LatticeMico8 Development Tools User Guide

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

October 2010

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.

Lattice Semiconductor Corporation

Lattice Semiconductor Corporation

LatticeMico8 Development Tools User Guide

Introduction

The LatticeMico8 is an 8-bit microcontroller optimized for Field Programmable Gate Arrays (FPGAs) and Crossover Programmable Logic Device architectures from Lattice. Combining a full 18-bit wide instruction set with 32 General Purpose registers, the LatticeMico8 is a flexible reference design written in Verilog and VHDL suitable for a wide variety of markets, including communications, consumer, computer, medical, industrial, and automotive. The core consumes minimal device resources, less than 200 Look Up Tables (LUTs) in the smallest configuration, while maintaining a broad feature set.

Lattice provides development tools, including a C language compiler, for users of the LatticeMico8 microcontroller. This document introduces readers to the LatticeMico8 tools and includes installation instructions, LatticeMico8 specific features of the tool chain, and details on the application binary interface that is implemented by the LatticeMico8 port of GNU binutils and GCC.

Toolset

The LatticeMico8 development tools consist of a LatticeMico8 port of version 4.4.3 of the GNU Compiler Collection (GCC) and version 2.18 of GNU Binary Utilities (binutils). These tools are a collection of command line executables hosted on a Linux/Unix or Cygwin environment. Cygwin provides a UNIX-like terminal emulation on the Windows platform.

These development tools include:

- Compiler
- Assembler
- Linker

- Librarian
- File converter
- Other file utilities
- C Library

Compiler

The compiler is the GNU Compiler Collection, or GCC version 4.4.3. The GCC included in this port is targeted for the Lattice Mico8 processor, and it is configured to compile C language programs. Go to http://gcc.gnu.org for more information on the GNU Compiler Collection.

Assembler, Linker, Librarian and More

The GNU Binary Utilities is a collection of programming tools for the manipulation of object code in various object file formats. Binutils were created to give the GNU system the facility to compile and link programs.

Binutils included in this port is configured for the LatticeMico8 target, and each of the programs is prefixed with the target name. So you have programs such as:

`lm8-elf-as` The Assembler

`lm8-elf-ld` The Linker

`lm8-elf-ar` Create, modify, and extract from archives (libraries).

`lm8-elf-ranlib` Generate index to archive (library) contents.

`lm8-elf-objcopy` Copy and translate object files.

`lm8-elf-objdump` Display information from object files including disassembly.

`lm8-elf-size` List section sizes and total size.

`lm8-elf-nm` List symbols from object files.

`lm8-elf-strings` List printable strings from files.

`lm8-elf-strip` Discard symbols.

`lm8-elf-readelf` Display the contents of ELF format files.

`lm8-elf-addr2line` Convert addresses to file and line.

See the binutils user manual for more information on what each program can do. Information on program options specific to the Lattice Mico8 processor appears later in this document. For more information refer to http://www.gnu.org/software/binutils/.

LatticeMico8 Development Tools Installation

This section describes how to install the LatticeMico8 Development Tools. You can download the tools from the Lattice Semiconductor Web site at

http://www.latticesemi.com/mico8/

The following sections describe the installation methods.

Installing LatticeMico8 Development Tools on a Windows PC

To install the LatticeMico8 Development Tools:

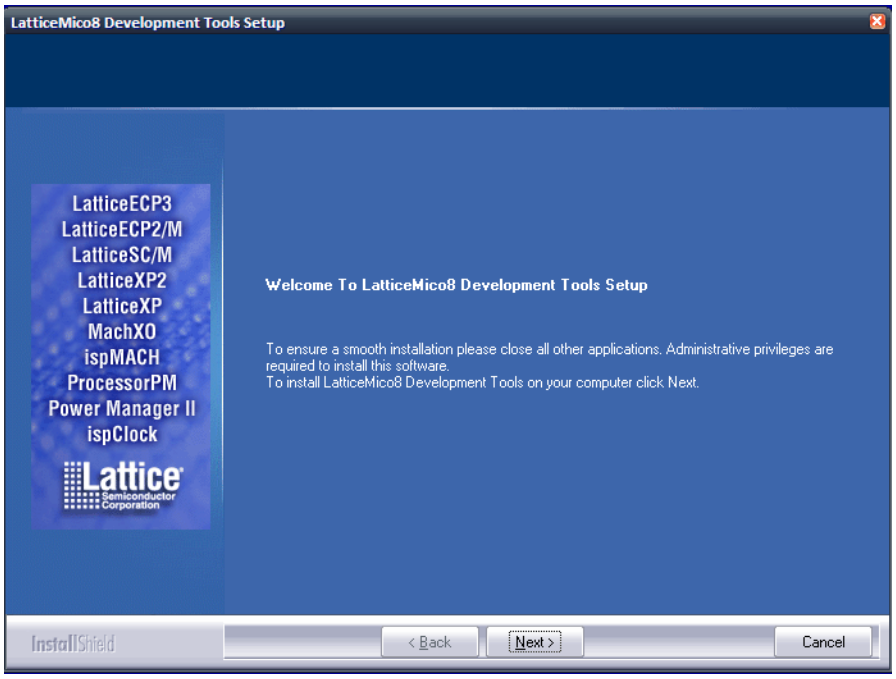
1. Go to the LatticeMico8 Web page at the following URL:

   http://www.latticesemi.com/mico8/

2. Download the LatticeMico8_<version_number>.exe executable.
3. Save the executable as a file.
4. Double-click the executable file name to begin the installation.

   The LatticeMico8 Development Tools Setup dialog box appears, as shown in Figure 1.

