Appendix A: Cerebro Code, Main File, Joseph Colosimo

```
; vim: ts=4 sw=4 fdm=marker
;------------------------------------------------------------------------------

; C E R E B R O
; J. Colosimo
; 6.115 Final Project
;
; CEREBRO
; main file
;------------------------------------------------------------------------------



;------------------------------------------------------------------------------

; CONSTANTS
;------------------------------------------------------------------------------

; ADDRESSES {{{
; EEG UART addresses
    Va_eeg equ 0FE00h
    Va_eeg_rxb equ (Va_eeg + 00h)   ; receiver buffer reg (dlab = 0)
    Va_eeg_ier equ (Va_eeg + 01h)   ; interrupt enable reg (dlab = 0)
    Va_eeg_dll equ (Va_eeg + 00h)   ; divisor latch (lsb) (dlab = 1)
    Va_eeg_dlm equ (Va_eeg + 01h)   ; divisor latch (msb) (dlab = 1)
    Va_eeg_lcr equ (Va_eeg + 03h)   ; line control reg
    Va_eeg_lsr equ (Va_eeg + 05h)   ; line status reg
; LED panel UART addresses
    Va_led equ 0FE20h
    Va_led_thr equ (Va_led + 00h)   ; transmitter buffer reg (dlab = 0)
    Va_led_ier equ (Va_led + 01h)   ; interrupt enable reg (dlab = 0)
    Va_led_dll equ (Va_led + 00h)   ; divisor latch (lsb) (dlab = 1)
    Va_led_dlm equ (Va_led + 01h)   ; divisor latch (msb) (dlab = 1)
    Va_led_lcr equ (Va_led + 03h)   ; line control reg
    Va_led_lsr equ (Va_led + 05h)   ; line status reg
; Equalizer board addresses
    Va_eqb_mux equ P1
    Va_eqb_adc equ 0FE10h
; }}}

; MEMORY {{{
; packet processing
    Vm_eeg_pptr_d equ 050h  ; packet pointer default address 0x50 -> 0x50+32=0
        x6F
    Vm_eeg_smst   equ 030h  ; current state machine state
    Vm_eeg_plen   equ 031h  ; packet length
    Vm_eeg_pptr   equ 032h  ; packet pointer
    Vm_eeg_csum   equ 033h  ; running checksum
; payload values
    Vm_eeg_sgnl   equ 034h      ; signal quality
    Vm_eeg_attn   equ 035h      ; attention
    Vm_eeg_mdtn   equ 036h      ; meditation
    Vm_eeg_drdy   equ 037h      ; data ready for processing
    Vm_eeg_dlta   equ 038h      ; delta (start of fft values) 0x38 -> 0x38
```

```
       +23=0x4F
        Vm_eeg_thta equ Vm_eeg_dlta+03d      ;    theta
        Vm_eeg_lalp equ Vm_eeg_dlta+06d      ;    low alpha
        Vm_eeg_halp equ Vm_eeg_dlta+09d      ;    high alpha
        Vm_eeg_lbet equ Vm_eeg_dlta+012d     ;    low beta
        Vm_eeg_hbet equ Vm_eeg_dlta+015d     ;    high beta
        Vm_eeg_lgam equ Vm_eeg_dlta+018d     ;    low gamma
        Vm_eeg_mgam equ Vm_eeg_dlta+021d     ;    mid gamma
    Vm_eeg_csum_got equ 070h     ; received checksum (for debugging)

    Vm_eeg_attn_lst equ 07Dh     ; previous attention value
    Vm_eeg_mdtn_lst equ 07Eh     ; previous meditation value

; equalizer board container memory
    Vm_eqb_vals equ 0B0h              ; 0xB0 -> 0xB7

; LED panel output buffer (used by F_lp_setpixels)
    Vm_led_rgbargs equ 071h                   ; 0x71 -> 0x71+12=0x7C
    Vm_led_rgbarg0 equ (Vm_led_rgbargs)       ; pixel 0
    Vm_led_rgbarg1 equ (Vm_led_rgbargs+3)     ; pixel 1
    Vm_led_rgbarg2 equ (Vm_led_rgbargs+6)     ; pixel 2
    Vm_led_rgbarg3 equ (Vm_led_rgbargs+9)     ; pixel 3

; transition system input buffer
    Vm_led_argbuf equ 081h                    ; 0x81 -> 0x81+12=0x8C
    Vm_led_argbuf0 equ (Vm_led_argbuf)        ; pixel 0
    Vm_led_argbuf1 equ (Vm_led_argbuf+3)      ; pixel 1
