ME 305 Mechatronics

Department of Mechanical Engineering
Cal Poly
Spring Quarter - 2022

Lab 2: Cooperative Multi-Tasking for LED Blinking

As discussed in class, the objective of the lab this week is to blink two pairs of LEDs at independent rates. To accomplish this task, cooperative multi-tasking will be required. Each task is to be designed as a finite state machine.

- Rather than wiring the bits of Port T to be the number of tenths of a second to delay as we did in Lab 1, we will use a delay routine that delays only 1.00 ms. A `bgnd` instruction near the top of your code should provide the user the option to use the Debugger to set the number of milliseconds in the period for each LED pair. A sixteen-bit unsigned integer should be used for the number of milliseconds to delay each pair of LEDs. This will allow a maximum period of slightly over 65.5 seconds.

- Two electrical circuits that each drive two separate LEDs with two separate BS170 MOSFETs will be provided to you. Note: Use bit_4 and bit_5 of PORT P to drive the MOSFETs for the green and red LED in the first pair of LEDs, respectively, and use bit_6 and bit_7 of PORT P to drive the MOSFETs for the green and red LED in the second pair of LEDs, respectively. **CAUTION:** Wiring the pins of an output port to ground or +5V can damage the drive electronics for that pin. Anytime that you declare an I/O port pin to be an output pin, make sure that pin is not connected in a manner that will cause the pin to source or sink more than 20 mA. For example, wiring an output pin to +5V and setting its logic state to 0 will force the output pin to sink a substantial current, possibly rendering the pin useless in the future. Similarly, wiring an output pin to ground and setting its logic state to 1 will force the output pin to attempt to source a substantial current, possibly rendering the pin useless in the future.

- As in Lab 1B, write a program that turns each pair of LEDs ON and OFF according to the following cycle:

G_LED	ON	and	R_LED	OFF
G_LED	OFF	and	R_LED	OFF
G_LED	OFF	and	R_LED	ON
G_LED	OFF	and	R_LED	OFF
G_LED	ON	and	R_LED	ON
G_LED	OFF	and	R_LED	OFF

The shell code that you are to modify for Lab 2 can be found in the last pages of this handout. Complete this code so that it runs one pair of LEDs at the expected rate. Debug this code until you are certain that it executes properly. Then, modify the code to run a second pair of LEDs simultaneously, but at a different rate that is specified independently from the rate for the first pair of LEDs.

Note: Use the Debugger as necessary to ensure that the code for each state is working properly. As an overall check on the flow of your program, an excellent debugging paradigm is to put a `bgnd`

instruction before and after each `jsr TASK` instruction in your main program. Add your state variables for each task to the Data window in the Debugger. Then use the Debugger to run your code at full speed from one task to the next, which allows you to 'single step' from task to task. Check the sequence of states during execution against the state transition diagram for each task to make sure that the overall flow of your program is as intended. To check that any specific state is working properly, use the Debugger to single step into the appropriate task and take a closer look at the execution of the code for that state. An alternative and perhaps better approach to setting up your program so that you can single step from task to task is to do the following: when the execution of your code stops at the `bgnd` that you included near the top of your code, use the Debugger to set a breakpoint at each `jsr TASK` instruction in your main program, and then run full speed from one breakpoint to the next.

