



RSDK Supplementary Library Module

Realtek Semiconductor Corp.

Release 1.2.11

February 15, 2008

Realtek Proprietary and Confidential

RSDK Supplementary Library Module1.2.11

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	CP3 Library	1
1.1	Introduction	1
1.2	Interface	1
2	GDB Remote I/O Library	7
2.1	Introduction	7
2.2	Interface	7
3	Profiler Library	17
3.1	Introduction	17
3.2	Interface	18
4	RLX Code Coverage Library	29
4.1	Introduction	29
4.2	Interface	29
A	Example - CP3 Library	33
A.1	Example program	33
A.2	Example C run-time	33
A.3	Example makefile	36
A.4	Example linker script	36
B	Example - GDB Library	41
B.1	Example program	41
B.2	Example C run-time	41
B.3	Example makefile	43
B.4	Example linker script	44

C	Example - Profiler Library	49
C.1	Example program	49
C.2	Example C run-time	50
C.3	Example makefile	53
C.4	Example linker script	53
C.5	Example interrupt handler	57
D	Example - RLX Code Coverage Library	61
D.1	Example program	61
D.2	Example C run-time	62
D.3	Example makefile	64
D.4	Example linker script	64

List of Tables

2.1	rlx_gdb_open flags	8
2.2	rlx_gdb_open modes	8
2.3	rlx_gdb_lseek flag	12
2.4	rlx_gdb_printf format	14
2.5	rlx_gdb_printf format	15

List of Figures

Chapter 1 CP3 Library

1.1 Introduction

Co-Processor3 (CP3) implements a modest set of 48-bit wide counters for monitoring certain performance parameters. There are four counters, each of which can be programmed to count one of the set of parameters. The counters are general read/write registers in CP3.

The CP3 library is a set of subroutines that makes use of the CP3 to achieve high accuracy CPU performance statistics.

In order to eliminate the impact on the cache and memory system casted by the CP3 library itself, the CP3 library should be put into the uncacheable region of the memory. This can be done by dedicating a ELF section which roots in proper memory region for CP3 library in the linker script. Examples are shown in [Appendix A](#).

1.2 Interface

The APIs for the CP3 performance counter library are summarized as follows:

- **rlx_cp3_init** – init CP3 counters
- **rlx_cp3_start** – start CP3 counting
- **rlx_cp3_get_counters** – save CP3 counters
- **rlx_cp3_print_counters** – print CP3 counters
- **rlx_cp3_stop** – stop CP3 counting

The synopses of these APIs are described in the following pages.

- **RLX_CP3_INIT**

NAME

rlx_cp3_init – init CP3 counters

SYNOPSIS

```
#include "rlx_library.h"

void
rlx_cp3_init(void)
```

DESCRIPTION

rlx_cp3_init initializes the CP3 performance counting mechanism. First, it enables CP3 performance counting by updating STATUS register (CP0 general register \$12). Then, it resets the CP3 counters to zero.

RETURN VALUE

This function does not return any value.

- **RLX_CP3_START**

NAME

rlx_cp3_start – start CP3 counting

SYNOPSIS

```
#include "rlx_library.h"

void
rlx_cp3_start(unsigned int mode)
```

DESCRIPTION

rlx_cp3_start marks the start of the CP3 counting. The **mode** is an unsigned int that defines the counting mode. The counting mode is a combination of 1-byte constant that denotes what each counter in the CP3 registers is for. Valid constants are listed as follows:

```
/* set counter controls:          0          1          2          3
 * counter0:          cycles (0x10)  (0x10)  (0x10)  (0x10)
 * counter1:          ifetches (0x11)  (0x11)  (0x11)
 * counter1:          dmiss busy                                (0x1b)
 * counter2:          ld+stores (0x16)                                (0x16)
 * counter2:          icache misses                (0x12)  (0x12)
 * counter3:          dcache misses (0x1a)  (0x1a)                                (0x1a)
 * counter3:          imiss busy                                (0x13)
 */
```

Four frequently used counting modes are pre-defined in `rlx_cp3.h`. For example, `CP3_PERFMODE1 (0x1a121110)` stands for (dcache misses, icache misses, ifetches, cycles) counting mode.

```
#define CP3_PERFMODE0 0x1a161110
#define CP3_PERFMODE1 0x1a121110
#define CP3_PERFMODE2 0x13121110
#define CP3_PERFMODE3 0x1a161b10
```

RETURN VALUE

This function does not return any value.

- **RLX_CP3_STOP**

NAME

rlx_cp3_stop – stop CP3 counting

SYNOPSIS

```
#include "rlx_library.h"

void
rlx_cp3_stop(void)
```

DESCRIPTION

rlx_cp3_stop marks the stop of the CP3 counting.

- **RLX_CP3_GET_COUNTERS**

NAME

rlx_cp3_get_counters - save CP3 counters to an external array

SYNOPSIS

```
#include "rlx_library.h"

void
rlx_cp3_get_counters(CP3_COUNTER *counters)
```

DESCRIPTION

rlx_cp3_get_counters retrieves the numbers stored in the CP3 counters and saves them into **counters**. **counters** is a pointer to a CP3_COUNTER array of four elements.

RETURN VALUE

This function does not return any value.

- **RLX_CP3_PRINT_COUNTERS**

NAME

rlx_cp3_print_counters – print readable CP3 counters

SYNOPSIS

```
#include "rlx_library.h"

void
rlx_cp3_print_counters(unsigned int mode, CP3_COUNTER *counters)
```

DESCRIPTION

rlx_cp3_print_counters interprets the saved CP3 numbers and displays them in a human-readable form. It takes two parameters. **mode** is the counting mode that is used when starting the CP3 counters. **counters** is a pointer to the array of saved numbers. The example output is shown as follows:

```
counter control = 0x1a121110
counter0 = 000097571862 CPU cycles.
counter1 = 000102877702 Instructions.
counter2 = 000000003016 Icache Misses.
counter3 = 000000000269 Dcache Misses.
```

RETURN VALUE

This function does not return any value.

Chapter 2 GDB Remote I/O Library

2.1 Introduction

The GDB Remote Serial Protocol (RSP) is a simple, ASCII message-based protocol suitable for use on serial lines, local area networks, or just about any other communications medium that can support at least half-duplex data exchange.

The GDB remote I/O library uses a predefined address, **0x80000090**, for passing parameters. At this moment, this address is not configurable. Please make sure the application does not use the memory within 1K range starting at **0x80000090**.

The GDB remote I/O library is a set of subroutines that bases on the RSP protocol to emulate I/O interactions during remote debugging.

2.2 Interface

The APIs for the GDB Remote I/O library are summarized as follows:

- **rlx_gdb_open** – remote file open
- **rlx_gdb_close** – remote file close
- **rlx_gdb_read** – remote file read
- **rlx_gdb_write** – remote file write
- **rlx_gdb_lseek** – remote file lseek
- **rlx_gdb_fstat** – remote file fstat
- **rlx_gdb_isatty** – remote isatty
- **rlx_gdb_printf** – remote printf
- **rlx_gdb_fprintf** – remote fprintf

The synopses of these APIs are described in the following pages.

- **RLX_GDB_OPEN**

NAME

rlx_gdb_open – remote file open via GDB remote protocol

SYNOPSIS

```
#include "rlx_library.h"

int
rlx_gdb_open(const char *pathname, int flags, mode_t mode)
```

DESCRIPTION

rlx_gdb_open implements the **open** system call via the GDB remote protocol.

flags is the bitwise OR of the following values:

Table 2.1: rlx_gdb_open flags

Flag	Mask	Description
O_RDONLY	0x0	open for reading only
O_WRONLY	0x1	open for writing only
O_RDWR	0x2	open for reading and writing
O_APPEND	0x8	open in append mode
O_CREAT	0x200	create if it does not exist
O_TRUNC	0x400	truncate to zero if in writing mode (O_RDWR O_WRONLY)
O_EXCL	0x800	return error if the file already exists in O_CREAT mode

Table 2.2: rlx_gdb_open modes

Mode	Mask	Description
S_IWOTH	0x2	others have write permission
S_IROTH	0x4	others have read permission
S_IWGRP	0x10	group has write permission
S_IRGRP	0x20	group has read permission
S_IWUSR	0x80	user has write permission
S_IRUSR	0x100	user has read permission

RETURN VALUE

rlx_gdb_open returns the new file descriptor or -1 if error.

- **RLX_GDB_CLOSE**

NAME

rlx_gdb_close – remote file close via GDB remote protocol

SYNOPSIS

```
#include "rlx_library.h"

int
rlx_gdb_close(int fd)
```

DESCRIPTION

rlx_gdb_close implements the file close system call via the GDB remote protocol.

RETURN VALUE

rlx_gdb_close returns zero on success, non-zero otherwise.

- **RLX_GDB_READ**

NAME

rlx_gdb_read – remote file read via GDB remote protocol

SYNOPSIS

```
#include "rlx_library.h"

int
rlx_gdb_read(int fd, void *buf, unsigned int count)
```

DESCRIPTION

rlx_gdb_read implements the file read system call via the GDB remote protocol.

RETURN VALUE

On success, the number of bytes read is returned. Zero indicates end of file. If count is zero, read returns zero as well.

- **RLX_GDB_WRITE**

NAME

rlx_gdb_write - remote file write via GDB remote protocol

SYNOPSIS

```
#include "rlx_library.h"

int
rlx_gdb_write(int fd, const void *buf, int count)
```

DESCRIPTION

rlx_gdb_write implements the file write system call via the GDB remote protocol.

RETURN VALUE

On success, the number of bytes written is returned. Zero indicates nothing was written.

- **RLX_GDB_LSEEK**

NAME

rlx_gdb_lseek – remote file lseek via GDB remote protocol

SYNOPSIS

```
#include "rlx_library.h"

int
rlx_gdb_lseek(int fd, int offset, int flag)
```

DESCRIPTION

rlx_gdb_lseek implements the file lseek system call via the GDB remote protocol.

Table 2.3: rlx_gdb_lseek flag

Flag	Mask	Description
SEEK_SET	0x0	the offset is set to offset bytes
SEEK_CUR	0x1	the offset is set to its current position plus offset bytes
SEEK_END	0x2	the offset is set to the size of the file plus offset bytes

RETURN VALUE

On success, the resulting unsigned offset in bytes from the beginning of the file is returned.

- **RLX_GDB_ISATTY**

NAME

rlx_gdb_isatty – remote isatty via GDB remote protocol

SYNOPSIS

```
#include "rlx_library.h"

void
rlx_gdb_isatty(int desc)
```

DESCRIPTION

rlx_gdb_isatty implements the isatty call via the GDB remote protocol.