    Vm_led_argbuf2 equ (Vm_led_argbuf+6)      ; pixel 2
    Vm_led_argbuf3 equ (Vm_led_argbuf+9)      ; pixel 3

; transition system internal buffers
    Vm_led_rgbargs_cur equ 090h      ; pointer to current RGB arg buffer
    Vm_led_xfade equ 0A0h            ; crossfader value
    Vm_led_rgbargsA equ 091h         ; buffer A 0x91 -> 0x91+12=0x9C
    Vm_led_rgbargsB equ 0A1h         ; buffer B 0xA1 -> 0xA1+12=0xAC

; ewma values
    Vm_ewma_p0hue equ 0C0h
    Vm_ewma_p0val equ (Vm_ewma_p0hue+1)
    Vm_ewma_p1hue equ (Vm_ewma_p0hue+2)
    Vm_ewma_p1val equ (Vm_ewma_p0hue+3)
    Vm_ewma_p2hue equ (Vm_ewma_p0hue+4)
    Vm_ewma_p2val equ (Vm_ewma_p0hue+5)
    Vm_ewma_p3hue equ (Vm_ewma_p0hue+6)
    Vm_ewma_p3val equ (Vm_ewma_p0hue+7)
; }}}

org 00h
ljmp main

;--------------------------------------------------------------------------------

; INTERRUPTS
;--------------------------------------------------------------------------------
```

```
org 03h
    ljmp F_ec_int

org 0Bh
    ljmp F_th_int

;------------------------------------------------------------------------------------------

; MAIN LOOP
;------------------------------------------------------------------------------------------

org 0100h
main:
    lcall F_sc_initserial
    lcall F_ec_initeeg
    lcall F_eq_initeqb
    lcall F_lp_initled
    lcall F_th_inittimer
    setb EA

    ; set data ready off
    mov R0, #Vm_eeg_drdy
    mov @R0, #00h

    ; initialize first buffer to B (so that it switches to A on first data
    ; loading)
    mov R0, #Vm_led_rgbargs_cur
    mov @R0, #Vm_led_rgbargsB

    ; initialize LED buffer memory
    mov R0, #Vm_led_argbuf
    mov R1, #012d
    L_main_initinbuf:
        mov @R0, #0h
        inc R0
        djnz R1, L_main_initinbuf

    mov R0, #Vm_led_rgbargsA
    mov R1, #012d
    L_main_initAbuf:
        mov @R0, #0h
        inc R0
        djnz R1, L_main_initAbuf

    mov R0, #Vm_led_rgbargsB
    mov R1, #012d
    L_main_initBbuf:
        mov @R0, #0h
        inc R0
        djnz R1, L_main_initBbuf

    mov R0, #Vm_ewma_p0hue
    mov R1, #08d
```

```
L_main_initewmabuf:
    mov @R0, #0h
    inc R0
    djnz R1, L_main_initewmabuf

L_main_loop:
    ; packets happen at roughly 1-second intervals
    mov R0, #Vm_eeg_drdy
    mov A, @R0
    jz L_main_loop    ; if data isn't ready yet, don't process anything

    ; read equalizer board values
    lcall F_eq_scan

    ; convert the second value to something between 1 and 4 for the
    ; transition speed
    mov R0, #Vm_eqb_vals+1
    mov A, @R0
    mov B, #064d
    div AB
    inc A         ; A ranges from 1 - 4
    inc A         ; A ranges from 2 - 5
    inc A         ; A ranges from 3 - 6
    mov @R0, A

    ; convert the 3rd value to something between 0 and for the algo
    ; selection
    mov R0, #Vm_eqb_vals+2
    mov A, @R0
    mov B, #064d
    div AB
    mov @R0, A

    ; print retrieved data to serial
    lcall F_dbg_prtdata

    ; process data -> RGB args color
    lcall F_sp_process

    ; copy buffer to free transition buffer
    lcall F_th_copy

    ; initiate transition
    lcall F_th_start

    ; update last values, but only if signal quality is good
    mov R0, #Vm_eeg_sgnl
    mov A, @R0
    jnz L_main_loop_skiplst

    mov R0, #Vm_eeg_attn
    mov A, @R0
    mov R0, #Vm_eeg_attn_lst
    mov @R0, A
```

```
