Evaluating the Linux Port to LatticeMico32 System User Guide

Lattice Semiconductor Corporation
5555 NE Moore Court
Hillsboro, OR 97124
(503) 268-8000

February 2008

Copyright

Trademarks

Disclaimers

to correct any errors contained herein or to advise any user of this document of any correction if such be made. LSC recommends its customers obtain the latest version of the relevant information to establish, before ordering, that the information being relied upon is current.

Type Conventions Used in This Document

Convention	Meaning or Use
Bold	Items in the user interface that you select or click. Text that you type into the user interface.
<*Italic*>	Variables in commands, code syntax, and path names.
Ctrl+L	Press the two keys at the same time.
`Courier`	Code examples. Messages, reports, and prompts from the software.
`...`	Omitted material in a line of code.
`.` `.` `.`	Omitted lines in code and report examples.
`[ ]`	Optional items in syntax descriptions. In bus specifications, the brackets are required.
`( )`	Grouped items in syntax descriptions.
`{ }`	Repeatable items in syntax descriptions.
`|`	A choice between items in syntax descriptions.

Contents

Evaluating the Linux Port to LatticeMico32 System User Guide

This guide shows you how to evaluate the Linux port to the LatticeMico32 System architecture once you have successfully built it and loaded it on the board, as described in the *Linux Port to LatticeMico32 System Reference Guide*. In addition, it demonstrates some of the userland applications available.

Prerequisites

To evaluate the Linux port to LatticeMico32 System, you must have Linux running on the board. The *Linux Port to LatticeMico32 System Quick-Start Guide* explains how to install the distributed pre-built binaries on a board by using a pre-generated bitstream. In addition, the *Linux Port to LatticeMico32 System Reference Guide* outlines several ways to rebuild the environment from scratch and install the resulting binaries.

Furthermore, a set-up network environment is required, including a network interface on the board configured by kernel command-line boot options. The examples in this guide are based on the assumption that the board's IP address is 192.168.0.100.

Using Linux

Serial Console

A common way to communicate with the board is through a serial console over a RS-232 connection. Use PuTTY, HyperTerminal, Picocom, or minicom to establish a connection. The configuration of the application used depends on the configuration of the UART of the present platform.

If you use the distributed pre-generated bitstream, the following configuration is required:

- 115200 baud rate
- 8 data bits
- 1 stop bit
- No parity
- No flow control

The “Setting Up the Remote Serial Console” section of the *Linux Port to LatticeMico32 System Quick-Start Guide* describes the configuration of PuTTY and HyperTerminal in detail.

Login and Shell

Establish a connection between the serial console and the board. After the kernel is booted, a login prompt appears. Log in as user “root” with the password “lattice.” Upon successful log in, a short welcome message is displayed, and the BusyBox built-in msh shell appears.

```
lm32_eval_ecp250 login: root
Password: Welcome to Linux on LatticeMico32!

Theobroma Systems

For further information check:
http://www.theobroma-systems.com/mico32/
http://www.latticesemi.com/

Jan  1 00:01:30 login[20]: root login on 'ttyS0'

BusyBox v1.7.1 (2007-11-15 12:00:00 CET) built-in shell (msh)
Enter 'help' for a list of built-in commands.

root:~#
```

The msh shell is comparable to, but not as extensive as, the well-known bash shell. It is the default shell for the system and can be used for simple shell scripts.

ping

To see if the board can be reached over the network, use the ping <*IP_address*> command. If the board can be reached, this command should provide output that shows how long it took to transmit the packets to the board.

telnet

Using a telnet client is another way of establishing a connection with the board, but this time over the network. After you connect the telnet client with the board on default port 23, a login prompt identical to the one shown for the serial console appears. Log in as “root,” and the msh shell appears.

The telnet client enables you to handle Linux the same way as the serial console, but it is not available until Linux has finished booting.

ftp

By default, the board provides an FTP server that you can use to transfer data from your host computer to the board and vice versa. Connect to the server, then provide the “root” user name and the “lattice” password to log in.

