RSDK Toolchain User Guide

Realtek Semiconductor Corp.
Release 1.3.6
April 24, 2009

RSDK Toolchain User Guide for Release 1.3.6

Contents

List of Tables

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 1.3.6 . 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 1.3.6 and table 1.2 shows the list of RLX processor core supported in 1.3.6 .

Software Component

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

Table 1.1: Software Components

Software	Original Version	RSDK Version
gcc	3.4.6	3.4.6-1.3.6
binutils	2.16.94	2.16.94-1.3.6
insight	6.4	6.4-1.3.6
newlib	1.14.0	1.14.0-1.3.6
uclibc	0.9.28	0.9.28-1.3.6

Example 1: RSDK Version number

```
% rsdk-elf-gcc --version
rsdk-elf-gcc (GCC) 3.4.6-1.3.5
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 2.16.94-1.3.5
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 2.16.94-1.3.5
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.
```

Supported Processor Cores

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

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

Supported Platform

The list of supported platforms is summarized in table 1.3.

Table 1.3: Supported RSDK platforms

Platform	Version	Package name
Linux	RedHat 7.3 and above	rsdk-1.3.6 -linux.tar.gz
Cygwin	Cygwin 1.5.10 and above	rsdk-1.3.6 -cygwin.tar.gz

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 1.4: Supported C Libraries

Library	Version
Newlib	1.14.0
uClibc	0.9.28

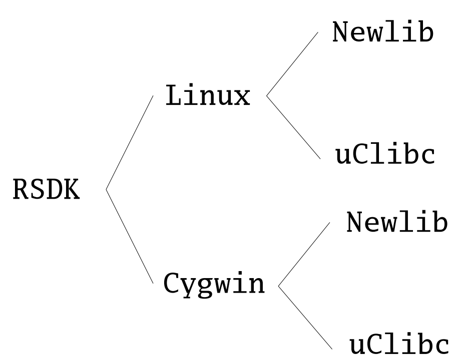
Installation

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

rsdk-1.3.6 -linux.tar.gz
rsdk-1.3.6 -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 symbolicly 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.

Example 2: RSDK installation procedure

```
step  1: % cd TARGET_DIR

step 2a: NEWLIB toolchain
step 2a: % gzip -cd rsdk-{VERSION}-{PLATFORM}-newlib.tar.gz | tar xvf -
step 3a: % ln -s rsdk-{VERSION}/{PLATFORM}/{LIBC} rsdk

step 2b: UCLIBC toolchain
step 2a: % gzip -cd rsdk-{VERSION}-{PLATFORM}-uclibc.tar.gz | tar xvf -
step 2b: % ln -s rsdk-{VERSION}/{PLATFORM}/uclibc rsdk

step 3: 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 1.3.6 , gcc has been upgraded from 3.2.3 to 3.4.6 for performance improvement and bug fixes. The list of changes is summarized in the following subsections.

General Changes

Relative path search

GCC searches a list of predefined paths for binaries, libraries, and header files during compilation. In RSDK 1.3.6 , GCC has been patched to search paths relative to the GCC binary to ensure maximal portability.

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
-meb | -EB	Enable big-endian
-march=4180	Set default target to LX4180 if none is specified

Machine-dependent options

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

Architecture: LX4180, RLX4181, RLX5181, LX5280
Summary: Specify target processor core
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.

-mt0-t3

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

By MIPS ABI 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, incremented number of 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 single instruction multiple data support
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	typedef int v2hi __attribute__((mode(V2HI))	VNB

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
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

Table 2.3: Built-in functions defined for SIMD instructions

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

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: Add an extra nop for FPGA boards
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

Example 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:

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:

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

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

Description:

In general, profiling 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.

-mno-data-in-code

Architecture: LX4180, RLX4181, RLX5181, LX5280
Summary: Separate data and code section
Dependency: None
Status: New
Description:

When specified, the compiler will not emit codes that embed data in the text section, instead, the compiler will allocate data symbols explicitly in the data section. This results in better IMEM/ICACHE utilization by removing data dependency from the text segment. This option is especially useful in MIPS16 mode where code size is critical.

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
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
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: Current
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: Current
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.

-frlxgcov

Architecture: LX4180, RLX4181, RLX5181, LX5280
Summary: Enable code coverage analysis using RLX library
Dependency: None
Status: New
Description:

When -frlxgcov is specified, compiler will emit codes to do coverage analysis for basic blocks. This option is similar to the combination of '-fprofile-arcs -ftest-coverage' except that it uses the RSDK supplementary library which supports remote file I/O over GDB remote serial protocol.

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

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 GCOV relative .da file generation
Dependency: -fprofile-arcs -ftest-coverage
Status: New
Description:

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

-fmerge-constants

Architecture: LX4180, RLX4181, RLX5181, LX5280
Summary: Attempt to merge identical string constants across compilation units
Dependency: -O and above
Status: New
Description:

If this option is enabled, compiler will attempt to merge identical string constants across compilation units by putting string literals and/or floating point constants in dedicate sections .rodata.str1.4 and .rodata.cst4. The benefit is that the code size can be reduced because duplicate constants are removed. For Linux kernel 2.4, the reduction in code size is in the magnitude of mega bytes.

In original GCC 3.4, this option is the default for optimized compilation if the assembler and linker support it. This option was enabled at levels -O, -O2, -O3, and -Os. In other words, if optimization is turned on, the compiler will remove duplicate constants by merging them into dedicate sections.

In RSDK 1.3, this option is always turned off unless explicitly switched on by specifying -fmerge-constants.

NOTE: If this option is turned on, the linker script must explicitly include the two dedicate sections .rodata.str1.4 and .rodata.cst4 if they are not already dealt with. Example is shown as follows:

```
. . . .

