Keil include path additions

If using the Keil or IAR toolchains, locate the setenv.bat file and update the file path environment variables specified in that file. As of the current release, the environment variables are shown below with their default values.

```
SET KEIL_BIN_BASE=C:\Keil\ARM\BIN40
SET KEIL_RVCT=C:\Keil\ARM\RV40
SET IAR_ROOT=C:\Program Files\IAR Systems\Embedded Workbench 5.0 Evaluation
```

### 1.2.3 Pre-built IDE based projects

There are several pre-built IDE based projects for IAR Embedded Workbench, Keil uVision4, and ARM Realview 3.1. These projects are found in some of the driver and startup examples. If you have the correct tool installed, the project can be started by clicking on the project icon.

*The pre-built IDE projects may need to be tweaked for the setup you are running on. For example, if you are using a different hardware debugger, the debugger options will need to be changed. The project assumes the BSP is installed in the /NXPMCU/SOFTWARE area. If it is installed elsewhere, you may need to change the paths in the project.*

## 1.3 IMPORTANT INFORMATION

Prior to build any code with this BSP, the 3 system revision defines need to be setup to their correct values. Locate the phy3250\_board.h file and set the following defines to the matching values indicated on the LCD module, CPU mode, and carrier board of your Phytex 3250 system. If these defines are set to the wrong values, the software may work correctly for your board.

```
/* Carrier board revision selection - The carrier board revision is a
   value of 1305.x, where x = 0 to 3 */
#define PHY3250_CARRIERBOARD_1305_X 0

/* Module board revision selection - The module board revision is a
   value of 1304.x, where x = 0 to 1 */
#define PHY3250_MODULEBOARD_1304_X 0
```

```
/* LCD revision selection - The LCD revision is a value of 1307.x,  
   where x = 0 to 1 */  
#define PHY3250_LCD_1307_X 0
```

## 2 BSP software

This BSP includes examples, startup code, and driver files applicable to the PHY3250 board and drivers files specific to the LPC32XX MCU. Examples that show how to use the CSP drivers provide a good starting point to understanding the LPC32XX MCU and its peripherals. Startup code and several startup examples are also provided to get up and running quickly if the kickstart or stage 1 loader are to be replaced.

### 2.1 Supported LPC32XX drivers

See the lpc32xx\_readme.txt file included in the CSPS/LPC32XX directory for supported drivers and additional information on the CSP.

### 2.2 Example code

The example code shows how to configure and use the LPC32XX peripheral drivers for various functions. The examples may include functions from multiple drivers (such as the touchscreen example, which uses functions from the Interrupt, LCD, and ADC/Touchscreen drivers). Although these examples are intended to be executed from S11, some of the examples may not have a visual output and are much better suited for single stepping through with a hardware debugger.

#### 2.2.1 Specific toolchain examples

Some pre-built projects are also provided for specific toolchains such as Keil uVision3 or ARM Realview 3.x. If you have the supported toolchain installed, you can click on the toolchain project file to open a ready-made project for the example. *Not every example has a pre-built project.*

### 2.3 Startup code and examples

The board startup code is included in the PHY3250/STARTUP folder. This code provides the initial setup for the board including initial mux setup, SDRAM initialization, and clock setup. This code is usually considered a minimum set of functions to get a board up and running. The startup code is used with an application by calling `c_entry` in the application.

*Note only 64MByte SDRAM modules are supported in the startup code.*

#### 2.3.1 Kickstart loader

The kickstart loader handles the boot ROM to stage 1 application transfer. There are several variants of the kickstart loader. The LPC32XX bootrom has a boot size limit of 1 FLASH block, or about 54Kbytes, whichever is smaller. The kickstart loader provides a mechanism to download larger applications into memory by bridging the size gap between the boot ROM and stage 1 application. The source code for the kickstart loader is included with the BSP.

©2006-2007 NXP Semiconductors. All rights reserved.

Customers can install their own stage 1 application to be booted by the kickstart loader. See the phy32xx\_bl.doc file for information on how kickstart loads the stage 1 application.

### **2.3.2 Stage 1 loader (S1L)**

The stage 1 loader is based on the startup code included in the BSP. The stage 1 loader is the default stage 1 application loaded by the kickstart loader when the board is powered on or reset. The stage 1 loader application provides various capabilities to load and run applications, examine memory, or change system configuration. More information about the stage 1 loader can be found in the lpc32xx\_bl.doc file.

## **2.4 Other applications**

Other applications and demos are available for the Phytex 3250 board such as pre-built WinCE and Linux executables. Documentation is included with those applications on how to setup and execute them on the Phytex 3250 board.

### **2.4.1 LPC3250 serial loader (LSL)**

The LPC3250 serial loader (LSL) is a Windows tool that connects to the LPC3250 through a serial port. This tool communicates with the LPC3250 when the LPC3250 is reset and provides a method to get code to the chip and board without a JTAG device, or even if the board boot devices have been erased.

More information about the LSL can be found in Section 5.3

## **3 Building and deploying an application**

Applications can be built for the Phytex board using one of the supported tool chains. Applications can be built to work with or without the supplied board startup code, depending on whether the S1L application will be used.

There are also various methods for getting your application to the Phytex board for execution such as S1L, LPC3250 serial loader (LSL) tool, or a hardware debugger. The application can be setup to automatically start when the board is powered up and can be programmed into NAND FLASH or boot via SD card or serial port.

### **3.1 Application types**

#### **3.1.1 Applications using the stage 1 loader**

Applications built to use the S1L are the simplest to build and deploy. These types of applications are loaded through the S1L's *load* command and use the system environment and configuration already setup by the S1L. These types of applications typically require only 'light' startup code, as most of the system startup has been handled by the S1L application. Most of the default examples included with the BSP fall into this category.

To build an example to load and execute through S1L, read Section 4.1.

### 3.1.2 Applications replacing the stage 1 loader

Applications replacing the stage 1 loader are designed to be loaded and executed with the kickstart loader. These types of applications require more robust system startup code to initialize SDRAM, chip clocking, LPC3250 pin muxing, etc. The startup code included with the BSP is a great reference to start with if you are doing this type of application. Most of the examples included with the BSP can be built to execute as the stage 1 application by making a new startup project and copying the example code into that project.

*The SPI and NOR kickstart loader variants setup SDRAM, while the small block NAND FLASH variant doesn't. If you need to boot from small block NAND FLASH, you will need to handle board initialization in the application loaded by the kickstart loader, or you can try tweaking the build options and kickstart loader code to get it smaller so it will fit in block 0 of small block NAND FLASH (15.5K max size).*

## 3.2 Application deployment methods

### 3.2.1 Using S1L

S1L provides methods for deploying an application through the serial port, SD/MMC cards, and NAND FLASH. Binary files and S-record files are supported. See the phy32xx\_bl.doc file for more information on how to use S1L.