Here is an example that uses a command-line FTP client:

```
$ ftp 192.168.0.100
Connected to 192.168.0.100.
220 lm32_eval_ecp250 FTP server (GNU inetutils 1.4.1) ready.
Name (192.168.0.100:theobroma): root
331 Password required for root.
Password:
230- Welcome to Linux on LatticeMico32!
230-
230-
230- Theobroma Systems
230-
230-
For further information check:
230- http://www.theobroma-systems.com/mico32/
230- http://www.latticesemi.com/
230-
30 User root logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

Alternatively, you can use your Web browser to see and download data residing on the board. Start your Web browser and open the URL ftp://root:lattice@192.168.0.100/ to access the FTP server.

Userland Applications

You can use a variety of common UNIX utilities and other userland applications to evaluate the Linux port to the LatticeMico32-based board, and this section introduces a few of them. Use the following command to see a list of all applications available in this release:

`ls /bin /usr/bin`

ifconfig

Ifconfig serves to configure and control the network interfaces from the command line, including setting the IP address and the net mask, and disabling or enabling a given interface.

Run ifconfig without a parameter to obtain a list of all configured interfaces:

```
eth0      Link encap:Ethernet  HWaddr 12:34:56:78:AB:CD
          inet addr:192.168.0.100  Bcast:192.168.255.255
          Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:2 Base address:0x8000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Eth0 is the network interface of the board, and lo is the loopback interface at the 127.0.0.1 loopback address.

Run ifconfig --help to obtain a list of available parameters:

```
Usage: ifconfig [-a] interface [address]

Configure a network interface

Options:

        [[-]broadcast [ADDRESS]] [[-]pointopoint [ADDRESS]]
        [netmask ADDRESS] [dstaddr ADDRESS] [hw ether ADDRESS]
        [metric NN] [mtu NN] [[-]trailers] [[-]arp]
        [[-]allmulti] [multicast] [[-]promisc] [txqueuelen NN]
        [[-]dynamic] [up|down] ...
```

To set the IP address of the eth0 interface, type:

`ifconfig eth0 <address>`

To set the net mask type, type:

```
ifconfig eth0 netmask <address>
```

To deactivate the network interface, type:

```
ifconfig eth0 down
```

To reactivate the network interface, type:

```
ifconfig eth0 up
```

dmesg

The dmesg (diagnostic message) command prints the kernel's message buffer, which contains important messages, from those printed during boot to those used for debugging software.

Use dmesg in combination with "more" to obtain an overview of the events:

```
dmesg | more
```

vi Editor

Vi is a powerful screen-oriented text editor for creating and editing text files. To edit an existing file, type:

```
vi <filename>
```

thttpd HTTP Server

Thttpd is a simple, small, single-threaded, fast, and secure HTTP server. Type **`thttpd --help`** to obtain a list of available command-line parameters:

```
root:~# thttpd --help
usage:  thttpd [-p port] [-d dir] [-r|-nor] [-u user]
        [-c cgipat] [-t throttles] [-h host] [-l logfile]
```

In the home directory of the root user (/root) is a shell script, http_server.sh, that you can use to start the thttpd server with the default configuration. This script sets the document root of the server to /var/www and declares all files located in the /var/www/cgi-bin directory as CGIs that will be executed by the server. To change these settings, edit the script by using the vi editor.

As long as the script is running, the HTTP server is available. You can stop the server and therefore the script by pressing Ctrl-C. Optionally, you can start the server in the background by executing ./http_server.sh &. Use "ps" and "kill" afterwards if you want to shut down the server.

Start the server by executing this script:

```
root:~# ./http_server.sh &
```

Start your preferred Web browser and open the URL http://192.168.0.100/. If the server has been started successfully, a test Web page should be provided by the board.

Hardware Drivers

Even the hardware drivers are accessible through the command line. This section describes how to set the LEDs and the 7-segment LED by using the “echo” command.

Control LEDs

The LED driver is used to set and unset the board’s dedicated status LEDs. Each LED is addressed through a separate file.

