Application Note

DA14580 Porting a Keil uVision Project to the GNU Tool Chain

AN-B-024

Abstract

Software development for DA14580 is currently done using Keil uVision, which provides an optimizing ARM compiler and an IDE for the MS Windows platform. The Keil compiler may be used free of license for code sizes up to 32 KiB, but a license is required for larger images. This limitation, along with a trend among software developers towards Linux and free and open-source software, has created the need for a GNU tool chain as an alternative to the Keil compiler. This document briefly describes the information included in a Keil uVision project. It analyses the equivalent files that are used with a GNU tool chain. Next, a script is introduced that automates the conversion of a Keil uVision project into a Makefile. Finally, results are presented on speed and size of two benchmarks built with different compiler options

Contents

Tables

1 Terms and definitions

BLE	Bluetooth Low Energy
FOSS	Free and Open-Source Software
GCC	GNU Compiler
GDB	GNU Debugger
IDE	Integrated Development Environment
KiB	Kbyte (1024 bytes)
LTO	Link-Time Optimisation

2 References

1. ARM GCC tool chain, release `gcc-arm-none-eabi-4_8-2014q1`
2. Keil uVision `V5.1.0.0`

3 Introduction

Software development for DA14580 is currently done using Keil uVision, which provides an optimizing ARM compiler and an IDE that runs on an MS Windows platform. The Keil tool chain is free of license for code sizes up to 32 KiB, but a license is required for larger images. This limitation, along with the need to have a development suit available for Linux and FOSS, has aroused interest in using a GNU tool chain as an alternative to the Keil compiler. This document describes how a Keil uVision project can be ported so that it can be built using a GNU tool chain.

The following tool chains where used to produce the results presented in this document:

- ARM GCC tool chain, release `gcc-arm-none-eabi-4_8-2014q1`
- Keil uVision `V5.1.0.0`

4 Keil uVision projects

4.1 Keil uVision project files

Keil uVision uses files with the extension `.uvproj` to store all the information of a project. This includes:

- Target configuration
- Compilation and linking flags
- Debugger options
- List of files that comprise the project

The `.uvproj` files are actually XML files, so their parsing is fairly easy. Within the scope of this document, only the list of source files are of interest, because the target is fixed (i.e. ARM Cortex M0) and the compilation/linking flags are tool chain dependent and essentially fixed among all the BLE applications. The debugger options are irrelevant for a GNU tool chain, as GDB is the norm in this case.

4.2 Keil scatter files

Keil uses the notion of a scatter file, which describes the memory layout of an executable (i.e. how its code and data are loaded or allocated in the memory). Scatter files are written in plain text using a specific syntax. Unfortunately, the GNU linker does not support scatter files; instead, custom linker scripts can be written to produce the same effect. Currently, only one (common) scatter file is used by all BLE applications (`dk_apps/scatterfiles/scatterfile_common.sct`). Therefore, this common scatter file was translated into a linker script that can be used by all BLE applications, at least as an initial version.

4.3 Keil start-up code

After a CPU reset, execution eventually reaches the *reset handler*, which performs any necessary initialisations and then jumps to the entry point of the application. The code of the reset handler, along with the definition of all interrupt handlers, is included in an assembly file, typically named `boot_vectors.s`. Some of the initialisations carried out by the reset handler are tool chain dependent, because they involve actions on the binary representation of the scatter file (i.e. copying of code or data from storage to their actual run-time locations and filling of certain memory areas with zeroes). Furthermore, the syntax of the ARM assembly varies between assemblers, so the start-up code needs to be rewritten for the GNU tool chain. Once again, this code is usually invariable among different projects, so porting needs to be done only once.

4.4 Keil compiler extensions

The Keil compiler is generally compatible with GCC, in terms of the extensions to standard C that it provides; however, it also supports a few extensions not supported by GCC. There are parts of the BLE code base that use such extensions. The two most notable cases are:

- The "at" __attribute__, e.g.
  `volatile uint8 descript[EM_SYSMEM_SIZE] __attribute__((at(EM_SYSMEM_START)));`
  A workaround would be to use the `section __attribute__` and set a unique section; that section can be then placed at the desired address (e.g. `EM_SYSTEM_START`) with the proper commands in the scatter file (for Keil) or linker script (for GCC). Example:
  ```
  volatile uint8 descript[EM_SYSMEM_SIZE]
  __attribute__((section("atEM_SYSMEM_START")));
  ```
- The "zero_init" __attribute__, e.g.
  `bool sys_startup_flag __attribute__((section("retention_mem_area0"), zero_init));`
  A workaround for GCC would be to use a compilation flag that mutes warnings about unknown attributes (i.e. `CFLAGS+=-Wno-attributes`) and include the pertinent section (`"retention_mem_area0"`) in a zero-initialised segment (i.e. `.bss`) in the linker script.

In general, the code can be written such that it is compatible with both Keil and GCC (at the expense of more involved or cumbersome syntax or configuration). In the extreme cases where this would not be possible, redesigning of the code might be necessary.

5 Building code with a GNU tool chain

5.1 Makefile

Building with a GNU tool chain is typically carried out using `make` and a `Makefile`. The `Makefile` is a text file, which comprises *rules* for creating *targets* (one of which is the final executable image). In essence, the Makefile is the equivalent of the Keil project file, as far as building is concerned.

In the context of section 4, the `Makefile` should contain the following:

- The list of source files that constitute the project, as found in the `.uvproj` file.
- The compilation and linking flags. Most compilation flags can be transferred verbatim from Keil to GCC. There are more discrepancies in the linking flags. In either case, equivalent flags *do* exist. As mentioned before, these flags usually stay constant from one BLE project to the next.
- A reference to the linker script, which replaces Keil's scatter file. Again, the linker script is usually the same of all BLE projects.
- The target and debugger configuration: with the GNU tool chain this is done in the compilation flags and is constant among all projects.

It should be evident that the Makefile consists of a *common* part, which is expected to be the same for all BLE projects and a *project-dependent* part. For clarity, all the common definitions, rules and recipes are included in the file `common.mk` and the project-dependent definitions are included in the `Makefile` (which includes `common.mk`). For brevity and in the context of building a project with a GNU tool chain, the files `common.mk` and `Makefile` will hereafter be collectively referred to as *project build files*. These files can be found in the directory `tools/uvproj2Makefile/` of a BLE code release.

