Source Code

Kernel.asm

```
PERIPHERAL_BASE = $3F000000    ;defines a variable for addressing the peripheral base
GPIO_BASE =  $200000           ;defines a variable for addressing the base of the GPIOs
GPFSEL1 = $4                   ;defines a variable for addressing GPIO function select register 1
GPSET0 = $1C                   ;defines a variable for addressing GPIO pin output set 0
GPCLR0 = $28                   ;defines a variable for addressing GPIO pin output clear 0
TIMER_BASE = $3000             ;defines a variable for addressing the base of the system timer
TIMER_CNT = $4                 ;defines a variable for addressing the lower 32 bits of the system
                               ; timer counter

format binary as 'img'
include 'LIB\FASMARM.INC'

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                USED GENERAL PURPOSE REGISTERS                                    ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; There are 15 general purpose registers (R0 - R14), but two of them have a special status and ;
; should normally not be used as data registers: R13 holds the stack pointer and R14 the load  ;
; register. In this program, the following general purpose registers are in use - so when      ;
; changing the code, make sure you don't accidently overwrite any data:                        ;
;                                                                                              ;
; R0-R3: to load and store various data                                                        ;
; R5-R8: for the Timer macro                                                                   ;
; R11-R12: to store return addresses                                                           ;
;                                                                                              ;
; Registers R4, R9 and R10 are currently not being used.                                       ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                        STEP 1: TIMER                                         ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; As a first step, we write a macro command, which accepts a parameter named "time" and        ;
; addresses the system timer. As the system timer runs independently of the cpu clock rate, it ;
; enables us to get accurate timing (real-time) in microseconds.                               ;
;                                                                                              ;
; The system timer base is in ST (= 0x3F003000)                                                ;
; The system timer counter lower 32 bits is in CLO (= 0x3F003004)                              ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
macro Timer time {

  local .wait                  ;defines a local loop called ".wait"
  imm32 r8, time               ;defines R8 as a 32-bit register and moves the value of "time"
                               ; (given as parameter
                               ;when the Timer macro is called upon) into it. R8 = requested
                               ; duration in microseconds
```

```
  ;initialising the Timer
  mov r6, PERIPHERAL_BASE      ;R6 = 0x3F000000
  orr r6, r6,TIMER_BASE        ;R6 = (0x3F000000 | 0x00003000) = 0x3F003000
  ldr r7, [r6,TIMER_CNT]       ;R7 = saves the initial time in microseconds (we get that time
                               ; from the system timer at 0x3F003004)

  .wait:
   ldr r5, [r6,TIMER_CNT]      ;R5 = current time in microseconds (we get that time from the
                               ; system timer at 0x3F003004)
   sub r5, r5,r7               ;R5 = elapsed time (current time - initial time)
   cmp r5, r8                  ;compares the elapsed time with the requested duration
  blt .wait                    ;if R5 < requested duration, return to .wait and execute loop
                               ; again
}

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                            STEP 2: Defining the GPIOs as outputs                              ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; GPIO17 is in GPFSEL1 (= 0x3F200004) at bits 23-21.                                            ;
; GPIO18 is in GPFSEL1 (= 0x3F200004) at bits 26-24.                                            ;
;                                                                                               ;
; To define the GPIOs as outputs, we have to set their values in GPFSEL1 to 001. As both pins  ;
; are in the same register, we can define them as outputs in one and the same step. The pins   ;
; lie next to each other, so we want to set the values in bits 24 to 21 to 1001 (= decimal 9). ;
; With a following logical shift left, the first 1 is on bit 21 and the second 1 on bit 24.    ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;initialising the GPIOs
mov r0, PERIPHERAL_BASE        ;R6 = 0x3F000000
orr r0, r0, GPIO_BASE          ;R6 = (0x3F000000 | 0x00200000) = 0x3F200000
mov r1, #9
lsl r1, #21
str r1, [r0, GPFSEL1]          ;GPIO17 and GPIO18 ports are set as outputs.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                  STEP 3: The SOS morse code                                   ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Here it is: The morse code "SOS" as an endless loop.                                          ;
;                                                                                               ;
; Of course, the morse code is adjustable at will. We can write sentences containing words and  ;
; numbers, as all 26 letters of the latin alphabet and numbers 0-9 exist as separate branches.  ;
; Just write "bl letter_[your letter]" or "bl number_[your number]" inside the loop - or remove ;
; the loop altogether if you want to have your sentence sent only once.                         ;
;                                                                                               ;