.data    :
{
  _fdata = . ;
  *(.data)

  *(.rodata.cst4)             /* merged constant */
  *(.rodata.str1.4)           /* merged string literals */

 /* Align the initial ramdisk image (INITRD) on page boundaries. */
 . = ALIGN(4096);
 __rd_start = .;
 *(.initrd)
 __rd_end = .;
 . = ALIGN(4096);

  CONSTRUCTORS
}

. . . .
```

-fuse-uls

Architecture: RLX4181, RLX5181
Summary: Enable unaligned load/store instructions
Dependency: None
Status: New
Description:

When this option is specified, GCC will generate unaligned load/store instructions whenever possible.

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 this option is specified, compiler will emit instructions in functions prologue and epilogue to jump to the predefined profiling functions for general program profiling, e.g. user applications. These two options are identical.

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 1: Example of using __attribute__((far_call))

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

int func()
{
    ....
}

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

The purpose of this 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 2.

Program 2: Example of using __attribute__((mips16))

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

int func2()
{
    ....
}

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

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
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
-march|mtune|mcpu=4180	-D__m4180
-march|mtune|mcpu=4181	-D__m4181
-march|mtune|mcpu=5181	-D__m5181
-march|mtune|mcpu=5280	-D__m5280

Program 3: 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
```

Deprecated Options

-mgpopt | -mno-gpopt

The -mgpopt switch says to write all of the data declarations before the instructions in the text section, this allows the MIPS assembler to generate one word memory references instead of using two words for short global or static data items. This is on by default if optimization is selected.

Both options, -mgpopt and -mno-gpopt, have been deprecated since GCC 3.3. They have been merged into the -G option.

```
-G num
    Put global and static items less than or equal to num bytes into the
small data or bss section instead of the normal data or bss section.
This allows the data to be accessed using a single instruction.

    All modules should be compiled with the same -G num value.


  -G 0 ==> -mno-gpopt
```

If the num is 0, gpopt optimization is turned off, otherwise, gpopt will be effective.

Chapter 3 Binutils

The binutils tool set is based on the GNU binutils package. The binutils tool set includes assembler, linker, and object file manipulation utilities, such as ar, nm, objdump, and objcopy. In RSDK 1.3.6 , the binutils has been upgraded from version 2.14 to 2.16.1 to provide a more reliable and a more stable development environment for both MIPS1 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.

MIPS1 and MIPS16 mix mode

Linking MIPS16 objects into MIPS1 mode is not supported at this moment. Due to the current design in the binutils, the linker cannot handle linking MIPS16 objects within MIPS1 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 4.

In RSDK 1.3.6 , 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 5.

Program 4: Example of mixing MIPS1 and MIPS16 objects

```
...
1:
  jalx 1b+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 5: 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
```

Run-Time OPCODE Table

In RSDK 1.3.6 , binutils has been patched to support dynamic opcode tables. This is done by storing the opcode table in an external file and by loading the external file during run-time. The Run-Time opcode table mechanism enables a single toolchain to support multiple instruction sets. This feature is useful for projects with custom engines and user defined instructions.

An example of opcode table is shown as follows:

```
/* These instructions appear first so that the disassembler will find
   them first.  The assemblers uses a hash table based on the
   instruction name anyhow.  */
  name,    args,      match,      mask,       pinfo,      pinfo2,     membership
{"pref",   "k,o(b)",  0xcc000000, 0xfc000000, RD_b,       0,          I4|I32|G3},
{"prefx",  "h,t(b)",  0x4c00000f, 0xfc0007ff, RD_b|RD_t,  0,          I4|I33},
{"nop",    "",        0x00000000, 0xffffffff, 0,          INSN2_ALIAS,I1}
```

Instruction Fields

Each instruction in the opcode table contains seven arguments. They are **name**, **args**, **match**, **mask**, **pinfo**, **pinfo2**, and **membership**. These seven arguments define the mnemonic name and the format, as well as information for encoding and decoding for an instruction. The detail for each argument is explained in the following subsection.

- **name:**
  This field is the name of the instruction.
- **args:**
  This field is a string describing the arguments for this instruction.

  These are the characters which may appear in the args field of an instruction. They appear in the order in which the fields appear when the instruction is used. Commas and parentheses in the args string are ignored when assembling, and written into the output when disassembling.

