Kickstart and Stage 1 loader

Table of contents

1 Introduction 3

2 Kickstart loaders 5

2.1 kickstart loader variants 5

2.2 Setting up a kickstart loader variant 6

2.2.1 Small block NAND FLASH 6

2.2.2 SPI FLASH 6

2.2.3 nor FLASH 7

3 Stage 1 loader 8

3.1 Stage 1 loader operation process 8

3.2 Stage 1 loader resource usage 8

3.2.1 IRAM organization 9

3.2.2 Stage 1 loader persistent data storage 9

3.3 Stage 1 loader monitor program operations 10

3.3.1 Accessing the monitor program 10

4 Notes 12

4.1 WinCE and reserved block marking 12

List of tables

Table 1 IRAM organiztion for stage 1 loader 9

1 Introduction

There are several bootloaders in the NAND FLASH of the Phytec LPC3250 board that help make program loading and development easier. These consist of the kickstart loader and the stage 1 application loader. *Before continuing with the document, it is highly recommended that the "startup code" file included in the documentation area of the generic_32x0 BSP be examined. This document explains for reasoning and concepts for the boot loaders included in this BSP.*

The kickstart loader sits in block 0 of FLASH and is responsible for loading an application that starts in FLASH block 1. The main features of the kickstart loader are shown below:

- Allows loading images greater than 1 block length (stage 1 application) *(An image of about 54K is the maximum size that can be booted with the LPC3250 boot ROM) or 15.5K for small block FLASH*
- Loads stage 1 applications into memory

The stage 1 loader is the default application loaded by the kickstart loader. The stage 1 loader provides a monitor program with a collection of functions to help with application development. The main features of the stage 1 loader are shown below:

- Register and memory change and dump
  - Poke, peek, dump, fill
- Image load via a serial port , SDMMC card, or FLASH
  - Supports raw binary and S-record files
  - Images can be executed after loading
  - Images can be saved in FLASH
- NAND FLASH support
  - Erase of NAND blocks
  - Direct read and write of FLASH blocks and pages
  - Bad block management
  - Reserved block management for operating systems
- MMU functions
  - Data and instruction cache control
  - Virtual address translation enable/disable
  - Virtual address remapping
  - Page table dump
- System support functions
  - Baud rate control, clock control, system information
- Automatic load and run support
  - Automatic load and execution of images from FLASH, SDMMC, or via the terminal

The source code for the kickstart loader is provided in the Phytec LPC3250 Board Support Package (BSP).

2 Kickstart loaders

The kickstart loader provides initial boot capability of the 3250 device from NAND FLASH, SPI FLASH, or NOR FLASH. The Phytec 3250 board is shipped with the NAND small block version of the kickstartloader installed in block 0 of NAND FLASH. The purpose of this loader is to load the Stage 1 Loader application.

This BSP provides the latest version of the kickstart loader and several other variants of the kickstart loader. More variations that can be ported over to the Phytec board are also included in the generic_32x0 BSP.

2.1 kickstart loader variants

There are 2 variants of the kickstart loaded included in this BSP.

The small block NAND FLASH kickstart loader is meant to reside in block 0 and loads a stage 1 application from block 1 and on into memory. It should be noted that this version of the kickstart loader doesn't include startup code or MMU setup code and is meant to be very small (less than 15.5K) so it will easily fit inside block 0 of small block NAND FLASH. Although this version is meant for small block NAND FLASH, it can also be used with SPI FLASH without any changes (and will allow the 32x0 chip to boot from SPI and load the stage 1 application from small block NAND FLASH block 1). This version of the kickstart loader cannot load a stage 1 application into SDRAM, as it doesn't initialize SDRAM. The stage 1 application it loads needs to load into (another location in) IRAM and must enable SDRAM and the MMU and caches if it plans on using them. *This is the default version of the kickstart loader include on the Phytec board, but can be replaced with another variant. The Phytec boards ship with S1L as the default stage 1 application that loads and runs in IRAM. S1L initializes clocking, GPIO, SDRAM, the MMU, etc. and can be used to launch u-boot or other OS specific boot loaders. Note that u-boot (Linux) and eboot (WinCE) need to load in SDRAM and must load from S1L; they cannot load from this version of the kickstart loader directly.*

The SPI FLASH kickstart loader is meant to reside in SPI FLASH only. It's exactly the same version as the small block NAND FLASH kickstart loader except that it includes the code for initializing SDRAM, clocks, GPIOs, MMU, etc. The resulting binary for this kickstart loader is larger than the small block NAND FLASH version. Note that it will still load the stage 1 application from small block NAND FLASH at block 1. *This version of the kickstart loader is well suited for applications that need to load into SDRAM. U-boot and eboot can be loaded directly into SDRAM without needing to transition through another loader program.*

The NOR FLASH version of the kickstart load is meant to reside on NOR FLASH. It's exactly the same version as the SPI FLASH kickstart loader except that it includes some extra code for relocating the data regions to IRAM. Note that it will still load the stage 1 application from small block NAND FLASH at block 1.

