



REALTEK

RSDK Toolchain User Guide

Realtek Semiconductor Corp.

Release 1.2.8

December 7, 2006

Realtek Proprietary and Confidential

RSDK Toolchain User Guide for Release 1.2.8

This document is proprietary and confidential to Realtek Semiconductor Corp.
Copyright © 2006 Realtek Semiconductor Corp.
ALL RIGHTS RESERVED

MIPS, MIPS16, MIPS ABI, MIPSII, MIPSIV, MIPSV, MIPS32, R3000, R4000, and other MIPS common law marks are trademarks and/or registered trademarks of MIPS Technologies.

SmoothCore, Radiax, and NetVortex are trademarks of Lexra, Inc.

Contents

| | | |
|----------|--|-----------|
| 1 | RSDK | 1 |
| 2 | GCC | 5 |
| 3 | Binutils | 17 |
| 4 | Problem Report and Issue Tracking | 19 |
| 5 | Known Problems | 21 |
| A | RADIAX registers | 23 |
| B | Inline Assembly Format | 25 |
| C | RELEASE NOTE | 27 |
| D | Change Log | 31 |

List of Tables

| | | |
|-----|--|----|
| 1.1 | Software Components | 2 |
| 1.2 | Supported LX/RLX CPU cores | 3 |
| 1.3 | Supported RSDK platforms | 3 |
| 1.4 | Supported C Libraries | 3 |
| 2.1 | Default compiler options | 5 |
| 2.2 | Built-in data type for SIMD instructions | 7 |
| 2.3 | Built-in functions defined for SIMD instructions | 7 |
| 2.4 | Optional MAC-DIV instructions | 9 |
| 2.5 | Preprocessor definition mapping | 15 |

List of Figures

Chapter 1 RSDK

The Realtek Software Development Kit (RSDK) is a chain of software tools that empowers end-users to develop embedded applications that run on Realtek's in-house processor cores. The set of tools in RSDK can be divided into three groups, compilers, binary utilities, and C libraries. The tools in the first two groups are derived from the GNU compiler collection (GCC) and binutils respectively. The C libraries are based on newlib and uClibc.

The lists of changes and enhancements to the original GNU compiler collection and binutils are summarized in the following chapters. For the detailed usage of GNU compiler collection and binutils, please refer to the GNU website at <http://www.gnu.org>.

Version Numbering

The format of RSDK version number is shown as follows:

GNU version . RLX version . Patch level

The current release version is 1.2.8. Each version contains three numbers. They are GNU version, RLX version, and patch level. The GNU version number is mapped to a set of GNU tools which RSDK is based on. The RLX version number corresponds to the processor cores supported by the RSDK. The patch level number represents the number of updates in the same branch of RSDK. It is usually that the higher the number the less the bugs.

The RSDK version number will be appended to that of the original GNU tools during the building of RSDK toolchain. To check the RSDK version number as well as the original GNU version numbers, users can issue version display commands shown in program 1.

Table 1.1 shows the list of GNU tools supported in 1.2.8 and table 1.2 shows the list of RLX processor core supported in 1.2.8.

Software Component

The list of software components and their version numbers is summarized in table 1.1.

Supported Processor Cores

The list of supported processor cores is summarized in table 1.2.

Program 1: RSDK Version number

```
% rsdk-elf-gcc --version
rsdk-elf-gcc (GCC) 3.2.3-1.2.0
Copyright (C) 2002 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

% rsdk-elf-as --version
GNU assembler 2.16-1.2.0
Copyright 2005 Free Software Foundation, Inc.
This program is free software; you may redistribute it under the terms of
the GNU General Public License.  This program has absolutely no warranty.
This assembler was configured for a target of 'mips-elf'.

% rsdk-elf-ld --version
GNU ld version 2.16-1.2.0
Copyright 2005 Free Software Foundation, Inc.
This program is free software; you may redistribute it under the terms of
the GNU General Public License.  This program has absolutely no warranty.
```

Table 1.1: Software Components

| Software | Original Version | RSDK Version |
|----------|------------------|--------------|
| gcc | 3.2.3 | 3.2.3-1.2.8 |
| binutils | 2.16.1 | 2.16.1-1.2.8 |
| insight | 6.0 | 6.0-1.2.8 |
| newlib | 1.14.0 | 1.14.0-1.2.8 |
| uclibc | 0.9.27 | 0.9.27-1.2.8 |

Supported Platform

The list of supported platforms is summarized in table [1.3](#).

NOTE: The C library shipped with RedHat Linux may differ from the one the RSDK toolchain was built against. To ensure maximal compability, the following two packages are recommended if the RedHat Linux version is newer than 7.3.

```
compat-glibc-7.x-2.2.4.32.6
compat-libstdc++-7.3-2.96.128
```

NOTE: The Cygwin platform itself is not a stable environment for software development. Users might encounter various problems which are not directly related to the RSDK toolchains. To ensure maximal stability, users are advised to develop applications on Linux platforms whenever possible.

Supported C Libraries

The list of supported C libraries is summarized in table [1.4](#).