See Section 4.1 for how to deploy an application through S1L.

### 3.2.2 LPC3250 boot methods

The LPC3250 provides the ability to boot from the serial port, Synchronous Serial Port (SSP), or via NAND FLASH. This BSP supports the serial port and NAND FLASH boot methods.

## 4 Building and deploying applications

This section explains how to build and deploy application to the Phytex 3250 board using S1L or the LPC3250 serial loader (LSL) tool. Customers can build their own applications and restore the board to its original state if the application fails to boot.

All that's needed is a PC with a serial port, a serial cable between the PC and the Phytex 3250 board, a terminal program (such as TeraTerm), and one of the supported tool chains to build the application. Optionally, an SD/MMC card can be used to store and deploy large applications that are booted and executed through S1L.

Applications can be built with startup code that completely initializes the board or without startup code. Application built without startup code requires that the board

already be initialized before starting the application. Applications loaded through S1L already have the board configured when they are loaded and executed.

## 4.1 Building an application for use with S1L

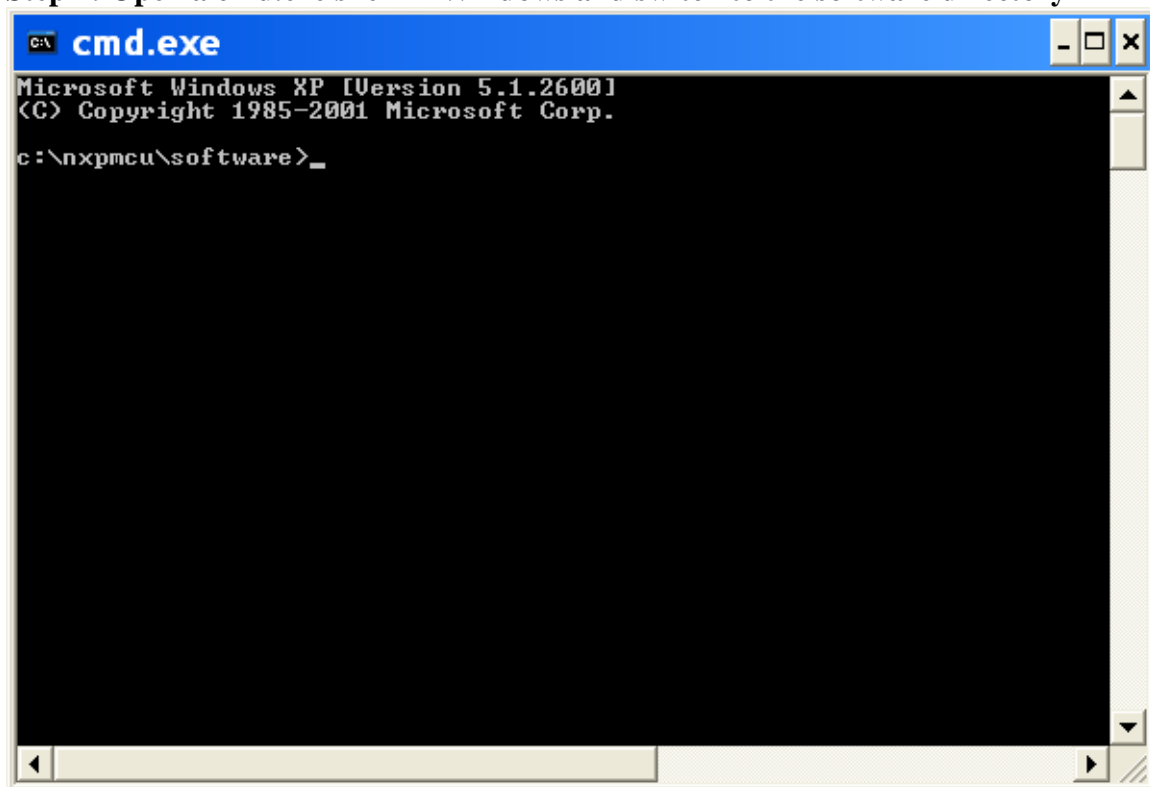
Applications that are loaded through S1L do not require full startup code and can be loaded and executed without any special hardware. These types of applications are usually load and run applications. If they don't work correctly or lock up the board, simply resetting the board will usually return the board to a good state.

These types of applications can be saved into NAND FLASH and booted from NAND FLASH (through S1L), loaded through the serial port, or loaded from an SD/MMC card. These applications are simple and quick to develop and shield the end user from the details of FLASH organization and system startup.

### 4.1.1 Building the application

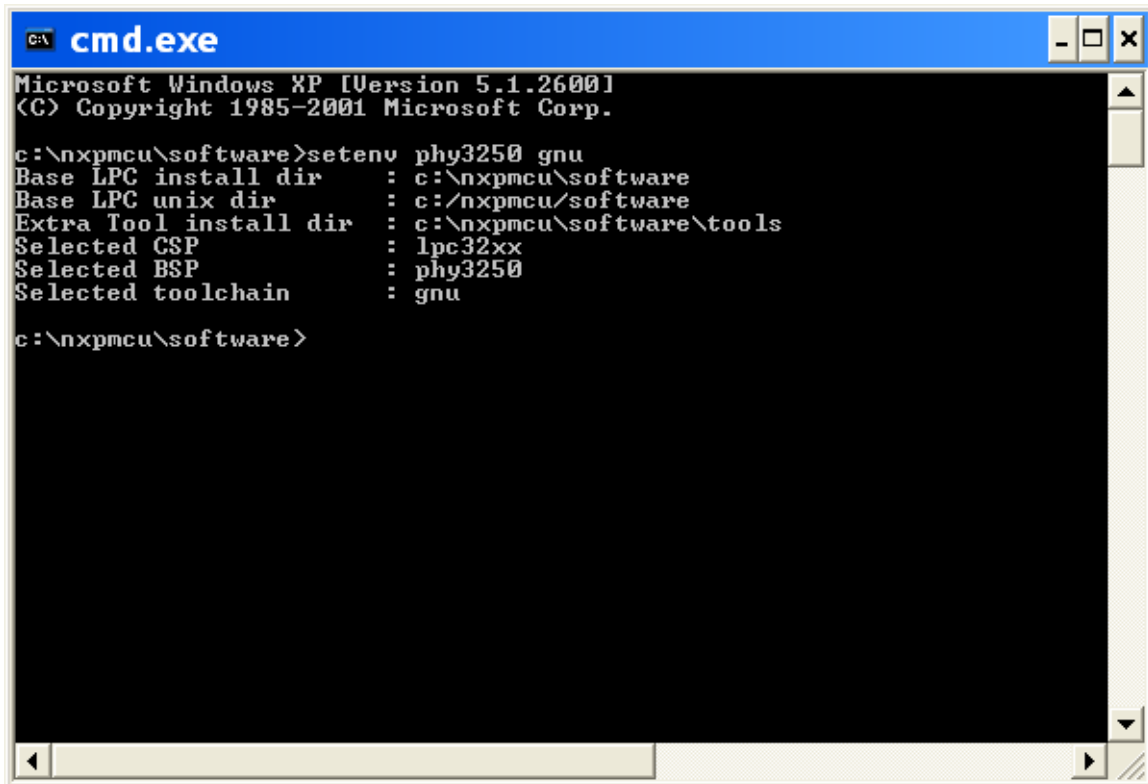
These steps detail how to build the timer application using the CodeSourcery GNU toolchain. This type of application will be built to load and execute through S1L.

