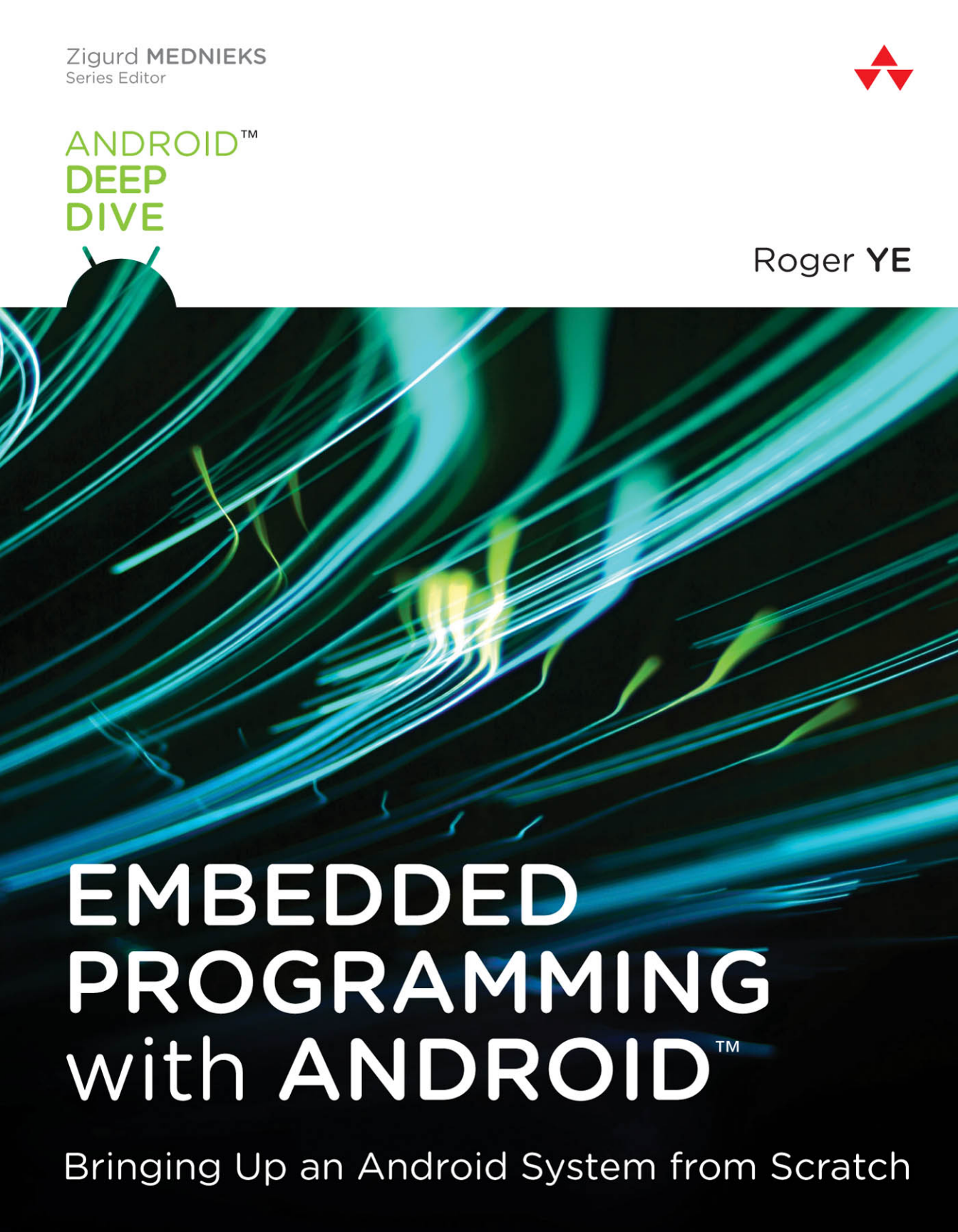


Zigurd MEDNIEKS  
Series Editor



ANDROID™  
DEEP  
DIVE

Roger YE

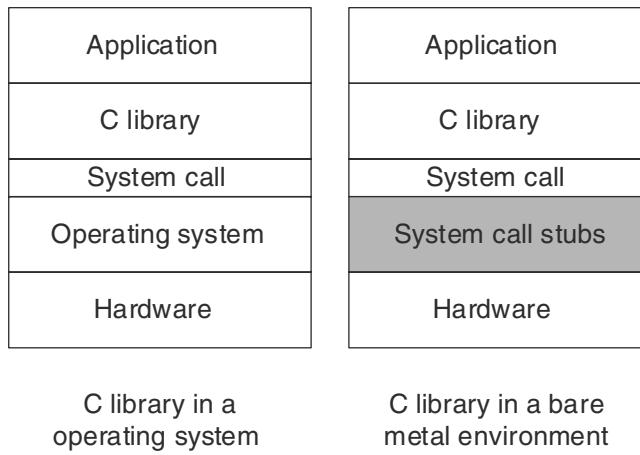


# EMBEDDED PROGRAMMING with ANDROID™

Bringing Up an Android System from Scratch

# Embedded Programming with Android™

---



**Figure 6.1** C library and system call

file-based I/O handling. The developer for an embedded system must be fully aware of the capability of the system and the C library that is built on the top of the system. The most popular C library variants in embedded system are Newlib, uclibc, and Bionic. They implement the standard C library based on their own systems with minor differences.

