```
/* Linker script for Silicon Labs EFM32PG12B devices                          */
/*                                                                            */
/* This file is subject to the license terms as defined in ARM's              */
/* CMSIS END USER LICENSE AGREEMENT.pdf, governing the use of                 */
/* Example Code.                                                              */
/*                                                                            */
/* Copyright 2017 Silicon Laboratories, Inc. http://www.silabs.com            */
/*                                                                            */
/* Version 5.1.2 */
/*                                                                            */

#if !defined(MBED_APP_START)
  #warning "MBED_APP_START not defined"
  #define MBED_APP_START 0x00000000
#endif

#if !defined(MBED_APP_SIZE)
  #define MBED_APP_SIZE 1048576
#endif

MEMORY
{
  FLASH (rx) : ORIGIN = MBED_APP_START, LENGTH = MBED_APP_SIZE
  RAM (rwx)  : ORIGIN = 0x20000000, LENGTH = 262144
}

/* MBED: mbed needs to be able to dynamically set the interrupt vector table.
 * We make room for the table at the very beginning of RAM, i.e. at
 * 0x20000000. We need (16+51 * sizeof(uint32_t) = 268 bytes for EFM32PG */
__vector_size = 0x10C;

/* Linker script to place sections and symbol values. Should be used together
 * with other linker script that defines memory regions FLASH and RAM.
 * It references following symbols, which must be defined in code:
 *   Reset_Handler : Entry of reset handler
 *
 * It defines following symbols, which code can use without definition:
 *   __exidx_start
 *   __exidx_end
 *   __copy_table_start__
 *   __copy_table_end__
 *   __zero_table_start__
 *   __zero_table_end__
 *   __etext
 *   __data_start__
 *   __preinit_array_start
 *   __preinit_array_end
 *   __init_array_start
 *   __init_array_end
 *   __fini_array_start
 *   __fini_array_end
 *   __data_end__
 *   __bss_start__
 *   __bss_end__
 *   __end__
 *   end
 *   __HeapLimit
 *   __StackLimit
 *   __StackTop
 *   __stack
 *   __Vectors_End
 *   __Vectors_Size
 */
ENTRY(Reset_Handler)

SECTIONS
{
  .text :
  {
```

```
70        KEEP(*(.vectors))
71        __Vectors_End = .;
72        __Vectors_Size = __Vectors_End - __Vectors;
73        __end__ = .;
74
75        *(.text*)
76
77        KEEP(*(.init))
78        KEEP(*(.fini))
79
80        /* .ctors */
81        *crtbegin.o(.ctors)
82        *crtbegin?.o(.ctors)
83        *(EXCLUDE_FILE(*crtend?.o *crtend.o) .ctors)
84        *(SORT(.ctors.*))
85        *(.ctors)
86
87        /* .dtors */
88        *crtbegin.o(.dtors)
89        *crtbegin?.o(.dtors)
90        *(EXCLUDE_FILE(*crtend?.o *crtend.o) .dtors)
91        *(SORT(.dtors.*))
92        *(.dtors)
93
94        *(.rodata*)
95
96        KEEP(*(.eh_frame*))
97      } > FLASH
98
99      .ARM.extab :
100     {
101       *(.ARM.extab* .gnu.linkonce.armextab.*)
102     } > FLASH
103
104     __exidx_start = .;
105     .ARM.exidx :
106     {
107       *(.ARM.exidx* .gnu.linkonce.armexidx.*)
108     } > FLASH
109     __exidx_end = .;
110
111     /* To copy multiple ROM to RAM sections,
112      * uncomment .copy.table section and,
113      * define __STARTUP_COPY_MULTIPLE in startup_ARMCMx.S */
114     /*
115     .copy.table :
116     {
117       . = ALIGN(4);
118       __copy_table_start__ = .;
119       LONG (__etext)
120       LONG (__data_start__)
121       LONG (__data_end__ - __data_start__)
122       LONG (__etext2)
123       LONG (__data2_start__)
124       LONG (__data2_end__ - __data2_start__)
125       __copy_table_end__ = .;
126     } > FLASH
127     */
128
129     /* To clear multiple BSS sections,
130      * uncomment .zero.table section and,
131      * define __STARTUP_CLEAR_BSS_MULTIPLE in startup_ARMCMx.S */
132     /*
133     .zero.table :
134     {
135       . = ALIGN(4);
136       __zero_table_start__ = .;
137       LONG (__bss_start__)
138       LONG (__bss_end__ - __bss_start__)
```

```
    LONG (__bss2_start__)
    LONG (__bss2_end__ - __bss2_start__)
    __zero_table_end__ = .;
  } > FLASH
  */

  __etext = .;

  .data : AT (__etext)
  {
    __data_start__ = .;
    PROVIDE( __start_vector_table__ = .);
    . += __vector_size;
    PROVIDE( __end_vector_table__ = .);
    *(vtable)
    *(.data*)
    . = ALIGN (4);
    *(.ram)

    . = ALIGN(4);
    /* preinit data */
    PROVIDE_HIDDEN (__preinit_array_start = .);
    KEEP(*(.preinit_array))
    PROVIDE_HIDDEN (__preinit_array_end = .);

    . = ALIGN(4);
    /* init data */
    PROVIDE_HIDDEN (__init_array_start = .);
    KEEP(*(SORT(.init_array.*)))
    KEEP(*(.init_array))
    PROVIDE_HIDDEN (__init_array_end = .);

    . = ALIGN(4);
    /* finit data */
    PROVIDE_HIDDEN (__fini_array_start = .);
    KEEP(*(SORT(.fini_array.*)))
    KEEP(*(.fini_array))
    PROVIDE_HIDDEN (__fini_array_end = .);

    KEEP(*(.jcr*))
    . = ALIGN(4);
    /* All data end */
    __data_end__ = .;

  } > RAM

  .bss :
  {
    . = ALIGN(4);
    __bss_start__ = .;
    *(.bss*)
    *(COMMON)
    . = ALIGN(4);
    __bss_end__ = .;
  } > RAM

  .heap (COPY):
  {
    __HeapBase = .;
    __end__ = .;
    end = __end__;
    _end = __end__;
    KEEP(*(.heap*))
    __HeapLimit = .;
  } > RAM

  /* .stack_dummy section doesn't contains any symbols. It is only
   * used for linker to calculate size of stack sections, and assign
   * values to stack symbols later */
```

```
208     .stack_dummy (COPY):
209     {
210       KEEP(*(.stack*))
211     } > RAM
212
213     /* Set stack top to end of RAM, and stack limit move down by
214      * size of stack_dummy section */
215     __StackTop = ORIGIN(RAM) + LENGTH(RAM);
216     __StackLimit = __StackTop - SIZEOF(.stack_dummy);
217     PROVIDE(__stack = __StackTop);
218
219     /* Check if data + heap + stack exceeds RAM limit */
220     ASSERT(__StackLimit >= __HeapLimit, "region RAM overflowed with stack")
221
222     /* Check if FLASH usage exceeds FLASH size */
223     ASSERT(ORIGIN(FLASH) + LENGTH(FLASH) >= (__etext + SIZEOF(.data)), "FLASH memory
        overflowed !")
224   }
225
```