**Step 1: Open a cmd.exe shell in Windows and switch to the software directory**



**Step 2 : Setup the build environment for the Phytex 3250 (PHY325) BSP and GNU**





```

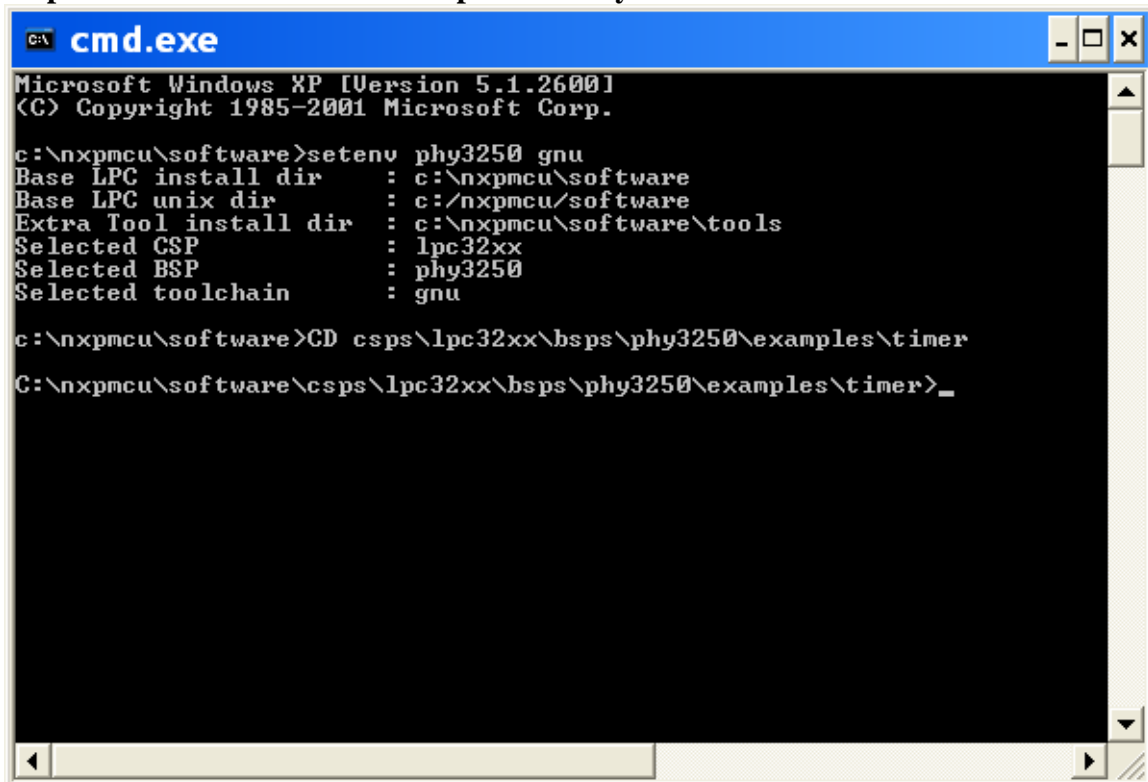
C:\> cmd.exe

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

c:\nxpmcu\software>setenv phy3250 gnu
Base LPC install dir      : c:\nxpmcu\software
Base LPC unix dir        : c:/nxpmcu/software
Extra Tool install dir    : c:\nxpmcu\software\tools
Selected CSP              : lpc32xx
Selected BSP              : phy3250
Selected toolchain        : gnu

c:\nxpmcu\software>
  
```

### Step 3 : Switch to the timer example directory in the PHY3250 BSP



```

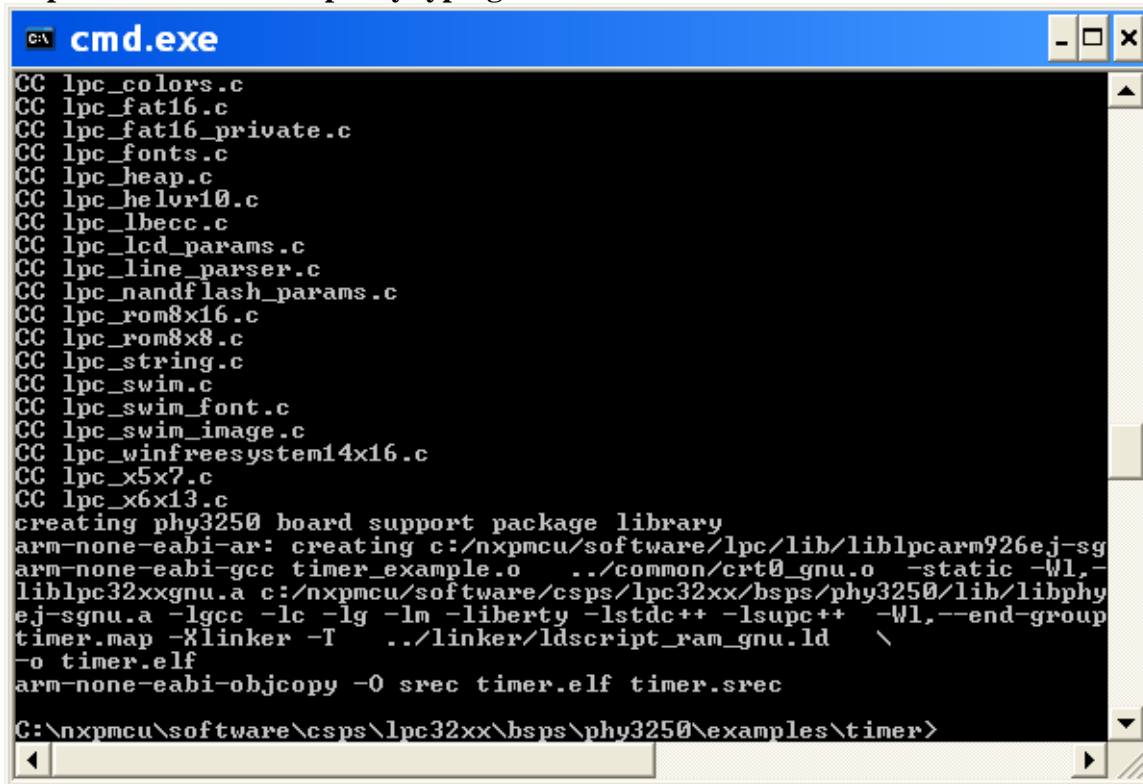
C:\> cmd.exe

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

c:\nxpmcu\software>setenv phy3250 gnu
Base LPC install dir      : c:\nxpmcu\software
Base LPC unix dir        : c:/nxpmcu/software
Extra Tool install dir    : c:\nxpmcu\software\tools
Selected CSP              : lpc32xx
Selected BSP              : phy3250
Selected toolchain        : gnu

c:\nxpmcu\software>CD c:\nxpmcu\software\csp\lpc32xx\bsp\phy3250\examples\timer
C:\nxpmcu\software\csp\lpc32xx\bsp\phy3250\examples\timer>_
  