RETURN VALUE

returns 1 if desc is an open descriptor connected to a terminal and 0 else.

- **RLX_GDB_PRINTF**

NAME

rlx_gdb_printf – remote printf via GDB remote protocol

SYNOPSIS

```
#include "rlx_library.h"

void
rlx_gdb_printf(char *format, ...)
```

DESCRIPTION

rlx_gdb_printf implements the printf call via the GDB remote protocol.

Table 2.4: rlx_gdb_printf format

Format	Description
s	print string
c	print character
x	print hex value
d	print decimal value

RETURN VALUE

This function does not return any value.

- **RLX_GDB_FPRINTF**

NAME

rlx_gdb_fprintf – remote fprintf via GDB remote protocol

SYNOPSIS

```
#include "rlx_library.h"

void
rlx_gdb_printf(int fd, char *format, ...)
```

DESCRIPTION

rlx_gdb_printf implements the fprintf call via the GDB remote protocol.

Table 2.5: rlx_gdb_printf format

Format	Description
s	print string
c	print character
x	print hex value
d	print decimal value

RETURN VALUE

This function does not return any value.

Chapter 3 Profiler Library

3.1 Introduction

RLX profiler library is a set of subroutines that collects performance statistics for user-mode applications at functional level.

To use the RLX profiler library, the compiler flags and linker options must be modified to generate specific object codes and to link proper libraries for function-level profiling. The OS must also provide functions to disable and enable interrupts for the profiler to call upon entering and leaving function prologues and function epilogue.

The default CP3 counting mode is 0x1b1a1110, which counts the following items: total number of CPU cycles, total number instruction fetches, total number of DCACHE misses, and total number of busy cycles caused by DCACHE misses.

- **Compiler** The compiler must insert codes to call profiler subroutines upon entering and leaving function prologue as well as function epilogue. In RSDK, this is done by adding **-pg** to the compiler flag.
- **Linker** The linker should put the profiler library in the uncacheable memory region to avoid impact on the memory and cache system casted by the profiler library. This can be done by dedicating ELF sections which root in proper memory location for the profiler library in the linker script.

The three sections reserved for the profiler library are **.rlxprof_text**, **.rlxprof_data**, and **.rlxprof_bss**. These sections should be put into the uncacheable region of the memory.

Two additional variables, **__rlxprof_bss_ent** and **__rlxprof_bss_end**, must be set to mark the begin and end address of the **.rlxprof_bss** section. These two variables are required for the library to correctly access the **.rlxprof_bss** section.

Example linker script is shown as follows:

```
/* Script for -z combreloc: combine and sort reloc sections */
OUTPUT_FORMAT("elf32-bigmips", "elf32-bigmips", "elf32-littlemips")
OUTPUT_ARCH(mips)
ENTRY(start)

SECTIONS
{
    /* profiler section */
    .rlxprof_text 0xa0200000 : { *(.rlxprof_text) rlx_*(.text) }
    .rlxprof_data ALIGN(0x8) : { *(.rlxprof_data) }

    __rlxprof_bss_ent = ABSOLUTE(.);
    .rlxprof_bss ALIGN(0x8) (NOLOAD) : { *(.rlxprof_bss) }
    __rlxprof_bss_end = ABSOLUTE(.);

    ...
}
```

3.2 Interface

The profiler library interface can be divided into three categories, API, Callback, and Hook. The APIs are subroutines that can be directly invoked by the user application. The Callback functions are library checkpoints that are called upon when specific events occur. The Hook functions are machine-dependent part that needs to be implemented by the developers.

The APIs are summarized as follows:

- **rlx_prof_init** – init profiler
- **rlx_prof_stop** – stop profiler
- **rlx_prof_save_result** – write rsdk_mon.out
- **rlx_prof_set_cp3_ctrl** – set CP3 counting mode

The CALLBACK functions are summarized as follows:

- **_mcount** – function prologue callback
- **_ecount** – function epilogue callback
- **rlx_prof_mcount** – called by _mcount
- **rlx_prof_ecount** – called by _ecount

The HOOK functions are summarized as follows:

- **rlx_prof_disable_int** – disable interrupt
- **rlx_prof_enable_int** – enable interrupt

- **RLX_PROF_INIT**

NAME

rlx_prof_init – init function profiling

SYNOPSIS

```
#include "rlx_library.h"

void
rlx_prof_init(void)
```

DESCRIPTION

rlx_prof_init initializes the function-level profiling mechanism. First, it zeros out the **.rlxprof_bss** section. Second, it starts the CP3 performance counting by calling the **rlx_cp3_init** and **rlx_cp3_start**.

RETURN VALUE

This function does not return any value.

- **RLX_PROF_STOP**

NAME

rlx_prof_stop - stop function profiling

SYNOPSIS

```
#include "rlx_library.h"

void
rlx_prof_stop(void)
```

DESCRIPTION

rlx_prof_stop marks the end of function profiling mechanism.

The proper place to call this function is in the crtn or crtend section after returning from the main function.

RETURN VALUE

This function does not return any value.

- **RLX_PROF_SAVE_RESULT**

NAME

rlx_prof_save_result - save profiling result to a local file

SYNOPSIS

```
#include "rlx_library.h"

void
rlx_prof_save_result(void)
```

DESCRIPTION

rlx_prof_save_result writes the profiling output to a local file , **rsdk_mon.out**, on the host via the remote GDB I/O library. The output file is a binary file that can be parsed by the **rsdk-elf-insight** program.

RETURN VALUE

This function does not return any value.

- **RLX_PROF_SET_CP3_CTRL**

NAME

rlx_prof_set_cp3_ctrl – set CP3 counting mode

SYNOPSIS

```
#include "rlx_library.h"

void
rlx_prof_set_cp3_ctrl(unsigned int mode)
```

DESCRIPTION

rlx_prof_set_cp3_ctrl sets the CP3 counting mode. The counting mode is a combination of 1-byte constant that denotes what each counter in the CP3 registers is for. Please refer to [chapter 1](#) for more information.

The default CP3 counting mode is 0x1b1a1110, which counts the following items: total number of CPU cycles, total number instruction fetches, total number of instruction fetch cache misses, and total number of busy cycles caused by instruction fetch cache.

RETURN VALUE

This function does not return any value.

- **_MCOUNT**

NAME

_mcount – function prologue callback function

SYNOPSIS

```
#include "rlx_library.h"

void
_mcount(void)
```

DESCRIPTION

_mcount is the callback function which the compiler will insert upon entering function prologue. This function, in turn, calls **rlx_prof_mcount** to collect calling graph and performance count of the function under investigation.

This function is implemented in the profiler library.

RETURN VALUE

This function does not return any value.

- **_ECOUNT**

NAME

_ecount – function epilogue callback function

SYNOPSIS

```
#include "rlx_library.h"

void
_ecount(void)
```

DESCRIPTION

_ecount is the callback function which the compiler will insert upon entering function epilogue. This function, in turn, calls **rlx_prof_ecount** to collect calling graph and performance count of the function under investigation. This function is implemented in the profiler library.

RETURN VALUE

This function does not return any value.

- **RLX_PROF_MCOUNT**

NAME

rlx_prof_mcount – 2nd level prologue callback function

SYNOPSIS

```
#include "rlx_library.h"

void
rlx_prof_mcount(void)
```

DESCRIPTION

rlx_prof_mcount is the 2nd level callback function which the **rlx_prof_func_ent** will call in function prologue. This function collects calling graph and performance count of the function under investigation.

This function is implemented in the profiler library.

RETURN VALUE

This function does not return any value.

- **RLX_PROF_ECOUNT**

NAME

rlx_prof_ecount – 2nd level epilogue callback function

SYNOPSIS

```
#include "rlx_library.h"

void
rlx_prof_ecount(void)
```

DESCRIPTION

rlx_prof_ecount is the 2nd level callback function which the **rlx_prof_func_end** will call in function epilogue. This function collects calling graph and performance count of the function under investigation.

This function is implemented in the profiler library.

RETURN VALUE

This function does not return any value.

- **RLX_PROF_ENABLE_INT**

NAME

rlx_prof_enable_int – enable interrupts

SYNOPSIS

```
#include "rlx_library.h"

void
rlx_prof_enable_int(void) __attribute__((section(".rlxprof_text")))
```

DESCRIPTION

rlx_prof_enable_int enables the interrupts. This function will be called by the profiler library after it has finished its task in function prologue or epilogue. This function should be implemented by the developer to meet the operating system and platform the application is running on.

By default, this function should be put in a special section **.rlxprof_text**. An example implementation is shown in [appendix C](#).

RETURN VALUE

This function does not return any value.

- **RLX_PROF_DISABLE_INT**

NAME

rlx_prof_disable_int - disables interrupts

SYNOPSIS

```
#include "rlx_library.h"

void
rlx_prof_disable_int(void) __attribute__((section(".rlxprof_text")))
```

DESCRIPTION

rlx_prof_disable_int disables the interrupts. This function will be called by the profiler library right before starting its task in function prologue or epilogue. This function should be implemented by the developer to meet the operation system and platform the application is running on.

By default, this function should be put in a special section **.rlxprof_text**. An example implementation is shown in appendix [C](#).

RETURN VALUE

This function does not return any value.

Chapter 4 RLX Code Coverage Library

4.1 Introduction

The RLX Code Coverage Library (RLXCOV) is a set of subroutines that help developers collect code coverage statistics in remote debugging environment. This library is similar to the GNU code coverage library (GCOV) except that the `rlxcov` library supports remote file I/O via GDB remote serial protocol.

To use the RLXCOV library, the compiler flags and linker options must be modified to generate and link proper codes and libraries to enable code coverage analysis.

- **Compiler:**

In order to enable code coverage analysis, the compiler option **-frlxcov** must be added to the compiler flag when generating codes.

- **Linker:**

The coverage analysis codes will be placed in a special **.rlxcov** section to avoid interference of the analysis statistics. Therefore, the linker script must be modified to explicitly allocate the **.rlxcov** section.

Two variables, **__rlxcov_ent** and **__rlxcov_end** must be set to mark the begin and end address of the **.rlxcov** section. These two variables are required for the library to read and parse symbols in the **.rlxcov** section.

Example linker script is shown as follows:

```
SECTIONS
{
    ....

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

    ....
}
```

Examples are shown in [Appendix D](#).

4.2 Interface

The APIs for the RLX code coverage analysis library are summarized as follows:

- **rlx_cov_init** – init code coverage library
- **rlx_cov_exit** – stop code coverage library

The synopses of these APIs are described in the following pages.