Change to the /sys/class/leds directory and execute “ls”:

```
root:~# cd /sys/class/leds
root:/sys/class/leds# ls
leds-ecp250_0  leds-ecp250_2  leds-ecp250_4  leds-ecp250_6
leds-ecp250_1  leds-ecp250_3  leds-ecp250_5  leds-ecp250_7
root:/sys/class/leds#
```

This directory contains eight subdirectories, one for each LED. To set or unset a LED, write to the brightness file in the corresponding directory.

Write 0 to the file to set, or turn on, the LED, and 1 to unset, or turn off, the LED. If you use the “echo” command for writing, you must omit the default new-line character at the end of the line with the -n parameter, as follows:

```
root:/sys/class/leds# echo -n "1" > leds-ecp250_0/brightness
root:/sys/class/leds# echo -n "1" > leds-ecp250_3/brightness
root:/sys/class/leds# echo -n "0" > leds-ecp250_3/brightness
```

Control 7-Segment LED

The delivered Linux port to LatticeMico32 System includes a driver to control the 7-segment display element of the LatticeECP2-50 evaluation board.

To control the LED, a four-byte-long string without a trailing new-line character must be written to the /dev/ecp250_7seg_0 file. The first two bytes control the left element, and the second two bytes control the right element. You cannot set both segments simultaneously. Only one of the two segments can be activated at a time, either the left or the right one.

Each two-byte instruction must conform to following grammar, in regular Perl expression syntax:

```
Instruction  :=  [ 0-9][ \.]
```

The first byte sets the value of the segment. A space turns the segment off. The second byte sets or unsets the dot of the corresponding segment.

Here are some examples of these instructions:

- “7.” is on the left segment, and the right segment is turned off:

  ```
  echo -n "7.  " > /dev/ecp250_7seg_0
  ```

  The -n parameter prevents a new line (`'\n'`) at the end of the string.

- The left segment is turned off, and the right segment is set to “4” without a dot:

```
echo -n "  4 " > /dev/ecp250_7seg_0
```

Demonstration Applications

Small and simple demonstration applications have been implemented to demonstrate the power of Linux in the Linux port to LatticeMico32 System. The first demonstration is a CGI Web application written in C to set and unset the board’s LEDs through a Web browser. The second demonstration consists of a server and client applications that communicate through a named pipe to control the LEDs.

LED Switch Web Application

The LED switch Web application demonstration consists of a CGI application that you can use to set or unset the board’s dedicated LEDs. The application is provided by an HTTP server and is controlled through a standard Web browser. It demonstrates how to use a Web application running on Linux to perform hardware-related tasks.

Description

You can find the source files for this demonstration in user/lm32_demo_leds. You can include and exclude these files from the build by configuring the set of userland applications to be built by the “menuconfig” command. See the “Configuring Userland Applications” section of the *Linux Port to LatticeMico32 System Reference Guide* for information.

The tree includes three significant files:

- lm32_leds.c – Contains the main function and functions to read and control the LED state
- cgi_proc.c – Contains functions to read and parse the HTTP GET variables, that is, the QUERY_STRING environment variable
- template.c – Contains the HTML templates

Building the demonstration generates a lm32_leds.cgi file that is executed by the Web server.

Configuring the Demonstration

The application is configured by modifying the header files. Template.h defines a macro that places the CGI binary on the Web server. Its default name is /cgi-bin/lm32_leds.cgi. The lm32_leds.h file contains the prefix and suffix for accessing the brightness file of the LED driver. Change this macro when using another driver.

Usage

Start the thttpd Web server by executing the /root/http_server.sh script. Start your preferred Web browser and open the URL http://192.168.0.100/cgi-bin/lm32_leds.cgi. A demonstration Web application for turning the board's LEDs on or off should appear. Click the switch links to switch the state of the corresponding LED.

LED Switch Command-Line Application

The LED switch command-line application demonstration consists of a stand-alone server and a client application communicating over a named pipe. The server application (lm32_led_server) waits for commands from the client (lm32_led_client) to control the board's LEDs.

Description

You can find the source files for this demonstration in user/lm32_cli_leds. You can include and exclude these files from the build by configuring the set of userland application to be built by the "menuconfig" command. See the "Configuring Userland Applications" section of the *Linux Port to LatticeMico32 System Reference Guide* for information.