          mov R0, #Vm_eeg_mdtn
          mov A, @R0
          mov R0, #Vm_eeg_mdtn_lst
          mov @R0, A
          ;------

          L_main_loop_skiplst:
          ; reset drdy
          mov R0, #Vm_eeg_drdy
          mov @R0, #00h

          sjmp L_main_loop

;------------------------------------------------------------------------------------------------

; FUNCTIONS
;------------------------------------------------------------------------------------------------

F_dbg_prtdata:   ; print out EEG data {{{
      mov R0, #Vm_eeg_sgnl      ; signal quality
      mov A, @R0
      lcall F_sc_prthex

          mov A, #' '
          lcall F_sc_tx

      mov R0, #Vm_eeg_attn      ; attention
      mov A, @R0
      lcall F_sc_prthex

          mov A, #' '
          lcall F_sc_tx

      mov R0, #Vm_eeg_mdtn      ; meditation
      mov A, @R0
      lcall F_sc_prthex

      lcall F_sc_crlf
      mov A, #' '
      lcall F_sc_tx

      mov R0, #Vm_eqb_vals      ; equalizer
      mov R1, #08d
      F_dbg_prtdata_eqb:
          mov A, @R0
          inc R0
          lcall F_sc_prthex
          mov A, #' '
          lcall F_sc_tx
          djnz R1, F_dbg_prtdata_eqb

      lcall F_sc_crlf
```

```
    mov R0, #Vm_eeg_dlta      ; FFT values
    mov R1, #024d
    F_dbg_prtdata_fft:
        mov A, #' '
        lcall F_sc_tx
        mov A, @R0
        inc R0
        lcall F_sc_prthex
        djnz R1, F_dbg_prtdata_fft

    lcall F_sc_crlf
    lcall F_sc_crlf

    ret
    ; }}}
;--------------------------------------------------------------------------------

; INCLUDES
;--------------------------------------------------------------------------------

#include eegctrl.asm
#include eqboard.asm
#include sigproc.asm
#include sercom.asm
#include colutils.asm
#include ledpanel.asm
#include trans.asm
```

Appendix A: Cerebro Code, EEG Controller Library, Joseph Colosimo

```
; vim: ts=4 sw=4 fdm=marker
;---------------------------------------------------------------------------

; C E R E B R O
; J. Colosimo
; 6.115 Final Project
;
; EEG PACKET CAPTURE LIBRARY
; prefix: ec
; reads packet data from 16C450 and stores signal strength, the 8 EEG values,
; and the attention and meditation eSense values
;---------------------------------------------------------------------------



;---------------------------------------------------------------------------

; NEEDED VALUES
;---------------------------------------------------------------------------

; UART:  Va_eeg_dll, Va_eeg_dlm, Va_eeg_ier, Va_eeg_lcr, Va_eeg_lsr,
   Va_eeg_rxb
; Vm_eeg_pptr_d - base address for payload storage
; Vm_eeg_smst   - address of memory to store state machine state
; Vm_eeg_plen   - address of memory to store packet length
; Vm_eeg_pptr   - address of memory to store payload pointer
; Vm_eeg_csum   - address of memory to store current checksum calculation
; Vm_eeg_sgnl   - address of memory to store signal value
; Vm_eeg_dlta   - address of memory to store first delta byte (start of 24-
   bytes
;                 of FFT values)
; Vm_eeg_attn   - address of memory to store attention value
; Vm_eeg_mdtn   - address of memory to store meditation value

;---------------------------------------------------------------------------

; CONSTANTS
;---------------------------------------------------------------------------

V_ec_st_sync1 equ 00h