```

#### Step 4 : Make the example by typing make



```

C:\nxpmcu\software\csps\lpc32xx\bsps\phy3250\examples\timer>make
CC lpc_colors.c
CC lpc_fat16.c
CC lpc_fat16_private.c
CC lpc_fonts.c
CC lpc_heap.c
CC lpc_hello10.c
CC lpc_lbecc.c
CC lpc_lcd_params.c
CC lpc_line_parser.c
CC lpc_nandflash_params.c
CC lpc_rom8x16.c
CC lpc_rom8x8.c
CC lpc_string.c
CC lpc_swim.c
CC lpc_swim_font.c
CC lpc_swim_image.c
CC lpc_winfreesystem14x16.c
CC lpc_x5x7.c
CC lpc_x6x13.c
creating phy3250 board support package library
arm-none-eabi-ar: creating c:/nxpmcu/software/lpc/lib/liblpcarm926ej-sg
arm-none-eabi-gcc timer_example.o ../common/crt0_gnu.o -static -Wl,-
liblpc32xxgnu.a c:/nxpmcu/software/csps/lpc32xx/bsps/phy3250/lib/libphy
ej-sgnu.a -lgcc -lc -lg -lm -liberty -lstdc++ -lsupc++ -Wl,--end-group
timer.map -Xlinker -T ../linker/ldscript_ram_gnu.ld \
-o timer.elf
arm-none-eabi-objcopy -O srec timer.elf timer.srec

C:\nxpmcu\software\csps\lpc32xx\bsps\phy3250\examples\timer>

```

The example build is complete! The timer.elf file can be loaded and executed through a hardware debugger, while the timer.srec file can be loaded and executed through S1L.

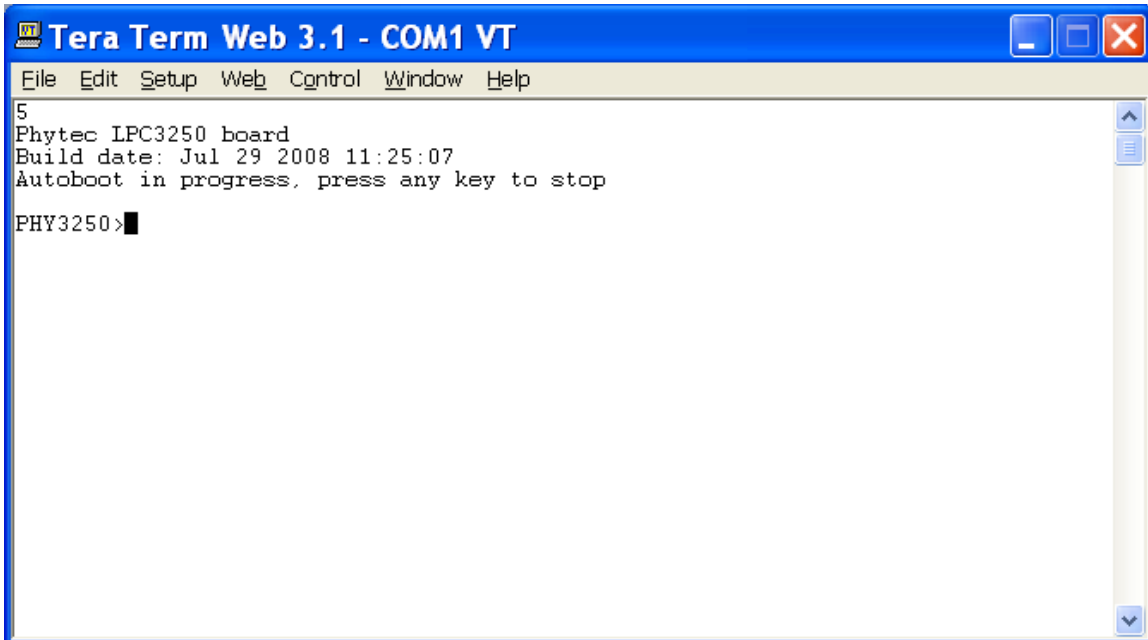
### 4.1.2 Deploying the application

The application can be loaded and executed on the board using S1L commands and load methods. The following 2 methods explain how to load and execute the timer example generated in Section 4.1.1 on the board using the serial port or an SD/MMC card.

#### 4.1.2.1 Deploying the application using the serial port

The following steps explain how to deploy an application to the Phytex board using S1L's serial port and load/exec commands.

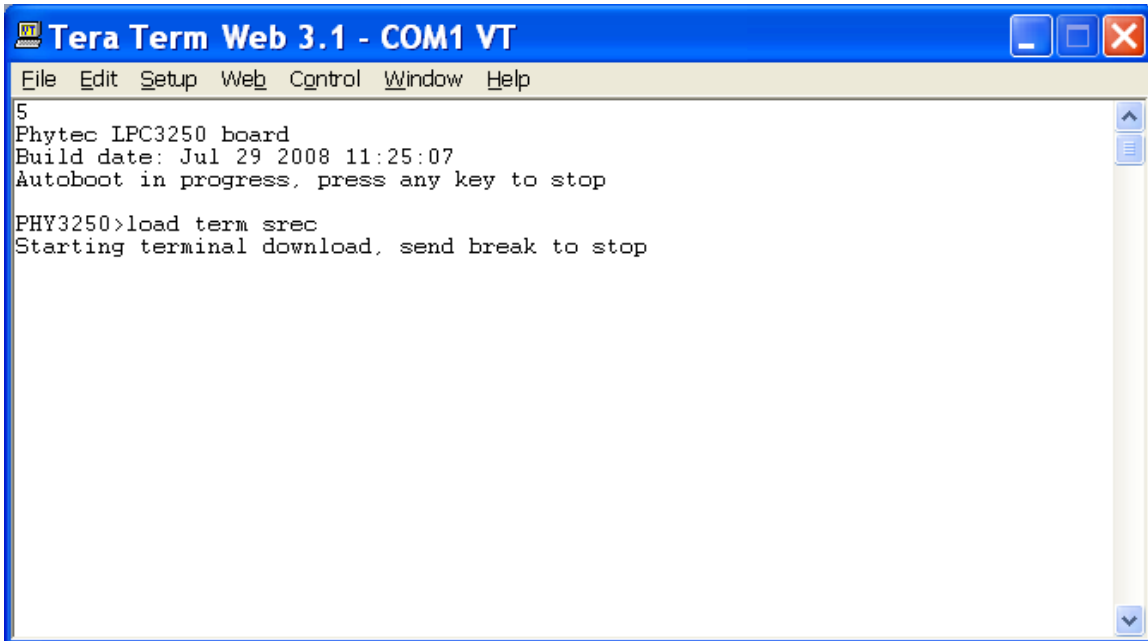
**Step 1 : Start your terminal program (115.2K-8-N-1) and connect to the board. A message similar to what is shown below will appear. You may need to press a key to get to the PHY3250> S1L prompt.**