The tree includes three significant C files:

- lm32_leds_client.c – Client application that parses the command-line arguments and writes to the named pipe
- lm32_leds_server.c – Server application that creates the named pipe, reads commands from it, and sets the LED states
- lm32_leds.c – File containing functions that read and parse the HTTP GET variables, that is, the QUERY_STRING environment variable

Building the demonstration generates two binaries: lm32_led_client, which is the client application, and lm32_led_server, which is the server.

Configuring the Demonstration

Modify the header files to configure the application. The lm32_cli_leds.h file defines a macro that specifies the name of the pipe file, /tmp/lm32_cli_leds, which is used by the server and the client. The lm32_leds.h file contains the prefix and suffix for accessing the brightness file of the LED driver. Change this macro when using another driver.

Usage

Open two telnet connections to the board, and log in to both. In the first telnet window, start the lm32_led_server application. In the second window, run lm32_led_client without a parameter to obtain the usage message:

```
root:~# lm32_led_client
LatticeMico32 Command-Line LED Demo - Client
Usage:
  lm32_led_client index [on|off]

Switch LED with the given index on or off. If no state is
given, the LED state is set to the opposite of its current
state.
```

Use this client to communicate with the server. The first parameter defines the index of the LED. If no second parameter is given, the LED state is set to the opposite of its current state. To explicitly set the state, use the second parameter. On tells the server to set the LED, and off tells the server to turn off the LED.

Benchmarking the Board

A benchmark is used to assess the performance of the board by comparing it to the performance of other platforms. This section describes the Netperf and Httperf benchmarking applications. These benchmarks are mainly used to measure the performance of network-related tasks.

Netperf

Netperf is a benchmark that you can use to measure the performance of many different types of networking. It provides tests for both unidirectional throughput and end-to-end latency.

You can obtain Netperf at http://www.netperf.org/, but since the application uses fork() function calls, it is not usable on architectures lacking an MMU without modifications. For this reason, a pre-built Netperf binary for the LatticeMico32 architecture is available at http://www.theobroma-systems.com/mico32/. Download this binary to perform the benchmark.

For this version, all fork() function calls have been replaced by vfork(). However, the application does not shut down after performing the tests, so you must terminate it manually.

As a starting point, here is a list of example tests:

```
root:~# ./netperf -p 9999
```

```
root:~# ./netperf -t UDP_STREAM -p 9999 -- -m 32768
root:~# ./netperf -p 9999 -H 192.168.0.1
root:~# ./netperf -p 9999 -H 192.168.0.1 -t UDP_STREAM -- -m
        32768
```

In addition, you can run Netperf from your host machine to benchmark the board's performance. Download the original version of Netperf and build the binary by using the default compiler.

Use the following tests as a starting point:

```
$ ./netperf -p 9999 -H 192.168.0.100
$ ./netperf -t UDP_STREAM -p 9999 -H 10.0.0.201 -- -m 32768
```

Httperf

Httperf is a tool for measuring Web server performance. It provides a flexible utility for generating various HTTP workloads and for measuring server performance. Download Httperf from http://www.hpl.hp.com/research/linux/httperf/ and build the test suite for your host machine. Httperf is executed only on the host machine. You do not have to build a version for LatticeMico32 System using the cross-compiler.

Read the manual on the home page of the project to obtain a description of the suite and an overview of the extensive list of command-line parameters that are available. Perform the tests in which you are most interested.

As a starting point for benchmarking the board by using Httperf, here is a short list of possible test calls:

- To test functionality with one single connection, type the following:

  ```
  $ httperf --hog --server 192.168.0.100
  ```

- To test functionality with 100 connections, 20 connections per second, and 5 seconds of timeout, type the following:

  ```
  $ httperf --hog --server 192.168.0.100 \
   --num-conn=100 --ra 20 --timeout 5
  ```

- To test functionality with 10 interleaved sessions, each with 5 requests, 2 seconds between requests, and the creation of a new session each second, type the following:

  ```
  $ httperf --hog --server 192.168.0.100 \
  --wsess=10,5,2 --rate 1 --timeout 5
  ```

Index