V_ec_st_sync2 equ 01h
V_ec_st_plnth equ 02h
V_ec_st_payld equ 03h
V_ec_st_chksm equ 04h

V_ec_val_sync equ 0AAh

;---------------------------------------------------------------------------

; FUNCTIONS
;---------------------------------------------------------------------------

F_ec_int:    ; interrupt handler {{{
```

```
; --- accounting ---
; R0 : current state    <-> mem
; R1 : captured byte    <-> mem
; R2 : packet length    <-> mem
; R3 : running checksum <-> mem
; R4 : packet buffer pointer <-> mem

; save A, B, and registers {{{
push acc
push B
push DPH
push DPL
push 00h     ; R0
push 01h     ; R1
push 02h     ; R2
push 03h     ; R3
push 04h     ; R4
; }}}

; get memory values {{{
; note: these are all indirect memory accesses, so the extra memory (80-FF
   )
; can be used, too
mov R1, #Vm_eeg_smst
mov A, @R1
mov R0, A

mov R1, #Vm_eeg_plen
mov A, @R1
mov R2, A

mov R1, #Vm_eeg_csum
mov A, @R1
mov R3, A

mov R1, #Vm_eeg_pptr
mov A, @R1
mov R4, A
; }}}

; this interrupt is called when we have heard something on the UART
; interrupt port

F_ec_int_getbyte:
    ; check line status reg to make sure we have a byte
    ;mov DPTR, #Va_eeg_lsr
    ;movx A, @DPTR
    ;anl A, #01h         ; bit 0 is data ready
    ;jz F_ec_int_end

    ; reading receiver buffer clears the interrupt, so do that
    mov DPTR, #Va_eeg_rxb
    movx A, @DPTR
    mov R1, A
```

```
F_ec_int_updatesm:
    ; run one step of the state machine with the new byte

    F_ec_int_us_sync1:
        cjne R0, #V_ec_st_sync1, F_ec_int_us_sync2      ; state == sync1 ?

        cjne R1, #V_ec_val_sync, F_ec_int_us_sync1_nosync ; received sync
           byte?
            ; reset state machine counters
            mov R3, #0h              ; reset running checksum to 0
            mov R4, #Vm_eeg_pptr_d   ; set packet pointer to starting
               address
            mov R2, #0h              ; invalidate packet length (FIXME: not
                necessary)

            ; new state -> sync2
            mov R0, #V_ec_st_sync2
            ljmp F_ec_int_us_sync1_done

        F_ec_int_us_sync1_nosync:
            mov R0, #V_ec_st_sync1   ; if no sync received, reset state
               machine

        F_ec_int_us_sync1_done:
            ljmp F_ec_int_us_done

    F_ec_int_us_sync2:
        cjne R0, #V_ec_st_sync2, F_ec_int_us_plnth      ; state == sync2 ?

        cjne R1, #V_ec_val_sync, F_ec_int_us_sync2_nosync ; received sync
           byte?
            ; new state -> plnth
            mov R0, #V_ec_st_plnth
            ljmp F_ec_int_us_sync2_done

        F_ec_int_us_sync2_nosync:
            mov R0, #V_ec_st_sync1   ; if no sync received, reset state
               machine

        F_ec_int_us_sync2_done:
            ljmp F_ec_int_us_done

    F_ec_int_us_plnth:
        cjne R0, #V_ec_st_plnth, F_ec_int_us_payld      ; state == plnth ?

        ; TODO: check packet size here?
        ; packets can concievably be up to 160 bytes long, but I've never
        ; seen anything other than 32-byte packets

        mov A, R1   ; save packet length
        add A, #01h
        mov R2, A
        ; new state -> payld
        mov R0, #V_ec_st_payld

        F_ec_int_us_plnth_done:
            ljmp F_ec_int_us_done

    F_ec_int_us_payld:
        cjne R0, #V_ec_st_payld, F_ec_int_us_chksm       ; state == payld ?