5.1.1 The invariable part of the project build files (common.mk)

The current version of `tools/uvproj2Makefile/common.mk` is listed below:

```
<  1> #
<  2> # Common variables and recipies used by Makefiles.
<  3> # The following variables are defined:
<  4> #  CC: C cross-compiler
```

```
<  5> #  CPP: cross-preprocessor
<  6> #  OBJCOPY: cross-objcopy
<  7> #  USE_NANO: link flags to select newlib-nano
<  8> #  USE_SEMIHOST: link flags to enable semihosting
<  9> #  USE_NOHOST: link flags to disable semihosting
< 10> #  MAP: link flags to create a map file
< 11> #  LDSCRIPTS: link flags to use a custom link script, generated from
$(LINK_SCRIPT).S
< 12> #
< 13>
< 14> CROSS_COMPILE = arm-none-eabi-
< 15> CC = $(CROSS_COMPILE)GCC
< 16> CPP = $(CROSS_COMPILE)cpp
< 17> OBJCOPY = $(CROSS_COMPILE)objcopy
< 18>
< 19> # verbosity switch
< 20> V ?= 0
< 21> ifeq ($(V),0)
< 22>  V_CC = @echo "  CC    " $@;
< 23>  V_CPP = @echo "  CPP   " $@;
< 24>  V_LINK = @echo "  LINK  " $@;
< 25>  V_OBJCOPY = @echo "  OBJCPY" $@;
< 26>  V_CLEAN = @echo "  CLEAN ";
< 27>  V_SED = @echo "  SED   " $@;
< 28>  V_GAWK = @echo "  GAWK  " $@;
< 29> else
< 30>  V_OPT = '-v'
< 31> endif
< 32>
< 33> # Use newlib-nano. To disable it, specify USE_NANO=
< 34> USE_NANO := --specs=nano.specs
< 35>
< 36> # Use semihosting or not
< 37> USE_SEMIHOST := --specs=rdimon.specs
< 38> USE_NOHOST := --specs=nosys.specs
< 39>
< 40> # Create map file
< 41> MAP = -Wl,-Map=$(O)/lst/$(TARGET_ELF).map
< 42>
< 43> ARCH_FLAGS = -mthumb -mcpu=cortex-m$(CORTEX_M)
< 44>
< 45> # general compilation flags
< 46> CFLAGS += $(ARCH_FLAGS) $(STARTUP_DEFS) -std=gnu99
< 47>
< 48> ifeq ($(V),2)
< 49>  CFLAGS += --verbose
< 50>  LDFLAGS += -Wl,--verbose
< 51> endif
< 52>
< 53> ROM_SYMDEF := rom.symdef
< 54> ROM_SYMBOLS := rom.symbols
< 55> LINK_SCRIPT := 580.lds
< 56>
< 57> LDSCRIPTS := -L. -T $(LINK_SCRIPT)
< 58>
< 59>
< 60> # how to compile C files
< 61> %.o : %.c
< 62>  $(V_CC)$(CC) $(CFLAGS) -c $< -o $@
< 63>
```

```
< 64> # how to compile assembly files
< 65> %.o : %.S
< 66>  $(V_CC)$(CC) $(CFLAGS) -c $< -o $@
< 67>
< 68>
< 69> all: $(O)/$(TARGET_ELF).hex $(O)/$(TARGET_ELF).bin
< 70>
< 71> clean:
< 72>  $(V_CLEAN)for file in $(OBJ); do rm -f $(V_OPT) "$$file"; done
< 73>  @rm -rf $(V_OPT) $(O)
< 74>  @rm -f $(V_OPT) $(ROM_SYMDEF) $(ROM_SYMBOLS) $(LINK_SCRIPT)
< 75>
< 76> $(O)/ :
< 77>  @mkdir -p $(V_OPT) $(O)/lst
< 78>
< 79> # how to create the main ELF target
< 80> $(O)/$(TARGET_ELF).axf: $(O)/ $(ROM_SYMBOLS) $(LINK_SCRIPT) $(OBJ)
< 81>  $(V_LINK)$(CC) $(CFLAGS) $(LDFLAGS) $(OBJ) $(patch_objs) -o $@
< 82>
< 83> # how to create the final hex file
< 84> $(O)/$(TARGET_ELF).hex: $(O)/$(TARGET_ELF).axf
< 85>  $(V_OBJCOPY)$(OBJCOPY) -O ihex $< $@
< 86>
< 87> # how to create the final binary file
< 88> $(O)/$(TARGET_ELF).bin: $(O)/$(TARGET_ELF).axf
< 89>  $(V_OBJCOPY)$(OBJCOPY) -O binary $< $@
< 90>
< 91> # how to create the linker script
< 92> $(LINK_SCRIPT): $(LINK_SCRIPT).S
< 93>  $(V_CPP)$(CPP) -P $< -o $@
< 94>
< 95> # how to create a clean, sorted list of known symbols, used by ROM code
< 96> $(ROM_SYMDEF): $(ROM_MAP_FILE)
< 97>  $(V_SED)sed -n -e 's/  */ /g p' $< | sed -e '/^[;#]/d' | \
< 98>         sort | dos2unix > $@
< 99>
<100> # how to create a file with the known symbols, to be used in the linker script
<101> $(ROM_SYMBOLS): $(ROM_SYMDEF)
<102>  $(V_GAWK)gawk '{printf "%s = %s ;\n", $$3, $$1}' $< > $@
```

Lines 4 to 11 list the variables defined in this file that can be used by a project `Makefile`.

Lines 69 to 102 contain the definitions of targets and receipes. The main targets, which normally should be used from the command line, are:

- `all`: it generates the final binary images:
  a. `$(O)/$(TARGET_ELF).axf`: the ELF executable.
  b. `$(O)/$(TARGET_ELF).hex`: the executable image in Intel hex format.
  c. `$(O)/$(TARGET_ELF).bin`: the executable in raw binary format.
- `clean`: it cleans all the output and intermediate files.

In addition, `common.mk` expects to find some project-dependent information in certain variables (e.g. filenames, paths etc). These variables are described in section 5.1.2.

Finally, the variable `$(V)` can be defined in the command line to modify the *screen* output of the project build files (the generated *binary* images are not affected, though):

- Brief output (lines 22 to 28); default behaviour or when invoked, for example, with "`make V=0`":

```
> make
  SED     rom.symdef
```

```
  GAWK   rom.symbols
  CPP    580.lds
  CC     ../../../src/plf/refip/src/arch/boot/rvds/system_ARMCM0.o
  CC     ../../../src/plf/refip/src/arch/main/ble/hardfault_handler.o
  CC     ../../../src/plf/refip/src/arch/main/ble/arch_main.o
  CC     ../../../src/plf/refip/src/arch/main/ble/arch_main.o
In file included from
./../../../src/modules/app/src/app_project/template_fh/system/app_sleep.h:10:0,
                 from ../../../src/plf/refip/src/arch/main/ble/arch_main.c:66:
./../../../src/plf/refip/src/driver/gpio/gpio.h:119:20: warning: inline function
'GPIO_GetPinStatus' declared but never defined [enabled by default]
 extern inline bool GPIO_GetPinStatus( GPIO_PORT port, GPIO_PIN pin );
                    ^
  CC     startup_ARMCM0.o
  LINK   out/full_emb_sysram.axf
  OBJCPY out/full_emb_sysram.hex
  OBJCPY out/full_emb_sysram.bin
```

In this case the Makefile prints only a brief description of the command/action being carried out. This makes any warnings or error messages stand out and not be missed in a lengthy, verbose output.