Table 1.2: Supported LX/RLX CPU cores

| Processor Core | RTL Release Version | MIPS1 | MIPS16 | RADIAX |
|----------------|---------------------|-------|--------|--------|
| LX4180 | 4.0.2 | Yes | Yes | No |
| LX5280 | 1.9.3 | Yes | Yes | Yes |
| RLX4181 | 1.0 | Yes | Yes | No |
| RLX5181 | 1.1 | Yes | Yes | Yes |

Table 1.3: Supported RSDK platforms

| Platform | Version | Package name |
|----------|-------------------------|--------------------------|
| Linux | RedHat 7.3 and above | rsdk-1.2.8-linux.tar.gz |
| Cygwin | Cygwin 1.5.10 and above | rsdk-1.2.8-cygwin.tar.gz |

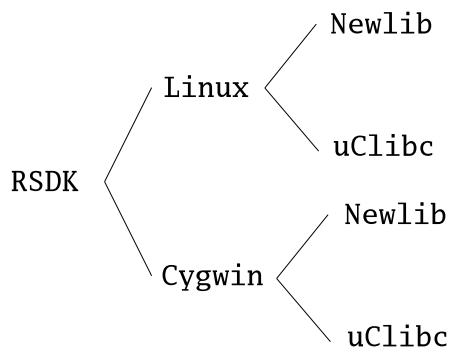
Installation

The RSDK packages are available as two tarballs, one for each platform. The tarballs are as follows:

rsdk-1.2.8-linux.tar.gz
rsdk-1.2.8-cygwin.tar.gz

For each platform, two RSDK toolchains are provided, one for each C library. These toolchains are completely independant. Users should only need to add the path of desired RSDK toolchain to the path list or to symbolically link the desired RSDK toolchain to a common path that is searchable in user shell.

The structure of the RSDK tarball is shown as follows:



The installation procedure is shown in program 2. The list of supported libraries and their versions is summarized in table 1.4.

Table 1.4: Supported C Libraries

| Library | Version |
|---------|---------|
| Newlib | 1.14.0 |
| uClibc | 0.9.27 |

Program 2: RSDK installation

```
step 1: cd TARGET_DIR  
step 2: gzip -cd rsdk-{VERSION}-{PLATFORM}.tar.gz | tar xvf -  
step 3 (newlib): ln -s rsdk-{VERSION}/{PLATFORM}/newlib rsdk  
step 3 (uclibc): ln -s rsdk-{VERSION}/{PLATFORM}/uclibc rsdk  
step 4: set path=(TARGET_DIR/rsdk/bin $path)
```

Chapter 2 GCC

GCC is the GNU Compiler Collection, which includes front ends for multiple languages such as C, C++, Objective-C, Fortran, Java, and Ada, as well as libraries for these languages (libstdc++, libgcj,...).

In RSDK 1.2.8, gcc has been upgraded from 3.2 to 3.2.3 for generic bug fixes and for better MIPS16 support. The list of changes is summarized in the following subsections.

General Changes

Relative path search

GCC search certain paths for binaries, libraries, and includes files during compilation. To ensure maximal portability, GCC has been patched to search paths relative to the GCC binary.

Default options

The following options, listed in table 2.1, are enabled by default.

Table 2.1: Default compiler options

| Options | Description |
|--------------|---|
| -msoft-float | Enable software floating point support |
| -meh | Enable big-endian |
| -march=4180 | Set default target to 4180 if none is specified |

Machine-dependent options

-march|mtune|mcpu=4180|4181|5181|5280

Architecture: LX4180, RLX4181, RLX5181, LX5280

Summary: Specify target processor core

Dependency: None

Status: Current

Description:

The march|mtune|mcpu option sets the target processor core to the one specified in the option.

If none of the -march, -mtune, and -mcpu options is specified, compiler will automatically add -march=4180 to the option list.

-mzero-overhead**-munsafe-zero-overhead****Architecture:** RLX5181, LX5280**Summary:** Enable zero-overhead loop optimization**Dependency:** -O2 or above**Status:** Current**Description:**

Zero-overhead loop is only supported on RLX5181 and LX5280 processor cores. To enable zero-overhead loop optimization, either -mzero-overhead or -munsafe-zero-overhead must be specified. To optimize loops, three special registers, LPS0, LPE0, and LPC0, will be used by the compiler to enable zero-overhead loop optimization. The loop iteration count is stored in the lower 16-bit of the LPC0 register. Therefore, the maximal loop count is 2^{16} .

When the -mzero-overhead is enabled, the compiler will exclude loops whose iteration count cannot be determined at compile time. When -munsafe-zero-overhead is enabled, the compiler will optimize all the applicable loops even though their iteration counts are unknown at compile time. In this case, users must take extra caution to ensure that the loop iteration counts will not exceed 2^{16} .

NOTE: for nested loops, the zero-overhead loop optimization will be applied to the inner most loop only.

NOTE: -mzero-overhead is not effective unless the optimization level is -O2 or above.

-mt0-t3**Architecture:** LX4180, RLX4181, RLX5181, LX5280**Summary:** Expand function parameter registers from four to eight**Dependency:** None**Status:** Current**Description:**

By MIPS convention, only four registers, a0, a1, a2, and a3, are used to pass function parameters. When calling a function with more than four parameters, extra parameters are pushed into and popped out of the stack before and after entering the callee. There are two major drawbacks for this approach. First, extra memory accesses for loading and storing these parameters will certainly degrade the performance. Second, loading data from memory has the load delay penalty and may incur cache miss, which makes things even worse. Therefore, it is beneficial to expand the number of function parameter passing registers from four to eight at the cost of breaching ABI compatibility. When -mt0-t3 option is specified, compiler will use four additional registers, t0, t1, t2, and t3, for passing function parameters. The total number of registers that are reserved for passing parameters is increased from four to eight.

NOTE: this option is not ABI compatible. If this option is used, it should be applied to all the source files in the application.

NOTE: When using inline assembly with this option enabled, users must take caution not to destroy the extra registers, t0-t4. These extra registers should be saved first if they will be used in the inline assembly codes and should be restored after use.

-msimd**Architecture:** RLX5181, LX5280**Summary:** Expand function parameter registers from four to eight**Dependency:** -mradiax**Status:** Current**Description:**

RLX5181 and LX5280 support Single Instruction Multiple Data (SIMD) instructions. A SIMD instruction operates on multiple values contained in a single register at the same time.

