Building the Linux Port to LatticeMico32 System User Guide

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

February 2008

Copyright

Copyright © 2008 Lattice Semiconductor Corporation.

This document may not, in whole or part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form without prior written consent from Lattice Semiconductor Corporation.

Trademarks

Lattice Semiconductor Corporation, L Lattice Semiconductor Corporation (logo), L (stylized), L (design), Lattice (design), LSC, E^2CMOS, Extreme Performance, FlashBAK, flexiFlash, flexiMAC, flexiPCS, FreedomChip, GAL, GDX, Generic Array Logic, HDL Explorer, IPexpress, ISP, ispATE, ispClock, ispDOWNLOAD, ispGAL, ispGDS, ispGDX, ispGDXV, ispGDX2, ispGENERATOR, ispJTAG, ispLEVER, ispLeverCORE, ispLSI, ispMACH, ispPAC, ispTRACY, ispTURBO, ispVIRTUAL MACHINE, ispVM, ispXP, ispXPGA, ispXPLD, LatticeEC, LatticeECP, LatticeECP-DSP, LatticeECP2, LatticeECP2M, LatticeMico8, LatticeMico32, LatticeSC, LatticeSCM, LatticeXP, LatticeXP2, MACH, MachXO, MACO, ORCA, PAC, PAC-Designer, PAL, Performance Analyst, PURESPEED, Reveal, Silicon Forest, Speedlocked, Speed Locking, SuperBIG, SuperCOOL, SuperFAST, SuperWIDE, sysCLOCK, sysCONFIG, sysDSP, sysHSI, sysI/O, sysMEM, The Simple Machine for Complex Design, TransFR, UltraMOS, and specific product designations are either registered trademarks or trademarks of Lattice Semiconductor Corporation or its subsidiaries in the United States and/or other countries. ISP, Bringing the Best Together, and More of the Best are service marks of Lattice Semiconductor Corporation.

HyperTransport is a licensed trademark of the HyperTransport Technology Consortium in the U.S. and other jurisdictions.

Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

Disclaimers

NO WARRANTIES: THE INFORMATION PROVIDED IN THIS DOCUMENT IS “AS IS” WITHOUT ANY EXPRESS OR IMPLIED WARRANTY OF ANY KIND INCLUDING WARRANTIES OF ACCURACY, COMPLETENESS, MERCHANTABILITY, NONINFRINGEMENT OF INTELLECTUAL PROPERTY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL LATTICE SEMICONDUCTOR CORPORATION (LSC) OR ITS SUPPLIERS BE LIABLE FOR ANY DAMAGES WHATSOEVER (WHETHER DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL, INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OF OR INABILITY TO USE THE INFORMATION PROVIDED IN THIS DOCUMENT, EVEN IF LSC HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. BECAUSE SOME JURISDICTIONS PROHIBIT THE EXCLUSION OR LIMITATION OF CERTAIN LIABILITY, SOME OF THE ABOVE LIMITATIONS MAY NOT APPLY TO YOU.

LSC may make changes to these materials, specifications, or information, or to the products described herein, at any time without notice. LSC makes no commitment to update this documentation. LSC reserves the right to discontinue any product or service without notice and assumes no obligation

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

Building the Linux Port to LatticeMico32 System User Guide

This guide explains how to build a Linux-based system for the LatticeMico32 architecture. It shows you how to build the LatticeMico32 tool chain, uClibc, U-Boot, Linux kernel, and Linux userland.

These instructions assume that you have successfully configured the build environment, as described in the *Linux Port to LatticeMico32 System Reference Guide*. They also assume that you will evaluate Linux on a LatticeECP2-50-based evaluation board programmed with the fully featured pre-generated bitstream provided.

The source package is released as a tarball package named lm32linux-<*yyyymmdd*>.tar.gz.

Building the Port in the Linux Environment

This section describes how to build the Linux port to LatticeMico32 System in the Linux environment.

Required Tools

To successfully build the entire system, you must have access to the following tools:

- binutils
- bison
- diffutils
- flex
- gettext
- GCC

- groff
- libtool
- make version 3.8 or later
- ncurses
- readline
- tar and gzip for unpacking

Required Packages

- LatticeMico32 Linux source package: lm32linux-<*yyyymmdd*>.tar.gz
- LatticeMico32 Linux pre-built tool chains for Linux: lm32linux-<*yyyymmdd*>toolchains_linux.tar.gz
- MSBConfigParser.jar Java package

Building the Port

The build process is composed of a sequence of make commands. See “Building Targets” on page 8 for a detailed description of the make targets used in this process.