- Normal output (line 30); more verbose output, when invoked, for example, with "`make V=1`":

```
> make V=1
mkdir: created directory «out»
mkdir: created directory «out/lst»
sed -n -e 's/  */ /g p' ../../../misc/rom_symdef.txt | sed -e '/^[;#]/d' | \
    sort | dos2unix > rom.symdef
gawk '{printf "%s = %s ;\n", $3, $1}' rom.symdef > rom.symbols
arm-none-eabi-cpp -P 580.lds.S -o 580.lds
arm-none-eabi-GCC -flto -ffunction-sections -fdata-sections -Os -
fplugin=tree_switch_shortcut_elf -I ./../../../src/dialog/include -I
./../../../src/plf/refip/src/arch -I
./../../../src/plf/refip/src/arch/compiler/rvds -I
./../../../src/plf/refip/src/arch/boot/rvds -I
./../../../src/plf/refip/src/arch/ll/rvds -I
./../../../src/plf/refip/src/driver/reg -I ./../../../src/modules/common/api -I
./../../../src/modules/dbg/api -I ./../../../src/modules/display/api -I
./../../../src/modules/gtl/api -I ./../../../src/modules/ke/api -I
./../../../src/modules/ke/src -I ./../../../src/modules/nvds/api -I
./../../../src/modules/rf/api -I ./../../../src/modules/rwip/api -I
./../../../src/ip/ble/ll/src/rwble -I ./../../../src/ip/ble/ll/src/controller/em -
I ./../../../src/ip/ble/ll/src/controller/llc -I
./../../../src/ip/ble/ll/src/controller/lld -I
./../../../src/ip/ble/ll/src/controller/llm -I
./../../../src/plf/refip/src/driver/led -I
./../../../src/plf/refip/src/driver/timer -I
./../../../src/plf/refip/src/driver/syscntl -I
./../../../src/plf/refip/src/driver/emi -I
./../../../src/plf/refip/src/driver/uart -I
./../../../src/plf/refip/src/driver/flash -I
./../../../src/plf/refip/src/driver/gpio -I ./../../../src/ip/ble/hl/src/host/att
-I ./../../../src/ip/ble/hl/src/host/att/attc -I
./../../../src/ip/ble/hl/src/host/att/attm -I
./../../../src/ip/ble/hl/src/host/gap -I
./../../../src/ip/ble/hl/src/host/gap/gapc -I
./../../../src/ip/ble/hl/src/host/gap/gapm -I
./../../../src/ip/ble/hl/src/host/att/atts -I
./../../../src/ip/ble/hl/src/host/gatt -I
./../../../src/ip/ble/hl/src/host/gatt/gattc -I
./../../../src/ip/ble/hl/src/host/gatt/gattm -I
```

```
./../../../src/ip/ble/hl/src/host/l2c/l2cc -I
./../../../src/ip/ble/hl/src/host/l2c/l2cm -I
./../../../src/ip/ble/hl/src/host/smp/smpc -I
./../../../src/ip/ble/hl/src/host/smp/smpm -I
./../../../src/ip/ble/hl/src/profiles -I
./../../../src/ip/ble/hl/src/profiles/accel -I
./../../../src/ip/ble/hl/src/profiles/bas/basc -I
./../../../src/ip/ble/hl/src/profiles/bas/bass -I
./../../../src/ip/ble/hl/src/profiles/blp -I
./../../../src/ip/ble/hl/src/profiles/blp/blpc -I
./../../../src/ip/ble/hl/src/profiles/blp/blps -I
./../../../src/ip/ble/hl/src/profiles/dis/disc -I
./../../../src/ip/ble/hl/src/profiles/dis/diss -I
./../../../src/ip/ble/hl/src/profiles/find/findl -I
./../../../src/ip/ble/hl/src/profiles/find/findt -I
./../../../src/ip/ble/hl/src/profiles/hogp -I
./../../../src/ip/ble/hl/src/profiles/hogp/hogpbh -I
./../../../src/ip/ble/hl/src/profiles/hogp/hogpd -I
./../../../src/ip/ble/hl/src/profiles/hogp/hogprh -I
./../../../src/ip/ble/hl/src/profiles/hrp -I
./../../../src/ip/ble/hl/src/profiles/hrp/hrpc -I
./../../../src/ip/ble/hl/src/profiles/hrp/hrps -I
./../../../src/ip/ble/hl/src/profiles/htp -I
./../../../src/ip/ble/hl/src/profiles/htp/htpc -I
./../../../src/ip/ble/hl/src/profiles/htp/htpt -I
./../../../src/ip/ble/hl/src/profiles/prox/proxm -I
./../../../src/ip/ble/hl/src/profiles/prox/proxr -I
./../../../src/ip/ble/hl/src/profiles/scpp -I
./../../../src/ip/ble/hl/src/profiles/scpp/scppc -I
./../../../src/ip/ble/hl/src/profiles/scpp/scpps -I
./../../../src/plf/refip/src/driver/intc -I ./../../../src/ip/ble/hl/src/rwble_hl
-I ./../../../src/ip/ble/ll/src/hcic -I ./../../../src/ip/ble/hl/src/host/smp -I
./../../../src/modules/app/api -I ./../../../src/modules/gtl/src -I
./../../../src/ip/ble/hl/src/profiles/anp -I
./../../../src/ip/ble/hl/src/profiles/anp/anpc -I
./../../../src/ip/ble/hl/src/profiles/anp/anps -I
./../../../src/ip/ble/hl/src/profiles/cscp -I
./../../../src/ip/ble/hl/src/profiles/cscp/cscpc -I
./../../../src/ip/ble/hl/src/profiles/cscp/cscps -I
./../../../src/ip/ble/hl/src/profiles/glp -I
./../../../src/ip/ble/hl/src/profiles/glp/glpc -I
./../../../src/ip/ble/hl/src/profiles/glp/glps -I
./../../../src/ip/ble/hl/src/profiles/pasp -I
./../../../src/ip/ble/hl/src/profiles/pasp/paspc -I
./../../../src/ip/ble/hl/src/profiles/pasp/pasps -I
./../../../src/ip/ble/hl/src/profiles/rscp -I
./../../../src/ip/ble/hl/src/profiles/rscp/rscpc -I
./../../../src/ip/ble/hl/src/profiles/rscp/rscps -I
./../../../src/ip/ble/hl/src/profiles/tip -I
./../../../src/ip/ble/hl/src/profiles/tip/tipc -I
./../../../src/ip/ble/hl/src/profiles/tip/tips -I ./../../../src/modules/app/src -
I ./../../../src/plf/refip/src/driver/adc -I
./../../../src/plf/refip/src/driver/wkupct -I
./../../../src/plf/refip/src/driver/battery -I
./../../../src/modules/app/src/app_project/template_fh -I
./../../../src/modules/app/src/app_project/template_fh/system -include
da14580_config.h -Wno-attributes -mthumb -mcpu=cortex-m0 -D__STACK_SIZE=0x0600 -
D__HEAP_SIZE=0x0100 -D__STARTUP_CLEAR_BSS_MULTIPLE -std=gnu99 -c