- **RLX_COV_INIT**

NAME

rlx_cov_init – init code coverage library

SYNOPSIS

```
#include "rlx_library.h"

void
rlx_cov_init(void)
```

DESCRIPTION

rlx_cov_init initializes the RLXCOV code analysis mechanism. This function must be called exactly once for the application under investigation. Upon initialization, the RLXCOV reads the symbols in the .rlxcov section and initializes bookkeeping fields for each function in the application.

The proper place to call this function is in the crt0 or crtbegin before entering the main function.

RETURN VALUE

This function does not return any value.

- **RLX_COV_EXIT**

NAME

rlx_cov_exit – stop code coverage library

SYNOPSIS

```
#include "rlx_library.h"

void
rlx_cov_exit(void)
```

DESCRIPTION

rlx_cov_exit marks the end of the code coverage analysis. This function must be called exactly once for the application under investigation.

The proper place to call this function is in the `crtn` or `crtend` after returning from the main function.

RETURN VALUE

This function does not return any value.

Appendix A Example - CP3 Library

A.1 Example program

```
#include "rlx_library.h"

int main(int argc, char **argv)
{
    CP3_COUNTER counters[4];

    /* init CP3 counters */
    rlx_cp3_init();
    rlx_cp3_start(CP3_PERFMODE1);

    /* your code here */
    printf("do something here\n");

    /* stop CP3 counters */
    rlx_cp3_stop();
    rlx_cp3_get_counters(counters);
    rlx_cp3_print_counters(CP3_PERFMODE1, counters);

    return 0;
}
```

A.2 Example C run-time

```
/*
 * Realtek Semiconductor Corp.
 *
 * c.S:
 */
#include <regdef.h>
#include "rlx_library.h"

#define PERFMODE1 0x1a121110
#define UART_DATA_ADDRESS 0xbc00000c

.set nomips16

.globl _stack_base
_stack_base:
.rept 512*1024
```

```
.byte 0
.endr

.balign 16
.globl _stack
_stack:
.rept 8
.long 0
.endr

.data
.globl my_argv
my_argv:
.rept 16
.byte 0
.endr

.data
.globl my_arg_str
my_arg_str:
.rept 256
.byte 0
.endr

.comm cp3_counter,32

.text

.globl start
.globl _exit

.ent start
start:
/*
 * Initialize COP3 counters
 */
la t0, rlx_cp3_init
jalr t0
nop
la t0, rlx_cp3_start
li a0, CP3_PERFMODEL
jalr t0
nop

/*
** Initialize data.
*/
la v0, _edata
/*
** Align pointer.
*/
add v0, 3
srl v0, 2
sll v0, 2
```

```
la v1, end
loop0: sw zero, 0x0(v0)
sw zero, 0x4(v0)
sw zero, 0x8(v0)
sw zero, 0xc(v0)
addu v0, 16
blt v0, v1, loop0

/*
** Initialize the global data pointer.
*/
la gp, _gp
    la sp, _stack

/*
** Every good C program has a main.
*/
#ifdef __mips16
jalx main
#else
jal main
#endif
nop
move a0, v0
.end start

.ent _exit
_exit:
    move s0, a0
    la t0, rlx_cp3_get_counters
    la a0, cp3_counter
    jalr t0
    nop
    la t0, rlx_cp3_print_counters
    li a0, CP3_PERFMODE1
    la a1, cp3_counter
    jalr t0
    nop
    move a0, s0
    syscall 1
.end _exit

/*
** PutC
** Instruction used to perform character output
** from programs running during simulation.
*/
.globl __PutCharacter
.ent __PutCharacter
__PutCharacter:
    li a0, UART_DATA_ADDRESS
    sw a1, 0(a0)
j ra # Return
.end __PutCharacter
```

```
.comm    __errno,4
```

A.3 Example makefile

```
#
# Realtek Semiconductor Corp.
#
# RSDK CP3 Library
#
# Tony Wu (tonywu@realtek.com.tw)
# Jul. 07, 2006
#

#####
ARCH = 4180
PROGRAM = testcp3
RSDKDIR = /rsdk/rsdk-1.2.7/linux/newlib

CC = $(RSDKDIR)/bin/rsdk-elf-gcc
LD = $(RSDKDIR)/bin/rsdk-elf-ld

CFLAGS = -march=$(ARCH) -c -G0 -fno-pic -DRLX_UNCACHEABLE
IFLAGS = -I$(RSDKDIR)/include
LFLAGS = -Ttext 80000000 -e start -N -n
LIBS = -L$(RSDKDIR)/lib/$(ARCH) -lrlx -lm -lc -lgcc -L. -lrlxsim_gdb

#####
all: $(PROGRAM)

.S.o:
$(CC) -D__ASM__ -x assembler-with-cpp $(CFLAGS) $(IFLAGS) $<

.c.o:
$(CC) $(CFLAGS) $(IFLAGS) $<

testcp3: c.o cp3.o
$(LD) -T target-rlxsim.ld -o testcp3 $(LFLAGS) c.o cp3.o $(LIBS)

clean:
rm -f *.o $(PROGRAM)
```

A.4 Example linker script

```
/* Script for -z combreloc: combine and sort reloc sections */
OUTPUT_FORMAT("elf32-bigmips", "elf32-bigmips", "elf32-littlemips")
OUTPUT_ARCH(mips)
ENTRY(start)

SECTIONS
{
    /* profiler section */
    .rlxprof_text 0xa0200000 : { *(.rlxprof_text) rlx_*(.text) }
```

```

.rlxprof_data ALIGN(0x8) : { *(.rlxprof_data) }

__rlxprof_bss_begin = ABSOLUTE(.);
.rlxprof_bss ALIGN(0x8) (NOLOAD) : { *(.rlxprof_bss) }
__rlxprof_bss_end = ABSOLUTE(.);

/* Read-only sections, merged into text segment: */
PROVIDE (__executable_start = 0x0400000); . = 0x0400000;
.interp      : { *(.interp) }
.reginfo     : { *(.reginfo) }
.dynamic     : { *(.dynamic) }
.hash        : { *(.hash) }
.dynsym      : { *(.dynsym) }
.dynstr      : { *(.dynstr) }
.gnu.version : { *(.gnu.version) }
.gnu.version_d : { *(.gnu.version_d) }
.gnu.version_r : { *(.gnu.version_r) }
.rel.dyn     :
{
    *(.rel.init)
    *(.rel.text .rel.text.* .rel.gnu.linkonce.t.*)
    *(.rel.fini)
    *(.rel.rodata .rel.rodata.* .rel.gnu.linkonce.r.*)
    *(.rel.data.rel.ro*)
    *(.rel.data .rel.data.* .rel.gnu.linkonce.d.*)
    *(.rel.tdata .rel.tdata.* .rel.gnu.linkonce.td.*)
    *(.rel.tbss .rel.tbss.* .rel.gnu.linkonce.tb.*)
    *(.rel.ctors)
    *(.rel.dtors)
    *(.rel.got)
    *(.rel.sdata .rel.sdata.* .rel.gnu.linkonce.s.*)
    *(.rel.sbss .rel.sbss.* .rel.gnu.linkonce.sb.*)
    *(.rel.sdata2 .rel.sdata2.* .rel.gnu.linkonce.s2.*)
    *(.rel.sbss2 .rel.sbss2.* .rel.gnu.linkonce.sb2.*)
    *(.rel.bss .rel.bss.* .rel.gnu.linkonce.b.*)
}
.rela.dyn    :
{
    *(.rela.init)
    *(.rela.text .rela.text.* .rela.gnu.linkonce.t.*)
    *(.rela.fini)
    *(.rela.rodata .rela.rodata.* .rela.gnu.linkonce.r.*)
    *(.rela.data .rela.data.* .rela.gnu.linkonce.d.*)
    *(.rela.tdata .rela.tdata.* .rela.gnu.linkonce.td.*)
    *(.rela.tbss .rela.tbss.* .rela.gnu.linkonce.tb.*)
    *(.rela.ctors)
    *(.rela.dtors)
    *(.rela.got)
    *(.rela.sdata .rela.sdata.* .rela.gnu.linkonce.s.*)
    *(.rela.sbss .rela.sbss.* .rela.gnu.linkonce.sb.*)
    *(.rela.sdata2 .rela.sdata2.* .rela.gnu.linkonce.s2.*)
    *(.rela.sbss2 .rela.sbss2.* .rela.gnu.linkonce.sb2.*)
    *(.rela.bss .rela.bss.* .rela.gnu.linkonce.b.*)
}
.rel.plt     : { *(.rel.plt) }
.rela.plt    : { *(.rela.plt) }
.init        :
{
    KEEP (*(init))
} =0