The screenshot shows a terminal window titled "Tera Term Web 3.1 - COM1 VT". The menu bar includes File, Edit, Setup, Web, Control, Window, and Help. The terminal output displays the following text:

```
5
Phytec LPC3250 board
Build date: Jul 29 2008 11:25:07
Autoboot in progress, press any key to stop
PHY3250>
```

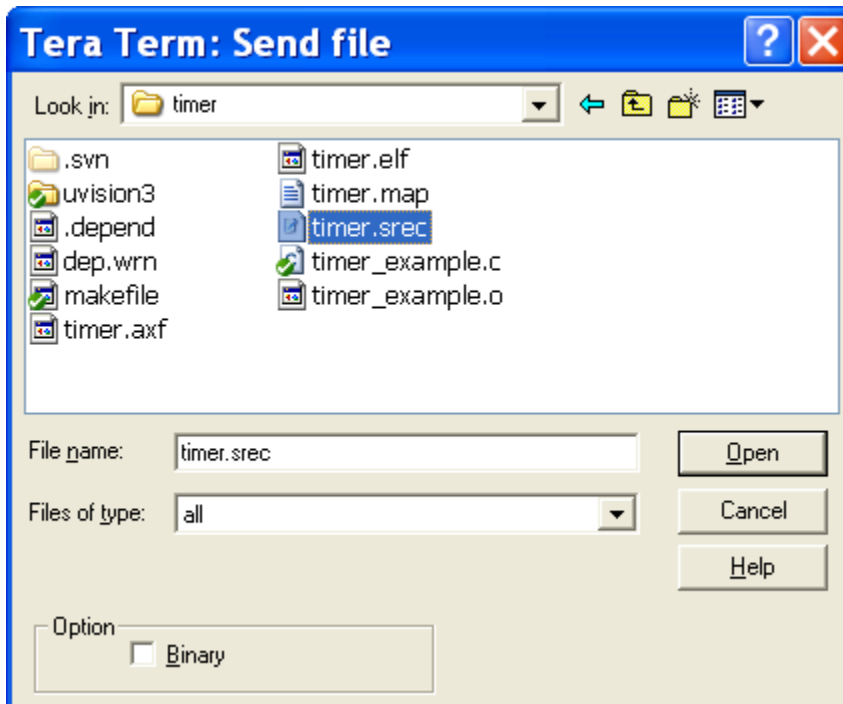
**Step 2 : Using the load command, tell S1L that you want it to receive a S-record file from the terminal.**



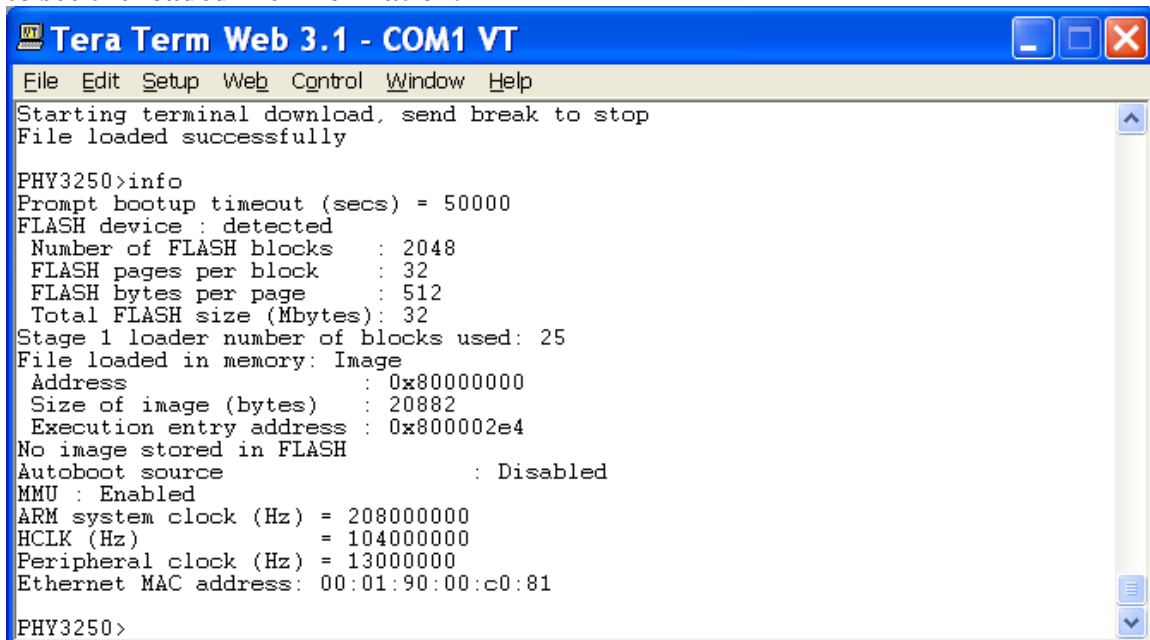
The screenshot shows the same terminal window as before, but with additional output after the command 'load term srec' has been entered:

```
5
Phytec LPC3250 board
Build date: Jul 29 2008 11:25:07
Autoboot in progress, press any key to stop
PHY3250>load term srec
Starting terminal download, send break to stop
```