        ; if we haven't hit the end of the packet length yet... save
        ; current byte, increment memory pointer, and add to
        ; checksum
        ; otherwise, new state is chksm

        djnz R2, F_ec_int_us_payld_save

        F_ec_int_us_payld_completed:
            ;mov R0, #V_ec_st_chksm   ; if not, move to checksum stage
            ;ljmp F_ec_int_us_payld_done
            ; if not, jump to checksumming
            mov R0, #V_ec_st_chksm   ; FIXME: not necessary
            ljmp F_ec_int_us_chksm

        F_ec_int_us_payld_save:
            mov B, R1         ; B <- R1
            mov A, R4
            mov R1, A         ; R1 = R4
            mov @R1, B        ; save current byte
            mov R1, B         ; R1 <- B

            inc R4            ; increment pointer
            mov A, R3
            add A, R1         ; add to checksum
            mov R3, A

        F_ec_int_us_payld_done:
            ljmp F_ec_int_us_done

    F_ec_int_us_chksm:
        ; TODO cjne with error condition (state machine reset?)
        ; cjne R0, #V_ec_st_chksm, F_ec_int_us_error     ; state == chksm ?

        ; check to see that we have a valid checksum
        mov A, R3
        cpl A
        mov R3, A    ; save checksum back

        ; FIXME: for debugging
        ;mov R3, A    ; save checksum back
        ;mov R0, #Vm_eeg_csum_got
        ;mov B, R1
        ;mov @R0, B

        ; new state -> sync1
        mov R0, #V_ec_st_sync1

        ; hack: use R1 = 01h
        cjne A, 01h, F_ec_int_us_chksm_bad

        F_ec_int_us_chksm_good:
            ; the checksum was good, so we can now do one of two options:
            ;   1. process the payload right now and extract data from it
            ;   2. send a signal for the main loop to process the payload

            ; try processing the payload here
            lcall F_ec_pld
            mov R1, #Vm_eeg_drdy
            mov @R1, #01h            ; data is now ready

            ljmp F_ec_int_us_done

        F_ec_int_us_chksm_bad:
            ; the checksum was bad, so don't do anything
            ; TODO: turn this into some kind of error condition (spit out
            ;       something on serial?)
            mov A, #'B'
            lcall F_sc_tx

            ljmp F_ec_int_us_done

F_ec_int_us_done:

F_ec_int_end:
    ; set memory values {{{
    mov R1, #Vm_eeg_smst
    mov A, R0
    mov @R1, A

    mov R1, #Vm_eeg_plen
    mov A, R2
    mov @R1, A

    mov R1, #Vm_eeg_csum
    mov A, R3
    mov @R1, A

    mov R1, #Vm_eeg_pptr
    mov A, R4
    mov @R1, A
    ; }}}

    ; restore A, B, and registers {{{
    pop 04h ; R4
    pop 03h ; R3
    pop 02h ; R2
    pop 01h ; R1
    pop 00h ; R0
    pop DPL
```

```
        pop DPH
        pop B
        pop acc
        ; }}}

        reti
    ; }}}

F_ec_initeeg:   ; initialize EEG UART {{{
    ; reset state machine
    push 00h ; R0
    mov R0, #Vm_eeg_smst
    mov @R0, #V_ec_st_sync1
    pop 00h ; R0

    ; set line control register
    ;   0b10000011
    ;   set-dlab no-break stick-parity odd-parity
    ;   no-parity 1-stop-bit 8-bit-words[2]
    mov A, #083h
    mov DPTR, #Va_eeg_lcr
    movx @DPTR, A

    ; set divisor
    ;   1.842e6/16/9600 ~= 12
    mov A, #0Ch
    mov DPTR, #Va_eeg_dll
    movx @DPTR, A
    mov A, #000h
    mov DPTR, #Va_eeg_dlm
    movx @DPTR, A

    ; clear divisor latch
    mov A, #003h
    mov DPTR, #Va_eeg_lcr
    movx @DPTR, A

    ; set interrupts
    ;   enable the interrupt on received data available