```

```
.plt          : { *(.plt) }
.text 0x80000000 :
{
    _ftext = . ;
    *(.text .stub .text.* .gnu.linkonce.t.*)
    KEEP (*(text.*personality*))
    /* .gnu.warning sections are handled specially by elf32.em.  */
    *(.gnu.warning)
    *(.mips16.fn.*) *(.mips16.call.*)
} =0
.fini         :
{
    KEEP (*(fini))
} =0
PROVIDE (__etext = .);
PROVIDE (_etext = .);
PROVIDE (etext = .);
.rodata       : { *(.rodata .rodata.* .gnu.linkonce.r.*) }
.rodata1      : { *(.rodata1) }
.sdata2       : { *(.sdata2 .sdata2.* .gnu.linkonce.s2.*) }
.sbss2        : { *(.sbss2 .sbss2.* .gnu.linkonce.sb2.*) }
.eh_frame_hdr : { *(.eh_frame_hdr) }
.eh_frame     : ONLY_IF_RO { KEEP (*(eh_frame)) }
.gcc_except_table : ONLY_IF_RO { KEEP (*(gcc_except_table)) *(gcc_except_table.*) }
/* Adjust the address for the data segment.  We want to adjust up to
   the same address within the page on the next page up.  */
. = ALIGN(0x40000) - ((0x40000 - .) & (0x40000 - 1)); . =
                                DATA_SEGMENT_ALIGN(0x40000, 0x1000);

/* Exception handling */
.eh_frame     : ONLY_IF_RW { KEEP (*(eh_frame)) }
.gcc_except_table : ONLY_IF_RW { KEEP (*(gcc_except_table)) *(gcc_except_table.*) }
/* Thread Local Storage sections */
.tdata       : { *(.tdata .tdata.* .gnu.linkonce.td.*) }
.tbss        : { *(.tbss .tbss.* .gnu.linkonce.tb.*) *(.tcommon) }
/* Ensure the __preinit_array_start label is properly aligned.  We
   could instead move the label definition inside the section, but
   the linker would then create the section even if it turns out to
   be empty, which isn't pretty.  */
. = ALIGN(32 / 8);
PROVIDE (__preinit_array_start = .);
.preinit_array : { KEEP (*(preinit_array)) }
PROVIDE (__preinit_array_end = .);
PROVIDE (__init_array_start = .);
.init_array   : { KEEP (*(init_array)) }
PROVIDE (__init_array_end = .);
PROVIDE (__fini_array_start = .);
.fini_array   : { KEEP (*(fini_array)) }
PROVIDE (__fini_array_end = .);
.ctors       :
{
    /* gcc uses crtbegin.o to find the start of
       the constructors, so we make sure it is
       first.  Because this is a wildcard, it
       doesn't matter if the user does not
       actually link against crtbegin.o; the
       linker won't look for a file to match a
       wildcard.  The wildcard also means that it
       doesn't matter which directory crtbegin.o
       is in.  */
    KEEP (*crtbegin*.o(.ctors))
}
```

```

/* We don't want to include the .ctor section from
   from the crtend.o file until after the sorted ctors.
   The .ctor section from the crtend file contains the
   end of ctors marker and it must be last */
KEEP (*(EXCLUDE_FILE (*crtend*.o ) .ctors))
KEEP (*(SORT(.ctors.*)))
KEEP (*(ctors))
}
.dtors      :
{
    KEEP (*crtbegin*.o(.dtors))
    KEEP (*(EXCLUDE_FILE (*crtend*.o ) .dtors))
    KEEP (*(SORT(.dtors.*)))
    KEEP (*(dtors))
}
.jcr        : { KEEP (*(jcr)) }
.data.rel.ro : { *(.data.rel.ro.local) *(.data.rel.ro*) }
. = DATA_SEGMENT_RELRO_END (0, .);
.data       :
{
    _fdata = . ;
    *(.data .data.* .gnu.linkonce.d.*)
    KEEP (*(gnu.linkonce.d.*personality*))
    SORT(CONSTRUCTORS)
}
.data1      : { *(.data1) }
. = .;
_gp = ALIGN(16) + 0x7ff0;
.got        : { *(.got.plt) *(.got) }
/* 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 .sdata.* .gnu.linkonce.s.*)
}
.lit8       : { *(.lit8) }
.lit4       : { *(.lit4) }
_edata = .;
PROVIDE (edata = .);
__bss_start = .;
_fbss = .;
.sbss       :
{
    PROVIDE (__sbss_start = .);
    PROVIDE (___sbss_start = .);
    *(.dynsbss)
    *(.sbss .sbss.* .gnu.linkonce.sb.*)
    *(.scommon)
    PROVIDE (__sbss_end = .);
    PROVIDE (___sbss_end = .);
}
.bss        :
{
    *(.dynbss)
    *(.bss .bss.* .gnu.linkonce.b.*)
    *(COMMON)
    /* Align here to ensure that the .bss section occupies space up to
       _end.  Align after .bss to ensure correct alignment even if the
       .bss section disappears because there are no input sections. */
}

```

```
. = ALIGN(32 / 8);
}
. = ALIGN(32 / 8);
_end = .;
PROVIDE (end = .);
. = DATA_SEGMENT_END (.);
/* Stabs debugging sections. */
.stab      0 : { *(.stab) }
.stabstr   0 : { *(.stabstr) }
.stab.excl 0 : { *(.stab.excl) }
.stab.exclstr 0 : { *(.stab.exclstr) }
.stab.index 0 : { *(.stab.index) }
.stab.indexstr 0 : { *(.stab.indexstr) }
.comment   0 : { *(.comment) }
/* DWARF debug sections.
   Symbols in the DWARF debugging sections are relative to the beginning
   of the section so we begin them at 0. */
/* DWARF 1 */
.debug      0 : { *(.debug) }
.line       0 : { *(.line) }
/* GNU DWARF 1 extensions */
.debug_srcinfo 0 : { *(.debug_srcinfo) }
.debug_sfnames 0 : { *(.debug_sfnames) }
/* DWARF 1.1 and DWARF 2 */
.debug_aranges 0 : { *(.debug_aranges) }
.debug_pubnames 0 : { *(.debug_pubnames) }
/* DWARF 2 */
.debug_info   0 : { *(.debug_info .gnu.linkonce.wi.*) }
.debug_abbrev 0 : { *(.debug_abbrev) }
.debug_line   0 : { *(.debug_line) }
.debug_frame  0 : { *(.debug_frame) }
.debug_str     0 : { *(.debug_str) }
.debug_loc     0 : { *(.debug_loc) }
.debug_macinfo 0 : { *(.debug_macinfo) }
/* SGI/MIPS DWARF 2 extensions */
.debug_weaknames 0 : { *(.debug_weaknames) }
.debug_funcnames 0 : { *(.debug_funcnames) }
.debug_typenames 0 : { *(.debug_typenames) }
.debug_varnames 0 : { *(.debug_varnames) }
.gptab.sdata : { *(.gptab.data) *(.gptab.sdata) }
.gptab.sbss : { *(.gptab.bss) *(.gptab.sbss) }
/DISCARD/ : { *(.note.GNU-stack) }
}
```

Appendix B Example - GDB Library

B.1 Example program

```
/*
 * Copyright (c) 2006, Realtek Semiconductor Corp.
 *
 * gdbio.c:
 *   GDB remote I/O example program
 *
 * Tony Wu (tonyw@realtek.com.tw)
 * Jul. 26, 2006
 */

#include <stdio.h>
#include "rlx_library.h"

int
main(int argc, char **argv)
{
    int fd;
    char buff[] = "This is a test string";

    fd = rlx_gdb_open("test.txt", O_CREAT | O_RDWR | O_TRUNC,
                     S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|S_IROTH|S_IWOTH);
    if (fd < 0)
    {
        rlx_gdb_printf("ERROR: unable to open the test.txt\n");
        return -1;
    }

    rlx_gdb_write(fd, buff, sizeof(buff));
    rlx_gdb_close(fd);
    return 0;
}
```

B.2 Example C run-time

```
/*
 * Realtek Semiconductor Corp.
 *
 * c.S:
 */
#include <regdef.h>
#include "rlx_library.h"
```

```
#define PERFMODE1 0x1a121110
#define UART_DATA_ADDRESS    0xbc00000c

.set nomips16

.globl _stack_base
_stack_base:
.rept 512*1024
.byte 0
.endr

.balign 16
.globl _stack
_stack:
.rept 8
.long 0
.endr

.data
.globl my_argv
my_argv:
.rept 16
.byte 0
.endr

.data
.globl my_arg_str
my_arg_str:
.rept 256
.byte 0
.endr

.comm cp3_counter,32

.text

.globl start
.globl _exit

.ent start
start:
/*
** Initialize data.
*/
la v0, _edata
/*
** Align pointer.
*/
add v0, 3
srl v0, 2
sll v0, 2
la v1, end
loop0: sw zero, 0x0(v0)
sw zero, 0x4(v0)
sw zero, 0x8(v0)
sw zero, 0xc(v0)
addu v0, 16
blt v0, v1, loop0
```

```

/*
** Initialize the global data pointer.
*/
la gp, _gp
    la sp, _stack

/*
** Every good C program has a main.
*/
#if defined(__mips16)
jalx main
#else
jal main
#endif
nop
move a0, v0
.end start

.ent _exit
_exit:
    syscall 1
.end _exit

/*
** PutC
** Instruction used to perform character output
** from programs running during simulation.
*/
.globl __PutCharacter
.ent __PutCharacter
__PutCharacter:
    li a0, UART_DATA_ADDRESS
    sw a1, 0(a0)
j ra # Return
.end __PutCharacter

.comm __errno,4

```

B.3 Example makefile

```

#
# Realtek Semiconductor Corp.
#
# RSDK GDB Remote IO Library
#
# Tony Wu (tonywu@realtek.com.tw)
# Jul. 07, 2006
#

#####
ARCH = 4180
PROGRAM = testgdbio
RSDKDIR = /rsdk/rsdk-1.2.7/linux/newlib

CC = $(RSDKDIR)/bin/rsdk-elf-gcc
LD = $(RSDKDIR)/bin/rsdk-elf-ld

CFLAGS = -march=$(ARCH) -c -G0 -fno-pic

```

```
IFLAGS = -I$(RSDKDIR)/include
LFLAGS = -Ttext 80000000 -e start -N -n
LIBS = -L$(RSDKDIR)/lib/$(ARCH) -lrlx -lm -lc -lgcc -L. -lrlxsim_gdb

#####
all: $(PROGRAM)

.S.o:
$(CC) -D__ASM__ -x assembler-with-cpp $(CFLAGS) $(IFLAGS) $<

.c.o:
$(CC) $(CFLAGS) $(IFLAGS) $<

testgdbio: c.o gdbio.o
$(LD) -T target-rlxsim.ld -o testgdbio $(LFLAGS) c.o gdbio.o $(LIBS)

clean:
rm -f *.o $(PROGRAM)
```