Each of these characters corresponds to a mask field defined above.

```
"<": 5 bit shift amount (OP_*_SHAMT)
">": shift amount between 32 and 63, stored after subtracting 32 (OP_*_SHAMT)
"a": 26 bit target address (OP_*_TARGET)
"b": 5 bit base register (OP_*_RS)
"c": 10 bit breakpoint code (OP_*_CODE)
"d": 5 bit destination register specifier (OP_*_RD)
"h": 5 bit prefx hint (OP_*_PREFX)
"i": 16 bit unsigned immediate (OP_*_IMMEDIATE)
"j": 16 bit signed immediate (OP_*_DELTA)
"k": 5 bit cache opcode in target register position (OP_*_CACHE)
       Also used for immediate operands in vr5400 vector insns.
"o": 16 bit signed offset (OP_*_DELTA)
"p": 16 bit PC relative branch target address (OP_*_DELTA)
"q": 10 bit extra breakpoint code (OP_*_CODE2)
"r": 5 bit same register used as both source and target (OP_*_RS)
"s": 5 bit source register specifier (OP_*_RS)
"t": 5 bit target register (OP_*_RT)
"u": 16 bit upper 16 bits of address (OP_*_IMMEDIATE)
"v": 5 bit same register used as both source and destination (OP_*_RS)
"w": 5 bit same register used as both target and destination (OP_*_RT)
"U": 5 bit same destination register in both OP_*_RD and OP_*_RT
       (used by clo and clz)
"C": 25 bit coprocessor function code (OP_*_COPZ)
"B": 20 bit syscall/breakpoint function code (OP_*_CODE20)
"J": 19 bit wait function code (OP_*_CODE19)
"x": accept and ignore register name
"z": must be zero register
"K": 5 bit Hardware Register (rdhwr instruction) (OP_*_RD)
"+A": 5 bit ins/ext position, which becomes LSB (OP_*_SHAMT).
Enforces: 0 <= pos < 32.
"+B": 5 bit ins size, which becomes MSB (OP_*_INSMSB).
Requires that "+A" or "+E" occur first to set position.
Enforces: 0 < (pos+size) <= 32.
"+C": 5 bit ext size, which becomes MSBD (OP_*_EXTMSBD).
Requires that "+A" or "+E" occur first to set position.
Enforces: 0 < (pos+size) <= 32.
(Also used by "dext" w/ different limits, but limits for
that are checked by the M_DEXT macro.)
"+E": 5 bit dins/dext position, which becomes LSB-32 (OP_*_SHAMT).
Enforces: 32 <= pos < 64.
"+F": 5 bit "dinsm" size, which becomes MSB-32 (OP_*_INSMSB).
Requires that "+A" or "+E" occur first to set position.
Enforces: 32 < (pos+size) <= 64.
"+G": 5 bit "dextm" size, which becomes MSBD-32 (OP_*_EXTMSBD).
Requires that "+A" or "+E" occur first to set position.
Enforces: 32 < (pos+size) <= 64.
"+H": 5 bit "dextu" size, which becomes MSBD (OP_*_EXTMSBD).
Requires that "+A" or "+E" occur first to set position.
Enforces: 32 < (pos+size) <= 64.

   Floating point instructions:
   "D" 5 bit destination register (OP_*_FD)
   "M" 3 bit compare condition code (OP_*_CCC) (only used for mips4 and up)
   "N" 3 bit branch condition code (OP_*_BCC) (only used for mips4 and up)
   "S" 5 bit fs source 1 register (OP_*_FS)
   "T" 5 bit ft source 2 register (OP_*_FT)
   "R" 5 bit fr source 3 register (OP_*_FR)
   "V" 5 bit same register used as floating source and destination (OP_*_FS)
```

```
   "W" 5 bit same register used as floating target and destination (OP_*_FT)

   Coprocessor instructions:
   "E" 5 bit target register (OP_*_RT)
   "G" 5 bit destination register (OP_*_RD)
   "H" 3 bit sel field for (d)mtc* and (d)mfc* (OP_*_SEL)
   "P" 5 bit performance-monitor register (OP_*_PERFREG)
   "e" 5 bit vector register byte specifier (OP_*_VECBYTE)
   "%" 3 bit immediate vr5400 vector alignment operand (OP_*_VECALIGN)
   see also "k" above
   "+D" Combined destination register ("G") and sel ("H") for CP0 ops,
for pretty-printing in disassembly only.

   Macro instructions:
   "A" General 32 bit expression
   "I" 32 bit immediate (value placed in imm_expr).
   "+I" 32 bit immediate (value placed in imm2_expr).
   "F" 64 bit floating point constant in .rdata
   "L" 64 bit floating point constant in .lit8
   "f" 32 bit floating point constant
   "l" 32 bit floating point constant in .lit4

   MDMX instruction operands (note that while these use the FP register
   fields, they accept both $fN and $vN names for the registers):
   "O" MDMX alignment offset (OP_*_ALN)
   "Q" MDMX vector/scalar/immediate source (OP_*_VSEL and OP_*_FT)
   "X" MDMX destination register (OP_*_FD)
   "Y" MDMX source register (OP_*_FS)
   "Z" MDMX source register (OP_*_FT)

   DSP ASE usage:
   "3" 3 bit unsigned immediate (OP_*_SA3)
   "4" 4 bit unsigned immediate (OP_*_SA4)
   "5" 8 bit unsigned immediate (OP_*_IMM8)
   "6" 5 bit unsigned immediate (OP_*_RS)
   "7" 2 bit dsp accumulator register (OP_*_DSPACC)
   "8" 6 bit unsigned immediate (OP_*_WRDSP)
   "9" 2 bit dsp accumulator register (OP_*_DSPACC_S)
   "0" 6 bit signed immediate (OP_*_DSPSFT)
   ":" 7 bit signed immediate (OP_*_DSPSFT_7)
   "'" 6 bit unsigned immediate (OP_*_RDDSP)
   "@" 10 bit signed immediate (OP_*_IMM10)

   MT ASE usage:
   "!" 1 bit immediate at bit 5
   "$" 1 bit immediate at bit 4
   "*" 2 bit dsp/smartmips accumulator register (OP_*_MTACC_T)
   "&" 2 bit dsp/smartmips accumulator register (OP_*_MTACC_D)
   "g" 5 bit coprocessor 1 and 2 destination register (OP_*_RD)
   "+t" 5 bit coprocessor 0 destination register (OP_*_RT)
   "+T" 5 bit coprocessor 0 destination register (OP_*_RT) - disassembly only

   Other:
   "()" parens surrounding optional value
   ","  separates operands
   "[]" brackets around index for vector-op scalar operand specifier (vr5400)
   "+"  Start of extension sequence.

   Characters used so far, for quick reference when adding more:
   "34567890"
```

```
"%[]<>(),+:'@!$*&"
"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
"abcdefghijklopqrstuvwxz"

Extension character sequences used so far ("+" followed by the
following), for quick reference when adding more:
"ABCDEFGHIT"
"t"
```