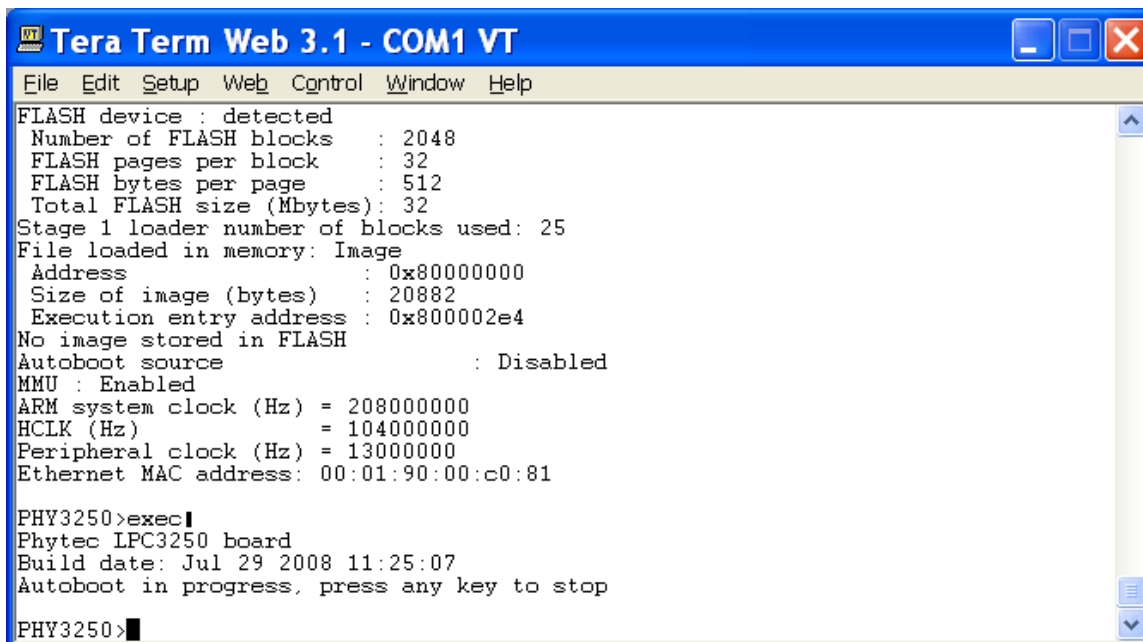
**Step 3 : Send the file from the terminal program as a binary or text file (no protocol).**



**Step 4 :** After the file has loaded, the prompt will return (binary files require a break to be sent from the terminal to return to the prompt). Type the info command to see the loaded file information.



**Step 5 :** The file loaded in memory is loaded at address 0x80000000 and is 20882 bytes in size. It is executed by starting the code at its execution entry address, which is 0x800002E4. Use the exec command to start the example. The example will run and after a few seconds, the prompt will return. The example has run successfully.



```

Tera Term Web 3.1 - COM1 VT
File Edit Setup Web Control Window Help
FLASH device : detected
Number of FLASH blocks : 2048
FLASH pages per block : 32
FLASH bytes per page : 512
Total FLASH size (Mbytes): 32
Stage 1 loader number of blocks used: 25
File loaded in memory: Image
Address : 0x80000000
Size of image (bytes) : 20882
Execution entry address : 0x800002e4
No image stored in FLASH
Autoboot source : Disabled
MMU : Enabled
ARM system clock (Hz) = 208000000
HCLK (Hz) = 104000000
Peripheral clock (Hz) = 13000000
Ethernet MAC address: 00:01:90:00:c0:81

PHY3250>exec
Phytec LPC3250 board
Build date: Jul 29 2008 11:25:07
Autoboot in progress, press any key to stop
PHY3250>

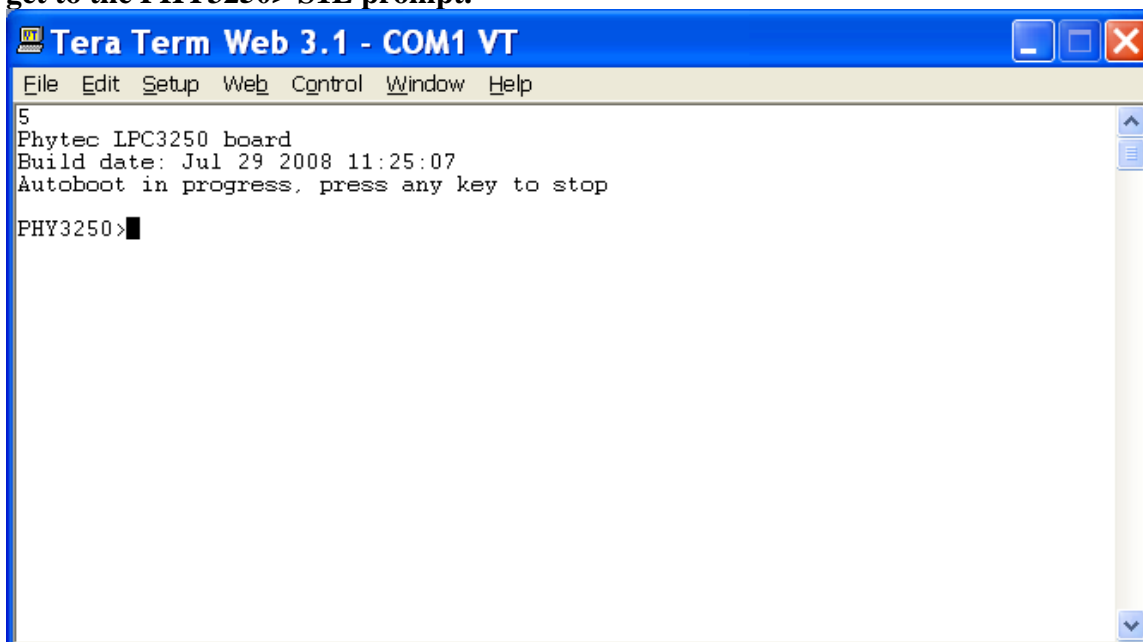
```

The same procedure can be used with the other examples included with the BSP.

#### 4.1.2.2 Deploying the application using an SD/MMC card

The following steps explain how to deploy an application to the Phytec board using Phytec board SD/MMC slot and an SD/MMC card.

**Step 1 : Start your terminal program (115.2K-8-N-1) and connect to the board. A message similar to what is shown below will appear. You may need to press a key to get to the PHY3250> S1L prompt.**