../../../src/plf/refip/src/arch/boot/rvds/system_ARMCM0.c -o
../../../src/plf/refip/src/arch/boot/rvds/system_ARMCM0.o
```

```
arm-none-eabi-GCC -flto -ffunction-sections -fdata-sections -Os -
fplugin=tree_switch_shortcut_elf -I ./../../../src/dialog/include -I
./../../../src/plf/refip/src/arch -I
./../../../src/plf/refip/src/arch/compiler/rvds -I
./../../../src/plf/refip/src/arch/boot/rvds -I
./../../../src/plf/refip/src/arch/ll/rvds -I
./../../../src/plf/refip/src/driver/reg -I ./../../../src/modules/common/api -I
./../../../src/modules/dbg/api -I ./../../../src/modules/display/api -I
./../../../src/modules/gtl/api -I ./../../../src/modules/ke/api -I
./../../../src/modules/ke/src -I ./../../../src/modules/nvds/api -I
./../../../src/modules/rf/api -I ./../../../src/modules/rwip/api -I
./../../../src/ip/ble/ll/src/rwble -I ./../../../src/ip/ble/ll/src/controller/em -
I ./../../../src/ip/ble/ll/src/controller/llc -I
./../../../src/ip/ble/ll/src/controller/lld -I
./../../../src/ip/ble/ll/src/controller/llm -I
./../../../src/plf/refip/src/driver/led -I
./../../../src/plf/refip/src/driver/timer -I
./../../../src/plf/refip/src/driver/syscntl -I
./../../../src/plf/refip/src/driver/emi -I
./../../../src/plf/refip/src/driver/uart -I
./../../../src/plf/refip/src/driver/flash -I
./../../../src/plf/refip/src/driver/gpio -I ./../../../src/ip/ble/hl/src/host/att
-I ./../../../src/ip/ble/hl/src/host/att/attc -I
./../../../src/ip/ble/hl/src/host/att/attm -I
./../../../src/ip/ble/hl/src/host/gap -I
./../../../src/ip/ble/hl/src/host/gap/gapc -I
./../../../src/ip/ble/hl/src/host/gap/gapm -I
./../../../src/ip/ble/hl/src/host/att/atts -I
./../../../src/ip/ble/hl/src/host/gatt -I
./../../../src/ip/ble/hl/src/host/gatt/gattc -I
./../../../src/ip/ble/hl/src/host/gatt/gattm -I
./../../../src/ip/ble/hl/src/host/l2c/l2cc -I
./../../../src/ip/ble/hl/src/host/l2c/l2cm -I
./../../../src/ip/ble/hl/src/host/smp/smpc -I
./../../../src/ip/ble/hl/src/host/smp/smpm -I
./../../../src/ip/ble/hl/src/profiles -I
./../../../src/ip/ble/hl/src/profiles/accel -I
./../../../src/ip/ble/hl/src/profiles/bas/basc -I
./../../../src/ip/ble/hl/src/profiles/bas/bass -I
./../../../src/ip/ble/hl/src/profiles/blp -I
./../../../src/ip/ble/hl/src/profiles/blp/blpc -I
./../../../src/ip/ble/hl/src/profiles/blp/blps -I
./../../../src/ip/ble/hl/src/profiles/dis/disc -I
./../../../src/ip/ble/hl/src/profiles/dis/diss -I
./../../../src/ip/ble/hl/src/profiles/find/findl -I
./../../../src/ip/ble/hl/src/profiles/find/findt -I
./../../../src/ip/ble/hl/src/profiles/hogp -I
./../../../src/ip/ble/hl/src/profiles/hogp/hogpbh -I
./../../../src/ip/ble/hl/src/profiles/hogp/hogpd -I
./../../../src/ip/ble/hl/src/profiles/hogp/hogprh -I
./../../../src/ip/ble/hl/src/profiles/hrp -I
./../../../src/ip/ble/hl/src/profiles/hrp/hrpc -I
./../../../src/ip/ble/hl/src/profiles/hrp/hrps -I
./../../../src/ip/ble/hl/src/profiles/htp -I
./../../../src/ip/ble/hl/src/profiles/htp/htpc -I
./../../../src/ip/ble/hl/src/profiles/htp/htpt -I
./../../../src/ip/ble/hl/src/profiles/prox/proxm -I
./../../../src/ip/ble/hl/src/profiles/prox/proxr -I
./../../../src/ip/ble/hl/src/profiles/scpp -I
./../../../src/ip/ble/hl/src/profiles/scpp/scppc -I
```

```
./../../../src/ip/ble/hl/src/profiles/scpp/scpps -I
./../../../src/plf/refip/src/driver/intc -I ./../../../src/ip/ble/hl/src/rwble_hl
-I ./../../../src/ip/ble/ll/src/hcic -I ./../../../src/ip/ble/hl/src/host/smp -I
./../../../src/modules/app/api -I ./../../../src/modules/gtl/src -I
./../../../src/ip/ble/hl/src/profiles/anp -I
./../../../src/ip/ble/hl/src/profiles/anp/anpc -I
./../../../src/ip/ble/hl/src/profiles/anp/anps -I
./../../../src/ip/ble/hl/src/profiles/cscp -I
./../../../src/ip/ble/hl/src/profiles/cscp/cscpc -I
./../../../src/ip/ble/hl/src/profiles/cscp/cscps -I
./../../../src/ip/ble/hl/src/profiles/glp -I
./../../../src/ip/ble/hl/src/profiles/glp/glpc -I
./../../../src/ip/ble/hl/src/profiles/glp/glps -I
./../../../src/ip/ble/hl/src/profiles/pasp -I
./../../../src/ip/ble/hl/src/profiles/pasp/paspc -I
./../../../src/ip/ble/hl/src/profiles/pasp/pasps -I
./../../../src/ip/ble/hl/src/profiles/rscp -I
./../../../src/ip/ble/hl/src/profiles/rscp/rscpc -I
./../../../src/ip/ble/hl/src/profiles/rscp/rscps -I
./../../../src/ip/ble/hl/src/profiles/tip -I
./../../../src/ip/ble/hl/src/profiles/tip/tipc -I
./../../../src/ip/ble/hl/src/profiles/tip/tips -I ./../../../src/modules/app/src -
I ./../../../src/plf/refip/src/driver/adc -I
./../../../src/plf/refip/src/driver/wkupct -I
./../../../src/plf/refip/src/driver/battery -I
./../../../src/modules/app/src/app_project/template_fh -I
./../../../src/modules/app/src/app_project/template_fh/system -include
da14580_config.h -Wno-attributes -mthumb -mcpu=cortex-m0 -D__STACK_SIZE=0x0600 -
D__HEAP_SIZE=0x0100 -D__STARTUP_CLEAR_BSS_MULTIPLE -std=gnu99 -c
../../../src/plf/refip/src/arch/main/ble/hardfault_handler.c -o
../../../src/plf/refip/src/arch/main/ble/hardfault_handler.o
arm-none-eabi-GCC -flto -ffunction-sections -fdata-sections -Os -
fplugin=tree_switch_shortcut_elf -I ./../../../src/dialog/include -I