Figure 1: LatticeMico8 Development Tools Setup Dialog Box



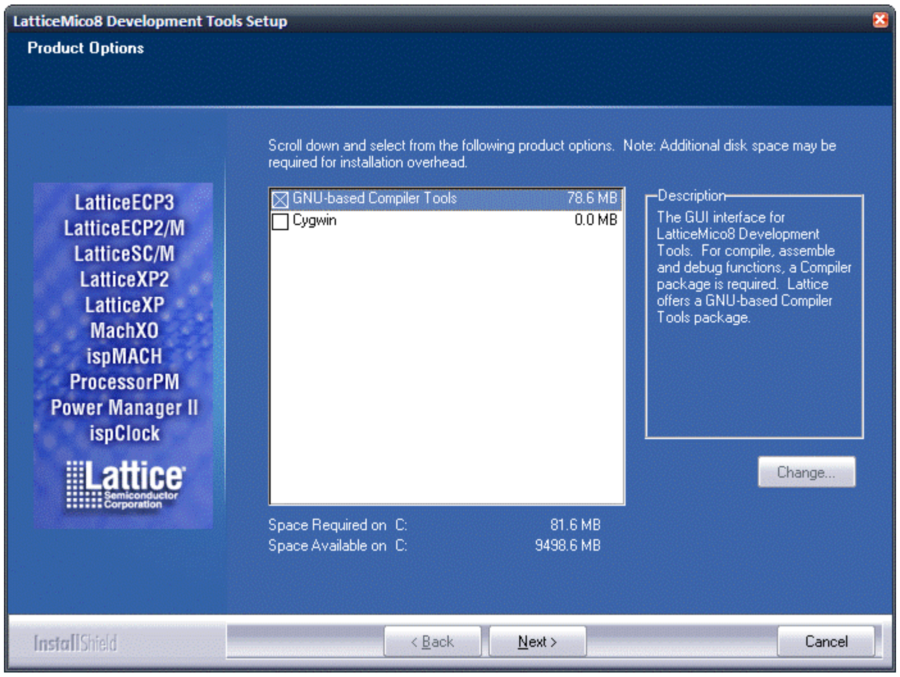
Welcome To LatticeMico8 Development Tools Setup

To ensure a smooth installation please close all other applications. Administrative privileges are required to install this software.
To install LatticeMico8 Development Tools on your computer click Next.

5. Click **Next**.

   The Product Options dialog box appears, as shown in Figure 2 on page 4.

Figure 2: Product Options Dialog Box



Scroll down and select from the following product options. Note: Additional disk space may be required for installation overhead.

☒ GNU-based Compiler Tools 78.6 MB
☐ Cygwin 0.0 MB

Description
The GUI interface for LatticeMico8 Development Tools. For compile, assemble and debug functions, a Compiler package is required. Lattice offers a GNU-based Compiler Tools package.

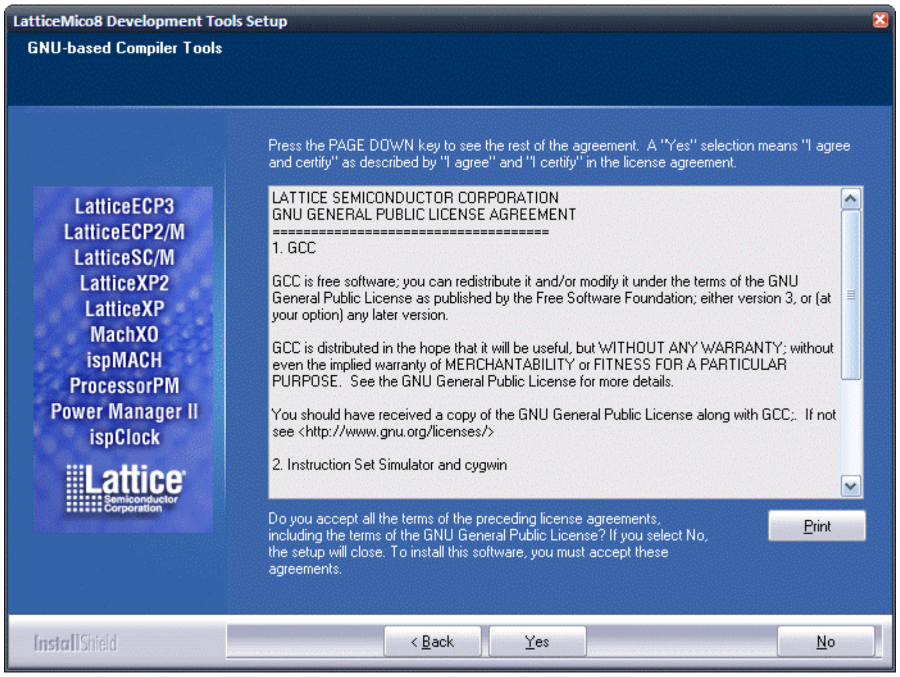
Space Required on C: 81.6 MB
Space Available on C: 9498.6 MB

When you select the GNU-based Compiler Tools option, the LatticeMico8 Development Tools install the C compiler tool chain. It is strongly recommended that you install Cygwin along with the GNU based compiler tools, because the compiler tools require Cygwin in order to be functional. This copy of Cygwin contains all the components that are required by the tools and will not interfere with any pre-existing installation of Cygwin.

6. Click **Next**.

7. Click **Yes** to accept the terms of the licensing agreement for LatticeMico8 Development Tools, shown in Figure 3.

Figure 3: Accepting the License for the GNU-Based Compiler Tools



Press the PAGE DOWN key to see the rest of the agreement. A "Yes" selection means "I agree and certify" as described by "I agree" and "I certify" in the license agreement.

LATTICE SEMICONDUCTOR CORPORATION
GNU GENERAL PUBLIC LICENSE AGREEMENT
==================================
1. GCC

GCC is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3, or (at your option) any later version.

GCC is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

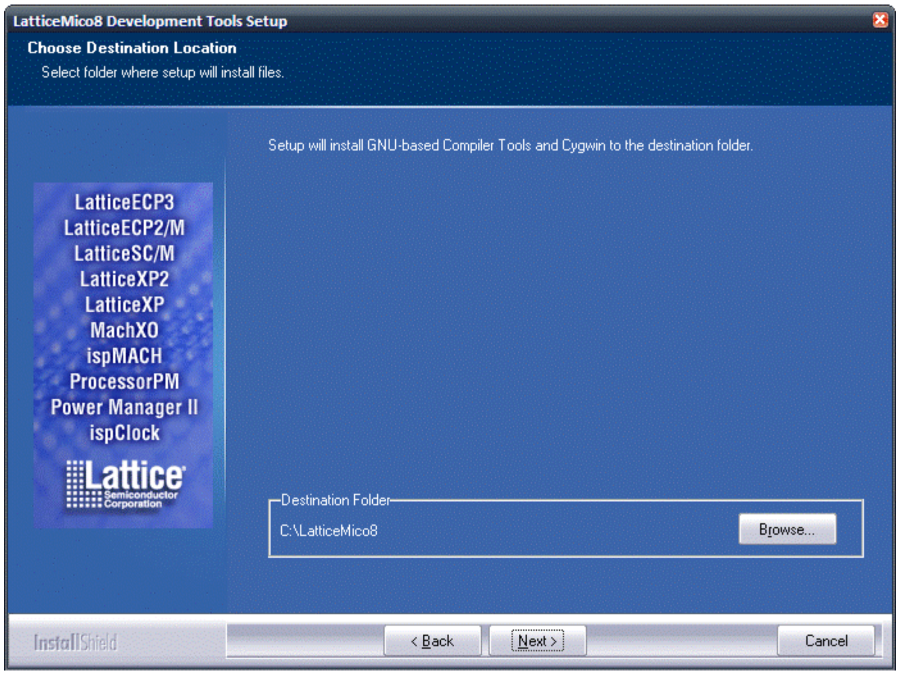
You should have received a copy of the GNU General Public License along with GCC;. If not see <http://www.gnu.org/licenses/>