; As the code structure contains several nested branches, it is important to bear in mind that  ;
; the command "bl" (branch with link) saves the current instruction set with all set flags from ;
; the program counter (R15) in R14 before branching into the subroutine.                        ;
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

loop:
  bl letter_s
  bl letter_o
  bl letter_s
  bl new_word
b loop

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                              STEP 4: The branch for the time unit                              ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; The branch for the time unit (one dit) can be found here.                                      ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

time_unit_dit:                  ;one dit is the basic time unit from which all elements can be
                                ; derived
  Timer 240000                  ;wait 240000 microseconds (= 240 milliseconds), the duration
  bx lr                         ;corresponding to a speed of 5 words per minute.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                        STEP 5: The branches for dit, dah, and new word                         ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; The branches for dit, dah, and new word can be found here. In here, the LED and audio speaker ;
; are turned on or off for the duration of one, three or seven dits (as for a dit, a dah or a   ;
; new word, respectively).                                                                       ;
;                                                                                                ;
; To turn the GPIOs on, we have to set bits 17 and 18 (for GPIO17 and GPIO18) in register       ;
; GPSET0 (= 0x3F20001C) to 1. The pins lie next to each other, so we want to set the values in  ;
; bits 18 to 17 to 11 (= decimal 3). With a following logical shift left, the first 1 is on bit ;
; 17 and the second 1 on bit 18.                                                                 ;
;                                                                                                ;
; To turn the GPIOs off, we have to set bits 17 and 18 (for GPIO17 and GPIO18) in register      ;
; GPCLR0 (= 0x3F200028) to 1. The pins lie next to each other, so we want to set the values in  ;
; bits 18 to 17 to 11 (= decimal 3). With a following logical shift left, the first 1 is on bit ;
; 17 and the second 1 on bit 18.                                                                 ;
;                                                                                                ;
; For a lower or higher audio pitch, some values in both dit and dah have to be adjusted. The   ;
; default frequency for the output is 2000Hz. For a higher pitch, the frequency has to become   ;
; higher and for a lower pitch, the frequency has to become lower. The frequency is set by the  ;
; value in the parameter of the Timer macro. As default, the value is set to 500, so every 500  ;
; microseconds, the signal is turned on and off. 1000000 microseconds (= 1 second) divided by   ;
; 500 result in the abovementioned 2000Hz. For the standard pitch A ("Kammerton") of 440Hz, the ;
; Timer parameter has to be set to 2273 (1000000 / 440). Keep in mind, however, that the signal ;
; length is also going to be longer or shorter, if only the Timer parameter is adjusted. If the ;
; default signal length of 240 milliseconds is not to be changed, use the following formula:    ;
; counter (value in the "mov r2" command = (240000 / (Timer + Timer).                            ;
;                                                                                                ;
```

```
; A note to the "new word" branch: one space between two words corresponds to seven dits. As    ;
; each letter has already a pause of one dah (= three dits) at the end of its last signal (as a ;
; space between letters), we have to insert only four more dits for a new word.                ;
;                                                                                               ;
; Remember that in our "loop" code, the command "bl" had the program counter saved in R14. To   ;
; return from a "bl" branch to the instruction where it was called upon, we can just use the    ;
; "bx: lr" command at the end of the branch (as it is done in the branch "time_unit_dit"        ;
; above). This leads to an exchange of the instruction sets in the link register (R14) and the  ;
; register for the program counter (R15).                                                       ;
;                                                                                               ;
; However, if inside a "bl" branch another "bl" is used, the content in the link register (R14) ;
; is lost, as it is overwritten with the new "bl" command. Therefore, it is necessary to save   ;
; the content in R14 in another register at the beginning of each "bl" branch that uses "bl"    ;
; commands for itself (nested branches). This enables us to return to the main program even if  ;
; we use several nested branches. To do so, we just have to rewrite the "bx" command so it      ;
; addresses the register with the saved original instruction set (e.g. "bx: r11).               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

dit:
  movs r11, r14                ;sets the root from which the subroutine was started into R11
                               ; (return address)
  mov r1, #3
  lsl r1, #17
  mov r2, #240                 ;a counter is used to turn GPIO17 and GPIO18 on and off 240 times
                               ; for 500 microseconds (= 0.5 milliseconds) each
  countdown_dit:               ;This corresponds to a frequency of 2000Hz for a duration of
                               ; 240 milliseconds.
   str r1, [r0, GPSET0]        ;LED and speaker turn on ...
   Timer 500                   ;wait 0.5 milliseconds
   str r1, [r0, GPCLR0]        ;LED and speaker turn off ...