./../../../src/plf/refip/src/arch -I
./../../../src/plf/refip/src/arch/compiler/rvds -I
./../../../src/plf/refip/src/arch/boot/rvds -I
./../../../src/plf/refip/src/arch/ll/rvds -I
./../../../src/plf/refip/src/driver/reg -I ./../../../src/modules/common/api -I
./../../../src/modules/dbg/api -I ./../../../src/modules/display/api -I
./../../../src/modules/gtl/api -I ./../../../src/modules/ke/api -I
./../../../src/modules/ke/src -I ./../../../src/modules/nvds/api -I
./../../../src/modules/rf/api -I ./../../../src/modules/rwip/api -I
./../../../src/ip/ble/ll/src/rwble -I ./../../../src/ip/ble/ll/src/controller/em -
I ./../../../src/ip/ble/ll/src/controller/llc -I
./../../../src/ip/ble/ll/src/controller/lld -I
./../../../src/ip/ble/ll/src/controller/llm -I
./../../../src/plf/refip/src/driver/led -I
./../../../src/plf/refip/src/driver/timer -I
./../../../src/plf/refip/src/driver/syscntl -I
./../../../src/plf/refip/src/driver/emi -I
./../../../src/plf/refip/src/driver/uart -I
./../../../src/plf/refip/src/driver/flash -I
./../../../src/plf/refip/src/driver/gpio -I ./../../../src/ip/ble/hl/src/host/att
-I ./../../../src/ip/ble/hl/src/host/att/attc -I
./../../../src/ip/ble/hl/src/host/att/attm -I
./../../../src/ip/ble/hl/src/host/gap -I
./../../../src/ip/ble/hl/src/host/gap/gapc -I
./../../../src/ip/ble/hl/src/host/gap/gapm -I
./../../../src/ip/ble/hl/src/host/att/atts -I
./../../../src/ip/ble/hl/src/host/gatt -I
```

```
./../../../src/ip/ble/hl/src/host/gatt/gattc -I
./../../../src/ip/ble/hl/src/host/gatt/gattm -I
./../../../src/ip/ble/hl/src/host/l2c/l2cc -I
./../../../src/ip/ble/hl/src/host/l2c/l2cm -I
./../../../src/ip/ble/hl/src/host/smp/smpc -I
./../../../src/ip/ble/hl/src/host/smp/smpm -I
./../../../src/ip/ble/hl/src/profiles -I
./../../../src/ip/ble/hl/src/profiles/accel -I
./../../../src/ip/ble/hl/src/profiles/bas/basc -I
./../../../src/ip/ble/hl/src/profiles/bas/bass -I
./../../../src/ip/ble/hl/src/profiles/blp -I
./../../../src/ip/ble/hl/src/profiles/blp/blpc -I
./../../../src/ip/ble/hl/src/profiles/blp/blps -I
./../../../src/ip/ble/hl/src/profiles/dis/disc -I
./../../../src/ip/ble/hl/src/profiles/dis/diss -I
./../../../src/ip/ble/hl/src/profiles/find/findl -I
./../../../src/ip/ble/hl/src/profiles/find/findt -I
./../../../src/ip/ble/hl/src/profiles/hogp -I
./../../../src/ip/ble/hl/src/profiles/hogp/hogpbh -I
./../../../src/ip/ble/hl/src/profiles/hogp/hogpd -I
./../../../src/ip/ble/hl/src/profiles/hogp/hogprh -I
./../../../src/ip/ble/hl/src/profiles/hrp -I
./../../../src/ip/ble/hl/src/profiles/hrp/hrpc -I
./../../../src/ip/ble/hl/src/profiles/hrp/hrps -I
./../../../src/ip/ble/hl/src/profiles/htp -I
./../../../src/ip/ble/hl/src/profiles/htp/htpc -I
./../../../src/ip/ble/hl/src/profiles/htp/htpt -I
./../../../src/ip/ble/hl/src/profiles/prox/proxm -I
./../../../src/ip/ble/hl/src/profiles/prox/proxr -I
./../../../src/ip/ble/hl/src/profiles/scpp -I
./../../../src/ip/ble/hl/src/profiles/scpp/scppc -I
./../../../src/ip/ble/hl/src/profiles/scpp/scpps -I
./../../../src/plf/refip/src/driver/intc -I ./../../../src/ip/ble/hl/src/rwble_hl
-I ./../../../src/ip/ble/ll/src/hcic -I ./../../../src/ip/ble/hl/src/host/smp -I
./../../../src/modules/app/api -I ./../../../src/modules/gtl/src -I
./../../../src/ip/ble/hl/src/profiles/anp -I
./../../../src/ip/ble/hl/src/profiles/anp/anpc -I
./../../../src/ip/ble/hl/src/profiles/anp/anps -I
./../../../src/ip/ble/hl/src/profiles/cscp -I
./../../../src/ip/ble/hl/src/profiles/cscp/cscpc -I
./../../../src/ip/ble/hl/src/profiles/cscp/cscps -I
./../../../src/ip/ble/hl/src/profiles/glp -I
./../../../src/ip/ble/hl/src/profiles/glp/glpc -I
./../../../src/ip/ble/hl/src/profiles/glp/glps -I
./../../../src/ip/ble/hl/src/profiles/pasp -I
./../../../src/ip/ble/hl/src/profiles/pasp/paspc -I
./../../../src/ip/ble/hl/src/profiles/pasp/pasps -I
./../../../src/ip/ble/hl/src/profiles/rscp -I
./../../../src/ip/ble/hl/src/profiles/rscp/rscpc -I
./../../../src/ip/ble/hl/src/profiles/rscp/rscps -I
./../../../src/ip/ble/hl/src/profiles/tip -I
./../../../src/ip/ble/hl/src/profiles/tip/tipc -I
./../../../src/ip/ble/hl/src/profiles/tip/tips -I ./../../../src/modules/app/src -
I ./../../../src/plf/refip/src/driver/adc -I
./../../../src/plf/refip/src/driver/wkupct -I
./../../../src/plf/refip/src/driver/battery -I
./../../../src/modules/app/src/app_project/template_fh -I
./../../../src/modules/app/src/app_project/template_fh/system -include
da14580_config.h -Wno-attributes -mthumb -mcpu=cortex-m0 -D__STACK_SIZE=0x0600 -
D__HEAP_SIZE=0x0100 -D__STARTUP_CLEAR_BSS_MULTIPLE -std=gnu99 -c
```

```
../../../src/plf/refip/src/arch/main/ble/arch_main.c -o
../../../src/plf/refip/src/arch/main/ble/arch_main.o
In file included from
./../../../src/modules/app/src/app_project/template_fh/system/app_sleep.h:10:0,
                 from ../../../src/plf/refip/src/arch/main/ble/arch_main.c:66:
./../../../src/plf/refip/src/driver/gpio/gpio.h:119:20: warning: inline function
'GPIO_GetPinStatus' declared but never defined [enabled by default]
 extern inline bool GPIO_GetPinStatus( GPIO_PORT port, GPIO_PIN pin );
                    ^
```