2. Instruction Set Simulator and cygwin

Do you accept all the terms of the preceding license agreements, including the terms of the GNU General Public License? If you select No, the setup will close. To install this software, you must accept these agreements.

The Choose Destination Location part of the LatticeMico8 Development Tools Setup dialog box now appears, as shown in Figure 4 on page 6, so that you can choose the folder in which the LatticeMico8 Development Tools will be installed. The default destination folder is C:\LatticeMico8.

Figure 4: Selecting the Destination Directory


LatticeMico8 Development Tools Setup

Choose Destination Location

Select folder where setup will install files.

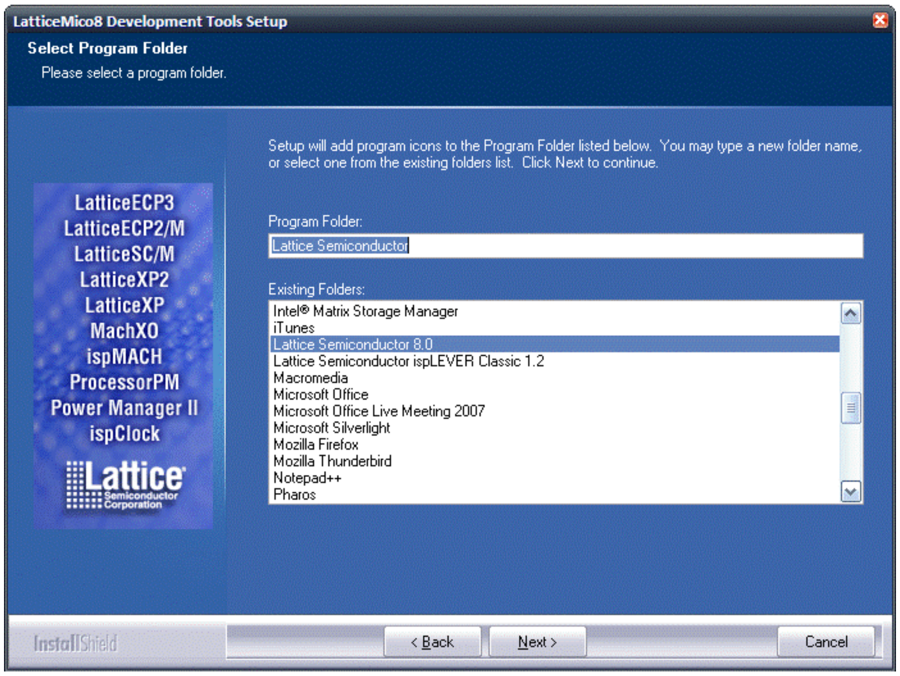
Setup will install GNU-based Compiler Tools and Cygwin to the destination folder.

Destination Folder: C:\LatticeMico8

8. To accept the default destination folder, click **Next**. Otherwise, click Browse to change the drive or destination folder, click **OK**, and follow the installation instructions on the screen.

9. In the Select Program Folder part of the dialog box, shown in Figure 5, select or type the name of the default program folder, which is the folder that contains the Lattice Semiconductor programs that you can choose through the Start menu. If ispLEVER is installed, the default folder is the same as that for ispLEVER.

Figure 5: Selecting the Program Folder



LatticeMico8 Development Tools Setup

Select Program Folder

Please select a program folder.

Setup will add program icons to the Program Folder listed below. You may type a new folder name, or select one from the existing folders list. Click Next to continue.

Program Folder:

Lattice Semiconductor

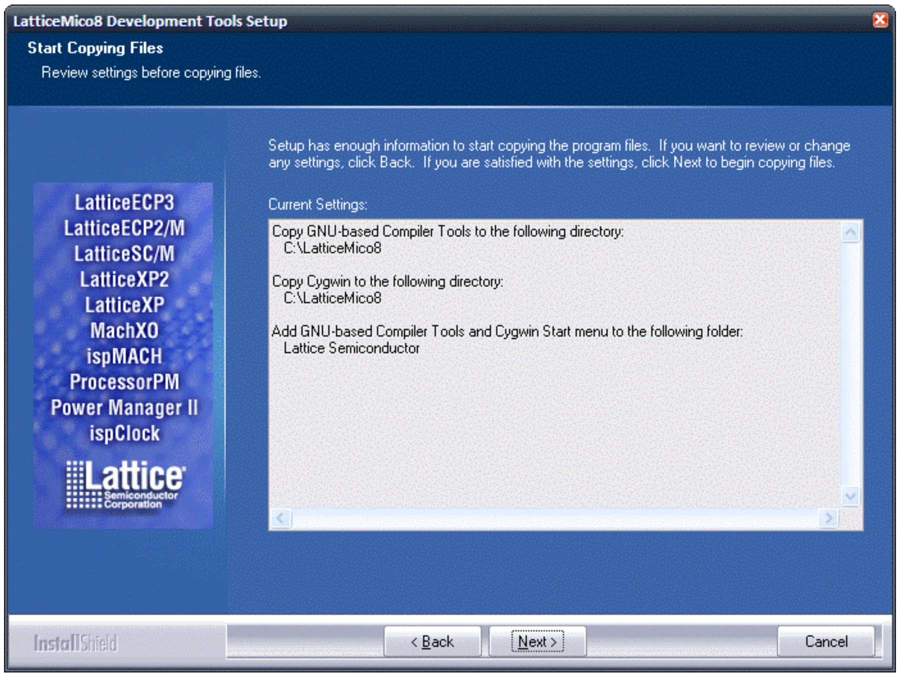
Existing Folders:

Intel® Matrix Storage Manager
iTunes
Lattice Semiconductor 8.0
Lattice Semiconductor ispLEVER Classic 1.2
Macromedia
Microsoft Office
Microsoft Office Live Meeting 2007
Microsoft Silverlight
Mozilla Firefox
Mozilla Thunderbird
Notepad++
Pharos

10. Click **Next**.

11. In the Start Copying Files part of the LatticeMico8 Development Tools Setup dialog box, shown in Figure 6 on page 8, click **Next**.