Table 2.2: Built-in data type for SIMD instructions

| Type | Definition | Internal Type |
|------|---|---------------|
| v2hi | <code>typedef int v2hi __attribute__((mode(V2HI)))</code> | VNB |

Table 2.3: Built-in functions defined for SIMD instructions

| Function | Argument 0 | Argument 1 | Return Type |
|--|------------|------------|----------------|
| <code>__builtin_lx5280_addr2</code> | V2HI | V2HI | V2HI |
| <code>__builtin_lx5280_subr2</code> | V2HI | V2HI | V2HI |
| <code>__builtin_lx5280_min2</code> | V2HI | V2HI | V2HI |
| <code>__builtin_lx5280_max2</code> | V2HI | V2HI | V2HI |
| <code>__builtin_lx5280_multa2_internal1</code> | V2HI | V2HI | V2HI |
| <code>__builtin_lx5280_multa2_internal2</code> | V2HI | V2HI | V2HI |
| <code>__builtin_lx5280_multa2_internal3</code> | V2HI | V2HI | DI (long long) |
| <code>__builtin_lx5280_mulna2_internal1</code> | V2HI | V2HI | V2HI |
| <code>__builtin_lx5280_mulna2_internal2</code> | V2HI | V2HI | V2HI |
| <code>__builtin_lx5280_mulna2_internal3</code> | V2HI | V2HI | DI (long long) |
| <code>__builtin_lx5280_madda2_internal1</code> | V2HI | V2HI | V2HI |
| <code>__builtin_lx5280_madda2_internal2</code> | V2HI | V2HI | V2HI |
| <code>__builtin_lx5280_madda2_internal3</code> | V2HI | V2HI | DI (long long) |
| <code>__builtin_lx5280_msuba2_internal1</code> | V2HI | V2HI | V2HI |
| <code>__builtin_lx5280_msuba2_internal2</code> | V2HI | V2HI | V2HI |
| <code>__builtin_lx5280_msuba2_internal3</code> | V2HI | V2HI | DI (long long) |
| <code>__builtin_lx5280_sltr2</code> | V2HI | SI (int) | V2HI |
| <code>__builtin_lx5280_sllv2</code> | V2HI | SI (int) | V2HI |
| <code>__builtin_lx5280_srlv2</code> | V2HI | SI (int) | V2HI |
| <code>__builtin_lx5280_srav2</code> | V2HI | SI (int) | V2HI |
| <code>__builtin_lx5280_absr2</code> | V2HI | V2HI | V2HI |

For `multa2`, `mulna2`, `madda2`, and `msuba2`, there are three different forms for their builtin functions. The three different forms are `internal1`, `internal2`, and `internal3`. For `internal1`, the destination register is the upper 32-bit HI of the accumulator. For `internal2`, the destination register is the lower 32-bit LO of the accumulator. For `internal3`, the destination register is the entire 64-bit accumulator, HI and LO.

-msave-restore-mmd

Architecture: RLX5181, LX5280

Summary: Preserve MMD state

Dependency: -mradiax

Status: Current

Program 3: Example of using built-in functions

```
int func2()
{
    return i > 0 ? i : -i;
}

int func1()
{
    v2hi data[32];
    v2hi sumv1, sumv2;
    long long sumv3;
    int sumv0 = 0;
    int i;

    for (i = 0; i < 32; i++)
        data[i] = __builtin_lx5280_addr2((v2hi) i, (v2hi) i);

    for (i = 0; i < 32; i++) {
        sumv1 = __builtin_lx5280_madda2_internal1(data[i], data[i]);
        sumv2 = __builtin_lx5280_madda2_internal2(data[i], data[i]);
        sumv3 = __builtin_lx5280_madda2_internal3(sumv2, data[i]);
    }

    shift_num = func2(-6);
    sumv2 = __builtin_lx5280_srav2(sumv2, shift_num);
    sumv0 = value & 0xffffffff;
    return (int) __builtin_lx5280_subr2(sumv2, (v2hi) sumv0);
}
```

Description:

MMD is a special register that is shared among many RADIAX instructions. The state of this MMD register is crucial for two reasons: First, compiler depends on the state of this MMD register to emit the right RADIAX instruction. Second, sequence of RADIAX instructions rely on the state of this MMD register to operate correctly. During compilation, the compiler keeps track of the state of MMD register carefully. However, if users change the state of the MMD register manually, for example, loading data into the MMD register in inline assembly codes, the result might be unpredictable. The `-msave-restore-mmd` is provided to add a safe net under this situation.

When `-msave-restore-mmd` is specified, compiler will automatically emit instructions that save the state of MMD register before the operation that changes its state and emit instructions that restore the state of MMD register after the operation result is retrieved.

-mfpga

Architecture: LX4180, RLX4181, RLX5181, LX5280

Summary: Expand function parameter registers from four to eight

Dependency: None

Status: Obsoleted

Description:

Setting the MMD register requires a delay slot before the data can be used. By default, only a single NOP will

be emitted. However, on some FPGA boards, it may take two NOPs to get the data ready. When `-mfpga` is used, two NOPs will be emitted after setting the MMD register. This option is obsoleted.

-mradiax

Architecture: RLX5181, LX5280

Summary: Enable RADIAX support for RLX5181 and LX5280

Dependency: None

Status: Current

Description:

The `-mradiax` option enables the RADIAX support for RLX5181 and LX5280. The RADIAX instruction extensions include MAC operations, vector-addressing, and enhanced extensions to the MIPS-I ALU instructions.

For RLX5181 and LX5280, `-mradiax` automatically implies `-mmac` by default.