In this case, the exact command (e.g. how the compiler is invoked) is printed out for each action. It is easy to miss warnings with such lengthy output, but this verbosity level is useful to see what is really happening or to debug problems with the Makefile.

- Verbose output (lines 49 to 50); very verbose output, when invoked, for example, with "`make V=2`". In this case, besides the output generated with "`make V=2`", extra diagnostic messages are printed by the compiler and the linker. This verbosity level could be helpful to debug problem with the tool chain (e.g. library search paths, system library selection etc).

5.1.2 The project-dependent part of the project build files (Makefile)

The common part of the project build files (i.e. the part that is shared between different projects) is actually invoked by a project-dependent `Makefile`. A Makefile template (`tools/uvproj2Makefile/Makefile.tmpl`) is listed below, which is used to create an actual Makefile as described in section 6:

```
<  1> #
<  2> # The Makefile must define the following variables, which are used by common.mk:
<  3> #  O: output directory
<  4> #  STACK_SIZE: number of bytes to reserve for the stack
<  5> #  HEAP_SIZE: number of bytes to reserve for the heap
<  6> #  CORTEX_M: the Cortex-M model to target
<  7> #  STARTUP_DEFS: define's that configure the start-up code
<  8> #  CFLAGS: flags used for compilation AND linking
<  9> #  LDFLAGS: flags used for linking
< 10> #  ROM_MAP_FILE:ROM map file
< 11> #  OBJ: list of object files to compile and link
< 12> #  TARGET_ELF: name of the target (used in the generated images)
< 13> #
< 14>
< 15> # output directory
< 16> O ?= out
< 17>
< 18> # stack size
< 19> STACK_SIZE = 0x0600
< 20> # heap size
< 21> HEAP_SIZE = 0x0100
< 22>
< 23> # Startup code
< 24> CORTEX_M := 0
< 25> STARTUP = startup_ARMCM$(CORTEX_M).S
< 26>
< 27> # startup configuration
< 28> STARTUP_DEFS += -D__STACK_SIZE=$(STACK_SIZE) -D__HEAP_SIZE=$(HEAP_SIZE)
< 29> #STARTUP_DEFS += -D__START=main
< 30> #STARTUP_DEFS += -D__STARTUP_COPY_MULTIPLE
< 31> STARTUP_DEFS += -D__STARTUP_CLEAR_BSS_MULTIPLE
< 32>
< 33> # -Os -flto -ffunction-sections -fdata-sections to compile for code size
< 34> CFLAGS += -flto
< 35> CFLAGS += -ffunction-sections -fdata-sections
```

```
< 36> # Link for code size
< 37> GC := -Wl,--gc-sections
< 38>
< 39> # optimisation flags
< 40> CFLAGS += -Os -fplugin=tree_switch_shortcut_elf
< 41>
< 42>
< 43> # include search paths
< 44> core_inc_search_paths := \
< 45> @@core_inc_search_paths@@
< 46>
< 47> app_inc_search_paths := \
< 48> @@app_inc_search_paths@@
< 49>
< 50> inc_search_paths := $(core_inc_search_paths) $(app_inc_search_paths)
< 51> CFLAGS += $(foreach d,$(inc_search_paths),-I $(d))
< 52>
< 53>
< 54> # global configuration
< 55> CFLAGS += -include da14580_config.h
< 56>
< 57> ROM_MAP_FILE := ../../../misc/rom_symdef.txt
< 58>
< 59> LDFLAGS += $(USE_NANO) $(USE_NOHOST) $(LDSCRIPTS) $(GC) $(MAP)
< 60>
< 61> # don't complain about unknown attributes (i.e. zero_init)
< 62> CFLAGS += -Wno-attributes
< 63>
< 64>
< 65> # CHECK: this flag prevents the warning
< 66> # "uses 2-byte wchar_t yet the output is to use 4-byte wchar_t;
< 67> # use of wchar_t values across objects may fail"
< 68> # revisit if things don't work as expected with wchar_t strings.
< 69> LDFLAGS += -Wl,--no-wchar-size-warning
< 70>
< 71>
< 72> # source files
< 73> core_src_cfiles := \
< 74> @@core_src_cfiles@@
< 75>
< 76> app_src_files := \
< 77> @@app_src_files@@
< 78>
< 79> src_cfiles := $(core_src_cfiles) $(app_src_files)
< 80>
< 81> src_Sfiles  := \
< 82>
< 83> obj_cfiles := $(src_cfiles:.c=.o)
< 84> obj_Sfiles := $(src_Sfiles:.S=.o)
< 85>
< 86> # patch objects
< 87> patch_objs := \
< 88> @@patch_objs@@
< 89>
< 90>
< 91> startup_obj := $(STARTUP:.S=.o)
< 92>
< 93>
< 94> OBJ := $(obj_cfiles) $(obj_Sfiles) $(startup_obj)
< 95>
```

```
< 96>
< 97> # target
< 98> TARGET_ELF := full_emb_sysram
< 99>
<100>
<101> include common.mk
```

The last line of the template includes common.mk, which does all the work, based on the definitions provided by the Makefile.

Lines 3 to 12 list the variables that each Makefile should define, for proper operation of the invoked common.mk.

Line 16 provides a default value for the variable **$(O)**, which signifies the output directory (i.e. where the final binary images will be stored.

The number of bytes to reserve for the stack and the heap are defined by the variables **$(STACK_SIZE)** and **$(HEAP_SIZE)**, respectively at lines 19 and 21.

The variable **$(CORTEX_M)** (in line 24) defines the model of the ARM Cortex-M family that the target has, so that the proper compilation and linking flags are set up. For DA14580 the correct value is 0.

In lines 28 to 31 the variable **$(STARTUP_DEFS)** is assigned some definitions that affect the start-up code (discussed in section 5.3). This is particularly useful when using a variant of the sample start-up files that come with the GCC ARM tool chain.

The canonical variable for defining the compilation flags (which are also used for linking) is **$(CFLAGS)**. Some basic flags are assigned to $(CFLAGS) in common.mk. More flags are appended by the Makefile template, to further customise compilation. It is important to point out that the Makefile should normally only *append* to $(CFLAGS), by using "+=" for the assignment, instead of "=" or ":=". If this guideline is not followed, the base compilation flags set by common.mk will be overwritten, potentially leading to compilation and/or linking problems or to the generation of improper code.

The analogous variable for the linking is **$(LDFLAGS)**. The advice for only appending to $(LDFLAGS) in the Makefile applies in this case, too. It should be noted that the recipe (in common.mk) for linking the executable uses both $(CFLAGS) and $(LDFLAGS), in that order.