; Shell code for LED Lab 2 Exercise

```
;*******************************************************************************
;* Lab 2 shell code for students                                               *
;*******************************************************************************
;* Summary:                                                                    *
;*  This code is designed for use with the 2016 hardware for ME305. This code accepts *
;*  two two-byte integers through the debugger and uses these value to adjust the     *
;*  timing of two pairs of LEDs connected to Port P.                           *
;*                                                                             *
;* Author: William R. Murray                                                   *
;*  Cal Poly University                                                        *
;*  January 2020                                                               *
;*                                                                             *
;* Revision History:                                                           *
;*  WRM 04/13/2022                                                             *
;*   - reduced fully functional Lab 2 code to an almost functioning shell as a       *
;*      starting point for students                                            *
;*                                                                             *
;*  ToDo:                                                                      *
;*   - students complete, test, and debug as necessary                         *
;*******************************************************************************

;/-----------------------------------------------------------------------------\
;| Include all associated files                                                |
;\-----------------------------------------------------------------------------/
; The following are external files to be included during assembly


;/-----------------------------------------------------------------------------\
;| External Definitions                                                        |
;\-----------------------------------------------------------------------------/
; All labels that are referenced by the linker need an external definition

              XDEF  main

;/-----------------------------------------------------------------------------\
;| External References                                                        |
;\-----------------------------------------------------------------------------/
; All labels from other files must have an external reference

              XREF  ENABLE_MOTOR, DISABLE_MOTOR
              XREF  STARTUP_MOTOR, UPDATE_MOTOR, CURRENT_MOTOR
              XREF  STARTUP_PWM, STARTUP_ATD0, STARTUP_ATD1
              XREF  OUTDACA, OUTDACB
              XREF  STARTUP_ENCODER, READ_ENCODER
              XREF  INITLCD, SETADDR, GETADDR, CURSOR_ON, CURSOR_OFF, DISP_OFF
              XREF  OUTCHAR, OUTCHAR_AT, OUTSTRING, OUTSTRING_AT
              XREF  INITKEY, LKEY_FLG, GETCHAR
              XREF  LCDTEMPLATE, UPDATELCD_L1, UPDATELCD_L2
              XREF  LVREF_BUF, LVACT_BUF, LERR_BUF,LEFF_BUF, LKP_BUF, LKI_BUF
              XREF  Entry, ISR_KEYPAD
```

```
;/---------------------------------------------------------------------------\
;| Assembler Equates                                                         |
;\---------------------------------------------------------------------------/
; Constant values can be equated here

PORTP         EQU   $0258             ; output port for LEDs
DDRP          EQU   $025A

G_LED_1       EQU   %00010000         ; green LED output pin for LED pair_1
R_LED_1       EQU   %00100000         ; red LED output pin for LED pair_1
LED_MSK_1     EQU   %00110000         ; LED pair_1
G_LED_2       EQU   %01000000         ; green LED output pin for LED pair_2
R_LED_2       EQU   %10000000         ; red LED output pin for LED pair_2
LED_MSK_2     EQU   %11000000         ; LED pair_2




;/---------------------------------------------------------------------------\
;| Variables in RAM                                                          |
;\---------------------------------------------------------------------------/
; The following variables are located in unpaged RAM

DEFAULT_RAM:  SECTION


;/---------------------------------------------------------------------------\
;|  Main Program Code                                                        |
;\---------------------------------------------------------------------------/
; This code uses cooperative multitasking for Lab 2 from ME 305

MyCode:       SECTION


main:
        clr   t1state                 ; initialize all tasks to state0
        clr   t2state
        clr   t3state

; Normally no code other than that to clear the state variables and call the tasks
; repeatedly should be in your main program.  However, in this lab we will make a
; one-time exception: the following code will set TICKS_1 and TICKS_2 to default values
; and the BGND will give the user an opportunity to change these values in the debugger.

        movw  #100, TICKS_1           ; set default for TICKS_1
        movw  #200, TICKS_2           ; set default for TICKS_2
        bgnd                          ; stop in DEBUGGER to allow user to alter TICKS

Top:
        jsr   TASK_1                  ; execute tasks endlessly
        jsr   TASK_2
        jsr   TASK_3
        bra   Top