Figure 6: Starting the Installation



Setup has enough information to start copying the program files. If you want to review or change any settings, click Back. If you are satisfied with the settings, click Next to begin copying files.

Current Settings:

Copy GNU-based Compiler Tools to the following directory:
C:\LatticeMico8

Copy Cygwin to the following directory:
C:\LatticeMico8

Add GNU-based Compiler Tools and Cygwin Start menu to the following folder:
Lattice Semiconductor

The installation begins. When it is finished, the LatticeMico8 Development Tools Installation Completing dialog box appears.

12. Click **Finish**.

13. Update the path variable to include the bin folder of the compiler tools installation. If you installed the tools in the default destination folder, the bin folder is at C:\LatticeMico8\gtools\bin. The Path variable can be modified by following the procedure below.

    To modify the Path variable for Windows XP:

    a. Chose **Start > Control Panel > System**

    b. Select the **Advanced** tab of the System Properties dialoag box, and click **Environment Variables** .

    c. In the Environment Variables dialog box, under System Variables, scroll to PATH and double-click it.

    d. In the Edit System Variable dialog box, modify the variable value for PATH by adding the location of the bin folder to the value for PATH.

       If you do not have the item PATH, you can click **New** in the Environment Variables dialog box to add a new variable. Type **PATH** in the “Variable name” box and enter the location of the class in the “Variable value” box.

    e. Close the window.

To modify the Path variable for Windows Vista:

f. Right-click the **My Computer** icon and choose **Properties** from the pop-up menu.

g. Select the **Advanced** tab ("Advanced system settings" link in Vista)

h. In the Edit System Variable dialog box, modify the variable value for PATH by adding the location of the bin folder to the value for PATH.

   If you do not have the item PATH, you can click **New** in the Environment Variables dialog box to add a new variable. Type **PATH** in the "Variable name" box and enter the location of the class in the "Variable value" box.

i. Close the window.

This completes the devlopment tools installation.

To use the tools:

◆ Open a Cygwin shell by double-clicking the cygwin.bat found under [installation path]\LatticeMico8\cygwin\.

Uninstalling the LatticeMico8 Development Tools on Windows

The LatticeMico8 development tools can be unistalled using the Add/Remove Programs tool in the Control Panel.

Installing LatticeMico8 Development Tools on Linux

1. Go to the LatticeMico8 Web page at the following URL:

   http://www.latticesemi.com/mico8/

2. Download the LatticeMico8_lm8-i386-linux_<version_number>.rpm

3. Save the file to your home directory.

4. Open up a terminal window and run the following command:

   ```
   #rpm -ivh --prefix <install path> lm8-3-i386-linux.rpm
   ```

   The installation begins. When it is finished, the command prompt appears again.

   Note

   GCC requires the MPFR library to be installed. This library can be obtained as an RPM package from RedHat. Make sure that this library is installed before using the compiler tools. Attempting to compile without installing this library will result in an error message.

5. Set the PATH environment variable to include <install path>/gtools/bin.

   This completes the Linux installation process.

Uninstalling the LatticeMico8 Development Tools on Linux

◆ Run the following command to uninstall the LatticeMico8 Development Tools

```
#rpm –e lm8
```

Compiling a Simple Program

After installing the development tools, as described in the previous section, you should be ready to compile a C program. Any text editor can be used to enter a C program. The most popular text editors include vi, ed, cat, emacs, joe and kate. Save the following program to the file hello.c using your favorite text editor.

```
int main()
{
   return 0;
}
```

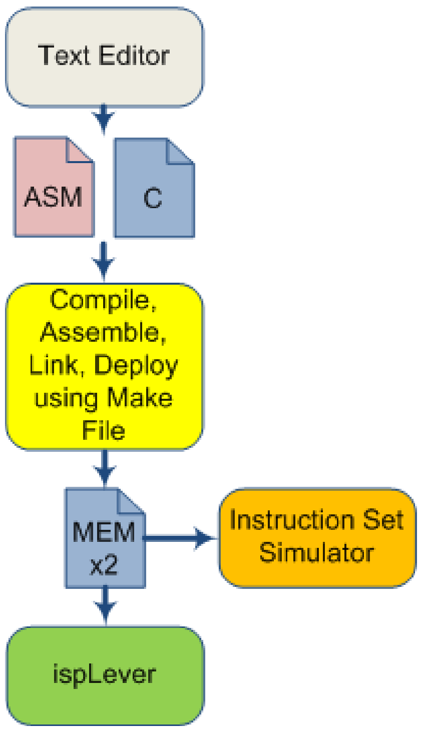
After entering the program, you can compile the program by executing the following command:

```
$ lm8-elf-gcc –Wall –Os –o hello.elf hello.c
```

The above command will compile hello.c, produce any warnings that the compiler deems fit (-Wall), optimize for size (-Os), and produce the output in the file hello.elf.

The contents of the hello file can now be assembled and linked. Once the linked object code is available, the memory file can be extracted from the linker output file and simulated through the Lattice Mico8 instruction set simulator or run on hardware. Figure 7 on page 11 shows the complete development flow. See “Making Your Software” on page 13 for information about the use of Make and makefiles.

Figure 7: LatticeMico8 Development Flow


Text Editor
ASM
C
Compile, Assemble, Link, Deploy using Make File
MEM x2
Instruction Set Simulator
ispLever

Linking Object Code

Object files are produced by a compiler as a result of processing the C or assembly source code file. Object files contain compact code and are often called "binaries."

The linker (lm8-elf-ld) is used to generate an executable or library by amalgamating parts of object files together.

Generating the .mem File

After linking your software, you can generate the .mem file to be included in your ispLever project by running the deployment tool. The deployment tool is a Perl script that extracts the LatticeMico8 executable program from the .ELF file.