B.4 Example linker script

```
/* Script for -z combreloc: combine and sort reloc sections */
OUTPUT_FORMAT("elf32-bigmips", "elf32-bigmips", "elf32-littlemips")
OUTPUT_ARCH(mips)
ENTRY(start)

SECTIONS
{
    /* Read-only sections, merged into text segment: */
    PROVIDE (__executable_start = 0x0400000); . = 0x0400000;
    .interp          : { *(.interp) }
    .reginfo         : { *(.reginfo) }
    .dynamic         : { *(.dynamic) }
    .hash            : { *(.hash) }
    .dynsym          : { *(.dynsym) }
    .dynstr          : { *(.dynstr) }
    .gnu.version     : { *(.gnu.version) }
    .gnu.version_d   : { *(.gnu.version_d) }
    .gnu.version_r   : { *(.gnu.version_r) }
    .rel.dyn         :
    {
        *(.rel.init)
        *(.rel.text .rel.text.* .rel.gnu.linkonce.t.*)
        *(.rel.fini)
        *(.rel.rodata .rel.rodata.* .rel.gnu.linkonce.r.*)
        *(.rel.data.rel.ro*)
        *(.rel.data .rel.data.* .rel.gnu.linkonce.d.*)
        *(.rel.tdata .rel.tdata.* .rel.gnu.linkonce.td.*)
        *(.rel.tbss .rel.tbss.* .rel.gnu.linkonce.tb.*)
        *(.rel.ctors)
        *(.rel.dtors)
        *(.rel.got)
        *(.rel.sdata .rel.sdata.* .rel.gnu.linkonce.s.*)
        *(.rel.sbss .rel.sbss.* .rel.gnu.linkonce.sb.*)
        *(.rel.sdata2 .rel.sdata2.* .rel.gnu.linkonce.s2.*)
        *(.rel.sbss2 .rel.sbss2.* .rel.gnu.linkonce.sb2.*)
        *(.rel.bss .rel.bss.* .rel.gnu.linkonce.b.*)
    }
```

```

    }
    .rela.dyn      :
    {
        *(.rela.init)
        *(.rela.text .rela.text.* .rela.gnu.linkonce.t.*)
        *(.rela.fini)
        *(.rela.rodata .rela.rodata.* .rela.gnu.linkonce.r.*)
        *(.rela.data .rela.data.* .rela.gnu.linkonce.d.*)
        *(.rela.tdata .rela.tdata.* .rela.gnu.linkonce.td.*)
        *(.rela.tbss .rela.tbss.* .rela.gnu.linkonce.tb.*)
        *(.rela.ctors)
        *(.rela.dtors)
        *(.rela.got)
        *(.rela.sdata .rela.sdata.* .rela.gnu.linkonce.s.*)
        *(.rela.sbss .rela.sbss.* .rela.gnu.linkonce.sb.*)
        *(.rela.sdata2 .rela.sdata2.* .rela.gnu.linkonce.s2.*)
        *(.rela.sbss2 .rela.sbss2.* .rela.gnu.linkonce.sb2.*)
        *(.rela.bss .rela.bss.* .rela.gnu.linkonce.b.*)
    }
    .rel.plt      : { *(.rel.plt) }
    .rela.plt     : { *(.rela.plt) }
    .init         :
    {
        KEEP (*(init))
    } =0
    .plt          : { *(.plt) }
    .text 0x80000000 :
    {
        _ftext = . ;
        *(.text .stub .text.* .gnu.linkonce.t.*)
        KEEP (*(text.*personality*))
        /* .gnu.warning sections are handled specially by elf32.em.  */
        *(.gnu.warning)
        *(.mips16.fn.*) *(.mips16.call.*)
    } =0
    .fini         :
    {
        KEEP (*(fini))
    } =0
    PROVIDE (__etext = .);
    PROVIDE (_etext = .);
    PROVIDE (etext = .);
    .rodata       : { *(.rodata .rodata.* .gnu.linkonce.r.*) }
    .rodata1      : { *(.rodata1) }
    .sdata2       : { *(.sdata2 .sdata2.* .gnu.linkonce.s2.*) }
    .sbss2        : { *(.sbss2 .sbss2.* .gnu.linkonce.sb2.*) }
    .eh_frame_hdr : { *(.eh_frame_hdr) }
    .eh_frame     : ONLY_IF_RO { KEEP (*(eh_frame)) }
    .gcc_except_table : ONLY_IF_RO { KEEP (*(gcc_except_table)) *(gcc_except_table.*) }
    /* Adjust the address for the data segment.  We want to adjust up to
       the same address within the page on the next page up.  */
    . = ALIGN (0x40000) - ((0x40000 - .) & (0x40000 - 1)); . =
                                   DATA_SEGMENT_ALIGN (0x40000, 0x1000);

    /* Exception handling */
    .eh_frame     : ONLY_IF_RW { KEEP (*(eh_frame)) }
    .gcc_except_table : ONLY_IF_RW { KEEP (*(gcc_except_table)) *(gcc_except_table.*) }
    /* Thread Local Storage sections */
    .tdata       : { *(.tdata .tdata.* .gnu.linkonce.td.*) }
    .tbss        : { *(.tbss .tbss.* .gnu.linkonce.tb.*) *(tcommon) }
    /* Ensure the __preinit_array_start label is properly aligned.  We

```

```
    could instead move the label definition inside the section, but
    the linker would then create the section even if it turns out to
    be empty, which isn't pretty.  */
. = ALIGN(32 / 8);
PROVIDE (__preinit_array_start = .);
.preinit_array : { KEEP (*.preinit_array) }
PROVIDE (__preinit_array_end = .);
PROVIDE (__init_array_start = .);
.init_array : { KEEP (*.init_array) }
PROVIDE (__init_array_end = .);
PROVIDE (__fini_array_start = .);
.fini_array : { KEEP (*.fini_array) }
PROVIDE (__fini_array_end = .);
.ctors :
{
    /* gcc uses crtbegin.o to find the start of
    the constructors, so we make sure it is
    first.  Because this is a wildcard, it
    doesn't matter if the user does not
    actually link against crtbegin.o; the
    linker won't look for a file to match a
    wildcard.  The wildcard also means that it
    doesn't matter which directory crtbegin.o
    is in.  */
    KEEP (*crtbegin*.o(.ctors))
    /* We don't want to include the .ctor section from
    from the crtend.o file until after the sorted ctors.
    The .ctor section from the crtend file contains the
    end of ctors marker and it must be last */
    KEEP (*(EXCLUDE_FILE (*crtend*.o) .ctors))
    KEEP (*(SORT(.ctors.*)))
    KEEP (*.ctors)
}
.dtors :
{
    KEEP (*crtbegin*.o(.dtors))
    KEEP (*(EXCLUDE_FILE (*crtend*.o) .dtors))
    KEEP (*(SORT(.dtors.*)))
    KEEP (*.dtors)
}
.jcr : { KEEP (*.jcr) }
.data.rel.ro : { *(.data.rel.ro.local) *(.data.rel.ro*) }
. = DATA_SEGMENT_RELRO_END (0, .);
.data :
{
    _fdata = . ;
    *(.data .data.* .gnu.linkonce.d.*)
    KEEP (*.gnu.linkonce.d.*personality*)
    SORT(CONSTRUCTORS)
}
.data1 : { *(.data1) }
. = .;
_gp = ALIGN(16) + 0x7ff0;
.got : { *(.got.plt) *(.got) }
/* 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 .sdata.* .gnu.linkonce.s.*)
}
```

```

}
.lit8      : { *(.lit8) }
.lit4      : { *(.lit4) }
_edata = .;
PROVIDE (edata = .);
__bss_start = .;
_fbss = .;
.sbss      :
{
    PROVIDE (__sbss_start = .);
    PROVIDE (___sbss_start = .);
    *(.dynsbss)
    *(.sbss .sbss.* .gnu.linkonce.sb.*)
    *(.scommon)
    PROVIDE (__sbss_end = .);
    PROVIDE (___sbss_end = .);
}
.bss      :
{
    *(.dynbss)
    *(.bss .bss.* .gnu.linkonce.b.*)
    *(COMMON)
    /* Align here to ensure that the .bss section occupies space up to
       _end.  Align after .bss to ensure correct alignment even if the
       .bss section disappears because there are no input sections.  */
    . = ALIGN(32 / 8);
}
. = ALIGN(32 / 8);
_end = .;
PROVIDE (end = .);
. = DATA_SEGMENT_END (.);
/* Stabs debugging sections.  */
.stab      0 : { *(.stab) }
.stabstr    0 : { *(.stabstr) }
.stab.excl  0 : { *(.stab.excl) }
.stab.exclstr 0 : { *(.stab.exclstr) }
.stab.index 0 : { *(.stab.index) }
.stab.indexstr 0 : { *(.stab.indexstr) }
.comment    0 : { *(.comment) }
/* DWARF debug sections.
   Symbols in the DWARF debugging sections are relative to the beginning
   of the section so we begin them at 0.  */
/* DWARF 1 */
.debug      0 : { *(.debug) }
.line       0 : { *(.line) }
/* GNU DWARF 1 extensions */
.debug_srcinfo 0 : { *(.debug_srcinfo) }
.debug_sfnames 0 : { *(.debug_sfnames) }
/* DWARF 1.1 and DWARF 2 */
.debug_aranges 0 : { *(.debug_aranges) }
.debug_pubnames 0 : { *(.debug_pubnames) }
/* DWARF 2 */
.debug_info   0 : { *(.debug_info .gnu.linkonce.wi.*) }
.debug_abbrev 0 : { *(.debug_abbrev) }
.debug_line   0 : { *(.debug_line) }
.debug_frame  0 : { *(.debug_frame) }
.debug_str     0 : { *(.debug_str) }
.debug_loc     0 : { *(.debug_loc) }
.debug_macinfo 0 : { *(.debug_macinfo) }
/* SGI/MIPS DWARF 2 extensions */

```

```
.debug_weaknames 0 : { *(.debug_weaknames) }  
.debug_funcnames 0 : { *(.debug_funcnames) }  
.debug_typenames 0 : { *(.debug_typenames) }  
.debug_varnames 0 : { *(.debug_varnames) }  
.gptab.sdata : { *(.gptab.data) *(.gptab.sdata) }  
.gptab.sbss : { *(.gptab.bss) *(.gptab.sbss) }  
/DISCARD/ : { *(.note.GNU-stack) }  
}
```

Appendix C Example - Profiler Library

C.1 Example program

```
#include <stdarg.h>

extern void abort (void);
extern void exit (int);

void bar (int n, int c)
{
    static int lastn = -1, lastc = -1;

    if (lastn != n)
    {
        if (lastc != lastn)
        abort ();
        lastc = 0;
        lastn = n;
    }

    if (c != (char) (lastc ^ (n << 3)))
        abort ();
    lastc++;
}

#define D(N) typedef struct { char x[N+1]; } A##N;
D(0) D(1) D(2) D(3) D(4) D(5) D(6) D(7)
D(8) D(9) D(10) D(11) D(12) D(13) D(14) D(15)
D(16) D(31) D(32) D(35) D(72)
#undef D

void foo (int size, ...)
{
    #define D(N) A##N a##N;
    D(0) D(1) D(2) D(3) D(4) D(5) D(6) D(7)
    D(8) D(9) D(10) D(11) D(12) D(13) D(14) D(15)
    D(16) D(31) D(32) D(35) D(72)
    #undef D
    va_list ap;
    int i;

    if (size != 21)
        abort ();
    va_start (ap, size);
    #define D(N) \
    a##N = va_arg (ap, typeof (a##N)); \
    for (i = 0; i < N; i++) \
        bar (N, a##N.x[i]);
```

```

D(0) D(1) D(2) D(3) D(4) D(5) D(6) D(7)
D(8) D(9) D(10) D(11) D(12) D(13) D(14) D(15)
D(16) D(31) D(32) D(35) D(72)
#undef D
    va_end (ap);
}

int main (void)
{
#define D(N) A##N a##N;
D(0) D(1) D(2) D(3) D(4) D(5) D(6) D(7)
D(8) D(9) D(10) D(11) D(12) D(13) D(14) D(15)
D(16) D(31) D(32) D(35) D(72)
#undef D
    int i;

#define D(N) \
    for (i = 0; i < N; i++) \
        a##N.x[i] = i ^ (N << 3);
D(0) D(1) D(2) D(3) D(4) D(5) D(6) D(7)
D(8) D(9) D(10) D(11) D(12) D(13) D(14) D(15)
D(16) D(31) D(32) D(35) D(72)
#undef D

    foo (21
#define D(N) , a##N
D(0) D(1) D(2) D(3) D(4) D(5) D(6) D(7)
D(8) D(9) D(10) D(11) D(12) D(13) D(14) D(15)
D(16) D(31) D(32) D(35) D(72)
#undef D
    );
    exit (0);
}