- **match:**
  The basic opcode for the instruction. When assembling, this opcode is modified by the arguments to produce the actual opcode that is used. If pinfo is INSN_MACRO, then this is 0.

- **mask:**
  If pinfo is not INSN_MACRO, then this is a bit mask for the relevant portions of the opcode when disassembling. If the actual opcode anded with the match field equals the opcode field, then we have found the correct instruction. If pinfo is INSN_MACRO, then this fiend is the macro identifier.

- **pinfo:**
  For a macro, this is INSN_MACRO. Otherwise, it is a collection of bits describing the instruction, notably any relevant hazard information.

  These are the bits which may be set in the pinfo field of an instructions, if it is not equal to INSN_MACRO.

```
WR_d: /* Modifies the general purpose register in OP_*_RD.  */
WR_t: /* Modifies the general purpose register in OP_*_RT.  */
WR_31: /* Modifies general purpose register 31.  */
WR_D: /* Modifies the floating point register in OP_*_FD.  */
WR_S: /* Modifies the floating point register in OP_*_FS.  */
WR_T: /* Modifies the floating point register in OP_*_FT.  */
RD_s: /* Reads the general purpose register in OP_*_RS.  */
RD_t: /* Reads the general purpose register in OP_*_RT.  */
RD_S: /* Reads the floating point register in OP_*_FS.  */
RD_T: /* Reads the floating point register in OP_*_FT.  */
RD_R: /* Reads the floating point register in OP_*_FR.  */
WR_CC: /* Modifies coprocessor condition code.  */
RD_CC: /* Reads coprocessor condition code.  */

/* TLB operation.  */
#define INSN_TLB                    0x00002000
RD_C0: /* Reads coprocessor register other than floating point register.  */
RD_C1: /* Reads coprocessor register other than floating point register.  */
RD_C2: /* Reads coprocessor register other than floating point register.  */
RD_C3: /* Reads coprocessor register other than floating point register.  */

/* Instruction loads value from memory, requiring delay.  */
#define INSN_LOAD_MEMORY_DELAY      0x00008000

/* Instruction loads value from coprocessor, requiring delay.  */
#define INSN_LOAD_COPROC_DELAY     0x00010000
/* Instruction has unconditional branch delay slot.  */
#define INSN_UNCOND_BRANCH_DELAY    0x00020000
/* Instruction has conditional branch delay slot.  */
#define INSN_COND_BRANCH_DELAY      0x00040000
/* Conditional branch likely: if branch not taken, insn nullified.  */
#define INSN_COND_BRANCH_LIKELY     0x00080000
/* Moves to coprocessor register, requiring delay.  */
#define INSN_COPROC_MOVE_DELAY      0x00100000
/* Loads coprocessor register from memory, requiring delay.  */
#define INSN_COPROC_MEMORY_DELAY    0x00200000
/* Reads the HI register.  */
```

```
#define INSN_READ_HI     0x00400000
/* Reads the LO register.  */
#define INSN_READ_LO     0x00800000
/* Modifies the HI register.  */
#define INSN_WRITE_HI     0x01000000
/* Modifies the LO register.  */
#define INSN_WRITE_LO     0x02000000
/* Takes a trap (easier to keep out of delay slot).  */
#define INSN_TRAP                   0x04000000
/* Instruction stores value into memory.  */
#define INSN_STORE_MEMORY     0x08000000
/* Instruction uses single precision floating point.  */
#define FP_S     0x10000000
/* Instruction uses double precision floating point.  */
#define FP_D     0x20000000
/* Instruction is part of the tx39's integer multiply family.    */
#define INSN_MULT                   0x40000000
/* Instruction synchronize shared memory.  */
#define INSN_SYNC     0x80000000

/* These are the bits which may be set in the pinfo2 field of an
   instruction. */

/* Instruction is a simple alias (I.E. "move" for daddu/addu/or) */
#define INSN2_ALIAS     0x00000001
/* Instruction reads MDMX accumulator. */
#define INSN2_READ_MDMX_ACC     0x00000002
/* Instruction writes MDMX accumulator. */
#define INSN2_WRITE_MDMX_ACC     0x00000004

/* Instruction is actually a macro.  It should be ignored by the
   disassembler, and requires special treatment by the assembler.  */
#define INSN_MACRO                  0xffffffff

/* Masks used to mark instructions to indicate which MIPS ISA level
   they were introduced in.  ISAs, as defined below, are logical
   ORs of these bits, indicating that they support the instructions
   defined at the given level.  */
```

- **pinfo2:**
  A collection of additional bits describing the instruction.