To provide better debugging capabilities, a special implementation of the system call stubs is provided in an embedded system called the semihosting or semihost C library. Semihosting or semihost is a mechanism that enables code running on an embedded system to communicate and use the input/output facilities on a host computer that is running a debugger. The host environment being discussed here is the environment introduced in Chapter 3—that is, the environment that we use to do development work, such as coding, compilation, and debugging. The semihost/semihosting library variant provides implementations of most of the standard C library functions, including file I/O. The file I/O will be directed through the debugger and will be performed on the host system. For example, `printf/scanf` will use the debugger console window and `fread/fwrite` will operate on files on the host system. This emulated I/O can be used only in a debugging environment. The semihosting variant is typically used together with a kind of debugging environment during development. For product release, however, the semihosting variant should not be used. We will explore this process in the second example (c06e2) in this chapter.

Many C library implementations are available for embedded programming. The choice of which to use depends heavily on the selection of the toolchain. The commercial toolchain known as RealView Development Suite (from ARM) has its own C library implementation. This variant is similar to those offered by other silicon vendors such as TI, NXP, and Freescale. For open source toolchains, Newlib is the most popular choice for the C library. As we are using Sourcery CodeBench Lite in this book, we will use Newlib's offering as an example when we discuss C library support in a bare metal programming environment.

**Note**

For the same source code of a C library, variants can be built for different target platforms. For example, Newlib C in Sourcery CodeBench Lite can be built for the ARM platform in the toolchain `arm-none-eabi` or for the MIPS platform in the toolchain `mips-sde-elf`. Similarly, the GNU C library in Sourcery CodeBench Lite can be built for the ARM platform in the toolchain `arm-none-linux-gnueabi` or for the MIPS platform in the toolchain `mips-linux-gnu`. Please refer to the Sourcery CodeBench Lite website for more details about various builds of Newlib C (<http://www.mentor.com/embedded-software/sourcery-tools/sourcery-codebench/editions/lite-edition/>).

## Newlib C Library

In Chapter 5, we learned how to build a program starting from the hardware reset in an assembly language environment. We also created simple startup code that initializes the programming environment for the C language and transfers control from the assembly language to the C language. In a real product, the startup code is somewhat more complicated than what we have seen in our examples. However, the basic concept underlying the code is the same. In this chapter, we will use the commercial implementation of startup code (CS3) from Mentor Graphics' Sourcery CodeBench Lite rather than trying to build complicated startup code on our own. In this way, we can focus on the essentials of building an embedded system instead of implementing everything by ourselves.

In this chapter, we will create two code examples. We will use them to illustrate how we can use C libraries in an embedded system development environment. In the first code example (c06e1), we will implement the necessary stubs in Newlib so that we can run the example code using Newlib on the goldfish platform. In the second code example (c06e2), we will test the semihosting environment supported by QEMU so that we can see how semihosting can help in the embedded system development environment.

In the first code example, the following files are created based on the example code in Chapter 5. These files are divided into two groups: common files and project-specific files. Common files are generic files that will be reused throughout the book; project-specific files are used for only this example.

Common files:

- `startup_cs3.S`: The startup code that performs the necessary setup before the control is transferred to the C code. It is based on the Sourcery CodeBench CS3 startup code.
- `syscalls_cs3.c`: The implementation of the system service stubs.
- `serial_goldfish.c`: The C code to implement serial functions. It is the same as in Chapter 5.
- `goldfish_uart.S`: The assembly code for implementing serial port-specific functions. It is the same as in Chapter 5.

Project-specific files:

- `c06e1.c`: The code for the test driver implementing `main()` function.
- `Makefile`: The makefile for building this project.
- `c06e1.ld`: The linker script for the project.

## Common Startup Code Sequence

In Chapter 3, we introduced the ARM Toolchain Sourcery CodeBench. We use the free edition Sourcery CodeBench Lite in this book. CS3 is the low-level board support library provided as part of Sourcery CodeBench. It provides a consistent set of conventions for processor and board-level initialization, language runtime setup, and interrupt and trap handler definition.

Sourcery CodeBench supports a number of built-in reference platforms. For each supported system, CS3 provides a set of linker scripts describing the system's memory map. A board support library provides generic reset, startup, and interrupt handlers. All of these scripts and libraries follow a standard set of conventions across a range of processors and boards.

## CS3 Linker Scripts

CS3 may provide multiple linker scripts for different configurations using the same board. For example, on some boards, CS3 may support running the program from either RAM or ROM (flash memory). In CS3 terminology, each of these different configurations is referred to as a profile.

Because goldfish is not supported by Sourcery CodeBench directly, we have to create a new link script file for goldfish to work with the CS3 startup sequence, as shown in Example 6.1. The link script for the general profile (`arm-none-eabi/lib/generic.ld`) is used as the basis to be modified for the goldfish platform.