-mmac

Architecture: LX4180, RLX4181, RLX5181, LX5280

Summary: Enable optional Multiply/Divide/Accumulator support

Dependency: None

Status: New

Description:

All the LX/RLX processor cores support an optional Multiply/Divide/Accumulate module (MAC-DIV) which further enhances mathematical operations. This MAC-DIV module is configurable using the `lconfig` utility. When `-mmac` option is specified, compiler will emit the instructions listed in table 2.4 whenever possible. It is users' responsibilities to ensure the MAC-DIV module exists in the target processor core.

The list of instructions for the optional MAC-DIV module is summarized as follows:

Table 2.4: Optional MAC-DIV instructions

| Mnemonic | Operation | Latency | Repeat Delay | Description |
|----------|----------------------------|---------|--------------|--|
| MTHI | HI <- Rs | - | - | pre-load accumulator, or restore saved HI |
| MTLO | LO <- Rs | - | - | pre-load accumulator, or restore saved LO |
| MFHI | Rd <- HI | 1 | - | read accumulator, or part of 64-bit result |
| MFLO | Rd <- LO | 1 | - | read accumulator, or part of 64-bit result |
| MULT | HI,LO <- Rs*Rt | 5 | - | 32x32 signed multiply 64-bit result |
| MULTU | HI,LO <- Rs*Rt | 5 | - | 32x32 unsigned multiply 64-bit result |
| MADH | HI <- HI+Rs[15:0]*Rt[15:0] | 3 | 0 | 16x16 signed multiply, with 32-bit signed add to accum |
| MADL | LO <- LO+Rs[15:0]*Rt[15:0] | 3 | 0 | 16x16 signed multiply, with 32-bit signed add to accum |
| MAZH | HI <- 0+Rs[15:0]*Rt[15:0] | 3 | 0 | 16x16 signed multiply, add to pre-zeroed 32-bit accum |
| MAZL | LO <- 0+Rs[15:0]*Rt[15:0] | 3 | 0 | 16x16 signed multiply, add to pre-zeroed 32-bit accum |
| MSBH | HI <- HI-Rs[15:0]*Rt[15:0] | 3 | 0 | 16x16 signed multiply, with 32-bit signed sub from accum |
| MSBL | LO <- LO-Rs[15:0]*Rt[15:0] | 3 | 0 | 16x16 signed multiply, with 32-bit signed sub from accum |
| MSZH | HI <- 0-Rs[15:0]*Rt[15:0] | 3 | 0 | 16x16 signed multiply, sub from pre-zeroed 32-bit accum |
| MSZL | LO <- 0-Rs[15:0]*Rt[15:0] | 3 | 0 | 16x16 signed multiply, sub from pre-zeroed 32-bit accum |
| DIV | HI <- Rs/Rt; LO <- Rs/Rt | 35 | - | 32 by 32 signed divide with reminder |
| DIVU | HI <- Rs/Rt; LO <- Rs/Rt | 35 | - | 32 by 32 unsigned divide with reminder |

For RLX5181 and LX5280, `-mradiax` automatically implies `-mmac` by default.

-mcache-profile**Architecture:** LX4180, RLX4181, RLX5181, LX5280**Summary:** Enable optional Multiply/Divide/Accumulator support**Dependency:** None**Status:** Obsoleted**Description:**

In general, profiler functions are placed in the uncacheable memory region to remove effects casted by the memory cache and to achieve more accurate results. The trade-off is the profiling speed because compiler will have to generate more instructions to cope with long jumps and memory access latency is inevitably higher. However, if the profiling speed is a concern, users can force compiler to map the address in the cacheable region by using `-mcache-profile` option.

Compiler Options

-finhibit-ltw**Architecture:** RLX4181**Summary:** Disable load twin-word instruction, ltw**Dependency:** None**Status:** Obsoleted**Description:**

Disable load twin-word instruction. When `-finhibit-ltw` is specified, compiler will emit a sequence of load byte and shift instructions instead of ltw instruction. ltw will trigger exception if the load address is not twin-word aligned i.e. 8-byte aligned. Since RSDK 1.2.0, this option is obsoleted as ltw is disabled by default.

-finhibit-lt**-finhibit-st****Architecture:** RLX5181, LX5280**Summary:** Disable load/store twin-word instruction, lt and st**Dependency:** None**Status:** Obsoleted**Description:**

When `-finhibit-lt` is specified, compiler will emit a sequence of load byte and shift instructions instead of lt instruction. Likewise, when `-finhibit-st` is specified, compiler will emit a sequence of store byte and shift instructions instead of st instruction. lt and st will trigger exception if the load or store address is not twin-word aligned i.e. 8-byte aligned. Since RSDK 1.2.0, these options are obsoleted as lt and st are disabled by default.

-finhibit-lw**-finhibit-sw****Architecture:** LX4180, RLX4181, RLX5181, LX5280**Summary:** Disable load/store word instruction, lw and sw**Dependency:** None**Status:** Obsoleted**Description:**

When `-finhibit-lw` is specified, compiler will emit a sequence of load byte and shift instructions instead of `lw` instruction. Likewise, when `-finhibit-sw` is specified, compiler will emit a sequence of store byte and shift instructions instead of `sw` instruction. `lw` and `sw` will trigger exception if the load or store address is not word aligned i.e. 4-byte aligned.

-fltw

Architecture: RLX4181

Summary: Enable load twin-word instruction, `ltw`

Dependency: None

Status: Obsoleted, use `-ftword` instead

Description:

When `-fltw` is specified, compiler will emit `ltw` instruction instead of a sequence of load byte and shift instructions whenever possible. This option is added since RSDK 1.2.0 and is merged into `-ftword` in RSDK 1.2.4.