- **membership:**
  A collection of bits describing the instruction sets of which this instruction or macro is a member.

  For RLX processor cores, the following ISA sets are defined:

  – INSN_ISA4180
  – INSN_ISA4181
  – INSN_ISA5181
  – INSN_ISA5280

Custom UDI Instructions

To add custom UDI instructions to the binutils, a text-based UDI instruction table must be constructed. This UDI instruction table is used to encode assembly code and to decode between machine object code.

UDI Instruction File

Example UDI instruction file is shown as follows:

```
/*
 * RLX UDI instruction list.
 */
struct mips_opcode rlx_udi_opcodes[] =
{
/* Lexra opcode extensions. Register mode */
{"udi0", "d,v,t", 0x00000038, 0xfc0007ff, WR_d|RD_s|RD_t, 0, RLX2 },
{"udi1", "d,v,t", 0x0000003a, 0xfc0007ff, WR_d|RD_s|RD_t, 0, RLX2 },
{"udi2", "d,v,t", 0x0000003b, 0xfc0007ff, WR_d|RD_s|RD_t, 0, RLX2 },
{"udi3", "d,v,t", 0x0000003c, 0xfc0007ff, WR_d|RD_s|RD_t, 0, RLX2 },
{"udi4", "d,v,t", 0x0000003e, 0xfc0007ff, WR_d|RD_s|RD_t, 0, RLX2 },
{"udi5", "d,v,t", 0x0000003f, 0xfc0007ff, WR_d|RD_s|RD_t, 0, RLX2 },

/* Lexra opcode extensions. Immediate mode */
{"udi0i", "t,r,j", 0x60000000, 0xfc000000, WR_t | RD_s, 0, RLX2 },
{"udi1i", "t,r,j", 0x64000000, 0xfc000000, WR_t | RD_s, 0, RLX2 },
{"udi2i", "t,r,j", 0x68000000, 0xfc000000, WR_t | RD_s, 0, RLX2 },
{"udi3i", "t,r,j", 0x6c000000, 0xfc000000, WR_t | RD_s, 0, RLX2 },
};
```

Instruction File Manipulation

The manipulation of ISA file is done via a utility program, rsdk-elf-opcutil, which is shipped with the RSDK toolchain package.

The usage of rsdk-elf-opcutil is shown as follows:

```
sh% ./rsdk-elf-opcutil

RLX Binutils OPCODE Util v1.3

usage: ./rsdk-elf-opcutil [-hlv] [-i [isa.bin]] [-r isa.bin]
        -h: help
        -l: list opcode table tags
        -d: show ISA info of the default file
        -i: show ISA info of the file
        -r: replace opcode table
        -v: verbose output
```

- **-l**: the -l option lists the current supported opcode tables. The output includes name, description, number of ISA, and number of UDI instructions of each opcode table. Example is shown as follows:

  ```
  sh% ./rsdk-elf-opcutil -l

  RLX Binutils OPCODE Util v1.3

  optree[0] = RLX, RLX opcode v1.3 rev 1, num_isa = 1144, num_udi = 10
  optree[1] = DVR, DVR opcode v1.3 rev 1, num_isa = 1144, num_udi = 88
  ```

- **-d**: the -d option shows details of the current opcode table. The output includes the file name, tag, number of ISA, and number of UDI instructions of the opcode table. Example is shown as follows:

```
sh% ./rsdk-elf-opcutil -d

RLX Binutils OPCODE Util v1.3

Filename: ../mips-elf/bin/rlx-isa.bin
TAG: RLX opcode v1.3 rev 1
ISA: 1144 instructions
UDI: 10 instructions
```

- **-i**: the -i option shows details of the specified opcode file. The output includes the file name, tag, number of ISA, and number of UDI instructions of the opcode table. Example is shown as follows:

```
sh% ./rsdk-elf-opcutil -i rlx-isa.bin

RLX Binutils OPCODE Util v1.3

Filename: rlx-isa.bin
TAG: RLX opcode v1.3 rev 1
ISA: 1144 instructions
UDI: 10 instructions
```

- **-r**: the -r option replaces the current opcode table file with the specified opcode tag. Users can use -l option to find out the supported opcode tags.

```
sh% ./rsdk-elf-opcutil -l

RLX Binutils OPCODE Util v1.3

optree[0] = RLX, RLX opcode v1.3 rev 1, num_isa = 1144, num_udi = 10
optree[1] = VENUS, DVR-VENUS opcode v1.3 rev 1, num_isa = 1144, num_udi = 88
optree[2] = MARS, DVR-MARS opcode v1.3 rev 1, num_isa = 1144, num_udi = 250

sh% ./rsdk-elf-opcutil -r VENUS

RLX Binutils OPCODE Util v1.3

Changing opcode table to VENUS ...

TAG: VENUS
REV: DVR-VENUS opcode v1.3 rev 1
ISA: 1144 instructions
UDI: 88 instructions
```

Chapter 4 Problem Report

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.