Use the following command:

```
#perl lm8-deployer.pl <elf filename>
```

Simulation

The software tools for LatticeMico8 include an Instruction Set Simulator for the microcontroller, which allows programs developed for the microcontroller to be run and debugged on a host platform. The Simulator can also be used to generate a disassembly listing of a LatticeMico8 program. The Simulator takes as input the memory output file of the Assembler. It emulates the instruction execution of the LatticeMico8 in software. Note that the Simulator does not handle interrupts.

```
#isp8sim -option1 -option2 ... <prom filename>
<scratch pad filename>
```

Table 1: Command Line Options

Option	Comment
-p	<Program Rom Size>Default is 512 opcodes.
-ix	Program file is in hexadecimal format (default). This is the file generated by the Assembler with the -vx options (default).
-ib	Program file is in binary format. This is the file generated by the Assembler with the -vb option.
-t	Trace the execution of the program. The Simulator will generate a trace as it executes each instruction. It will also print the modified value of any register (if the instruction modifies a register value).
-d	Generate a disassembly of the program specified by the PROM file.

Example: Command Line, Compile to Simulator Consider the following sequence of commands:

```
#lm8-elf-gcc -Os -mcall-prologues -mcmodel=large -o test.elf
test.c

#perl lm8-deployer.pl test.elf

#isp8sim -p 2560 -s 16 -g 16 -ix prom_init.mem
scratchpad_init.mem > test.txtout
```

The options set for the program lm8-elf-gcc are only the ones specified. Of the options set, `-mcall-prologues` and `-mcmodel=large` are Mico8 specific. As stated in other parts of this document, `-mcmodel=` can have the values “small” “medium” or “large.”

“Small” will cause the compiler to use on 8-bit addressing, which limits you to 256 bytes of data.

“Medium” will cause the compiler to use 16-bit addressing, which is also the default when unspecified.

“Large” will cause 32-bit addressing to be used. Setting this option includes extending the stack and frame pointers to be 32 bits.

By using `-mcall-prologues`, the code size is reduced.

If `-mint8` is used, the common `int` data type will be 8-bits instead of the normal 16-bits.

If `-m16regs` is used, only the first 16 registers (r0 to r15) will be used by the compiler.

If `-mcall-stack-size=` is used, it sets the size of the call stack implemented in the target processor from the default of 16. This will control the depth that function calls can be made.

In the above example "-Os" was set to reduce the size of the code generated. If "-O0" is used, there will be no compiler optimization.

Making Your Software

The complete development flow can be automated by using the "GNU make." The "make" is a program that is widely used to build software. "Make" reads and executes makefiles, which are descriptions of how to build something. Makefiles typically do things such as group files together; set lists of compiler and linker flags; list rules of how to compile source code to object code, how to link object files, how to convert files from one type to another, and many other things.

The following is an example of a simple makefile for building LatticeMico8 software:

```
#----------------------------------------------------------
# Name of Project
#----------------------------------------------------------
PROJECT_NAME = Software
PROJECT_EXE  = $(PROJECT_NAME).elf

#----------------------------------------------------------
# Include Source Files of Project
#----------------------------------------------------------
include setup.mk
export

#----------------------------------------------------------
# Define Executables to be used for various stages of
Compilation.
#----------------------------------------------------------
CC = lm8-elf-gcc
LD = lm8-elf-gcc
AS = lm8-elf-as
AR = lm8-elf-ar
SZ = lm8-elf-size

#----------------------------------------------------------
# LM8 Memory Mode
#----------------------------------------------------------
MMODE := medium
```

```
#----------------------------------------------------------
# Preprocessor Flags
#----------------------------------------------------------
CPPFLAGS +=

#----------------------------------------------------------
# Compiler Flags
#----------------------------------------------------------
CFLAGS  = $(PFLAGS) -mcall-prologues -mcmodel=$(MMODE) -Os

#----------------------------------------------------------
# Assembler Flags
#----------------------------------------------------------
ASFLAGS = $(CFLAGS)

#----------------------------------------------------------
# Linker Flags
#----------------------------------------------------------
LDFLAGS = $(CFLAGS)

OBJS += $(sort $(CSRCS:.c=.o))
OBJS += $(sort $(ASRCS:.S=.o))

$(PROJECT_EXE): clean $(OBJS) $(HSRCS)
  @echo
  @echo building application...
  $(LD) -T lm8_linker_$(MMODE).ld -o $@ $(OBJS) $(LDFLAGS)
  rm -f *.o
  $(SZ) $(PROJECT_EXE)
  @echo
  @echo generating initialization files...
  perl lm8-deployer.pl ./ $(PROJECT_EXE)
  @echo
  @echo DONE

clean:
  @echo cleaning
  $(foreach object, $(OBJS), $(call fn_remove_file, $(object)))
  rm -f *.o
  rm -f *.elf
```

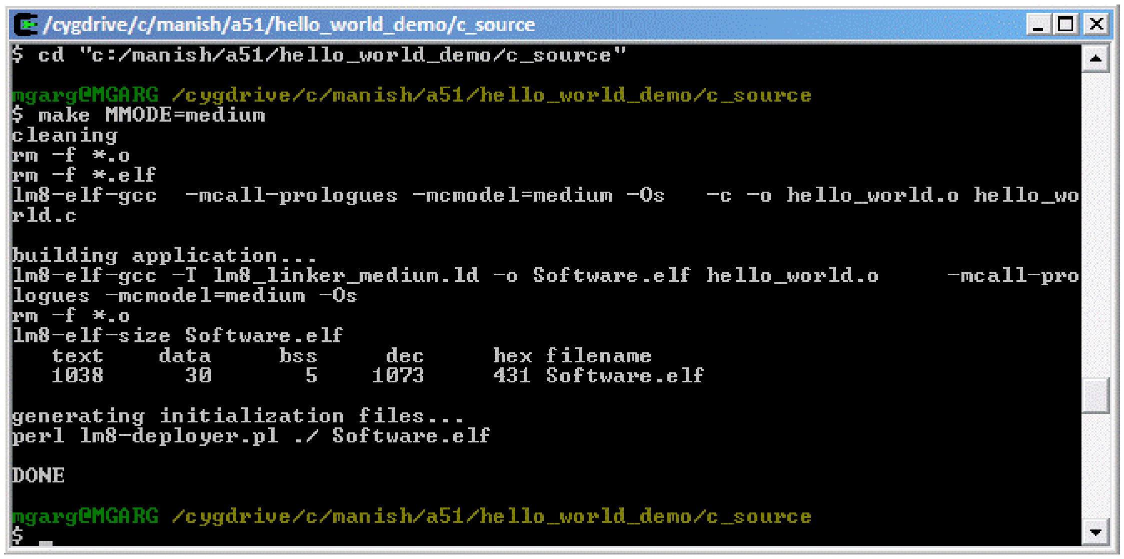
This makefile can be executed using the following command.

```
#make MMODE=medium
```

Figure 8 on page 15 shows the process of executing the makefile.