    mov A, #001h
    mov DPTR, #Va_eeg_ier
    movx @DPTR, A

    setb EX0
    setb P3.2
    setb PX0    ; set high priority on the external interrupt
    ;setb IT0
    ; FIXME: don't forget to set EA

    ret
    ; }}}

F_ec_pld:   ; process payload {{{
    ; --- accounting ---
; R0 : current state     <- 0x
; R1 : captured byte     <- 0x
; R2 : packet length
; R3 : running checksum
; R4 : packet buffer pointer

; save A, B, and registers {{{
push acc
push B
push 00h     ; R0
push 01h     ; R1
push 02h     ; R2
; }}}

; this is cheating a bit -- I'm not following the proper payload
   processing
; guidelines because the mindset always outputs packets in the same form

mov R0, #Vm_eeg_pptr_d+001d      ; signal quality
mov R1, #Vm_eeg_sgnl
mov A, @R0
mov @R1, A

mov R0, #Vm_eeg_pptr_d+029d      ; attention
mov R1, #Vm_eeg_attn
mov A, @R0
mov @R1, A

mov R0, #Vm_eeg_pptr_d+031d      ; meditation
mov R1, #Vm_eeg_mdtn
mov A, @R0
mov @R1, A

; FFT values
; to capture these, make a memory block starting at Vm_eeg_dlta
; each value is 3 bytes long
; there are 8 values => 24 bytes
; delta, theta, loalpha, hialpha, lobeta, hibeta, logamma, mdgamma
mov R0, #Vm_eeg_pptr_d+004d
mov R1, #Vm_eeg_dlta
mov R2, #024d
F_ec_pld_fftloop:
    mov A, @R0
    mov @R1, A
    inc R0
    inc R1
    djnz R2, F_ec_pld_fftloop

F_ec_pld_done:
    ; restore A, B, and registers {{{
    pop 02h ; R2
    pop 01h ; R1
    pop 00h ; R0
    pop B
    pop acc
    ; }}}

    ret
; }}}
```

Appendix A: Cerebro Code, Equalizer Board Library, Joseph Colosimo

```
; vim: ts=4 sw=4 fdm=marker
;-----------------------------------------------------------------------------

; C E R E B R O
; J. Colosimo
; 6.115 Final Project
;
; EQUALIZER BOARD CONTROLLER
; prefix: eq
; enables scanning the equalizer and reading its values into a buffer
;-----------------------------------------------------------------------------



;-----------------------------------------------------------------------------

; NEEDED VALUES
;-----------------------------------------------------------------------------

; Va_eqb_mux, Va_eqb_adc, Vm_eqb_val

;-----------------------------------------------------------------------------

; CONSTANTS
;-----------------------------------------------------------------------------


F_eq_scan:          ; scan through equalizer {{{
    ; --- accounting ---
    ; R0 : pointer to buffer
    ; R1 : counter

    ; save A, B, and registers {{{
    push acc
    push B
    push DPH
    push DPL
    push 00h      ; R0
    push 01h      ; R1
    push 02h      ; R2
    ; }}}

    mov DPTR, #Va_eqb_adc     ; set DPTR to the ADC
    mov Va_eqb_mux, #0F0h     ; set first mux address
    mov R0, #Vm_eqb_vals      ; initialize to buffer start
    mov R1, #08h              ; counter
    F_eq_scan_loop:
        ; read value
        mov  DPTR, #Va_eqb_adc
        movx @DPTR, A    ; initiate conversion
        ; wait until conversion is done
        F_eq_scan_loop_wait: jb P3.3, F_eq_scan_loop_wait
        movx A, @DPTR    ; read result
        mov @R0, A        ; set buffer value
        inc R0            ; increment pointer
        inc Va_eqb_mux    ; go to next address on the mux

        mov R2, #0h
        F_eq_scan_loop_wait2: djnz R2, F_eq_scan_loop_wait2

        djnz R1, F_eq_scan_loop

    F_eq_scan_done:
        ; restore A, B, and registers {{{
        pop 02h ; R2
        pop 01h ; R1
        pop 00h ; R0
        pop DPL
        pop DPH
        pop B
        pop acc