```

C.2 Example C run-time

```

/*
 * Realtek Semiconductor Corp.
 *
 * c.S:
 */
#include <regdef.h>
#include "rlx_library.h"

#define UART_DATA_ADDRESS    0xbc00000c

.set nomips16

.globl _stack_base
_stack_base:
.rept 512*1024
.byte 0
.endr

.balign 16
.globl _stack
_stack:
.rept 8

```

```
.long 0
.endr

.data
.globl my_argv
my_argv:
.rept 16
.byte 0
.endr

.data
.globl my_arg_str
my_arg_str:
.rept 256
.byte 0
.endr

.comm cp3_counter,32

.text

.globl start
.globl _exit

.ent start
start:
    /*
     * init profiler
     */
    la t0, rlx_prof_init
    jalr ra, t0
    nop

    /*
     * set counting mode
     */
    la t0, rlx_prof_set_cp3_ctrl
    li a0, CP3_PERFMODEL1
    jalr ra, t0
    nop

    /*
     * start profiler
     */
    la t0, rlx_prof_start
    jalr ra, t0
    nop

    /*
     ** Initialize data.
     */
    la v0, _edata

    /*
     ** Align pointer.
     */
    add v0, 3
    srl v0, 2
    sll v0, 2
```

```
la v1, end
loop0: sw zero, 0x0(v0)
sw zero, 0x4(v0)
sw zero, 0x8(v0)
sw zero, 0xc(v0)
addu v0, 16
blt v0, v1, loop0

/*
** Initialize the global data pointer.
*/
la gp, _gp
    la sp, _stack

/*
** Every good C program has a main.
*/
#ifdef __mips16
jalx main
#else
jal main
#endif
nop
move a0, v0
.end start

.ent _exit
_exit:
/*
* save current return value
*/
move s0, a0

/*
* stop profiler
*/
la t0, rlx_prof_stop
jalr ra,t0
nop

/*
* save profiler output
*/
la t0, rlx_prof_save_result
jalr ra,t0
nop
move a0, s0
syscall 1
.end _exit

/*
** PutC
** Instruction used to perform character output
** from programs running during simulation.
*/
.globl __PutCharacter
.ent __PutCharacter
__PutCharacter:
li a0, UART_DATA_ADDRESS
sw a1, 0(a0)
```

```

j ra # Return
.end __PutCharacter

.comm __errno,4

```

C.3 Example makefile

```

#
# Realtek Semiconductor Corp.
#
# RSDK Profiler Library
#
# Tony Wu (tonywu@realtek.com.tw)
# Jul. 07, 2006
#

#####
ARCH = 4180
PROGRAM = testprof
RSDKDIR = /rsdk/rsdk-1.2.7/linux/newlib

CC = $(RSDKDIR)/bin/rsdk-elf-gcc
LD = $(RSDKDIR)/bin/rsdk-elf-ld

CFLAGS = -march=$(ARCH) -c -G0 -fno-pic -pg -DRLX_UNCACHEABLE
IFLAGS = -I$(RSDKDIR)/include
LFLAGS = -Ttext 80000000 -e start -N -n
LIBS = -L$(RSDKDIR)/lib/$(ARCH) -lrlx -lm -lc -lgcc -L. -lrlxsim_gdb

#####
all: $(PROGRAM)

.S.o:
$(CC) -D__ASM__ -x assembler-with-cpp $(CFLAGS) $(IFLAGS) $<

.c.o:
$(CC) $(CFLAGS) $(IFLAGS) $<

testprof: c.o prof.o rlx_prof_int.o
$(LD) -T target-rlxsim.ld -o testprof $(LFLAGS) c.o prof.o rlx_prof_int.o $(LIBS)

clean:
rm -f *.o $(PROGRAM)

```

C.4 Example linker script

```

/* Script for -z combreloc: combine and sort reloc sections */
OUTPUT_FORMAT("elf32-bigmips", "elf32-bigmips", "elf32-littlemips")
OUTPUT_ARCH(mips)
ENTRY(start)

SECTIONS
{
    /* profiler section */
    .rlxprof_text 0xa0200000 : { *(.rlxprof_text) rlx_*(.text) }
}

```

```
.rlxprof_data ALIGN(0x8) : { *(.rlxprof_data) }
__rlxprof_bss_begin = ABSOLUTE(.);
.rlxprof_bss ALIGN(0x8) (NOLOAD) : { *(.rlxprof_bss) }
__rlxprof_bss_end = ABSOLUTE(.);

/* Read-only sections, merged into text segment: */
PROVIDE (__executable_start = 0x0400000); . = 0x0400000;
.interp : { *(.interp) }
.reginfo : { *(.reginfo) }
.dynamic : { *(.dynamic) }
.hash : { *(.hash) }
.dynsym : { *(.dynsym) }
.dynstr : { *(.dynstr) }
.gnu.version : { *(.gnu.version) }
.gnu.version_d : { *(.gnu.version_d) }
.gnu.version_r : { *(.gnu.version_r) }
.rel.dyn :
{
    *(.rel.init)
    *(.rel.text .rel.text.* .rel.gnu.linkonce.t.*)
    *(.rel.fini)
    *(.rel.rodata .rel.rodata.* .rel.gnu.linkonce.r.*)
    *(.rel.data.rel.ro*)
    *(.rel.data .rel.data.* .rel.gnu.linkonce.d.*)
    *(.rel.tdata .rel.tdata.* .rel.gnu.linkonce.td.*)
    *(.rel.tbss .rel.tbss.* .rel.gnu.linkonce.tb.*)
    *(.rel.ctors)
    *(.rel.dtors)
    *(.rel.got)
    *(.rel.sdata .rel.sdata.* .rel.gnu.linkonce.s.*)
    *(.rel.sbss .rel.sbss.* .rel.gnu.linkonce.sb.*)
    *(.rel.sdata2 .rel.sdata2.* .rel.gnu.linkonce.s2.*)
    *(.rel.sbss2 .rel.sbss2.* .rel.gnu.linkonce.sb2.*)
    *(.rel.bss .rel.bss.* .rel.gnu.linkonce.b.*)
}
.rela.dyn :
{
    *(.rela.init)
    *(.rela.text .rela.text.* .rela.gnu.linkonce.t.*)
    *(.rela.fini)
    *(.rela.rodata .rela.rodata.* .rela.gnu.linkonce.r.*)
    *(.rela.data .rela.data.* .rela.gnu.linkonce.d.*)
    *(.rela.tdata .rela.tdata.* .rela.gnu.linkonce.td.*)
    *(.rela.tbss .rela.tbss.* .rela.gnu.linkonce.tb.*)
    *(.rela.ctors)
    *(.rela.dtors)
    *(.rela.got)
    *(.rela.sdata .rela.sdata.* .rela.gnu.linkonce.s.*)
    *(.rela.sbss .rela.sbss.* .rela.gnu.linkonce.sb.*)
    *(.rela.sdata2 .rela.sdata2.* .rela.gnu.linkonce.s2.*)
    *(.rela.sbss2 .rela.sbss2.* .rela.gnu.linkonce.sb2.*)
    *(.rela.bss .rela.bss.* .rela.gnu.linkonce.b.*)
}
.rel.plt : { *(.rel.plt) }
.rela.plt : { *(.rela.plt) }
.init :
{
    KEEP (*(init))
} =0
.plt : { *(.plt) }
```

```

.text 0x80000000 :
{
    _ftext = . ;
    *(.text .stub .text.* .gnu.linkonce.t.*)
    KEEP (*( .text.*personality*))
    /* .gnu.warning sections are handled specially by elf32.em.  */
    *(.gnu.warning)
    *(.mips16.fn.*) *(.mips16.call.*)
} =0
.fini :
{
    KEEP (*( .fini))
} =0
PROVIDE (__etext = .);
PROVIDE (_etext = .);
PROVIDE (etext = .);
.rodata : { *(.rodata .rodata.* .gnu.linkonce.r.*) }
.rodata1 : { *(.rodata1) }
.sdata2 : { *(.sdata2 .sdata2.* .gnu.linkonce.s2.*) }
.sbss2 : { *(.sbss2 .sbss2.* .gnu.linkonce.sb2.*) }
.eh_frame_hdr : { *(.eh_frame_hdr) }
.eh_frame : ONLY_IF_RO { KEEP (*( .eh_frame)) }
.gcc_except_table : ONLY_IF_RO { KEEP (*( .gcc_except_table)) *(.gcc_except_table.*) }
/* Adjust the address for the data segment.  We want to adjust up to
   the same address within the page on the next page up.  */
. = ALIGN (0x40000) - ((0x40000 - .) & (0x40000 - 1)); . =
                                DATA_SEGMENT_ALIGN (0x40000, 0x1000);

/* Exception handling */
.eh_frame : ONLY_IF_RW { KEEP (*( .eh_frame)) }
.gcc_except_table : ONLY_IF_RW { KEEP (*( .gcc_except_table)) *(.gcc_except_table.*) }
/* Thread Local Storage sections */
.tdata : { *(.tdata .tdata.* .gnu.linkonce.td.*) }
.tbss : { *(.tbss .tbss.* .gnu.linkonce.tb.*) *(.tcommon) }
/* Ensure the __preinit_array_start label is properly aligned.  We
   could instead move the label definition inside the section, but
   the linker would then create the section even if it turns out to
   be empty, which isn't pretty.  */
. = ALIGN(32 / 8);
PROVIDE (__preinit_array_start = .);
.preinit_array : { KEEP (*( .preinit_array)) }
PROVIDE (__preinit_array_end = .);
PROVIDE (__init_array_start = .);
.init_array : { KEEP (*( .init_array)) }
PROVIDE (__init_array_end = .);
PROVIDE (__fini_array_start = .);
.fini_array : { KEEP (*( .fini_array)) }
PROVIDE (__fini_array_end = .);
.ctors :
{
    /* gcc uses crtbegin.o to find the start of
       the constructors, so we make sure it is
       first.  Because this is a wildcard, it
       doesn't matter if the user does not
       actually link against crtbegin.o; the
       linker won't look for a file to match a
       wildcard.  The wildcard also means that it
       doesn't matter which directory crtbegin.o
       is in.  */
    KEEP (*crtbegin*.o(.ctors))
    /* We don't want to include the .ctor section from