For more information on the make program and writing makefiles, see the “make” user manual at the GNU Manuals Online at http://www.gnu.org/software/make/.

Figure 8: Executing the makefile



```
$ cd "c:/manish/a51/hello_world_demo/c_source"

mgarg@MGARG /cygdrive/c/manish/a51/hello_world_demo/c_source
$ make MMODE=medium
cleaning
rm -f *.o
rm -f *.elf
lm8-elf-gcc  -mcall-prologues -mcmodel=medium -Os   -c -o hello_world.o hello_wo
rld.c

building application...
lm8-elf-gcc -T lm8_linker_medium.ld -o Software.elf hello_world.o     -mcall-pro
logues -mcmodel=medium -Os
rm -f *.o
lm8-elf-size Software.elf
   text    data     bss     dec     hex filename
   1038      30       5    1073     431 Software.elf

generating initialization files...
perl lm8-deployer.pl ./ Software.elf

DONE

mgarg@MGARG /cygdrive/c/manish/a51/hello_world_demo/c_source
$ _
```

C Library

Libgcc is the Standard C Library for the lm8-elf GCC. It contains many of the standard C routines, and it contains many non-standard routines that are specific and useful for the Lattice Mico8 processor.

LatticeMico8 Specific Features of GNU Binutils

The LatticeMico8 port of the GNU Binutils provides full support for the GNU assembler, linker, and disassembler. The Lattice Mico8 port uses the ELF file format that supports most of the features provided by the file format.

Supported Binutils Features

The following features are supported by the Lattice Mico8 binutils port:

- Full support for the GNU assembler
- Full support for the disassembler (objdump -d, and the rest of the options).
- Support for the GNU linker
- Support for most of the features provided by the ELF file format.

Unsupported Binutils Features

The following features cannot be supported by the Lattice Mico8 microprocessor:

- Support for shared libraries
- Support for Thread Local Storage
- Position independent executables

LatticeMico8 Specific Command Line Options

Neither the assembler nor the disassembler support any additional command line options. The Lattice Mico8 port of the GNU linker, however, supports the following additional command line arguments for selecting the addressing mode of the target configuration. The following options are mutually exclusive:

Option	Comment
-mcmodel=small	Causes 8-bit addressing to be used.
-mcmodel=medium	Causes 16-bit addressing to be used. (This is the default in the event that no switch is provided.)
-mcmodel=large	Causes 32-bit addressing to be used.

LatticeMico8 Specific Features of the GNU Compiler Collection

The provided port of the GNU Compiler Collection (GCC) to the Lattice Mico8 processor only supports GCC's C language frontend with most of the features found therein.

Supported GCC Features

The provided gcc port to the LatticeMico8 microcontroller supports all the features supported by GCC's C language frontend, as documented in "Using the GNU Compiler Collection" except for the ones mentioned in the section below.

Unsupported GCC Features or Limitations

The following features are not supported by the LatticeMico8 GCC port or have limited support:

- Language frontends other than C (including C++, Objective C, Objective C++, Java, FORTRAN and Ada) are not supported.
- Maximum +/– 2K relative executable program addresses for branching or calls.

  It has been observed that typically programs between 100-200 lines of C code can compile and link within this restriction easily. Bigger programs may generate linking errors.
- No floating point support.
- Limited support for nested functions.
- No support for newlib. Functions such as `printf()`, `scanf()`, `memset()`, `memcpy()`, `strcpy()` are not available.
- Calling functions through function pointers (indirect calls) are not supported. These are declarations of the type `void (*func)()`.

- Built-in functions `__builtin_longjmp()` and `__builtin_setjmp()`are not supported.
- Exceptions are not supported. These are used to implement `__attribute__((cleanup()))`.
- DWARF2 stack unwinding is not supported
- No support for trampolines. Trampolines are required to support nested functions.

LatticeMico8 Specific Command Line Options

The following command line options are supported by the Lattice Mico8 port of GCC in addition to all the options mentioned in "Using the GNU Compiler Collection".

Option	Comment
-mint8	Causes the common "int" type to be 8-bit wide instead of the standard 16-bit width.
-mcmodel=small	Causes 8-bit addressing to be used.
-mcmodel=medium	Causes 16-bit addressing to be used (default).
-mcmodel=large	Causes 32-bit addressing to be used. Setting this option includes extending the stack and frame pointers to be 32 bits.
-m16regs	If this option is set, only the first 16 registers (r0-r15) will be used by the compiler.
-mcall-stack-size=	Sets the size of the call stack implemented in the target processor.
-mcall-prologues	Don't inline the function prologue/epilogue.

LatticeMico8 Specific Function Attributes

The following function attributes are supported by the Lattice Mico8 port of GCC in addition to all the options mentioned in "Using the GNU Compiler Collection."

```
void __IRQ (void) __attribute__ ((interrupt));
```

The compiler will generate code to set up an interrupt stack frame suitable for an interrupt handler in the prologue of the function that has the attribute "interrupt" set. The function implementing the interrupt handler must be named __IRQ so that the interrupt handler can link correctly.

LatticeMico8 Specific Built-in Functions

The following built-in functions are supported by the Lattice Mico8 port of GCC in addition to all the built-in functions mentioned in "Using the GNU Compiler Collection," except for the built-in functions

`__builtin_longjmp() and __builtin_setjmp().`

Function	Comment
`void_builtin_export (char value, size_t port)`	Generates an "export" or "exporti" instruction.
`char __builtin_import (size_t port)`	Generates an "import" or "importi" instruction. The result of the import instruction is the returned value.

Note

The size of the size_t type reflects the size of pointers.