   Timer 500                   ;wait 0.5 milliseconds
   sub r2, #1
   cmp r2, #0
  bne countdown_dit
  bl time_unit_dit             ;wait for one dit
  bx r11

dah:
  movs r11, r14                ;sets the root from which the subroutine was started into R11
                               ; (return address)
  mov r1, #3
  lsl r1, #17
  mov r2, #720                 ;a counter is used to turn GPIO17 and GPIO18 on and off 720 times
                               ; for 500 microseconds (= 0.5 milliseconds) each
                               ;This corresponds to a frequency of 2000Hz for a duration of
                               ; 720 milliseconds.
  countdown_dah:
   str r1, [r0, GPSET0]        ;LED and speaker turn on ...
```

```
    Timer 500                   ;wait 0.5 milliseconds
    str r1, [r0, GPCLR0]        ;LED and speaker turn off ...
    Timer 500                   ;wait 0.5 milliseconds
    sub r2, #1
    cmp r2, #0
  bne countdown_dah
  bl time_unit_dit              ;wait for one dit
  bx r11

new_word:
  movs r12, r14                 ;sets the root from which the subroutine was started into R12
                                ; (return address)
  mov r3, #4
  countdown_new_word:           ;a counter is used to call upon the branch time_unit_dit four
                                ; times.
    bl time_unit_dit
    sub r3, #1
    cmp r3, #0
  bne countdown_new_word
  bx r12

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                       STEP 6: The branches for all letters and numbers                          ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; The branches for all letters and numbers can be found here.                                      ;
;                                                                                                  ;
; A note to the insertion of "bl time_unit_dit" at the end of each branch: one space between       ;
; two letters corresponds to three dits. As each letter has already a pause of one dit before      ;
; the last command "bx" inside the dit or dah element (as a space between signals), we have to     ;
; insert only two more dits for a new letter.                                                      ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

letter_a:                       ;di-dah
  movs r12, r14
  bl dit
  bl dah
  bl time_unit_dit
  bl time_unit_dit
  bx r12

letter_b:                       ;dah-di-di-dit
  movs r12, r14
  bl dah
  bl dit
  bl dit
  bl dit
  bl time_unit_dit
  bl time_unit_dit
```

```
  bx r12

letter_c:                       ;dah-di-dah-dit
  movs r12, r14
  bl dah
  bl dit
  bl dah
  bl dit
  bl time_unit_dit
  bl time_unit_dit
  bx r12

letter_d:                       ;dah-di-dit
  movs r12, r14
  bl dah
  bl dit
  bl dit
  bl time_unit_dit
  bl time_unit_dit
  bx r12

letter_e:                       ;dit
  movs r12, r14
  bl dit
  bl time_unit_dit
  bl time_unit_dit
  bx r12

letter_f:                       ;di-di-dah-dit
  movs r12, r14
  bl dit
  bl dit
  bl dah
  bl dit
  bl time_unit_dit
  bl time_unit_dit
  bx r12

letter_g:                       ;dah-dah-dit
  movs r12, r14
  bl dah
  bl dah
  bl dit
  bl time_unit_dit
  bl time_unit_dit
  bx r12

letter_h:                       ;di-di-di-dit
```

```
  movs r12, r14
  bl dit
  bl dit
  bl dit
  bl dit
  bl time_unit_dit
  bl time_unit_dit
  bx r12

letter_i:                       ;di-dit
  movs r12, r14
  bl dit
  bl dit
  bl time_unit_dit
  bl time_unit_dit
  bx r12

letter_j:                       ;di-dah-dah-dah
  movs r12, r14
  bl dit
  bl dah
  bl dah
  bl dah
  bl time_unit_dit
  bl time_unit_dit
  bx r12

letter_k:                       ;dah-di-dah
  movs r12, r14
  bl dah
  bl dit
  bl dah