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 Porting Linux kernel 2.4 to RSDK 1.3

When Linux Kernel 2.4 was being developed, the main GCC compiler version used was GCC 3.2. However, in RSDK 1.3, the base GCC compiler used is GCC 3.4. GCC 3.4 is a better compiler in terms of strictness of the compiler and performance of the compiled code. There might exist minor incompatibility issues for source codes that are not fully compliant with GCC 3.4 coding standard. The kernel 2.4 has hit some of the traps.

There are a number of changes that need to be applied on kernel 2.4 for the kernel to work with GCC 3.4. The list of incompatibilities is summarized as follows and is explained in details in the following subsections:

- __FUNCTION__ GCC extension
- save_static_function macro
- kernel linker script

__FUNCTION__ extension

The __FUNCTION__ is a GCC extension that does does string catenation with the current function name. This extension has been changed since GCC 3.4 and might be removed completely in future releases. Therefore, the kernel source needs to be modified accordingly.

save_static_function

In kernel 2.4, save_static_function is a macro that save additional registers before falling through the next function. It is usually invoked as follows:

```
save_static_function(sys_sigsuspend);
static_unused int _sys_sigsuspend(struct pt_regs regs)
{
    ........
}
```

While it used to work in GCC 3.2 (RSDK 1.2), it is not working in GCC 3.4 (RSDK 1.3) due to the dead code elimination and function reordering in GCC 3.4 optimization phase. The result is a unusable kernel binary.

There are a number of patches, which can be found on google, for the save_static_function to built with GCC 3.4. One of the patches, from http://osdir.com/ml/ports.mips.general/2004-12/msg00039.html, is shown as follows:

```
Index: kernel/Makefile
===================================================================
RCS file: /linux/linux/arch/mips/kernel/Makefile,v
retrieving revision 1.2
diff -c -p -r1.2 Makefile
*** kernel/Makefile     2 Dec 2004 19:50:05 -0000       1.2
--- kernel/Makefile     3 Dec 2004 03:00:44 -0000
*************** obj-y           += branch.o cpu-probe.o irq.o pro
*** 18,23 ****
--- 18,27 ----
```

```
                   traps.o ptrace.o reset.o semaphore.o setup.o syscall.o \
                   sysmips.o ipc.o scall_o32.o time.o unaligned.o

+ check_gcc = $(shell if $(CC) $(1) -S -o /dev/null -xc /dev/null > /dev/null
2>&1; then echo "$(1)"; else echo "$(2)"; fi)
+
+ syscall.o signal.o : override CFLAGS += $(call check_gcc,
-fno-unit-at-a-time,)
+
  obj-$(CONFIG_MODULES)         += mips_ksyms.o

  obj-$(CONFIG_CPU_R3000)               += r2300_fpu.o r2300_switch.o
Index: kernel/signal.c
===================================================================
RCS file: /linux/linux/arch/mips/kernel/signal.c,v
retrieving revision 1.1.1.2
diff -c -p -r1.1.1.2 signal.c
*** kernel/signal.c     1 Dec 2004 21:50:39 -0000       1.1.1.2
--- kernel/signal.c     3 Dec 2004 03:00:44 -0000
***************
*** 18,23 ****
--- 18,24 ----
  #include <linux/errno.h>
  #include <linux/wait.h>
  #include <linux/unistd.h>
+ #include <linux/compiler.h>

  #include <asm/asm.h>
  #include <asm/bitops.h>
*************** int copy_siginfo_to_user(siginfo_t *to,
*** 76,82 ****
   * Atomically swap in the new signal mask, and wait for a signal.
   */
  save_static_function(sys_sigsuspend);
! static_unused int _sys_sigsuspend(struct pt_regs regs)
  {
        sigset_t *uset, saveset, newset;

--- 77,84 ----
   * Atomically swap in the new signal mask, and wait for a signal.
   */
  save_static_function(sys_sigsuspend);
! __attribute_used__ static int
! _sys_sigsuspend(struct pt_regs regs)
  {
        sigset_t *uset, saveset, newset;

*************** static_unused int _sys_sigsuspend(struct
*** 102,108 ****
  }

  save_static_function(sys_rt_sigsuspend);
! static_unused int _sys_rt_sigsuspend(struct pt_regs regs)
  {
        sigset_t *unewset, saveset, newset;
          size_t sigsetsize;
--- 104,111 ----
  }

  save_static_function(sys_rt_sigsuspend);
```

```
! __attribute_used__ static int
! _sys_rt_sigsuspend(struct pt_regs regs)
  {
        sigset_t *unewset, saveset, newset;
          size_t sigsetsize;
Index: kernel/syscall.c
===================================================================
RCS file: /linux/linux/arch/mips/kernel/syscall.c,v
retrieving revision 1.1.1.2
diff -c -p -r1.1.1.2 syscall.c
*** kernel/syscall.c    1 Dec 2004 21:50:39 -0000       1.1.1.2
--- kernel/syscall.c    3 Dec 2004 03:00:44 -0000
***************
*** 25,30 ****
--- 25,31 ----
  #include <linux/slab.h>
  #include <linux/utsname.h>
  #include <linux/unistd.h>
+ #include <linux/compiler.h>
  #include <asm/branch.h>
  #include <asm/offset.h>
  #include <asm/ptrace.h>
*************** sys_mmap2(unsigned long addr, unsigned l
*** 158,164 ****
  }

  save_static_function(sys_fork);
! static_unused int _sys_fork(struct pt_regs regs)
  {
        int res;

--- 159,166 ----
  }

  save_static_function(sys_fork);
! __attribute_used__ static int
! _sys_fork(struct pt_regs regs)
  {
        int res;

*************** static_unused int _sys_fork(struct pt_re
*** 168,174 ****


  save_static_function(sys_clone);
! static_unused int _sys_clone(struct pt_regs regs)
  {
        unsigned long clone_flags;
        unsigned long newsp;
--- 170,177 ----


  save_static_function(sys_clone);
! __attribute_used__ static int
! _sys_clone(struct pt_regs regs)
  {
        unsigned long clone_flags;
        unsigned long newsp;
```