;-------------TASK_1 Pattern_1------------------------------------------------
TASK_1: ldaa  t1state                 ; get current t1state and branch accordingly
        beq   t1state0
        deca
        beq   t1state1
        deca
```

```
        beq   t1state2
        deca
        beq   t1state3
        deca
        beq   t1state4
        deca
        beq   t1state5
        deca
        beq   t1state6
        rts                             ; undefined state - do nothing but return

t1state0:                               ; init TASK_1 (not G, not R)
        bclr  PORTP, LED_MSK_1          ; ensure that LEDs are off when initialized
        bset  DDRP, LED_MSK_1           ; set LED_MSK_1 pins as PORTS outputs
        movb  #$01, t1state             ; set next state
        rts

t1state1:                               ; G, not R
        bset  PORTP, G_LED_1            ; set state1 pattern on LEDs
        tst   DONE_1                    ; check TASK_1 done flag
        beq   exit_t1s1                 ; if not done, return
        movb  #$02, t1state             ;   otherwise if done, set next state
exit_t1s1:
        rts

t1state2:                               ; not G, not R
        bclr  PORTP, G_LED_1            ; set state2 pattern on LEDs
        tst   DONE_1                    ; check TASK_1 done flag
        beq   exit_t1s2                 ; if not done, return
        movb  #$03, t1state             ;   otherwise if done, set next state
exit_t1s2:
        rts

t1state3:                               ; not G, R
        bset  PORTP, R_LED_1            ; set state3 pattern on LEDs
        tst   DONE_1                    ; check TASK_1 done flag
        beq   exit_t1s3                 ; if not done, return
        movb  #$04, t1state             ;   otherwise if done, set next state
exit_t1s3:
        rts

t1state4                                ; not G, not R
        bclr  PORTP, R_LED_1            ; set state4 pattern on LEDs
        tst   DONE_1                    ; check TASK_1 done flag
        beq   exit_t1s4                 ; if not done, return
        movb  #$05, t1state             ;   otherwise if done, set next state
exit_t1s4:
        rts

t1state5:                               ; G, R
        bset  PORTP, LED_MSK_1          ; set state5 pattern on LEDs
        tst   DONE_1                    ; check TASK_1 done flag
        beq   exit_t1s5                 ; if not done, return
        movb  #$06, t1state             ;   otherwise if done, set next state
exit_t1s5:
        rts

t1state6:                               ; not G, not R
        bclr  PORTP, LED_MSK_1          ; set state6 pattern on LEDs
        tst   DONE_1                    ; check TASK_1 done flag
```

```
        beq   exit_t1s6               ; if not done, return
        movb  #$01, t1state           ;   otherwise if done, set next state
exit_t1s6:
        rts                           ; exit TASK_1


;-------------TASK_2 Timing_1-----------------------------------------------------
TASK_2: ldaa  t2state                 ; get current t2state and branch accordingly
        beq   t2state0
        deca
        beq   t2state1
        rts                           ; undefined state - do nothing but return

t2state0:                             ; initialization for TASK_2
        movw  TICKS_1, COUNT_1        ; init COUNT_1
        clr   DONE_1                  ; init DONE_1 to FALSE
        movb  #$01, t2state           ; set next state
        rts

t2state1:                             ; Countdown_1
        ldaa  DONE_1
        cmpa  #$01
        bne   t2s1a                   ; skip reinitialization if DONE_1 is FALSE
t2s1a:                                ; ???
exit_t2s2:
        rts                           ; exit TASK_2


;-------------TASK_3 Delay 1ms----------------------------------------------------
TASK_3: ldaa  t3state                 ; get current t3state and branch accordingly
        beq   t3state0
        deca
        beq   t3state1
        rts                           ; undefined state - do nothing but return

t3state0:                             ; initialization for TASK_3
                                      ; no initialization required
        movb  #$01, t3state           ; set next state
        rts

t3state1:
        jsr   DELAY_1ms
        rts                           ; exit TASK_3


;/-------------------------------------------------------------------------------\
;| Subroutines                                                                   |
;/-------------------------------------------------------------------------------/

; Add subroutines here:

DELAY_1ms:
        ldy   #$0584
INNER:                                ; inside loop
        cpy   #0
        beq   EXIT
        dey
        bra   INNER
EXIT:
        rts                           ; exit DELAY_1ms
```

```
;/-------------------------------------------------------------------------------\
;| Messages                                                                      |
;/-------------------------------------------------------------------------------/

; Add ASCII messages here:




;/-------------------------------------------------------------------------------\
;| Vectors                                                                       |
;\-------------------------------------------------------------------------------/

; Add interrupt and reset vectors here:
        ORG   $FFFE                     ; reset vector address
        DC.W  Entry
```