The variable **$(ROM_MAP_FILE)** is used to define the path to a text file that contains the list of symbols (functions and variables) of the code that resides in the ROM. This file is included in the code release and only the path to it might have to be adjusted.

The variable **$(OBJ)** is used to list all the object files that should be linked to produce the executable. The Makefile template first generates several lists of source files (separately for C and assembly files) and then transforms the source file names into object file names.

Finally, the variable **$(TARGET_ELF)** provides the base name for the images to be generated. For example, if $(TARGET_ELF) is set to my_app, the images my_app.axf (ELF), my_app.hex (Intel hex) and my_app.bin (binary) will be created.

5.2 GNU linker script

The user of the GNU linker (ld) may use custom linker scripts to dictate the memory layout of an executable. These are equivalent to Keil uVision's scatter files. The linker script template tools/uvproj2Makefile/580.lds.S replicates the memory layout defined by the common scatter file dk_apps/scatterfiles/scatterfile_common.sct. It is a *template*, because it contains some C pre-processor directives. The project build files, described in section 5.1, create a real linker script named 580.lds from this template.

5.3 Start-up code for GCC

The start-up code is written in assembly and usually does the following:

1. Initialises some registers, e.g. to set up memory mappings.

2. Enforces the memory layout by:
   a. Copying code and/or data sections to their execution (in Keil's terminology) or virtual memory (in GNU ld terminology) address.
   b. Filling certain sections (e.g. `.bss`) with zeroes.
3. Jumps to the entry point of the application. This typically is the function `_start()` in the system library, which eventually calls `main()`.

Evidently, steps 1 and 3 are usually the same between different projects, but step 2 is prone to change from one project to the next, due to different memory usage. However, generic start-up code may be found at `tools/uvproj2Makefile/startup_ARMCM0.S`. With the proper value assigned to `$(STARTUP_DEFS)` in the `Makefile` (see 5.1.2), it should cover most cases without any changes in its code.

In addition, the start-up source file defines handlers for certain IRQs and provides default handlers for the rest of the IRQs. The default interrupt handlers merely go into an infinite loop.

6 Converting a Keil project file into a Makefile

The process of manually creating a `Makefile` from a Keil uVision project file should be clear, but it is tedious and error-prone. Therefore, a script was developed to automate the process; it can be found at `tools/uvproj2Makefile/uvproj2Makefile`. It is a Bash shell script, which uses `sed` to extract information from the `.uvproj` file. As noted in section 4.1 the `.uvproj` file is an XML file. However, `uvproj2Makefile` does not parse it as an XML document but rather as a text file: it looks for specific XML tags, though, before attempting to extract the sought information.

The script accepts the following command-line options:

```
Usage: ./tools/uvproj2Makefile/uvproj2Makefile [options]
        -i UVPROJ              input .uvproj file (must be defined)
        -o OUTPUT_MAKEFILE     output Makefile (default: Makefile.out)
        -t MAKEFILE_TEMPLATE   Makefile template (default: Makefile.tmpl)
        -v:                    show version
```

The script uses a `.uvproj` file as input and needs a Makefile template with certain placeholders in it. The name of the output Makefile is configurable. When it parses a `.uvproj` file, the script searches for certain XML tags, extracts strings and puts them into specific placeholders in the Makefile template, as described in Table 1.

Table 1: How XML tags are used to fill placeholders in the Makefile template.

XML tag	Type of extracted string	Placeholder in Makefile template
IncludePath	Include search path	@@core_inc_search_paths@@ @@app_inc_search_paths@@ Note 1
FileType	C source files Note 3	@@core_src_cfiles@@ @@app_src_files@@ Note 1
FileType	ELF object files Note 3	@@patch_objs@@

Note 1 Files are heuristically split into "core" and "application".

Note 2 C sources files have <FileType>1<\/FileType>.

Note 3 ELF object files have <FileType>3<\/FileType>.

Usually there is more than one string for each XML tag type; in this case, these multiple strings are concatenated in the resulting `Makefile`. In addition, the script attempts to split the C source files and the include search paths into "core" and "application", based on a simple heuristic: if a path contains

the string "`app_project`" it is classified as "application; otherwise, it is classified as "core." If this heuristic fails to classify correctly a path, there is no consequence, as the two sets of paths are eventually merged back into one set before being used. The split is done only for the clarity of the `Makefile`.

7 Benchmarks

This section discusses the results of two benchmarks, which were used to compare the performance of the ARM GCC and Keil's compiler. Probably the most important metric of a BLE application is its memory footprint, as memory is the most limited resource on a DA14580-based device. The second metric is the code efficiency in terms of speed. To assess these two metrics, a template BLE project and the well-known Dhrystone benchmark were used.

7.1 Building parameters

Both compilers provide a large number of parameters to fine-tune the optimisations performed while building code. Optimisation strategies typically aim at either small size or high speed; higher speed sometimes leads to increased speed (due to loop unrolling, function inlining etc.). Another optimisation option is *link-time* (in GCC's terminology), or *cross-module* (in Keil's terminology), optimisation, which performs optimisations during linking, when all of the object code is available, rather than the traditional intra-module optimisation. The two strategies can be combined, in theory. Unfortunately, during testing it was found that combining optimise-for-speed flags with LTO caused GCC to produce non-working images; it is unclear whether this is a bug or some misconfiguration. Therefore, when building with ARM GCC, LTO should be used only with the optimise-for-size flag (i.e. `-Os`).

A different building option to consider is the variant of the system library to use. The system library provides the standard C functions, for tasks such as memory copies, string operations, mathematical functions, file handling etc. However, most embedded systems do not require a full-fledged system library (e.g. floating-point operations or file handling are redundant). Both ARM GCC and Keil provide stripped-down system libraries, which are optimised for size. In case of ARM GCC it is called Newlib-nano, while Keil calls it MicroLIB. The impact of using these variants is evident in the results presented in the next section.

7.2 Dhrystone results

The Dhrystone benchmark was executed, built with different options, for 300,000 iterations. Repetition of a run produced the exact same score, which is to be expected for a bare-metal system without caches, such as DA14580.

Table 2: Dhrystone results with full-fledged system library

	ARM GCC					Keil µVision 5			
Opt. Level	Default Note 1	Os Note 2	Os+LTO Note 5	O2 Note 3	O3 Note 4	Default Note 1	O3,size Note 6	O2 Note 3	O3 Note 4
µs/loop	102.43	77.43	48.43	42.43	40.57	39.97	55.63	39.97	36.50
Dhrystones	9762	12914	20647	23566	24651	25021	17975	25021	27397
DMIPS	5.56	7.35	11.75	13.41	14.03	14.24	10.23	14.24	15.59
DMIPS/MHz	**0.35**	**0.46**	**0.73**	**0.84**	**0.88**	**0.89**	**0.64**	**0.89**	**0.97**
Binary size	12540	10856	10216	10608	10576	11768	11196	11768	11996

Note 1 Default optimisation

Note 2 Optmise for size

Note 3 High optimise-for-speed level

Note 4 Highest optimise-for-speed level

Note 5 Default optimisation combination used in BLE projects with GCC

Note 6 Default optimisation combination used in BLE projects with Keil

Table 2 shows the results for different optimisation options, when the normal full-fledged system library is used. In addition, the size in bytes of the resulting application is given. The absolute best numbers for the two basic metrics (speed and size) are highlighted.