        ; }}}

        ret
    ; }}}

F_eq_initeqb:    ; set up equalizer board {{{
    mov Va_eqb_mux, #0F0h
    ret
    ; }}}
```

Appendix A: Cerebro Code, Signal Processing Library, Joseph Colosimo

```
; vim: ts=4 sw=4 fdm=marker
;-------------------------------------------------------------------------------

; C E R E B R O
; J. Colosimo
; 6.115 Final Project
;
; SIGNAL PROCESSOR
; prefix: sp
; algorithms for signal processing
; NOTE: the output is in the Vm_led_argbuf buffer
;-------------------------------------------------------------------------------



;-------------------------------------------------------------------------------

; NEEDED VALUES
;-------------------------------------------------------------------------------

; Vm_eeg_sgnl, Vm_eeg_dlta, Vm_eeg_attn, Vm_eeg_mdtn
; Vm_led_argbuf

;-------------------------------------------------------------------------------

; CONSTANTS
;-------------------------------------------------------------------------------


;-------------------------------------------------------------------------------

; FUNCTIONS
;-------------------------------------------------------------------------------

F_sp_process:   ; main signal processing function {{{
    ; --- accounting ---
    ; R0 :
    ; R1 :
    ; R2 :
    ; R3 :
    ; R4 :
    ; R5 :
    ; R6 :
    ; R7 :

    ; save A, B, and registers {{{
    push acc
    push B
    push 00h    ; R0
    push 01h    ; R1
    push 02h    ; R2
    push 03h    ; R3
    push 04h    ; R4
    push 05h    ; R5
    push 06h    ; R6
    push 07h    ; R7
    ; }}}

    ; check the signal quality
    ; don't proceed unless the signal quality is perfect
    ; and otherwise, show a display based on signal quality
    ljmp F_sp_process_sigokay

    mov R0, #Vm_eeg_sgnl
    mov A, @R0
    jz F_sp_process_sigokay

    F_sp_process_sigbad:
        ; check to make sure that we don't have attention and meditation
           values
        ; before declaring the signal bad (we don't want to keep going back to
        ; bad signal mode if we can help it)
        mov R1, #Vm_eeg_attn
        mov A, @R1
        jnz F_sp_process_sigokay
        mov R1, #Vm_eeg_mdtn
        mov A, @R1
        jnz F_sp_process_sigokay

        ; okay, both of those values were 0, so the signal was bad
        lcall F_sp_sigbad
        ljmp F_sp_process_done

    F_sp_process_sigokay:
        lcall F_sp_sigokay
        ljmp F_sp_process_done

    F_sp_process_done:
        ; restore A, B, and registers {{{
        pop 07h ; R7
        pop 06h ; R6
        pop 05h ; R5
        pop 04h ; R4
        pop 03h ; R3
        pop 02h ; R2
        pop 01h ; R1
        pop 00h ; R0
        pop B
        pop acc
        ; }}}

        ret
    ; }}}

F_sp_sigbad:    ; display bar if signal is bad {{{
    ; --- accounting ---
    ; R0 : signal quality
; R1 : rgb pointer
; R2 : counter

; no register storage and restoration needed since it's only called by
; F_sp_process

mov R0, #Vm_eeg_sgnl
mov A, @R0
jz F_sp_sigbad_sensokay_end
mov R0, A

; if the signal value is 200, then the sensors aren't touching the skin
cjne A, #0200d, F_sp_sigbad_sensokay

F_sp_sigbad_sensbad:
    ; if the sensors are bad, set the whole panel red
    mov R0, #Vm_led_argbuf
    mov R1, #012d
    F_sp_sigbad_sensbad_loop:
        mov @R0, #0h
        inc R0
        djnz R1, F_sp_sigbad_sensbad_loop

    mov R0, #Vm_led_argbuf
    mov @R0, #030h
    mov R0, #Vm_led_argbuf+3
    mov @R0, #030h
    mov R0, #Vm_led_argbuf+6