Before you perform the following stepsk, make sure that the VERBOSE environment variable is not set. To unset VERBOSE, enter this command if you are using csh:

```
unsetenv VERBOSE
```

If you are using bsh, enter the following command:

```
export -n VERBOSE
```

To build the Linux port to LatticeMico32:

1. Remove the source tarball from the archive in the preliminary generated Cygwin managed mount (see “Required Packages” on page 4):

   ```
   tar -xzf lm32linux-<yyyymmdd>.tar.gz
   ```

2. If you intend to build the LatticeMico32 Linux tool chain from scratch, which takes approximately five hours, skip steps 3 through 9, following.
3. Remove the pre-built LatticeMico32 Linux tool chain build for Linux from the archive in the preliminary generated Cygwin managed mount:

   ```
   tar –xzf lm32linux-<yyyymmdd>toolchains_linux.tar.gz
   ```

4. Copy the pre-built tool chain into the sources directory:

   ```
   cp -R lm32linux-<yyyymmdd>toolchains_linux/* ./lm32linux-
   <yyyymmdd>/
   ```

5. Create the dist directory in MSBConfigParser directory in the toolchains directory:

   ```
   mkdir ./lm32linux-<yyyymmdd>/toolchains/MSBConfigParser/dist
   ```

6. Copy MSBConfigParser.jar in the dist directory created in the previous step:

   cp <*path_to_MSBConfigParser.jar*> ./lm32linux-<*yyyymmdd*>/toolchains/MSBConfigParser/dist/

7. Export the CLASSPATH variable to provide all the path name information to the MSBConfigParser.jar file. If you are using csh, enter the following:

   setenv CLASSPATH /<*full_path*>/lm32linux-<*yyyymmdd*>/toolchains/MSBConfigParser/dist/MSBConfigParser.jar

   If you are using bsh, enter the following:

   export CLASSPATH=/<*full_path*>/lm32linux-<*yyyymmdd*>/toolchains/MSBConfigParser/dist/MSBConfigParser.jar

8. Move to the Linux sources directory:

   cd lm32linux-<*yyyymmdd*>

9. Execute the install_toolchains.sh script:

   ```
   ./install_toolchains.sh
   ```

   This script modifies the KERNEL_HEADERS variable value in the toolchains/installdir/config.uClibc file to reflect the correct path, for example, KERNEL_HEADERS="/home/<user>/lm32linux/lm32linux-<*yyyymmdd*>/linux-2.6.x/include."

10. Move to the created directory:

    cd lm32linux-<*yyyymmdd*>/

11. If you are not building the tool chain from scratch, skip this step. To build the LatticeMico32 tool chain, type the following command:

    make vendor_toolchains

12. Build U-Boot for the fully featured pre-generated bitstream:

    make u_boot

    Note

    If you want to build U-Boot for a custom bitstream, see "Building Targets" on page 8.

13. Configure the build according to your needs by executing the following command:

    make menuconfig

14. Build all dependencies with this command:

    make dep

15. Build U-Boot, the Linux kernel, and the Linux userland, and generate the images and the ROM file system:

    make

The build process is completed.

Generated Files

The resulting Linux U-Boot vmlinux.img image and the ext2 initial RAM disk U-Boot initrd.img image are placed in the images/ subdirectory of the source directory. You can find the file system in the romfs/ directory.

Building the Port in the Windows Environment

This section describes how to use Cygwin to build the Linux port to LatticeMico32 System in the Windows host environment.

Building the Linux port to LatticeMico32 System in a Windows environment requires you to install Cygwin. You can obtain the latest Cygwin release from http://www.cygwin.com. The Cygwin setup.exe application is used to install the base system, as well as the pre-compiled GNU tools provided by Cygwin.

Several tools in Cygwin are required:

- Base packages installed by default
- binutils
- bison
- diffutils
- flex
- gettext
- GCC
- gcc-core
- groff
- libiconv
- ibtool
- make version 3.8 or later
- ncurses
- readline
- tar and gzip for unpacking

Required Packages

- LatticeMico32 Linux source package: lm32linux-<*yyyymmdd*>.tar.gz
- LatticeMico32 Linux pre-built tool chains for Windows: lm32linux-<*yyyymmdd*>toolchains_w32_c.tar.gz
- MSBConfigParser.jar Java package

Preliminary Tasks

Since the names of some files in the tarball are spelled the same way but have different case, the tarball must be unpacked on a case-sensitive Cygwin managed mount.

To create a new directory and unpack it on a "managed" mount:

1. Type the following command to create a directory named lm32linux (<*name*> is the login user name):

   mkdir /home/<*user*>/lm32linux

2. To map this directory as a managed mount, enter the following command:

   mount -o managed C:/cygwin/home/<*user*>/lm32linux/ /home/<*user*>/lm32linux

   C:/Cygwin in this example must eventually be replaced by the correct Windows file system location of the Cygwin root directory on the host system.