Briefly, Keil gives a higher speed than GCC and GCC gives a smaller size, for equivalent building options. Interestingly, the optimisation flags used by default in BLE projects, give both better speed and better size in the case of GCC than in the case of Keil. It should be noted that LTO did not have any effect in the Dhrystone scores and only marginally decreased the binary size with Keil, while it improved the score considerably with GCC.

In almost all cases, equivalent optimisation flags make a difference of 5 % to 10 % in favour of GCC or Keil, for both metrics.

Table 3: Dhrystone results with stripped-down system libraries

	ARM GCC with Newlib-nano					Keil µVision 5 with MicroLIB			
Opt. Level	Default Note 1	Os Note 2	Os+LTO Note 5	O2 Note 3	O3 Note 4	Default Note 1	O3,size Note 6	O2 Note 3	O3 Note 4
µs/loop	102.43	77.43	48.43	42.43	40.57	75.33	107.33	71.90	71.90
Dhrystones	9762	12914	20647	23566	24651	13274	9317	13908	13908
DMIPS	5.56	7.35	11.75	13.41	14.03	7.56	5.30	7.92	7.92
DMIPS/MHz	**0.35**	**0.46**	**0.73**	**0.84**	**0.88**	**0.47**	**0.33**	**0.49**	**0.49**
Binary size	8228	6444	5804	6196	6164	8208	7720	8436	8436

Note 1 Default optimisation

Note 2 Optmise for size

Note 3 High optimise-for-speed level

Note 4 Highest optimise-for-speed level

Note 5 Default optimisation combination used in BLE projects with GCC

Note 6 Default optimisation combination used in BLE projects with Keil

Table 3 presents the results when Dhrystone is built with the stripped-down system libraries (Newlib-nano or MicroLIB). An important difference between GCC and Keil is that Newlib-nano (comes with GCC) does not have any impact on speed, but MicroLIB (comes with Keil) brings the scores down by about 50 %. This very high penalty imposed by MicroLIB might have to do with the implementation of memory and string functions in this particular library. In terms of size, Newlib-nano decreases size by about 4 KiB, while MicroLIB brings the size down by about 3.5 KiB. In this context, GCC with the highest optimise-for-speed level (`-O3`) is probably the best trade-off between speed and size.

7.3 BLE template project results

The second test case is the BLE template project (`fh_project_template`), which just performs advertisement. For this application it only makes sense to check the memory footprint, when building with a different tool chain and with different building options.

Table 4: Memory footprint of the BLE template project with full-fledged system libraries

	ARM GCC				Keil µVision 5					
Opt. level	Os	Os +LTO	O2	O3	O3,size	O2	O3	O3,size +LTO	O2 +LTO	O3 +LTO
text	14048	13184	14652	14708	10256	10480	10516	9796	9948	9956
data	1176	1172	1180	1180	0	0	0	0	0	0
bss	11458	11462	11460	11460	13608	13608	13608	13608	13608	13608
total	26682	25818	27292	27348	23864	24088	24124	23404	23556	23564

Table 4 presents the static memory footprint, when using the full-fledged system libraries. In this case, the sizes when enabling LTO in Keil are also given, as there is a considerable difference with the non-LTO case. The static memory footprint constitutes of the following parts:

- Text: the code of the application.
- Data: the static, initialised data of the application.
- BSS: the zero-initialised data of the application; they are not stored in the application image, only the start and end addresses of the memory area need to be stored.

In this case, Keil with the optimisation flags used by default in the BLE projects plus LTO, give the smallest memory footprint. In general, Keil's LTO decreases the footprint by about 0.5 KiB. GCC's images have about 4 KiB larger code, 2 KiB less zero-initialised data and introduce over 1 KiB of initialised data, for equivalent optimisation flags. In total, GCC produces a memory footprint that is about 10 % to 13 % larger than when building with Keil.

Table 5: Memory footprint of the BLE template project with stripped-down system libraries

	ARM GCC with Newlib-nano				Keil µVision 5 with MicroLIB					
Opt. level	Os	Os +LTO	O2	O3	O3,size	O2	O3	O3,size +LTO	O2 +LTO	O3 +LTO
text	13272	12416	13844	13940	9592	9812	9848	9132	9280	9288
data	208	204	212	212	0	0	0	0	0	0
bss	11458	11462	11460	11460	13252	13252	13252	13252	13252	13252
total	24938	24082	25516	25612	22844	23064	23100	22384	22532	22540

Table 5 presents the analogous results when the stripped-down system libraries are used. The general trend is the same as before, with Keil providing a smaller memory footprint. In general, MicroLIB decreases the footprint by 1 KiB, while Newlib-nano decreases it by over 1.5 KiB. The memory footprint of the BLE template project when built with GCC is about 7 % to 10 % larger than when built with Keil, for equivalent optimisation flags.

7.4 General comments

The two benchmarks used to compare GCC and Keil do not give a clear answer which of the two is *better* for every case. However, the differences both in size and in speed are generally around 10 % for either of them, depending on the case, for equivalent building options. An important note is that Keil's MicroLIB imposed a 50 % penalty on Dhrystone scores, which might be relevant for some applications.

8 Conclusions

This document described how a BLE Keil uVision project can be replaced by a Makefile, which can then be used to build the project with GCC. This is an attractive capability, to avoid any licensing limitations imposed by non-open source tool chains.

A script has been developed to automate this conversion. The resulting `Makefile`, start-up code and linker script may require manual fine tuning for better performance or customisation, but they should be able to generate working images out-of-the-box.

Finally, some benchmarks regarding speed and size were presented, which do not give a clear choice between GCC and Keil for every case. However, the differences in either speed or size are in the order of 10 %.

9 Revision history

Revision	Date	Description
1.2	23-Dec-2021	Updated logo, disclaimer, copyright.
1.1	16-Jul-2014	Include benchmark results.
1.0	23-May-2014	Initial version.

Status definitions

Status	Definition
DRAFT	The content of this document is under review and subject to formal approval, which may result in modifications or additions.
APPROVED or unmarked	The content of this document has been approved for publication.

RoHS Compliance

Dialog Semiconductor complies to European Directive 2001/95/EC and from 2 January 2013 onwards to European Directive 2011/65/EU concerning Restriction of Hazardous Substances (RoHS/RoHS2).
Dialog Semiconductor's statement on RoHS can be found on the customer portal https://support.diasemi.com/. RoHS certificates from our suppliers are available on request.