Kernel 2.4 linker script

If -fmerge-constants compiler option is enabled, the linux kernel linker script, **arch/mips/ld.script.in** and/or **arch/mips/ld.script** must be modified to accommodate the two extra sections, **.rodata.cst4** and **.rodat.str1.4.** The modified linker script for kernel 2.4 is shown as follows:

```
OUTPUT_ARCH(mips)
ENTRY(kernel_entry)
SECTIONS
{
  /* Read-only sections, merged into text segment: */
  /*. = @@LOADADDR@@;*/
  . = 0x80000000 ;
  .init          : { *(.init) } =0
  .text      :
  {
    _ftext = . ;
    *(.text)
    *(.rodata)
    *(.rodata1)
    /* .gnu.warning sections are handled specially by elf32.em.  */
    *(.gnu.warning)
  } =0
  .kstrtab : { *(.kstrtab) }

  . = ALIGN(16); /* Exception table */
  __start___ex_table = .;
  __ex_table : { *(__ex_table) }
  __stop___ex_table = .;

  __start___dbe_table = .; /* Exception table for data bus errors */
  __dbe_table : { *(__dbe_table) }
  __stop___dbe_table = .;

  __start___ksymtab = .; /* Kernel symbol table */
  __ksymtab : { *(__ksymtab) }
  __stop___ksymtab = .;

  _etext = .;

  . = ALIGN(8192);
  .data.init_task : { *(.data.init_task) }

  /* Startup code */
  . = ALIGN(4096);
  __init_begin = .;
  .text.init : { *(.text.init) }
  .data.init : { *(.data.init) }
  . = ALIGN(16);
  __setup_start = .;
  .setup.init : { *(.setup.init) }
  __setup_end = .;
  __initcall_start = .;
  .initcall.init : { *(.initcall.init) }
  __initcall_end = .;
  . = ALIGN(4096); /* Align double page for init_task_union */
  __init_end = .;

  . = ALIGN(4096);
```

```
.data.page_aligned : { *(.data.idt) }

. = ALIGN(32);
.data.cacheline_aligned : { *(.data.cacheline_aligned) }

.fini      : { *(.fini)    } =0
.reginfo : { *(.reginfo) }
/* Adjust the address for the data segment.  We want to adjust up to
   the same address within the page on the next page up.  It would
   be more correct to do this:
     . = .;
   The current expression does not correctly handle the case of a
   text segment ending precisely at the end of a page; it causes the
   data segment to skip a page.  The above expression does not have
   this problem, but it will currently (2/95) cause BFD to allocate
   a single segment, combining both text and data, for this case.
   This will prevent the text segment from being shared among
   multiple executions of the program; I think that is more
   important than losing a page of the virtual address space (note
   that no actual memory is lost; the page which is skipped can not
   be referenced).  */
. = .;
.data    :
{
  _fdata = . ;
  *(.data)

  *(.rodata.cst4)
  *(.rodata.str1.4)

 /* Align the initial ramdisk image (INITRD) on page boundaries. */
 . = ALIGN(4096);
 __rd_start = .;
 *(.initrd)
 __rd_end = .;
 . = ALIGN(4096);

  CONSTRUCTORS
}
.data1   : { *(.data1) }
_gp = . + 0x8000;
.lit8 : { *(.lit8) }
.lit4 : { *(.lit4) }
.ctors         : { *(.ctors)   }
.dtors         : { *(.dtors)   }
.got           : { *(.got.plt) *(.got) }
.dynamic       : { *(.dynamic) }
/* We want the small data sections together, so single-instruction offsets
   can access them all, and initialized data all before uninitialized, so
   we can shorten the on-disk segment size.  */
.sdata     : { *(.sdata) }
. = ALIGN(4);
_edata  =  .;
PROVIDE (edata = .);

__bss_start = .;
_fbss = .;
.sbss      : { *(.sbss) *(.scommon) }
.bss       :
{
```

```
   *(.dynbss)
   *(.bss)
   *(COMMON)
   .  = ALIGN(4);
  _end = . ;
  PROVIDE (end = .);
  }

  /* Sections to be discarded */
  /DISCARD/ :
  {
        *(.text.exit)
        *(.data.exit)
        *(.exitcall.exit)
  }

  /* This is the MIPS specific mdebug section.  */
  .mdebug : { *(.mdebug) }
  /* These are needed for ELF backends which have not yet been
     converted to the new style linker.  */
  .stab 0 : { *(.stab) }
  .stabstr 0 : { *(.stabstr) }
  /* DWARF debug sections.
     Symbols in the .debug DWARF section are relative to the beginning of the
     section so we begin .debug at 0.  It's not clear yet what needs to happen
     for the others.   */
  .debug          0 : { *(.debug) }
  .debug_srcinfo  0 : { *(.debug_srcinfo) }
  .debug_aranges  0 : { *(.debug_aranges) }
  .debug_pubnames 0 : { *(.debug_pubnames) }
  .debug_sfnames  0 : { *(.debug_sfnames) }
  .line           0 : { *(.line) }
  /* These must appear regardless of  .  */
  .gptab.sdata : { *(.gptab.data) *(.gptab.sdata) }
  .gptab.sbss : { *(.gptab.bss) *(.gptab.sbss) }
  .comment : { *(.comment) }
  .note : { *(.note) }
}
```