   Managed mount is an experimental feature of Cygwin and should be used only for directories that are initially unpopulated and are going to be completely managed by Cygwin.

3. To verify a successful managed mount, enter the following command:

   mkdir /home/<*user*>/lm32linux/tmp:0.0

If the mount command was successful, the mkdir command will succeed. If the mount command was unsuccessful the mkdir command will fail.

Building the Port

The build process is composed of a sequence of “make” commands. See “Building Targets” on page 8 for a detailed description of the make targets used in this process.

To build the port:

1. Remove the source tarball from the archive in the preliminary generated Cygwin managed mount (see “Required Packages” on page 4):

   tar -xzf lm32linux-<*yyyymmdd*>.tar.gz

2. If you intend to build the LatticeMico32 Linux tool chain from scratch, which takes approximately five hours, skip steps 3 through 9, following.

3. Remove the pre-built LatticeMico32 Linux tool chain build for Windows from the archive in the preliminary generated Cygwin managed mount:

   tar –xzf lm32linux-<*yyyymmdd*>toolchains_w32_c.tar.gz

4. Copy the pre-built tool chain into the sources directory:

   cp -R lm32linux-<*yyyymmdd*>toolchains_w32/* ./lm32linux-<*yyyymmdd*>/

5. Create the dist directory in MSBConfigParser directory in the toolchains directory:

   mkdir ./lm32linux-<*yyyymmdd*>/toolchains/MSBConfigParser/dist

6. Copy MSBConfigParser.jar in the dist directory created in the previous step:

   cp <*path_to_MSBConfigParser.jar*> ./lm32linux-<*yyyymmdd*>/toolchains/MSBConfigParser/dist/

7. Export the CLASSPATH variable to provide all the path name information to the MSBConfigParser.jar file:

   export CLASSPATH=/home/<*user*>/lm32linux/lm32linux-<*yyyymmdd*>/toolchains/MSBConfigParser/dist/MSBConfigParser.jar

8. Move to the Linux sources directory:

   cd lm32linux-<*yyyymmdd*>

9. Execute the install_toolchains.sh script:

   ```
   ./install_toolchains.sh
   ```

   This script modifies the KERNEL_HEADERS variable value in the toolchains/installdir/config.uClibc file to reflect the correct path, for example, KERNEL_HEADERS="/home/<user>/lm32linux/lm32linux-<*yyyymmdd*>/linux-2.6.x/include."

10. If you are not building the tool chain from scratch, skip this step. Enter the following commands to build the tool chain (build time is approximately five hours) in the lm32linux-<*yyyymmdd*> directory:

    make vendor_toolchains

    This command produces a lot of output. The last lines of the output should like this:

    ```
    ...
    /usr/bin/install -c -m 644 './genext2fs.8'
    '/<your_build_directory>/lm32linux-beta20071012/toolchains/
    genext2fs/share/man/man8/genext2fs.8'
    make[2]: Leaving directory
    `/<your_build_directory>/lm32linux-beta20071012/toolchains/
    genext2fs'
    make[1]: Leaving directory
    `/<your_build_directory>/lm32linux-beta20071012/toolchains/
    genext2fs'
    ```

11. Build U-Boot for the fully featured pre-generated bitstream.

    make u_boot

    Note

    If you want to build U-Boot for a custom bitstream, see "Building Targets" on page 8.

12. Configure the build according to your needs by executing the following command:

    make menuconfig

13. For default settings, you must do the following:

    a. From the Kernel/Library/Defaults Selection menu, select the **Default all settings (lose changes)** checkbox.

b. Choose **Exit** twice.

c. Click **Yes** to save all changes.

The last lines of the expected output are the following:

```
...
Compile all sources at once into an object (DOMULTI) [N/y/?]
n
Manuel's hidden warnings (UCLIBC_MJN3_ONLY) [N/y/?] n
#
# configuration written to .config
#
make[2]: Leaving directory `/<your_build_directory>/
lm32linux-beta20071012/uClibc'
make[1]: Leaving directory `/<your_build_directory>/
lm32linux-beta20071012'
```

14. Build U-Boot, the Linux kernel, and the Linux userland, and generate the images and the ROM file system:

    make

The build process is completed. The lines expected after the build process is finished are as follows:

```
/<your_build_directory>/lm32linux-beta20071012/u-boot/build/
tools/mkimage
-T ramdisk -n 'Lattice ext2 initrd U-Boot Image' -A LM32 -C
none \
  -d
  /<your_build_directory>/lm32linux-beta20071012/images/
  romfs.ext2
  /<your_build_directory>/lm32linux-beta20071012/images/
  initrd.img
  Image Name:  Lattice ext2 initrd U-Boot Image
  Created:     Wed Oct 24 13:22:56 2007
  Image Type:  LatticeMico32 Linux RAMDisk Image (uncompressed)
Data Size:    11776000 Bytes = 11500.00 kB = 11.23 MB
Load Address: 0x00000000
Entry Point:  0x00000000
make[2]: Leaving directory
`/<your_build_directory>/lm32linux-beta20071012/vendors/
Lattice/ECP250'
make[1]: Leaving directory
`/<your_build_directory>/lm32linux-beta20071012/vendors'
```