-flt

-fst

Architecture: RLX5181, LX5280

Summary: Enable load/store twin-word instruction, `lt` and `st`

Dependency: None

Status: Obsoleted, use `-ftword` instead

Description:

When `-flt` is specified, compiler will emit `lt` instruction instead of a sequence of load byte and shift instructions whenever possible. Likewise, when `-fst` is specified, compiler will emit `st` instruction instead of a sequence of store byte and shift instructions whenever possible. These options are added since RSDK 1.2.0 and are merged into `-ftword` in RSDK 1.2.4.

-ftword

Architecture: RLX4181, RLX5181, LX5280

Summary: Enable load/store twin-word instruction, `ltw` or `lt/st`

Dependency: None

Status: New

Description:

When `-ftword` is specified, compiler will emit `ltw` (RLX4181) instruction or `lt` (RLX5181/LX5280) instruction instead of a sequence of load byte and shift instructions whenever possible. Likewise, when `-ftword` is specified, compiler will emit `st` (RLX5181/LX5280) instruction instead of a sequence of store byte and shift instructions whenever possible. These options are added since RSDK 1.2.4.

-ftword-stack

Architecture: RLX4181, RLX5181, LX5280

Summary: Enable twin-word instructions in function prologue/epilogue

Dependency: None

Status: New

Description:

When `-ftword-stack` is specified, compiler will emit `lt/ltw` instruction instead of a sequence of load byte and shift instructions whenever possible during function calling. This option is added since RSDK 1.2.0.

-frlxcov

Architecture: LX4180, RLX4181, RLX5181, LX5280

Summary: Enable code coverage analysis using RLX library

Dependency: None

Status: New

Description:

When `-frlxcov` is specified, compiler will emit codes to do coverage analysis for basic blocks. This option is similar to the combination of `'-fprofiling-arcs -ftest-coverage'` except that it uses remote file I/O via GDB remote serial protocol to support remote debugging.

The coverage analysis codes will be placed in **.rlxcov** section. Therefore, the linker script must be modified to explicitly allocate the **.rlxcov** section. This option is added since RSDK 1.2.7.

For more information about the usage of the RLX code coverage library, please refer to the RSDK supplementary library module userguide.

Program 4: Example linker script

```
SECTIONS
{
    ....

    __rlxcov_ent = ABSOLUTE(.);
    .rlxcov BLOCK(0x8) : { *(.rlxcov) }
    __rlxcov_end = ABSOLUTE(.);

    ....
}
```

The `'-fprofile-arcs -ftest-coverage'` options have been reverted to comply with GNU standard. If specified, the standard GNU code coverage analysis will be used. Please refer to GNU website for more information.

-fdafile-relative

Architecture: LX4180, RLX4181, RLX5181, LX5280

Summary: Enable Linux profiling

Dependency: `-fprofile-arcs -ftest-coverage`

Status: New

Description:

By default, the GCOV generates `.da` file in the path where the source files reside. When this option is specified, the GCOV will generate `.da` file where the executable is invoked.

Profiling options

-plinux

Architecture: LX4180, RLX4181, RLX5181, LX5280

Summary: Enable Linux profiling

Dependency: None

Status: Current

Description:

When this option is specified, compiler will emit instructions in functions prologue and epilogue to jump to the predefined profiling functions for Linux profiling.

-pros

Architecture: LX4180, RLX4181, RLX5181, LX5280

Summary: Enable ROS profiling

Dependency: None

Status: Current

Description:

When this option is specified, compiler will emit instructions in functions prologue and epilogue to jump to the predefined profiling functions for ROS profiling.

-pg

-p

Architecture: LX4180, RLX4181, RLX5181, LX5280

Summary: Enable general program profiling

Dependency: None

Status: Current

Description:

When **-pg** or **-p** is specified, compiler will emit specific codes in functions prologue and epilogue to jump to the predefined functions to profile user applications at function-level. These two options are identical.

For more information, please refer to the RSDK supplementary library module for an example of profiler implementation.

Attributes

__attribute__((far_call))

Architecture: LX4180, RLX4181, RLX5181, LX5280

Summary: Mark the specified function as a long jump

Dependency: None

Status: Current

Description:

By default, the range for a jump instruction is within 256MB. If the jump target is more than 256MB away, then

a single jump instruction can not reach the desired target. In this case, the jump instruction must be modified to a load and a jump instructions. The former loads the target address to a specific register and the later does the actual jump to the address stored in that register. By using the `far_call` attribute, users can explicitly specify certain functions as long jumps and compiler will emit the right instructions when these functions are called.

Program 5: Example of using `__attribute__((far_call))`

```
int func() __attribute__((far_call))

int func()
{
    ....
}

int myfunc()
{
    ....
    func();
}
```

The purpose of attribute is similar to that of the compiler option `-mlong-calls`. The difference is that the compiler option, `-mlong-calls`, applies to all the functions in the application, while `__attribute__((far_call))` allows users to do finer control and apply to specific functions only.

`__attribute__((mips16))`

`__attribute__((nomips16))`

Architecture: LX4180, RLX4181, RLX5181, LX5280

Summary: Mark the specified function to be compiled in MIPS16/NONMIPS16 mode

Dependency: None

Status: New

Description:

The `__attribute__((mips16))` enables users to insert MIPS16 codes into MIPS1 applications without compiling the entire source as a MIPS16 code. In other words, with this attribute, users can fine control certain functions to be in MIPS16 mode and balance the trade-off between code performance and code size.

Likewise, the `__attribute__((nomips16))` enables users to insert MIPS1 codes into MIPS16 applications without compiling the entire source as a MIPS1 code.

The example is shown in program 6.