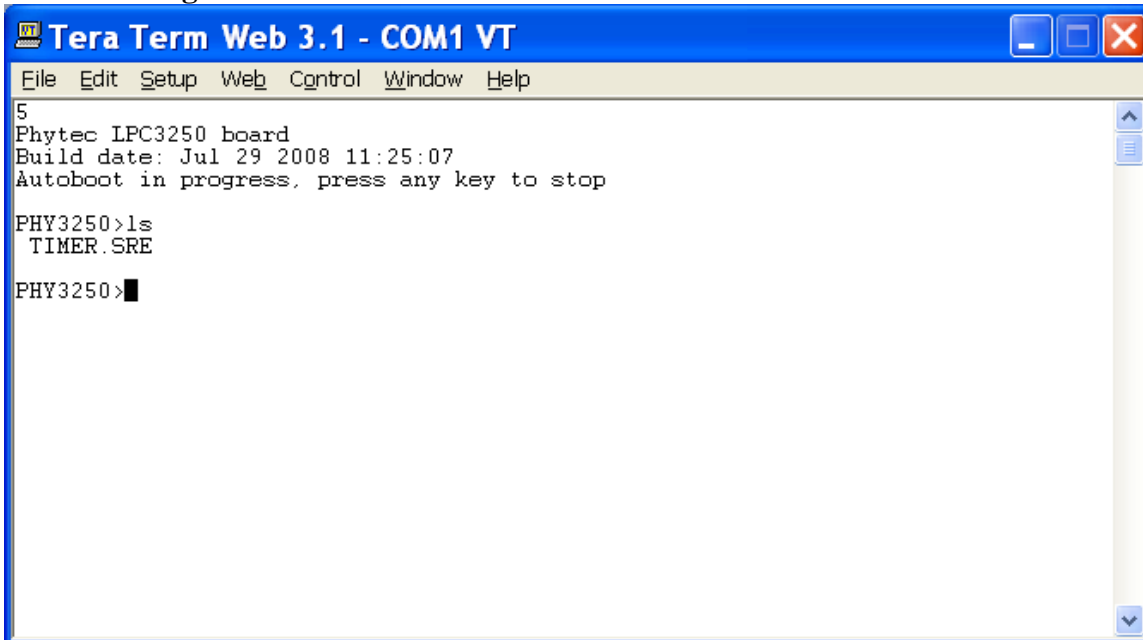


```

Tera Term Web 3.1 - COM1 VT
File Edit Setup Web Control Window Help
5
Phytec LPC3250 board
Build date: Jul 29 2008 11:25:07
Autoboot in progress, press any key to stop
PHY3250>

```

**Step 2 : Place your program (timer.srec) into the root directory of an SD or MMC card and insert the card into the SD/MMC slot on the board. It is recommended that the name be renamed to an 8.3 format name (such as timer.sre) to make loading the file easier. Once the card has been inserted into the board, use the ls command to get a list of file in the root director of the card.**



```

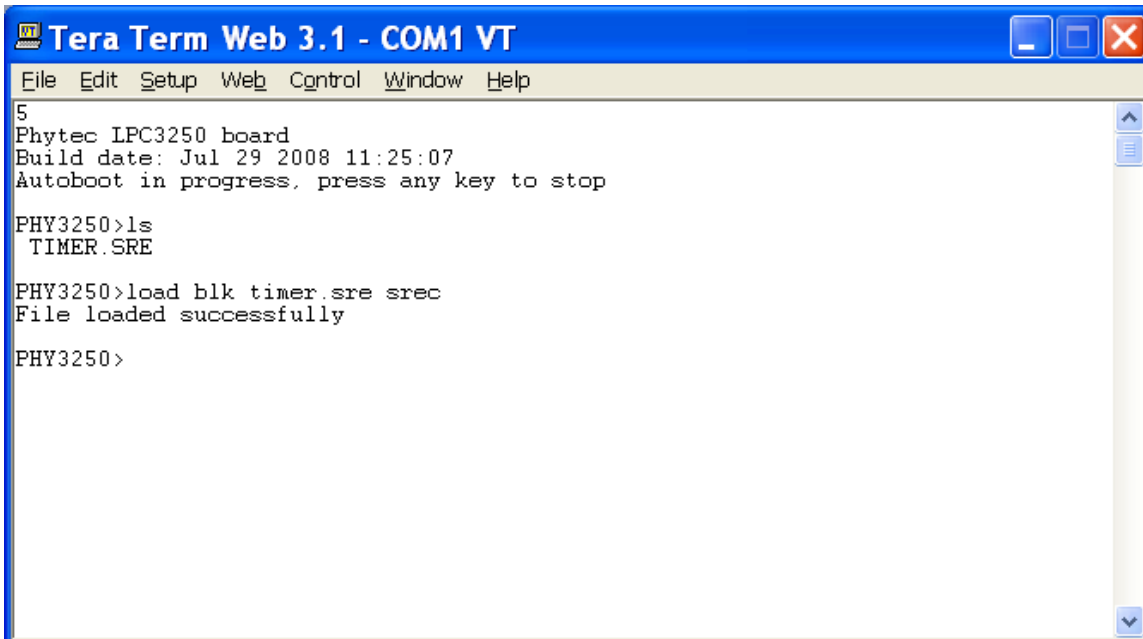
Tera Term Web 3.1 - COM1 VT
File Edit Setup Web Control Window Help
5
Phytec LPC3250 board
Build date: Jul 29 2008 11:25:07
Autoboot in progress, press any key to stop

PHY3250>ls
  TIMER.SRE

PHY3250>

```

**Step 3 : Use the load command to get the S-record file into the board's memory. The syntax of the load command changes slightly from the serial loaded version of the command.**



```

Tera Term Web 3.1 - COM1 VT
File Edit Setup Web Control Window Help
5
Phytec LPC3250 board
Build date: Jul 29 2008 11:25:07
Autoboot in progress, press any key to stop

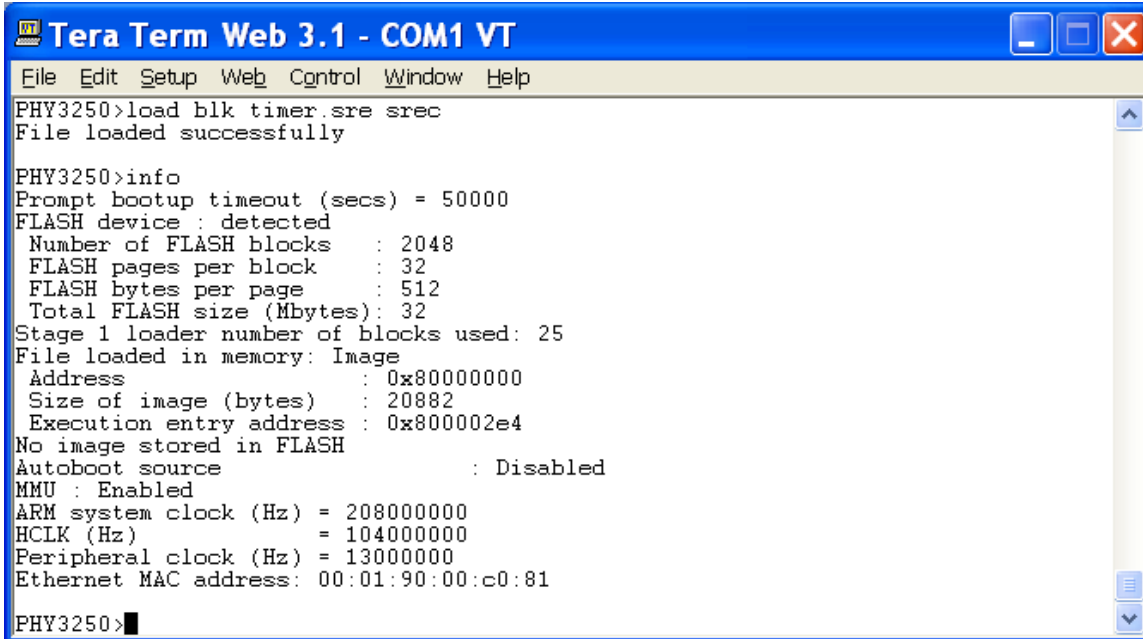
PHY3250>ls
  TIMER.SRE

PHY3250>load blk timer.sre srec
File loaded successfully

PHY3250>

```

**Step 4 :** After the file has loaded, the prompt will return. Type the info command to see the loaded file information.