2.2 Setting up a kickstart loader variant

In most cases, you probably don't need to change the default kickstart loader. However, if you're developing your own board, you may not want an extra boot layer such as S1L. In these cases, it would be best to setup the kickstart loader to boot the desired application directly into memory.

2.2.1 Small block NAND FLASH

If you boot from small block NAND FLASH, you have a limit of 15.5Kbytes on your boot image size for block 0. This will make it tough to fit board initialization code (SDRAM, clocks, GPIOs, etc) and the stage 1 application loader code into a simgle image under 15.5K. If this is your situation, you have several choices:

Optimize the startup and kickstart loader code to be smaller
The default code has lots of room for optimization, especially in the clock setup area. Some of the functions can be hardcoded instead of using the setup libraries in the LPC32XX CDL. For example, directly programming the main system PLL with a hardcoded value instead of using the system PLL setup functions can save a lot of memory.

Try different compilation options or toolchains
Each supported toolchain generates a different code size. Try changing the compiler options to disable extra debug code or size based optimization.

Move code out of the kickstart loader and into your stage 1 application
The default build options in the BSP use this approach. The code for initializing the board (SDRAM, clocks, GPIOs, etc.) is not in the kickstart loader and is in the stage 1 application instead. This makes loading the stage 1 application in SDRAM not possible from the kicktsart loader.

2.2.2 SPI FLASH

If you boot from SPI FLASH, you have a limit of 54Kbytes on your boot image size from NAND FLASH. This allows enough room for the board initialization code to be present in the kickstart loader.

Since the kickstart loader initializes SDRAM, the stage 1 application can be loaded directly into SDRAM. The variant included with this board allows the kickstart loader to boot from SPI FLASH, initialize SDRAM, and then load the stage 1 application from small block NAND FLASH block 1 and on into SDRAM.

By editing the load address and load size for the kickstart loader and then burning it into SPI FLASH, u-boot and eboot can be loaded directly into SDRAM without the use of S1L.

2.2.3 nor FLASH

If you boot from NOR FLASH, you don’t have a limit on the boot image size. You can boot directly into the stage 1 application as the kickstart loader is no longer needed. A version of the kickstart loader is included with this BSP that builds for NOR FLASH and initializes the board and then loads the stage 1 application from NOR FLASH (at offset 128K).

3 Stage 1 loader

The default stage 1 application loaded by the kickstart loader is the stage 1 loader. This application initializes the board and chip and provides a monitor program for simple development and program execution options.

3.1 Stage 1 loader operation process

The stage 1 loader is started at address 0x8000 once it is loaded from the kickstart loader. The stage 1 loader first initializes the board with the code from the startup files before starting the monitor program. The startup code sets up the board in states as shown below:

MMU page table located at the end of IRAM (section table only)
Default MMU virtual to physical address mapping
MMU is enabled; data and instruction caches are enabled
Sets up ARM clock to 208MHz, bus clock to 104MHz, and PCLK to 13MHz
Initializes SDRAM at address 0x80000000
Sets up initial GPIO directions and states and some peripheral muxing
Disables all peripheral clocks
Sets up static memory timing
Initializes all stacks in IRAM

Once startup is completed, the monitor program is started. Depending on how the stage 1 loader is configured, the monitor program may attempt to load and execute and image, or go to the monitor prompt.

3.2 Stage 1 loader resource usage

The stage 1 loader uses UART5, SSP0, the SD card controller, several timers, and the SLC FLASH controller during its operation. Data may be saved into FLASH or the serial EEPROM during its use. SDRAM is not directly used by the stage 1 loader, although some interactive operations may affect SDRAM.