NOTE: the compilation time will increase as the number of functions with MIPS16 attribute increases. The overhead is introduced by testing and switching between MIPS1 and MIPS16 mode on a per function basis. If the majority of the functions in a C file are MIPS16, users should consider compiling the entire file in MIPS16 mode using the `-mips16` compiler option.

Preprocessor definitions

`-D__m4180|__m4181|__m5181|__m5280`

Architecture: LX4180, RLX4181, RLX5181, LX5280

Summary: Define identifier for each processor

Program 6: Example of using `__attribute__((mips16))`

```

int __attribute__((mips16)) func1()
{
    ....
}

int func2()
{
    ....
}

int myfunc()
{
    ....    /* MIPS32 mode */
    func1(); /* MIPS16 mode */
    func2(); /* MIPS32 mode */
    ....    /* MIPS32 mode */
}

```

Dependency: None**Status:** New**Description:**

Due to differences among the ISAs of Realtek's processor cores, users may need fine-tune certain code segments for each core, e.g. fine-tune machine-dependent codes using inline assembly. When the target processor is set by specifying `-march|-mtune|-mcpu`, compiler will automatically add the corresponding preprocessor identifier for users to separate machine-dependent codes in the same segment. The preprocessor identifiers are shown in table 2.5.

Table 2.5: Preprocessor definition mapping

| Processor Core | Preprocessor definition |
|-------------------------------------|-------------------------|
| <code>-march mtune mcpu=4180</code> | <code>-D__m4180</code> |
| <code>-march mtune mcpu=4181</code> | <code>-D__m4181</code> |
| <code>-march mtune mcpu=5181</code> | <code>-D__m5181</code> |
| <code>-march mtune mcpu=5280</code> | <code>-D__m5280</code> |

Program 7: Example of using preprocessor definitions

```
#ifdef __m4180
    machine-dependent code for 4180
#endif

#ifdef __m4181
    machine-dependent code for 4181
#endif

#ifdef __m5181
    machine-dependent code for 5181
#endif

#ifdef __m5280
    machine-dependent code for 5280
#endif
```

Chapter 3 Binutils

The binutils tool is based on the GNU binutils suite. The binutils tool provides the fundamental assembler, linker, and object file manipulation utilities, such as objdump and objcopy. In RSDK 1.2.8, the binutils has been upgraded from version 2.14 to 2.16.1 to provide a more reliable and more stable development environment for both MIPS32 and MIPS16 applications.

The list of changes is summarized in the following subsections.

Assembler

-march=4180|4181|5181|5280

Architecture: LX4180, RLX4181, RLX5181, LX5280

Summary: Set the target processor core

Status: Current

Description:

The -march option sets the target processor core to the one specified in the option.

If none of the -march option is specified, the assembler will automatically add -march=4180 to the option list.

Mixing MIPS16 and MIPS32 objects warning

Linking MIPS16 objects into MIPS32 mode is not supported at this moment. Due to the current design in binutils 2.16.1, the linker cannot handle linking MIPS16 objects within MIPS32 mode, hence causes user application to fail when it tries to access the wrong address. This problem can be worked around by using explicit symbols instead of expressions. The example is shown in program [8](#).

In RSDK 1.2.8, the assembler has been modified to yield an error message on this case and to abort the assembling process.

Objdump

-mmips:4180|4181|5181|5280

Architecture: LX4180, RLX4181, RLX5181, LX5280

Summary: Set the target processor core

Status: Current

Description:

The -mmips option specifies the target ISA for the objdump utility. The usage is shown in program [9](#).

Program 8: Example of mixing MIPS16 and MIPS32 objects

```
...
1:
    jalx lb+18 # should jump to M32 (Fail)
    jalx M32   # should jump to M32 (Work)
    nop
    b fail16
    nop
    addiu v0, 0x4
    addiu v0, 0x8
    .set nomips16
    add v0, 0x8
M32:
    add v0, 0x8
    add v0, 0x10
...
```

Program 9: Objdump Example

```
rsdk-elf-objdump -d -m mips:4180 file.o
rsdk-elf-objdump -d -m mips:4181 file.o
rsdk-elf-objdump -d -m mips:5181 file.o
rsdk-elf-objdump -d -m mips:5280 file.o
```

Chapter 4 Problem Report and Issue Tracking

The official website for the processor and platform team is at the following URL:

<http://processor.realtek.com.tw>

On the official website, latest news, documentation, and releases of RSDK toolchain will be made available as soon as they are ready. The link to the issue tracking system can also be found on the processor website. Through the issue tracking system, any feature request and bug report will be handled in a systematic and timely fashion.

To report a problem, in addition to the detail problem description, please also clearly indicate the platform, the RSDK version, and exact way to reproduce the problem. The more details we have, the faster we can have the problem identified and nailed.

Chapter 5 Known Problems

MIPS16: File size limitation

Due to limitation in GCC 3.2, the compiler cannot handle source files of 14K lines or more in MIPS16 mode. A temporary workaround is to split the source files into multiple smaller ones. This issue will be addressed in RSDK release 1.3 which is actively under development.

MIPS16: la operand value range limitation

la is treated differently in MIPS1 and MIPS16. In MIPS16, la is an alias for addiu, which has only 15-bit of addressing power. In this case, la cannot handle PC relative load from a different section that is more than 15-bit away in address. Again, if the size of the source files is large, the compiler might emit codes that cannot be assembled by the assembler due to operand value range limitation. A temporary workaround is to split the source files into multiple smaller ones. This issue will be addressed in RSDK release 1.3 which is actively under development.