Application Binary Interface

Data Representation

Table 2: Fundamental Types

		Size of in code model			Alignment in code model (bytes)		
TYPE	**C TYPE**	**small**	**medium**	**large**	**small**	**medium**	**large**
Integral	signed char	1	1	1	1	1	1
Integral	unsigned char	1	1	1	1	1	1
Integral	signed short	2	2	2	2	2	2
Integral	unsigned short	2	2	2	2	2	2
Integral	signed int	2	2	2	2	2	2
Integral	unsigned int	2	2	2	2	2	2
Integral	signed long	4	4	4	4	4	4
Integral	unsigned long	4	4	4	4	4	4
Integral	signed long long	4	4	4	4	4	4
Integral	unsigned long long	4	4	4	4	4	4
Pointer	any-type * any-type (*)()	4	4	4	4	4	4
Floating-point	float	4	4	4	4	4	4

Table 2: Fundamental Types (Continued)

		Size of in code model			Alignment in code model (bytes)		
TYPE	**C TYPE**	**small**	**medium**	**large**	**small**	**medium**	**large**
Floating-point	double	4	4	4	4	4	4
Floating-point	long double	4	4	4	4	4	4

A NULL pointer of any type must be zero. All floating-point types are IEEE-754 compliant.

Aggregates and Unions

Aggregates (structures and arrays) and unions assume the alignment of the element requiring the highest alignment contained therein.

- An array has the alignment of the type of its elements.
- Structures and unions may be padded to meet alignment requirements. Each element is assigned to the lowest aligned address.

Bit-Fields

Structure and union definitions can have integral elements defined by a specified number of bits. Bit-fields follow the same alignment rules as aggregates and unions, with the following additions:

- Bit-fields are allocated from most to least significant (from left to right).
- A bit-field must entirely reside in a storage unit appropriate for its declared type.
- Bit-fields may share a storage unit with other struct/union elements, including elements that are not bit-fields. Struct elements occupy different parts of the storage unit.

Function Calling Sequence

This section describes the standard function calling sequence, including stack frame layout, register usage, and parameter passing. The standard calling sequence requirements apply only to global functions; however, it is recommended that all functions use the standard calling sequence.

Register Usage

Table 3: Small Code Model Register Usage (8-Bit Addressing)

Register	Preserved across function calls	Usage
R31	NO	Non-preserved temporary register
R30	NO	Non-preserved temporary register
R29	NO	Non-preserved temporary register
R28	NO	Non-preserved temporary register

Table 3: Small Code Model Register Usage (8-Bit Addressing) (Continued)

Register	Preserved across function calls	Usage
R27	YES	Preserved temporary register
R26	YES	Preserved temporary register
R25	YES	Preserved temporary register
R24	YES	Preserved temporary register
R23	YES	Preserved temporary register
R22	YES	Preserved temporary register
R21	YES	Preserved temporary register
R20	YES	Preserved temporary register
R19	YES	Preserved temporary register
R18	YES	Preserved temporary register
R17	YES	Preserved temporary register
R16	YES	Preserved temporary register
R15	YES	Frame pointer/preserved temporary register
R14	YES	Fixed—stack pointer
R13	YES	Preserved temporary
R12	NO	Non-preserved temporary
R11	NO	Non-preserved temporary
R10	NO	Non-preserved temporary
R9	YES	Preserved temporary
R7	NO	Function argument register 7
R6	NO	Function argument register 6
R5	NO	Function argument register 5
R4	NO	Function argument register 4
R3	NO	Function argument register/return value register 3
R2	NO	Function argument register/return value register 2
R1	NO	Function argument register/return value register 1
R0	NO	Function argument register/return value register 0

Some registers have assigned roles:

R14 – The stack pointer holds the limit of the current stack frame. The stack contents below the stack pointer are undefined. An exception to this rule is the usage of red zone space in leaf functions. The stack pointer must be aligned at all times.

R0 through R7 – Function parameters use up to 8 registers.

R0 through R3 – Return value of the function. These registers are undefined for void functions.

Table 4: Medium Code Model Register Usage (16-Bit Addressing)

Register	Preserved across function calls	Usage
R31	NO	Non-preserved temporary register
R30	NO	Non-preserved temporary register
R29	NO	Non-preserved temporary register
R28	NO	Non-preserved temporary register
R27	YES	Preserved temporary register
R26	YES	Preserved temporary register
R25	YES	Preserved temporary register
R24	YES	Preserved temporary register
R23	YES	Preserved temporary register
R22	YES	Preserved temporary register
R21	YES	Preserved temporary register
R20	YES	Preserved temporary register
R19	YES	Preserved temporary register
R18	YES	Preserved temporary register
R17	YES	Preserved temporary register
R16	YES	Preserved temporary register
R15	NO	Non-preserved temporary register
R14	YES	Preserved temporary register
R13	NO	Fixed – page pointer
R12	NO	Non-preserved temporary register
R11	YES	Frame pointer/preserved temporary register
R10	YES	Frame pointer/preserved temporary register
R9	YES	Fixed – stack pointer
R8	YES	Fixed – stack pointer
R7	NO	Function argument register 7
R6	NO	Function argument register 6
R5	NO	Function argument register 5
R4	NO	Function argument register 4

Table 4: Medium Code Model Register Usage (16-Bit Addressing) (Continued)

Register	Preserved across function calls	Usage
R3	NO	Function argument/return value register 3
R2	NO	Function argument/return value register 2
R1	NO	Function argument/return value register 1
R0	NO	Function argument/return value register 0

Some registers have assigned roles:

{R8, R9} – The stack pointer holds the limit of the current stack frame. The stack contents below the stack pointer are undefined. An exception to this rule is the usage of red zone space in leaf functions. The stack pointer must be aligned at all times.

R0 through R7 – Function parameters use up to 8 registers.

R0 through R3 – Return value of the function. These registers are undefined for void functions.

Table 5: Large Code Model Register Usage (32-Bit Addressing)

Register	Preserved across function calls	Usage
R31	YES	Frame pointer/preserved temporary register
R30	YES	Frame pointer/preserved temporary register
R29	YES	Frame pointer/preserved temporary register
R28	YES	Frame pointer/preserved temporary register
R27	YES	Fixed – stack pointer
R26	YES	Fixed – stack pointer
R25	YES	Fixed – stack pointer
R24	YES	Fixed – stack pointer
R23	YES	Preserved temporary register
R22	YES	Preserved temporary register
R21	YES	Preserved temporary register
R20	YES	Preserved temporary register
R19	YES	Preserved temporary register
R18	YES	Preserved temporary register
R17	YES	Preserved temporary register
R16	YES	Preserved temporary register

Table 5: Large Code Model Register Usage (32-Bit Addressing) (Continued)

Register	Preserved across function calls	Usage
R15	NO	Fixed – page pointer
R14	NO	Fixed – page pointer
R13	NO	Fixed – page pointer
R12	NO	Non-preserved temporary
R11	NO	Non-preserved temporary
R10	NO	Non-preserved temporary
R9	YES	Preserved temporary
R8	YES	Preserved temporary
R7	NO	Function argument register 7
R6	NO	Function argument register 6
R5	NO	Function argument register 5
R4	NO	Function argument register 4
R3	NO	Function argument/return value register 3
R2	NO	Function argument/return value register 2
R1	NO	Function argument/return value register 1
R0	NO	Function argument/return value register 0

Some registers have assigned roles:

{R24, R25, R26, R27} – The stack pointer holds the limit of the current stack frame. The stack contents below the stack pointer are undefined. An exception to this rule is the usage of red zone space in leaf functions. The stack pointer must be aligned at all times.