  bl time_unit_dit
  bl time_unit_dit
  bx r12

letter_l:                       ;di-dah-di-dit
  movs r12, r14
  bl dit
  bl dah
  bl dit
  bl dit
  bl time_unit_dit
  bl time_unit_dit
  bx r12

letter_m:                       ;dah-dah
  movs r12, r14
```

```
  bl dah
  bl dah
  bl time_unit_dit
  bl time_unit_dit
  bx r12

letter_n:                       ;dah-dit
  movs r12, r14
  bl dah
  bl dit
  bl time_unit_dit
  bl time_unit_dit
  bx r12

letter_o:                       ;dah-dah-dah
  movs r12, r14
  bl dah
  bl dah
  bl dah
  bl time_unit_dit
  bl time_unit_dit
  bx r12

letter_p:                       ;di-dah-dah-dit
  movs r12, r14
  bl dit
  bl dah
  bl dah
  bl dit
  bl time_unit_dit
  bl time_unit_dit
  bx r12

letter_q:                       ;dah-dah-di-dah
  movs r12, r14
  bl dah
  bl dah
  bl dit
  bl dah
  bl time_unit_dit
  bl time_unit_dit
  bx r12

letter_r:                       ;di-dah-dit
  movs r12, r14
  bl dit
  bl dah
  bl dit
```

```
  bl time_unit_dit
  bl time_unit_dit
  bx r12

letter_s:                       ;di-di-dit
  movs r12, r14
  bl dit
  bl dit
  bl dit
  bl time_unit_dit
  bl time_unit_dit
  bx r12

letter_t:                       ;dah
  movs r12, r14
  bl dah
  bl time_unit_dit
  bl time_unit_dit
  bx r12

letter_u:                       ;di-di-dah
  movs r12, r14
  bl dit
  bl dit
  bl dah
  bl time_unit_dit
  bl time_unit_dit
  bx r12

letter_v:                       ;di-di-di-dah
  movs r12, r14
  bl dit
  bl dit
  bl dit
  bl dah
  bl time_unit_dit
  bl time_unit_dit
  bx r12

letter_w:                       ;di-dah-dah
  movs r12, r14
  bl dit
  bl dah
  bl dah
  bl time_unit_dit
  bl time_unit_dit
  bx r12
```

```
letter_x:                       ;dah-di-di-dah
  movs r12, r14
  bl dah
  bl dit
  bl dit
  bl dah
  bl time_unit_dit
  bl time_unit_dit
  bx r12

letter_y:                       ;dah-di-dah-dah
  movs r12, r14
  bl dah
  bl dit
  bl dah
  bl dah
  bl time_unit_dit
  bl time_unit_dit
  bx r12

letter_z:                       ;dah-dah-di-dit
  movs r12, r14
  bl dah
  bl dah
  bl dit
  bl dit
  bl time_unit_dit
  bl time_unit_dit
  bx r12

number_0:                       ;dah-dah-dah-dah-dah
  movs r12, r14
  bl dah
  bl dah
  bl dah
  bl dah
  bl dah
  bl time_unit_dit
  bl time_unit_dit
  bx r12

number_1:                       ;di-dah-dah-dah-dah
  movs r12, r14
  bl dit
  bl dah
  bl dah
  bl dah
  bl dah
```

```
  bl time_unit_dit
  bl time_unit_dit
  bx r12

number_2:                       ;di-di-dah-dah-dah
  movs r12, r14
  bl dit
  bl dit
  bl dah
  bl dah
  bl dah
  bl time_unit_dit
  bl time_unit_dit
  bx r12

number_3:                       ;di-di-di-dah-dah
  movs r12, r14
  bl dit
  bl dit
  bl dit
  bl dah
  bl dah
  bl time_unit_dit
  bl time_unit_dit
  bx r12

number_4:                       ;di-di-di-di-dah
  movs r12, r14
  bl dit
  bl dit
  bl dit
  bl dit
  bl dah
  bl time_unit_dit
  bl time_unit_dit
  bx r12

number_5:                       ;di-di-di-di-dit
  movs r12, r14
  bl dit
  bl dit
  bl dit
  bl dit
  bl dit
  bl time_unit_dit
  bl time_unit_dit
  bx r12
```

```
number_6:                       ;dah-di-di-di-dit
  movs r12, r14
  bl dah
  bl dit
  bl dit
  bl dit
  bl dit
  bl time_unit_dit
  bl time_unit_dit
  bx r12

number_7:                       ;dah-dah-di-di-dit
  movs r12, r14
  bl dah
  bl dah
  bl dit
  bl dit
  bl dit
  bl time_unit_dit
  bl time_unit_dit
  bx r12

number_8:                       ;dah-dah-dah-di-dit
  movs r12, r14
  bl dah
  bl dah
  bl dah
  bl dit
  bl dit
  bl time_unit_dit
  bl time_unit_dit
  bx r12

number_9:                       ;dah-dah-dah-dah-dit
  movs r12, r14
  bl dah
  bl dah
  bl dah
  bl dah
  bl dit
  bl time_unit_dit
  bl time_unit_dit
  bx r12
```