Appendix A RADIAX registers

RADIAX register name translation

For processor cores that support DSP instructions, an additional set of registers are available for programming. The mnemonic names of RADIAX registers are added to: **\$(RSDK)/include/regdef.h**

The set of registers are shown as follows:

```
#define m0l      $1
#define m0h      $2
#define m0       $3
#define m1l      $5
#define m1h      $6
#define m1       $7
#define m2l      $9
#define m2h     $10
#define m2      $11
#define m3l     $13
#define m3h     $14
#define m3      $15
#define estatus  $0
#define ecause   $1
#define intvec   $2
#define cbs0     $0
#define cbs1     $1
#define cbs2     $2
#define cbe0     $4
#define cbe1     $5
#define cbe2     $6
#define lps0     $16
#define lpe0     $17
#define lpc0     $18
#define mmd      $24
```

The RADIAX registers are treated as regular MIPS registers in the way the register name translation is processed.

The register name translation can happen in two places:

- **preprocessor:**

If preprocessor is applicable and the **regdef.h** header file is included, the preprocessor can do the following translation:

addma.s m0l, m0l, m0l => addma.s \$1, \$1, \$1

- **assembler:**

When it comes to the assembler, the register names should be prefixed with a \$ character. For example, the assembler is able to do the translation of the following form:

addma.s \$m0l, \$m0l, \$m0l => addma.s \$1, \$1, \$1

Appendix B Inline Assembly Format

Form 1

The first form of inline assembly is shown as follows:

```
asm("move $6, $8");
```

The above code will be translated literally to the code segment shown below:

```
#APP
move $6, $8
#NO_APP
```

Form 2

The second form of inline assembly is shown as follows:

```
int v1;
int v2;

asm("move %0,%1" : "=d"(v1) : "d"(v2));
```

The above inline assembly code states that: copy the value of variable v2 to variable v1 while storing v1 and v2 in general registers. The compiler will generate the actual assembly code as follows:

```
#APP
move $6, $8
#NO_APP
```

NOTE: the actual register number is determined by compiler during register allocation so the actual register number might be different from the one shown above.

Form 3

The third form of inline assembly is shown as follows:

```
register int v1 asm("8");
int ret;

ret = ret + v1;
```

The compiler supports a special keyword, `asm`, for variable declaration. The above code forces compiler to allocate register 8 for variable `v1`. The translated assembly code is shown as follows:

```
#APP
addu   $8, $2, $8
#NO_APP
```

There are five alternatives to the above form. The `$8`, `%8`, and `#8` are internally supported in the compiler. The `name` stands for the alias of the register.

```
register int v1 asm("8");
register int v1 asm("$8");
register int v1 asm("%8");
register int v1 asm("#8");
register int v1 asm("name");
```

NOTE: The compiler optimization, `-O`, has the priority over register number assignment in this form. If `-O` is turned on, the compiler might assign a different register number than the one specified in the inline assembly code.

Appendix C RELEASE NOTE

RSDK Release 1.2

We are pleased to announce the release of RSDK version 1.2 on Mar. 17, 2006. RSDK is the toolchain which supports Realtek's in-house processor cores. Version 1.2.0 is the first release candidate for branch 1.2. Any latter update fixes bugs found in version 1.2.0.

What's new in release 1.2

1. Better MIPS16 support

RSDK 1.2.x provides better MIPS16 support. The GCC has been upgraded from 3.2 to 3.2.3. Several bugs related to MIPS16 have been fixed during the transition from branch 1.1 to branch 1.2. The binutils has been upgraded from 2.14 to 2.16.1, which solved many MIPS16 related linking problem. Finally, the `__attribute__((mips16))` attribute has been added to enable mixture of MIPS1/MIPS16 codes in a single file.

2. gcc-3.2.3

The gcc has been upgraded from 3.2 to 3.2.3. Load and store twin-word instructions in functions as well as function prologue and epilogue are disabled by default. (lt/st for RLX5181, LX5280 and ltw for RLX4181). They can be enabled by using the following options.

`-flt/-fst/-fltw`
enable lt/st (RLX5181,LX5280), ltw (RLX4181)

`-ftword-stack`
enable twin-word stack instruction in function's prologue and epilogue.

Below is the summary of the bug fixes and changes.

- * Fixed gcc delay slot optimization bug
- * Patched newlib strlen inline assembly for LX4180, RLX4181, RLX5181
- * Added `__m[4180,4181,5181,5280]` preprocessor definition in CPP_SPEC
- * Added `__attribute__((mips16))` support
- * Added configurable twin-word stack operations
- * Made soft-float a default option in mips16

3. binutils-2.16.1

The binutils has been upgraded from 2.14 to 2.16.1.
The binutils 2.16.1 provides better MIPS16 support.

Below is the summary of the bug fixes and changes.

- * Fixed branch delay slot optimization bug
- * Added error messages when mixing MIPS16 and MIPS32 objects.
- * Added instructions for the MAC-DIV module.
- * Added missing MIPS16 instructions

4. newlib-1.14.0

The newlib C library has been upgraded from 1.13.0 to 1.14.0

5. uClibc-0.9.27

The memcpy and memset functions in uClibc 0.9.27 have been patched to improve performance by using word copy as much as possible and to avoid unalign load/store instructions which are not supported on RLX/LX processor cores.

CPUs supported by RSDK release 1.2

1. LX4180:
up to RTL release 4.0.2
2. RLX4181
up to RTL release 1.1
3. LX5280
up to RTL release 1.9.3
4. RLX5181
up to RTL release 1.2

RSDK Release 1.1

We are pleased to announce the release of RSDK version 1.1 on Dec. 22, 2005. RSDK is the toolchain which supports Realtek's in-house processor cores. Version 1.1.0 is the first official release following the standard release procedure.