Appendix D RELEASE NOTE

```
----------------
RSDK Release 1.3
----------------

We are pleased to announce the release of RSDK version 1.3
on Sep. 29, 2006. RSDK is the software development kit that
supports Realtek's in-house processor cores. Version 1.3.0 is
the first stable release for branch 1.3.

What's new in release 1.3

1. gcc-3.4.6

The gcc has been upgraded from 3.2.3 to 3.4.6. gcc release 3.4 has a lot
of enhancement over gcc release 3.2, not only in code size reduction,
but also in performance optimization.
A brief summary is listed as follows:

  * Better inter-procedural optimization

  * More realistic code size estimates used by inlining for C

  * A new loop optimizer and two loop transformations --
       loop peeling and loop unswitching

  * Better register allocation

  * Better MIPS ABIs compliance

  * Better optimization in code size reduction

A new option -mno-data-in-code has been added to the 1.3 release.
If this option is enabled, gcc will not generate code that
embeds data in the text section. Instead, constant data will
always be allocated in the data section. The benefit is that the
text section can be more compact, hence a better IMEM/ICACHE
utilization. This option is especially useful in MIPS16 where
code size is critical.

Load and store twin-word instructions in function bodies, prologues,
and epilogues are disabled by default. (lt/st for RLX5181, LX5280
and ltw for RLX4181). The twin-word instructions can be enabled by
using the following options.

-ftword
enable lt/st (RLX5181,LX5280), ltw (RLX4181)

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

GCC will attempt to merge identical constants across compilation units through the -fmerge-constants option. This option is the default for optimized compilation if the assembler and linker support it. When this option is enabled, GCC will allocate constants in two dedicate sections, .rodata.cst4 and .rodata.str1.4, so that duplicate constants can be identified and removed. The linker script must be modified to handle these two sections if they are not already dealt with.

In RSDK 1.3, the -fmerge-constants option is turned off by default unless it is explicitly switched on. This is to ease the migration efforts for users from 1.2 to 1.3.

2. uClibc-0.9.28

The uClibc C library has been upgraded from 0.9.27 to 0.9.28. The memcpy and memset functions in uClibc 0.9.28 have also been patched for RLX/LX processor cores to improve performance by using word copy as much as possible and to avoid unsupported unalign load/store instructions.

3. Insight-6.4

The Insight/GDB has been upgraded from 6.0 to 6.4. Insight 6.4 provides better MIPS1, MIPS16, and MIPS1/MIPS16 mixed mode debugging support.

4. RSDK Supplementary Library Module

A supplementary library module has been added to enrich RSDK's capability in functional profiling, performance tuning, and remote debugging. The supplementary library includes the following four modules:

a. CP3 library - CP3 performance counter
b. Profiler library - function-level profiling support
c. GDB I/O - remote I/O via GDB remote serial protocol
d. RLXCOV - RLX code coverage analysis library

CPUs supported by RSDK release 1.3

1. LX4180:
   up to RTL release 4.0.2

2. RLX4181
   up to RTL release 1.2

3. LX5280
   up to RTL release 1.9.3

4. RLX5181
   up to RTL release 1.3

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
official release 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 instructions 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 and platform 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 E Change Log

```
version 1.3.5

    * Fix libgcc shared library naming
    * Update binutils to 2.16.94
    * Fix ld section start address miscalculation
    * Fix ltw relocation check
    * Fix ld no memory region warings for unused sections

version 1.3.4

    * Add libuls to RSDK supplementary library
    * Add rsdk-linux- symbolic link for uclibc toolchain
    * Fix insight mips sim compiler warnings
    * Fix RADIAX instruction parsing

version 1.3.3

    * Add DVR Mars OPCODE support
    * Add DVR Venus OPCODE support
    * Fix uClibc ld share library loader path
    * Fix mips16/mips1 inline function mix-mode compilation
    * Add kernel 2.4 porting guide

version 1.3.2

    * Add run-time opcode support
    * Add default asm/offset.h
    * Change GCC to put zeroed globa data in data/sdata
    * Fix gdb breakpoint address calculation on symbols

version 1.3.1

    * Add bi-endian support
    * Add pthread library debug support
    * Fix uClibc multilib support
    * Fix uClibc bi-endian support
    * Fix objdump segmentation fault on invalid encoding
    * Fix multa2 code generation
    * Fix RADIAX register name translation
    * Disable -KPIC by default
    * Fix NOP insertion for LT/ST

version 1.3.0

    * Add -fdafile-relative option
    * Disable -fmerge-constants by default
    * Initial version of RSDK toolchain branch 1.3
    * Upgrade gcc from 3.2.3 to 3.4.6
    * Upgrade uclibc from 0.9.27 to 0.9.28
```

```
* Upgrade insight/gdb from 6.0 to 6.4
```