When applications are loaded and executed with the stage 1 loader, the loader will disable all its used peripherals prior to executing the loaded application. Some peripheral and loader settings are passed unchanged to the application; these include the settings configured by the startup code. When the application is called, the system is in the following state:

MMU page table located at end of IRAM (section table only)
MMU may be enabled or disabled
Data and instruction caches may be enabled or disabled
Clock rates as configured by the monitor program
SDRAM initialized at address 0x80000000
GPIO directions, states, and muxing unchanged from startup code
All peripherals disabled

All stacks in IRAM

Applications called from the stage 1 loader may safely use the resources setup by the stage 1 loader. If the application wants to return to the stage 1 loader, the application must return the system back to its original state before exiting. Applications that require bigger stacks should setup their own stacks as part of the application. If an application overwrites the memory used by the stage 1 loader, returning from the application may cause the system to crash.

3.2.1 IRAM organization

IRAM is organized as shown in Table 1 for the application called from the stage 1 loader.

Table 1 IRAM organiztion for stage 1 loader

IRAM base	Size (bytes)	IRAM use
0x0003C000	16K	MMU page table
0x0003C000[1]	varies	Stacks (grow down as needed)
0x00008000	varies	Stage 1 loader code and data
0x00000000	233K	Unused (kickstart loader area before transfer to S1L)

[1]Stacks grown down.

3.2.2 Stage 1 loader persistent data storage

The stage 1 loader stores configuration data in the serial EEPROM located on SSP0. This data contains information such as the prompt string, baud rate, autoboot parameters, and image stored in FLASH. These values are used by the stage 1 loader between power cycles to remember the last stage 1 loader configuration. Some commands automatically updated this data (such as the prompt command) whenever a configurable options is changed. The first time the board is powered up, the data is populated with default values.

The following configurable items can be changed with the corresponding data stored in the serial EEPROM:

Prompt string
Autoboot timeout
Autoboot source, file type, load address, filename, execution address
Configured baud rate
Saved FLASH image status
System clock settings
Startup script

3.2.2.1 Resetting default values

If for some reason, the board is configured in such as way as to become unbootable after the stage 1 loader starts, holding down the BTN1 button as the board is booting will restore the default values for the configuration data. The restored values will only apply for that power cycle.

3.3 Stage 1 loader monitor program operations

This section explains the monitor program operations that can be executed from the command prompt of the stage 1 loader.

3.3.1 Accessing the monitor program

The monitor program can be accessed by connecting a serial cable to the bottom serial connector on the Phytec LPC3250 board. A terminal program (such as Teraterm) configured for 115.2Kbits/sec, 8 data bits, no parity, and 1 stop bit will be needed. If the default configuration is used, a prompt such as *PHY3250>* should appear on the terminal. If the default configuration has been changed, the prompt may be different or another program may have been configured to be autobooted.

Commands are entered at the monitor program prompt. All commands are case insensitive. Pressing DEL on the current command line will erase the current command entered. After each command has been typed, pressing enter will execute the command.

3.3.1.1 S1L commands

S1L commands and syntax are explained in the “start code” document in the documentation area of the generic_32x0 BSP.

3.3.1.2 Phytec 3250 S1L commands

The commands in this section are specific to the Phytec 3250 board.

3.3.1.2.1 clock

Resets the system clock rates. The ARM core clock, bus clock (HCLK), and the peripheral clock (PCLK) are all set based on the passed arguments. Care must be used with this command as driving the part too fast may cause the system to crash or become unstable.

Command syntax:
clock [ARM freq MHz] [HCLK divider (1, 2, 4)] [PERIPH_CLK divider (1 to 32)]

Examples:
clock 208 2 16 (Sets ARM clock to 208MHz, HCLK to 104MHz, and PCLK to 13MHz)
clock 150 75 12 (Sets ARM clock to 150MHz, HCLK to 75MHz, and PCLK to 12.5MHz)

Notes:
It is recommended that values be selected that prevent the HCLK from exceeding 104MHz and PCLK should be as close to 13MHz as possible. System timings dependent on the new clock settings will be optimized to get the best performance with this command. A small glitch on the terminal may occur as the clocks are adjusted.

Persistent configuration:
Clock settings are saved across power cycles. If the board fails to boot due to a bad clock setting, boot the board in the default (208MHz) configuration by holding down the BTN1 button when the board is reset.

3.3.1.2.2 bberase

Disables or enables erasure of bad blocks when using the erase command. If an argument of 1 is used with this command, use of the erase command will also erase any bad blocks. If an argument of 0 is used, bad blocks will always be skipped.

Command syntax:
bberase <0, 1>

4 Notes

4.1 WinCE and reserved block marking

If FLASH support is enabled in the WinCE build, WinCE may attempt to partition and format FLASH for its use. This will cause the kickstart and stage 1 loader to be erased, as well as any stored data in FLASH. Care should be taken to prevent WinCE from erasing any blocks needed by the boot loaders or other storage.