Features

1. Full regression test

This version of RSDK has underwent a complete regression test on each of its components in the hope to provide a stable and

reliable develop environment. While the regression test may not be completely error-proof, this will be one of the standard procedure for future releases and more corner cases will be covered with the continuous addition of tests.

2. Multi-platform

Both Linux and Cygwin development platforms are supported to deliver RSDK toolchain to developers' familiar environment. Two sets of toolchains are provided, one under Linux and the other under Cygwin.

3. Multi-libc

Both newlib and uClibc C libraries are supported to provide developers a flexible choice of target system. Under each platform, two separate toolchains are delivered, one for newlib and the other for uClibc.

4. Multi-lib

Multiple processors are supported. Developers may choose the target processor by using the `-march`, `-mtune`, or `-mcpu` switches. Supported processors are 4180, 4181, 5181, and 5280.

5. Basic MIPS16 support

This version of RSDK supports basic MIPS16 extension.

6. Issue Tracking

In parallel to the release of RSDK version 1.1, a issue tracking system (Mantis) is created to aid and enhance the usability of RSDK toolchain. Any bug or feature request may be submitted through the issue tracking system and will be handled in a hopefully timely fashion. The issue tracking system is available at http://cadinfo/cgi-bin/rcs_issue_dtd.pl under the "processor and platform" project.

What's in RSDK Release 1.1

The RSDK toolchain is derived from a collection of GNU utilities. The set of tools is summarized as follows:

1. gcc-3.2
C/C++ compiler
2. binutils-2.14
Linker, assembler, object file manipulation utilities
3. newlib-1.13.0
A trimmed-down version of C library with a flexible libgloss interface for porting to different systems

4. uClibc-0.9.27
A Linux-based C library. (All previous compiled binaries should be recompiled to link with this C library)
5. insight 6.0
A TK/TCL based graphical debugger interface and a GNU gdb debugger backend

CPUs supported by RSDK release 1.1

1. LX4180:
up to RTL release 4.0.2
2. RLX4181
up to RTL release 1.0
3. LX5280
up to RTL release 1.9.3
4. RLX5181
up to RTL release 1.1

For more information, please contact the DTD processor team:

Ching-Yeh Yu (cyyu@realtek.com.tw)
Chen-You Huang (jyhuang@realtek.com.tw)
Ching-Tung Wu (tonywu@realtek.com.tw)
Chi-Feng Wu (cfwu@realtek.com.tw)

Appendix D Change Log

version 1.2.8

- * Fix `rlx_library` CP3 metric
- * Add `libgcov` to `rlx_library` suite
- * Add `-fdafile-relative` to generate `.da` in relative path

version 1.2.7

- * Fix `-ftword-stack` to emit twin-word instructions only in function prologue/epilogue
- * Fix RADIAX register number check in inline assembly
- * Remove RADIAX register alias name translation
- * Fix `sltu` instruction constraints in MIPS16
- * Fix text/data mismatch under certain circumstances in MIPS16
- * Revert `-fprofiling-arcs` `-fctest-coverage` to comply with GNU
- * Add `-frlxcov` option for RLX code coverage analysis
- * Avoid inserting `load_memory` instruction in branch delay slot when mode switches between MIPS1/MIPS16
- * Add RSDK supplementary library module
- * Fix `__attribute__((far_call))` on library functions

version 1.2.6

- * Fix inline function test
- * Fix redundant function removal test
- * Increase `PARAM_MAX_INLINE_INSNS` from 600 to 1200

version 1.2.5

- * Add `*.s` to default assembly extension
- * Change insight to support RADIAX instruction by default
- * Add debug-mode C libraries to RSDK distribution for function profiling

version 1.2.4

- * Fix gcc to support `mdebug` format
- * Merge `-flt/-fst/-fltw` to `-ftword` in gcc
- * Fix `ATTRIBUTE_SENTINEL` definition in `ansidecl.h`
- * Fix `atomicity.h` in RSDK release
- * Add `ansidecl.h`, `symcat.h`, `bfd.h` to include directory in RSDK release
- * Fix `andsi` instruction pattern in MIPS16 mode

version 1.2.3

- * Set correct -march under MIPS1/MIPS16 mix mode

version 1.2.2

- * Fix ld segmentation fault in binutils
- * Enable __attribute__((mips16)) and __attribute__((nomips16)) in both mips16 and mips32
- * Clean up RADIAX and non-RADIAX mode for RLX5181/LX5280
- * Fix c++ include path search

version 1.2.1

- * Fix -msave-restore-mmd under alloca
- * Fix RADIAX instruction under non-RADIAX mode
- * Fix default arch under -mipsxx in binutils
- * Fix rlxutils under cygwin

version 1.2.0

- * Disable -mt0-t3 under MIPS16
- * Fix inlineable test for functions with variable args
- * Add deret instruction to binutils
- * Fix non-MIPS16 instruction under __attribute__((mips16))

version 1.2.0-rc3

- * Fix unaligned load/store instructions in uClibc string functions
- * Enhanced memcpy and memset functions in uClibc
- * Fix GCC sluggish compilation speed under attribute mips16
- * Fix assembler display message under duplicate arches

version 1.2.0-rc2

- * Fix MIPS16 return address macro
- * Fix RLX4181 test conditions for pattern "mulhisi3"
- * Fix MIPS16 movsi to load correct \$gp value.
- * Fix internal compiler error in dwarf2out_frame_debug_expr

version 1.2.0-rc1

- * Fix ltw displacement display
- * Fix assembler core dump under duplicate arches

version 1.2.0-pre

- * Initial version of RSDK toolchain branch 1.2
- * Upgrade gcc from 3.2 to 3.2.3
- * Upgrade binutils from 2.14 to 2.16.1
- * Upgrade newlib from 1.13.0 to 1.14.0