The most important resulting files, initrd.img and vmlinux.img, are stored in the images directory. You can find the Linux kernel with debugging symbols in linux-2.6.x/vmlinux. The U-Boot binary with and without debugging symbols is built into the u-boot/build directory.

Generated Files

The resulting Linux U-Boot vmlinux.img image and the ext2 initrd U-Boot initrd.img image, as well as the ROM file system romfs.ext2 file, are placed in the images/ subdirectory of the source directory.

Building the Port in the MacOS X Environment

The build process on MacOS X is identical to the build process on a Linux-based environment, described in “Building the Port in the Linux Environment” on page 1. The same tools and utilities are required, and the same commands are executed in order to build the whole system.

Since the names of some of the files in the tarball are spelled the same way but have different case, you must unpack the tarball on a case-sensitive file system. Create a new disk image and format it with a case-sensitive file system by using the Mac OS application disk utility.

Building Targets

The tarball release package contains all the parts necessary to build a Linux-based environment for the LatticeMico32 platform, including an extensive userland. The release package provides a root makefile to (re)build certain parts or the entire system environment. This section lists the available make targets that are either required or useful in the build process.

- make vendor_toolchains – This target (re)builds the entire LatticeMico32 tool chain, including (in order):
  - The binutils
  - A minimal GCC
  - A non-OS-aware GDB
  - uClibc with minimal GCC
  - A GCC with uClibc and libstdc++ support

  All of these tools are installed in the toolchains/installdir/ directory.

  Note

  Rebuilding the entire toolchain is a very time-consuming process.

- make vendor_toolchains_clean – This target removes the tool chain build directory and must be used if all tool chains are to be rebuilt.
- make vendor_toolchains_uninstall – Removes the tool chain tools from the installation directory (toolchains/installdir/) but preserves the build directory, which can be removed by make vendor_toolchains_clean.
- make menuconfig – Starts a console-based application to configure the build. The top-level menu consists of two items: Vendor/Product and Kernel/Library/Defaults.

In the first selection, the vendor (in this case, Lattice Semiconductor) and the product family (for example, LatticeECP2-50) of the target board is specified. The defaults are Lattice Semiconductor and LatticeECP2-50, respectively. The latter selection is used to configure the Linux kernel.

Note

Check the Default All Settings option the first time that you execute the build process.

To finish the configuration process, exit the menuconfig application and save the new configuration (answer the final question with **Yes**). Afterwards the configuration file will be distributed to the directories that require it.

- make u_boot – Builds U-Boot for the fully configured pre-generated bitstream. It depends on the successful execution of “make vendor_toolchains” (LatticeMico32 tool-chain build).
- make u_boot_custom – Builds U-Boot for a custom bitstream. For detailed instructions on configuring U-Boot for a custom bitstream, see the “U-Boot” chapter of the *Linux Port to LatticeMico32 System Reference Guide*.
- make linux – Builds the Linux kernel. It depends on the successful execution of “make vendor_toolchains” (LatticeMico32 tool-chain build).
- make romfs – Generates the ROM file system in the romfs/ directory. It depends on the successful execution of “make linux” (Linux kernel build) and the userland to be completely built.
- make image – Generates the Linux U-Boot vmlinux.img image and the ext2 initrd U-Boot initrd.img image. It depends on the successful execution of “make linux” (Linux kernel build), “make u-boot” (U-Boot build), and “make romfs” (ROM file system generation). The files are placed in the images/ directory.
- make – Do not execute “make” without a target without first executing “make vendor_toolchains,” “make menuconfig,” and “make dep.” “Make” performs the following tasks:
  - Builds the Linux kernel
  - Builds the Linux userland
  - Generates the file system in the romfs/ directory
  - Generates the ext2 romfs.ext2 file system file
  - Generates the Linux U-Boot vmlinux.img image
  - Generates ext2 initrd U-Boot initrd.img image

  The three files are placed in the images/ directory.

The required tools and the recommended sequence of targets for a complete build of the Linux port to LatticeMico32 depend on the host architecture and are addressed in “Building the Port in the Linux Environment” on page 1, “Building the Port in the Windows Environment” on page 4, and “Building the Port in the MacOS X Environment” on page 8.

Index