### Example 6.1 Linker Script (`code/c06/c06e1/c06e1.ld`)

---

```
OUTPUT_FORMAT ("elf32-littlearm", "elf32-bigarm", "elf32-littlearm")

ENTRY(__cs3_reset)

SEARCH_DIR(.)

GROUP(-lgcc -lc -lcs3 -lcs3hosted -lcs3arm)

MEMORY

{
    flash (rx) : ORIGIN = 0x00010000, LENGTH = 128K /* Defined ROM size and
the start address */
    ram (rwx) : ORIGIN = 0x00030000, LENGTH = 512M /* Defined RAM size and the
start address */
}
```

```

/* These force the linker to search for particular symbols from
 * the start of the link process and thus ensure the user's
 * overrides are picked up
 */
EXTERN(__cs3_reset __cs3_reset_generic)
EXTERN(__cs3_start_asm __cs3_start_asm_sim)
/* Bring in the interrupt routines and vector */
INCLUDE arm-names.inc
EXTERN(__cs3_interrupt_vector_arm)
EXTERN(__cs3_start_c main __cs3_stack __cs3_heap_end)
/* Force exit to be picked up in a hosted or OS environment
EXTERN(exit atexit) */
/* Provide fall-back values */
PROVIDE(__cs3_heap_start = _end);
PROVIDE(__cs3_heap_end = __cs3_region_start_ram + __cs3_region_size_ram);
PROVIDE(__cs3_region_num = (__cs3_regions_end - __cs3_regions) / 20);
/* Ensure that Newlib runs the finalizers
__libc_fini = _fini; */
PROVIDE(__cs3_stack = __cs3_region_start_ram + __cs3_region_size_ram);
SECTIONS
{
    .text :
    {
        CREATE_OBJECT_SYMBOLS
        __cs3_region_start_flash = .;          /* We put .text section in flash */
        _ftext = .;
        *(.cs3.region-head.flash)
        ASSERT (. == __cs3_region_start_flash, ".cs3.region-head.flash not permitted");
        __cs3_interrupt_vector = __cs3_interrupt_vector_arm;
        *(.cs3.interrupt_vector)
        /* Make sure we pulled in an interrupt vector */
        ASSERT (. != __cs3_interrupt_vector_arm, "No interrupt vector");
        PROVIDE(__cs3_reset = __cs3_reset_generic);
        *(.cs3.reset)

        __cs3_start_asm_sim = DEFINED(__cs3_start_asm) ? __cs3_start_asm : __cs3_
start_asm_sim;

```

```

*(.text.cs3.init)
*(.text .text.* .gnu.linkonce.t.*)
*(.plt)
*(.gnu.warning)
*(.glue_7t) *(.glue_7) *(.vfp11_veneer)
*(.ARM.extab* .gnu.linkonce.armextab.*)
*(.gcc_except_table)
} >flash
.eh_frame_hdr : ALIGN (4)
{
    KEEP (*(eh_frame_hdr))
    *(eh_frame_entry eh_frame_entry.*)
} >flash
.eh_frame : ALIGN (4)
{
    KEEP (*(eh_frame)) *(eh_frame.*)
} >flash
/* .ARM.exidx is sorted, so it has to go in its own output section */
PROVIDE_HIDDEN (__exidx_start = .);
.ARM.exidx :
{
    *(.ARM.exidx* .gnu.linkonce.armexidx.*)
} >flash
PROVIDE_HIDDEN (__exidx_end = .);
.rodata : ALIGN (4)
{
    *(.rodata .rodata.* .gnu.linkonce.r.*)

    . = ALIGN(4);
    KEEP(*(.init))

    . = ALIGN(4);
    __preinit_array_start = .;
    KEEP (*(.preinit_array))
    __preinit_array_end = .;

    . = ALIGN(4);

```

```

__init_array_start = .;
KEEP (*(SORT(.init_array.*)))
KEEP (*(init_array))
__init_array_end = .;

. = ALIGN(4);
KEEP(*(.fini))

. = ALIGN(4);
__fini_array_start = .;
KEEP (*(.fini_array))
KEEP (*(SORT(.fini_array.*)))
__fini_array_end = .;

. = ALIGN(0x4);
KEEP (*crtbegin.o(.ctors))
KEEP (*(EXCLUDE_FILE (*crtend.o) .ctors))
KEEP (*(SORT(.ctors.*)))
KEEP (*crtend.o(.ctors))

. = ALIGN(0x4);
KEEP (*crtbegin.o(.dtors))
KEEP (*(EXCLUDE_FILE (*crtend.o) .dtors))
KEEP (*(SORT(.dtors.*)))
KEEP (*crtend.o(.dtors))

. = ALIGN(4);
__cs3_regions = .;
LONG (0)
LONG (__cs3_region_init_ram)
LONG (__cs3_region_start_ram)
LONG (__cs3_region_init_size_ram)
LONG (__cs3_region_zero_size_ram)
__cs3_regions_end = .;

. = ALIGN (8);
_etext = .;
} >flash

```