```

```
    from the crtend.o file until after the sorted ctors.
    The .ctor section from the crtend file contains the
    end of ctors marker and it must be last */
KEEP (*(EXCLUDE_FILE (*crtend*.o ) .ctors))
KEEP (*(SORT(.ctors.*)))
KEEP (*(ctors))
}
.dtors      :
{
    KEEP (*crtbegin*.o(.dtors))
    KEEP (*(EXCLUDE_FILE (*crtend*.o ) .dtors))
    KEEP (*(SORT(.dtors.*)))
    KEEP (*(dtors))
}
.jcr        : { KEEP (*(jcr)) }
.data.rel.ro : { *(.data.rel.ro.local) *(.data.rel.ro*) }
. = DATA_SEGMENT_RELRO_END (0, .);
.data       :
{
    _fdata = . ;
    *(.data .data.* .gnu.linkonce.d.*)
    KEEP (*(gnu.linkonce.d.*personality*))
    SORT(CONSTRUCTORS)
}
.data1      : { *(.data1) }
. = .;
_gp = ALIGN(16) + 0x7ff0;
.got        : { *(.got.plt) *(.got) }
/* 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 .sdata.* .gnu.linkonce.s.*)
}
.lit8       : { *(.lit8) }
.lit4       : { *(.lit4) }
_edata = .;
PROVIDE (edata = .);
__bss_start = .;
_fbss = .;
.sbss       :
{
    PROVIDE (__sbss_start = .);
    PROVIDE (___sbss_start = .);
    *(.dynsbss)
    *(.sbss .sbss.* .gnu.linkonce.sb.*)
    *(.scommon)
    PROVIDE (__sbss_end = .);
    PROVIDE (___sbss_end = .);
}
.bss        :
{
    *(.dynbss)
    *(.bss .bss.* .gnu.linkonce.b.*)
    *(COMMON)
    /* Align here to ensure that the .bss section occupies space up to
       _end.  Align after .bss to ensure correct alignment even if the
       .bss section disappears because there are no input sections. */
    . = ALIGN(32 / 8);
}
```

```

}
. = ALIGN(32 / 8);
_end = .;
PROVIDE (end = .);
. = DATA_SEGMENT_END (.);
/* Stabs debugging sections. */
.stab          0 : { *(.stab) }
.stabstr       0 : { *(.stabstr) }
.stab.excl     0 : { *(.stab.excl) }
.stab.exclstr  0 : { *(.stab.exclstr) }
.stab.index    0 : { *(.stab.index) }
.stab.indexstr 0 : { *(.stab.indexstr) }
.comment       0 : { *(.comment) }
/* DWARF debug sections.
   Symbols in the DWARF debugging sections are relative to the beginning
   of the section so we begin them at 0. */
/* DWARF 1 */
.debug          0 : { *(.debug) }
.line          0 : { *(.line) }
/* GNU DWARF 1 extensions */
.debug_srcinfo  0 : { *(.debug_srcinfo) }
.debug_sfnames  0 : { *(.debug_sfnames) }
/* DWARF 1.1 and DWARF 2 */
.debug_aranges  0 : { *(.debug_aranges) }
.debug_pubnames 0 : { *(.debug_pubnames) }
/* DWARF 2 */
.debug_info     0 : { *(.debug_info.gnu.linkonce.wi.*) }
.debug_abbrev   0 : { *(.debug_abbrev) }
.debug_line     0 : { *(.debug_line) }
.debug_frame    0 : { *(.debug_frame) }
.debug_str      0 : { *(.debug_str) }
.debug_loc      0 : { *(.debug_loc) }
.debug_macinfo  0 : { *(.debug_macinfo) }
/* SGI/MIPS DWARF 2 extensions */
.debug_weaknames 0 : { *(.debug_weaknames) }
.debug_funcnames 0 : { *(.debug_funcnames) }
.debug_typenames 0 : { *(.debug_typenames) }
.debug_varnames  0 : { *(.debug_varnames) }
.gptab.sdata : { *(.gptab.data) *(.gptab.sdata) }
.gptab.sbss : { *(.gptab.bss) *(.gptab.sbss) }
/DISCARD/ : { *(.note.GNU-stack) }
}

```

C.5 Example interrupt handler

C example

```

/*
 * Copyright (c) 2006, Realtek Semiconductor Corp.
 *
 * rlx_prof_int.c:
 *   Enable/Disable interrupts
 *
 * Zhe Jiang (zhe_jiang@realtek.com.cn)
 * Tony Wu (tonywu@realtek.com.tw)
 *
 * Jul. 19, 2006

```

```

*/

#include <rlx_library.h>

#undef printf
int printf(const char *, ...) __attribute__((far_call));

void
rlx_prof_disable_int(void) __attribute__((section(".rlxprof_text")))
{
    /* disable interrupt here */
    printf("to be implemented\n");
}

void
rlx_prof_enable_int(void) __attribute__((section(".rlxprof_text")))
{
    /* enable interrupt here */
    printf("to be implemented\n");
}

```

ASM example

```

/*
 * Copyright (c) 2006, Realtek Semiconductor Corp.
 *
 * rlx_profiler_int.S:
 *   Enable/Disable interrupts
 *
 * Zhe Jiang (zhe_jiang@realsil.com.cn)
 * Tony Wu (tonywu@realtek.com.tw)
 *
 * Jul. 19, 2006
 */

#include <regdef.h>

.globl rlx_prof_disable_int
.section .rlxprof_text
.ent rlx_prof_disable_int
rlx_prof_disable_int:
    .set noreorder
    /* save the status register and disable interrupt */
    mfc0 t4,$12
    nop
    or t5,t4,$0
    lui t6,0xffff
    ori t6,0xfffc
    and t4,t4,t6
    mtc0 t4,$12
    nop
    nop
    lui t7,%hi(old_status_register)
    sw t5,%lo(old_status_register)(t7)
#if defined(_ROS_)
    /* add this code to stop timer */
    lui t4,0xbd01
    lw t4,0x84(t4)
    lui t5,%hi(profiler_timer_value)

```

```
sw    t4,%lo(profiler_timer_value)(t5)
lui   t6,0xbd01
ori   t4,0x202
sw    t4,0x84(t6)
jr    $31
sw    t4,0x84(t6)
#endif
.set  reorder
.end  _ros_enter_call_back

.globl rlx_prof_enable_int
.section .rlxprof_text
.ent   rlx_prof_enable_int
rlx_prof_enable_int:
.set   noreorder
/* restore the status register */
lui    t5,%hi(old_status_register)
lw     t5,%lo(old_status_register)(t5)
lui    t6,%hi(profiler_timer_value)
lw     t6,%lo(profiler_timer_value)(t6)
mtc0   t5,$12
nop
nop
#if defined(_ROS_)
/* restore the timer status */
lui    t7,0xbd01
jr     $31
sw     t6,0x84(t7)
#endif
.set   reorder
.end   rlx_prof_eanble_int
```


Appendix D Example - RLX Code Coverage Library

D.1 Example program

```
/*
 * Copyright (c) 2006, Realtek Semiconductor Corp.
 *
 * cov.c:
 *   RLXCOV example program
 *
 * Tony Wu (tonyw@realtek.com.tw)
 * Jul. 26, 2006
 */

#include <stdio.h>
#include "rlx_library.h"

int func1(void)
{
    printf("This is function 1");
    return 0;
}

int func2(void)
{
    printf("This is function 2");
    return 0;
}

int
main(int argc, char **argv)
{
    int n;

    printf("We should do something here\n");

    n = rand();
    if (n%2 == 0) {
        func1();
    } else {
        func2();
    }

    printf("Nah.\n");
    return 0;
}
```

D.2 Example C run-time

```
/*
 * Realtek Semiconductor Corp.
 *
 * c.S:
 */
#include <regdef.h>
#include "rlx_library.h"

#define PERFMODE1 0x1a121110
#define UART_DATA_ADDRESS 0xbc00000c

.set nomips16

.globl _stack_base
_stack_base:
.rept 512*1024
.byte 0
.endr

.balign 16
.globl _stack
_stack:
.rept 8
.long 0
.endr

.data
.globl my_argv
my_argv:
.rept 16
.byte 0
.endr

.data
.globl my_arg_str
my_arg_str:
.rept 256
.byte 0
.endr

.comm cp3_counter,32

.text

.globl start
.globl _exit

.ent start
start:
/*
 * Initialize COV library
 */
    la t0, rlx_cov_init
    jalr t0
    nop
```

```

/*
** Initialize data.
*/
la v0, _edata
/*
** Align pointer.
*/
add v0, 3
srl v0, 2
sll v0, 2
la v1, end
loop0: sw zero, 0x0(v0)
sw zero, 0x4(v0)
sw zero, 0x8(v0)
sw zero, 0xc(v0)
addu v0, 16
blt v0, v1, loop0

/*
** Initialize the global data pointer.
*/
la gp, _gp
    la sp, _stack

/*
** Every good C program has a main.
*/
#if defined(__mips16)
jalx main
#else
jal main
#endif
nop
move a0, v0
.end start

.ent _exit
_exit:
    move s0, a0
    la t0, rlx_cov_exit
    jalr t0
    nop
    move a0, s0
    syscall 1
.end _exit

/*
** PutC
** Instruction used to perform character output
** from programs running during simulation.
*/
.globl __PutCharacter
.ent __PutCharacter
__PutCharacter:
    li a0, UART_DATA_ADDRESS
    sw a1, 0(a0)
j ra # Return
.end __PutCharacter

.comm __errno, 4

```

D.3 Example makefile

```
#
# Realtek Semiconductor Corp.
#
# RSDK RLXCOV Library
#
# Tony Wu (tonywu@realtek.com.tw)
# Jul. 07, 2006
#

#####
ARCH = 4180
PROGRAM = testcov
RSDKDIR = /rsdk/rsdk-1.2.7/linux/newlib

CC = $(RSDKDIR)/bin/rsdk-elf-gcc
LD = $(RSDKDIR)/bin/rsdk-elf-ld

CFLAGS = -march=$(ARCH) -c -G0 -fno-pic -frlxcov
IFLAGS = -I$(RSDKDIR)/include
LFLAGS = -Ttext 80000000 -e start -N -n
LIBS = -L$(RSDKDIR)/lib/$(ARCH) -lrlx -lm -lc -lgcc -L. -lrlxsim_gdb

#####
all: $(PROGRAM)

.S.o:
$(CC) -D__ASM__ -x assembler-with-cpp $(CFLAGS) $(IFLAGS) $<

.c.o:
$(CC) $(CFLAGS) $(IFLAGS) $<

testcov: c.o cov.o
$(LD) -T target-rlxsim.ld -o testcov $(LFLAGS) c.o cov.o $(LIBS)

clean:
rm -f *.o $(PROGRAM)
```