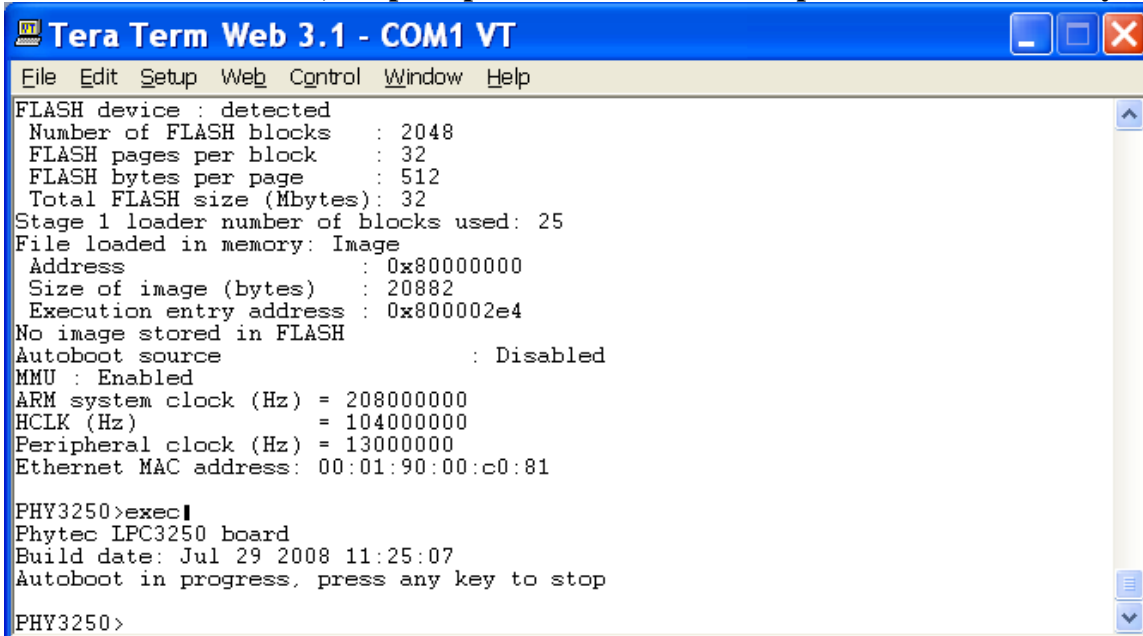
```

Tera Term Web 3.1 - COM1 VT
File Edit Setup Web Control Window Help
PHY3250>load blk timer.sre srec
File loaded successfully

PHY3250>info
Prompt bootup timeout (secs) = 50000
FLASH device : detected
  Number of FLASH blocks   : 2048
  FLASH pages per block   : 32
  FLASH bytes per page    : 512
  Total FLASH size (Mbytes): 32
Stage 1 loader number of blocks used: 25
File loaded in memory: Image
  Address      : 0x80000000
  Size of image (bytes) : 20882
  Execution entry address : 0x800002e4
No image stored in FLASH
Autoboot source      : Disabled
MMU : Enabled
ARM system clock (Hz) = 208000000
HCLK (Hz)             = 104000000
Peripheral clock (Hz) = 130000000
Ethernet MAC address: 00:01:90:00:c0:81

PHY3250>
  
```

**Step 5 :** The file loaded in memory is loaded at address 0x80000000 and is 20882 bytes in size. It is executed by starting the code at its execution entry address, which is 0x800002E4. Use the exec command to start the example. The example will run and after a few seconds, the prompt will return. The example has run successfully.



```

Tera Term Web 3.1 - COM1 VT
File Edit Setup Web Control Window Help
FLASH device : detected
  Number of FLASH blocks   : 2048
  FLASH pages per block   : 32
  FLASH bytes per page    : 512
  Total FLASH size (Mbytes): 32
Stage 1 loader number of blocks used: 25
File loaded in memory: Image
  Address      : 0x80000000
  Size of image (bytes) : 20882
  Execution entry address : 0x800002e4
No image stored in FLASH
Autoboot source      : Disabled
MMU : Enabled
ARM system clock (Hz) = 208000000
HCLK (Hz)             = 104000000
Peripheral clock (Hz) = 130000000
Ethernet MAC address: 00:01:90:00:c0:81

PHY3250>exec
Phytec LPC3250 board
Build date: Jul 29 2008 11:25:07
Autoboot in progress, press any key to stop

PHY3250>
  
```

The same procedure can be used with the other examples included with the BSP.

## 5 Miscellaneous

### 5.1 Where to get software

The latest Phytex 3250 BSP and LPC3250 CSP are available from NXP's website.

TeraTerm is free and can be downloaded from a number of sources such as SourceForge and tucows.

CodeSourcery GNU can be downloaded from [www.codesourcery.com](http://www.codesourcery.com).

### 5.2 Startup code, S1L, boot loaders, and deployment

This topic is covered in the generic\_32x0 BSP documentation included with the LPC32xx CDL.

### 5.3 Using the LPC3250 serial loader (LSL) tool

The LSL is meant to be used on various LPC devices and all of it's functionality may not apply to the LPC3250 or the Phytex 3250 board. *As of this time, only the primary boot (IRAM) option is supported with the LSL.*

Using the LSL requires setting up the primary boot executable filename. The image must always be built to load and execute starting at address 0x00000000. Once LSL is setup, the Load bin's/Start primary button can be pressed and the LSL will wait for a character from the LPC3250 boot ROM on reset.

The boot sequence is as follows:

LPC3250	LPC3250 serial loader
LPC3250 is reset	
LPC3250 sends a '5' on UART3.	LSL responds with a 'A'.
LPC3250 sends a '5' on UART3.	LSL responds with a 'U'.
	LSL sends the address to the board (0x00000000)
	LSL sends the image size to the board in bytes
	LSL sends the code to the board
LPC3250 boots loaded code at address 0x00000000	

### 5.4 Issues

#### 5.4.1 IAR toolchain

Examples built with IAR and executed through S1L will not return to the S1L prompt once execution is complete. This is because the system resources shared between S1L



and the examples (stacks) are modified by the IAR example. For IAR examples, press reset on the board to return to the S1L prompt.

Startup code examples will not currently work with IAR.