R0 through R7 – Function parameters use up to 8 registers.

R0 through R3 – Return value of the function. For void functions, these registers are undefined.

The Stack Frame

Each function has a frame on the run-time stack in addition to registers. This stack grows downward from high addresses. The table below shows the stack frame organization.

Table 6: The Stack Frame

FP relative position	SP relative position	Contents	Frame
FP+ (M-6)	SP + (N + M + 4)	Function argument byte M	Previous
. . .	. . .	. . .	
FP+ 0	SP + (N + 4)	Function argument byte 6	
FP – 1	SP + (N + 3)	Previous Fp value higher part, if saved	Current
FP – 2	SP + (N + 2)	Previous Fp value higher part, if saved	
FP – 3	SP + (N + 1)	Previous Fp value high part, if saved	
FP – 4	SP + N	Previous Fp value low part, if saved	
FP – 5	SP + (N - 1)	Local variable N	
. . .	. . .	. . .	
FP – (N + 5)	SP + 0	Local variable 0	
FP – N (N + 6)	SP - 1	Red zone area start	Future
. . .	. . .	. . .	
FP – (N + 37)	SP - 32	Red zone area end	

The stack pointer always points to the end of the latest allocated stack frame. The first 32 bytes below the stack frame are reserved for leaf functions that do not need to modify the stack pointer. Interrupt handlers must guarantee that they will not use this area.

Parameter Passing

Functions receive their first eight argument bytes in function argument registers R0-R7. If there are more than eight argument bytes, the remaining argument bytes are passed on the stack. Small structure and union arguments are passed in argument registers; other structure and union arguments are passed as pointers.

Functions Returning Scalars or No Value

A function that returns an integral or pointer value puts its result in the registers R0–R3. Void functions leave registers R0–R3 undefined.

Functions Returning Structures or Unions

A function that returns a small structure or union places the returned value in registers R0-R3. Other structures and unions are returned in memory, pointed by the "invisible" first function argument.

Interrupts

In the event of an interrupt, the stack pointer will be switched to the top of the interrupt stack minus 32 where all the registers will be saved as shown in Table 7, Table 8, and Table 9. The interrupt stack must be separate from the user stack.

Nested interrupts are not supported by this ABI.

Table 7: Interrupt Stack Layout for the Small Code Model

Position	Register
Top of interrupt stack-1	R11
Top of interrupt stack-2	R10
Top of interrupt stack-3	R31
Top of interrupt stack-4	R30
Top of interrupt stack-5	R29
Top of interrupt stack-6	R28
Top of interrupt stack-7	R7
Top of interrupt stack-8	R6
Top of interrupt stack9	R5
Top of interrupt stack-10	R4
Top of interrupt stack-11	R3
Top of interrupt stack-12	R2
Top of interrupt stack-13	R1
Top of interrupt stack-14	R0

Table 8: Interrupt Stack Layout for the Medium Code Model

Position	Register
Top of interrupt stack-1	R9
Top of interrupt stack-2	R8
Top of interrupt stack-3	R31
Top of interrupt stack-4	R30
Top of interrupt stack-5	R29
Top of interrupt stack-6	R28
Top of interrupt stack-7	R7
Top of interrupt stack-8	R6
Top of interrupt stack-9	R5
Top of interrupt stack-10	R4
Top of interrupt stack-11	R3
Top of interrupt stack-12	R2
Top of interrupt stack-13	R1
Top of interrupt stack-14	R0

Table 9: Interrupt Stack Layout for the Large Code Model

Position	Register
Top of interrupt stack-	R11
Top of interrupt stack-	R10
Top of interrupt stack-	R27
Top of interrupt stack-	R26
Top of interrupt stack-	R25
Top of interrupt stack-	R24
Top of interrupt stack-	R7
Top of interrupt stack-	R6
Top of interrupt stack-	R5
Top of interrupt stack-	R4
Top of interrupt stack-	R3
Top of interrupt stack-	R2
Top of interrupt stack-	R1
Top of interrupt stack-	R0

Scratchpad Memory

The first four bytes of the scratchpad memory area are reserved to set up the interrupt stack in the event of an interrupt.

Address	Use
0–3	Reserved
4–255	Application data

Further Reading

This guide only touches on the options, attributes and functions that are specific to the Lattice Mico8 port of GCC and GNU binutils. Both collections support many options, attributes, and functions that are described in detail in the following documents. The reader is advised to review them in addition to this guide.

- *Using the GNU Compiler Collection* by Richard M. Stallman and the GCC Developers Community. Available online at http://gcc.gnu.org/onlinedocs/gcc-4.4.3/gcc/
- *Using as*. Available online at http://sourceware.org/binutils/docs-2.18/as/index.html
- *Using ld*. Available online at http://sourceware.org/binutils/docs-2.18/ld/index.html
- *GNU Binary Utilities*. Available online at http://sourceware.org/binutils/docs-2.18/binutils/index.htm

Useful Links

Before asking for help online, try to find your answers in the followoing documentation.

- http://sourceforge.net [SourceForge]
- http://www.gnu.org/ [GNU Project]
- http://www.gnu.org/manual/ [GNU Manuals Online]
- http://sources.redhat.com/binutils/ [GNU Binutils]
- http://www.gnu.org/software/gcc/ [GNU Compiler Collection (GCC)]
- http://www.gnu.org/software/gcc/onlinedocs/ [GCC Manuals Online]
- http://srecord.sourceforge.net/ [SRecord]
- http://www.gnu.org/software/make/ [GNU Make]

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)

+1-503-268-8001 (Outside North America)

e-mail:techsupport@latticesemi.com

Internet: www.latticesemi.com