D.4 Example linker script

```
/* Script for -z combreloc: combine and sort reloc sections */
OUTPUT_FORMAT("elf32-bigmips", "elf32-bigmips", "elf32-littlemips")
OUTPUT_ARCH(mips)
ENTRY(start)

SECTIONS
{
    /* RLX code coverage section */
    __rlxcov_ent = ABSOLUTE(.);
    .rlxcov BLOCK(0x8) : { *(.rlxcov) rlx_*(.text) }
    __rlxcov_end = ABSOLUTE(.);

    /* Read-only sections, merged into text segment: */
    PROVIDE (__executable_start = 0x0400000); . = 0x0400000;
    .interp : { *(.interp) }
```

```

.reginfo      : { *(.reginfo) }
.dynamic      : { *(.dynamic) }
.hash         : { *(.hash) }
.dynsym       : { *(.dynsym) }
.dynstr       : { *(.dynstr) }
.gnu.version  : { *(.gnu.version) }
.gnu.version_d : { *(.gnu.version_d) }
.gnu.version_r : { *(.gnu.version_r) }
.rel.dyn      :
{
    *(.rel.init)
    *(.rel.text .rel.text.* .rel.gnu.linkonce.t.*)
    *(.rel.fini)
    *(.rel.rodata .rel.rodata.* .rel.gnu.linkonce.r.*)
    *(.rel.data.rel.ro*)
    *(.rel.data .rel.data.* .rel.gnu.linkonce.d.*)
    *(.rel.tdata .rel.tdata.* .rel.gnu.linkonce.td.*)
    *(.rel.tbss .rel.tbss.* .rel.gnu.linkonce.tb.*)
    *(.rel.ctors)
    *(.rel.dtors)
    *(.rel.got)
    *(.rel.sdata .rel.sdata.* .rel.gnu.linkonce.s.*)
    *(.rel.sbss .rel.sbss.* .rel.gnu.linkonce.sb.*)
    *(.rel.sdata2 .rel.sdata2.* .rel.gnu.linkonce.s2.*)
    *(.rel.sbss2 .rel.sbss2.* .rel.gnu.linkonce.sb2.*)
    *(.rel.bss .rel.bss.* .rel.gnu.linkonce.b.*)
}
.rela.dyn     :
{
    *(.rela.init)
    *(.rela.text .rela.text.* .rela.gnu.linkonce.t.*)
    *(.rela.fini)
    *(.rela.rodata .rela.rodata.* .rela.gnu.linkonce.r.*)
    *(.rela.data .rela.data.* .rela.gnu.linkonce.d.*)
    *(.rela.tdata .rela.tdata.* .rela.gnu.linkonce.td.*)
    *(.rela.tbss .rela.tbss.* .rela.gnu.linkonce.tb.*)
    *(.rela.ctors)
    *(.rela.dtors)
    *(.rela.got)
    *(.rela.sdata .rela.sdata.* .rela.gnu.linkonce.s.*)
    *(.rela.sbss .rela.sbss.* .rela.gnu.linkonce.sb.*)
    *(.rela.sdata2 .rela.sdata2.* .rela.gnu.linkonce.s2.*)
    *(.rela.sbss2 .rela.sbss2.* .rela.gnu.linkonce.sb2.*)
    *(.rela.bss .rela.bss.* .rela.gnu.linkonce.b.*)
}
.rel.plt      : { *(.rel.plt) }
.rela.plt     : { *(.rela.plt) }
.init         :
{
    KEEP (*(init))
} =0
.plt          : { *(.plt) }
.text 0x80000000 :
{
    _ftext = . ;
    *(.text .stub .text.* .gnu.linkonce.t.*)
    KEEP (*(text.*personality*))
    /* .gnu.warning sections are handled specially by elf32.em. */
    *(.gnu.warning)
    *(.mips16.fn.*) *(.mips16.call.*)

```

```
} =0
.fini      :
{
    KEEP (*( .fini ))
} =0
PROVIDE ( __etext = . );
PROVIDE ( _etext = . );
PROVIDE ( etext = . );
.rodata    : { *(.rodata .rodata.* .gnu.linkonce.r.*) }
.rodata1   : { *(.rodata1) }
.sdata2    : { *(.sdata2 .sdata2.* .gnu.linkonce.s2.*) }
.sbss2     : { *(.sbss2 .sbss2.* .gnu.linkonce.sb2.*) }
.eh_frame_hdr : { *(.eh_frame_hdr) }
.eh_frame  : ONLY_IF_RO { KEEP (*( .eh_frame )) }
.gcc_except_table : ONLY_IF_RO { KEEP (*( .gcc_except_table )) *(.gcc_except_table.*) }
/* Adjust the address for the data segment.  We want to adjust up to
   the same address within the page on the next page up.  */
. = ALIGN (0x40000) - ((0x40000 - .) & (0x40000 - 1)); . =
    DATA_SEGMENT_ALIGN (0x40000, 0x1000);
/* Exception handling */
.eh_frame  : ONLY_IF_RW { KEEP (*( .eh_frame )) }
.gcc_except_table : ONLY_IF_RW { KEEP (*( .gcc_except_table )) *(.gcc_except_table.*) }
/* Thread Local Storage sections */
.tdata     : { *(.tdata .tdata.* .gnu.linkonce.td.*) }
.tbss      : { *(.tbss .tbss.* .gnu.linkonce.tb.*) *(.tcommon) }
/* Ensure the __preinit_array_start label is properly aligned.  We
   could instead move the label definition inside the section, but
   the linker would then create the section even if it turns out to
   be empty, which isn't pretty.  */
. = ALIGN(32 / 8);
PROVIDE ( __preinit_array_start = . );
.preinit_array : { KEEP (*( .preinit_array )) }
PROVIDE ( __preinit_array_end = . );
PROVIDE ( __init_array_start = . );
.init_array   : { KEEP (*( .init_array )) }
PROVIDE ( __init_array_end = . );
PROVIDE ( __fini_array_start = . );
.fini_array   : { KEEP (*( .fini_array )) }
PROVIDE ( __fini_array_end = . );
.ctors        :
{
    /* gcc uses crtbegin.o to find the start of
       the constructors, so we make sure it is
       first.  Because this is a wildcard, it
       doesn't matter if the user does not
       actually link against crtbegin.o; the
       linker won't look for a file to match a
       wildcard.  The wildcard also means that it
       doesn't matter which directory crtbegin.o
       is in.  */
    KEEP (*crtbegin*.o(.ctors))
    /* We don't want to include the .ctor section from
       from the crtend.o file until after the sorted ctors.
       The .ctor section from the crtend file contains the
       end of ctors marker and it must be last */
    KEEP (*(EXCLUDE_FILE (*crtend*.o) .ctors))
    KEEP (*(SORT(.ctors.*)))
    KEEP (*(.ctors))
}
.dtors      :
```

```

{
    KEEP (*crtbegin*.o(.dtors))
    KEEP (*(EXCLUDE_FILE (*crtend*.o ) .dtors))
    KEEP (*(SORT(.dtors.*)))
    KEEP (*(.dtors))
}
.jcr          : { KEEP (*(.jcr)) }
.data.rel.ro : { *(.data.rel.ro.local) *(.data.rel.ro*) }
. = DATA_SEGMENT_RELRO_END (0, .);
.data         :
{
    _fdata = . ;
    *(.data .data.* .gnu.linkonce.d.*)
    KEEP (*(.gnu.linkonce.d.*personality*))
    SORT(CONSTRUCTORS)
}
.data1        : { *(.data1) }
. = .;
_gp = ALIGN(16) + 0x7ff0;
.got          : { *(.got.plt) *(.got) }
/* 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 .sdata.* .gnu.linkonce.s.*)
}
.lit8         : { *(.lit8) }
.lit4         : { *(.lit4) }
_edata = .;
PROVIDE (edata = .);
__bss_start = .;
_fbss = .;
.sbss         :
{
    PROVIDE (__sbss_start = .);
    PROVIDE (___sbss_start = .);
    *(.dynsbss)
    *(.sbss .sbss.* .gnu.linkonce.sb.*)
    *(.scommon)
    PROVIDE (__sbss_end = .);
    PROVIDE (___sbss_end = .);
}
.bss          :
{
    *(.dynbss)
    *(.bss .bss.* .gnu.linkonce.b.*)
    *(COMMON)
    /* Align here to ensure that the .bss section occupies space up to
       _end.  Align after .bss to ensure correct alignment even if the
       .bss section disappears because there are no input sections. */
    . = ALIGN(32 / 8);
}
. = ALIGN(32 / 8);
_end = .;
PROVIDE (end = .);
. = DATA_SEGMENT_END (.);
/* Stabs debugging sections. */
.stab         0 : { *(.stab) }
.stabstr      0 : { *(.stabstr) }

```

```
.stab.excl      0 : { *(.stab.excl) }
.stab.exclstr   0 : { *(.stab.exclstr) }
.stab.index     0 : { *(.stab.index) }
.stab.indexstr  0 : { *(.stab.indexstr) }
.comment        0 : { *(.comment) }
/* DWARF debug sections.
   Symbols in the DWARF debugging sections are relative to the beginning
   of the section so we begin them at 0. */
/* DWARF 1 */
.debug          0 : { *(.debug) }
.line           0 : { *(.line) }
/* GNU DWARF 1 extensions */
.debug_srcinfo  0 : { *(.debug_srcinfo) }
.debug_sfnames  0 : { *(.debug_sfnames) }
/* DWARF 1.1 and DWARF 2 */
.debug_aranges  0 : { *(.debug_aranges) }
.debug_pubnames 0 : { *(.debug_pubnames) }
/* DWARF 2 */
.debug_info     0 : { *(.debug_info .gnu.linkonce.wi.*) }
.debug_abbrev   0 : { *(.debug_abbrev) }
.debug_line     0 : { *(.debug_line) }
.debug_frame    0 : { *(.debug_frame) }
.debug_str      0 : { *(.debug_str) }
.debug_loc      0 : { *(.debug_loc) }
.debug_macinfo  0 : { *(.debug_macinfo) }
/* SGI/MIPS DWARF 2 extensions */
.debug_weaknames 0 : { *(.debug_weaknames) }
.debug_funcnames 0 : { *(.debug_funcnames) }
.debug_typenames 0 : { *(.debug_typenames) }
.debug_varnames  0 : { *(.debug_varnames) }
.gptab.sdata : { *(.gptab.data) *(.gptab.sdata) }
.gptab.sbss : { *(.gptab.bss) *(.gptab.sbss) }
/DISCARD/ : { *(.note.GNU-stack) }
}
```