    mov @R0, #030h
    mov R0, #Vm_led_argbuf+9
    mov @R0, #030h
    ret

F_sp_sigbad_sensokay:
    mov B, #020d    ; in practice, the signal quality varies between 0 and
                    ; 80, so divide among 4 pixels evenly
    div AB

    mov R2, B   ; temp
    mov B, A
    clr C
    mov A, #04d
    subb A, B   ; 4 - quotient gives signal quality low -> high
    mov R1, A   ; temp

    mov B, R2
    mov A, #020d
    clr C
    subb A, B   ; 20 - remainder gives sig quality remainder
    mov B, #04d
    mul AB      ; scale up

    mov B, A    ; B has remainder 0->79
    mov A, R1   ; A has quotient  0->4
```

```
        ; clear memory
        mov R2, #012d
        mov R1, #Vm_led_argbuf
        F_sp_sigbad_sensokay_memclear:
            mov @R1, #0h
            inc R1
            djnz R2, F_sp_sigbad_sensokay_memclear

        ; set panel
        F_sp_sigbad_sensokay_p0:
            mov R1, #Vm_led_argbuf+01d
            mov @R1, B
            cjne A, #00d, F_sp_sigbad_sensokay_p1
            ljmp F_sp_sigbad_sensokay_end

        F_sp_sigbad_sensokay_p1:
            mov R1, #Vm_led_argbuf+01d
            mov @R1, #080d
            mov R1, #Vm_led_argbuf+04d
            mov @R1, B
            cjne A, #01d, F_sp_sigbad_sensokay_p2
            ljmp F_sp_sigbad_sensokay_end

        F_sp_sigbad_sensokay_p2:
            mov R1, #Vm_led_argbuf+04d
            mov @R1, #080d
            mov R1, #Vm_led_argbuf+07d
            mov @R1, B
            cjne A, #02d, F_sp_sigbad_sensokay_p3
            ljmp F_sp_sigbad_sensokay_end

        F_sp_sigbad_sensokay_p3:
            mov R1, #Vm_led_argbuf+07d
            mov @R1, #080d
            mov R1, #Vm_led_argbuf+010d
            mov @R1, B
            cjne A, #03d, F_sp_sigbad_sensokay_p3
            ljmp F_sp_sigbad_sensokay_end

        F_sp_sigbad_sensokay_p4:
            mov R1, #Vm_led_argbuf+010d
            mov @R1, #080d
            ljmp F_sp_sigbad_sensokay_end

        F_sp_sigbad_sensokay_end:
            ret
    ; }}}

F_sp_sigokay:    ; signal processor {{{
    mov R0, #Vm_eqb_vals+2
    mov A, @R0

    F_sp_sigokay_check3:
        cjne A, #03h, F_sp_sigokay_check2
        lcall F_sp_algo3
        ljmp F_sp_sigokay_done

    F_sp_sigokay_check2:
        cjne A, #02h, F_sp_sigokay_check1
        lcall F_sp_algo2
        ljmp F_sp_sigokay_done

    F_sp_sigokay_check1:
        cjne A, #01h, F_sp_sigokay_check0
        lcall F_sp_algo1
        ljmp F_sp_sigokay_done

    F_sp_sigokay_check0:
        lcall F_sp_sigbad
        ljmp F_sp_sigokay_done

    F_sp_sigokay_done:
        ret
    ; }}}

F_sp_algo1: ; {{{
    mov R0, #Vm_eeg_sgnl
    mov A, @R0
    jnz F_sp_algo1_done

    ; set attention color {{{
    ; set attention hue
    ; we want it to vary from red -> green, which is about 0->85
    ; attention varies from 0->100 -- close enough
    mov B, #085d     ; 120/360x255 = 85
    mov R0, #Vm_eeg_attn
    mov A, @R0
    mov R3, A        ; R0 temporarily in R3
    ; set attention saturation
    mov R1, #0FFh
    ; set attention value
    ; take the delta, which is typically on the order of 10 and scale it
    mov R0, #Vm_eeg_attn
    mov A, @R0
    mov R0, #Vm_eeg_attn_lst
    mov B, @R0
    clr C
    subb A, B
    jnb OV, F_sp_sigokay_skipattncpl
        cpl A
        inc A
    F_sp_sigokay_skipattncpl:
    mov B, #04d
    mul AB
    mov R2, A

    mov A, R3
    mov R0, A         ; restore R0
    lcall F_cu_hsv2rgb        ; HSV -> RGB
    mov A, R0
    mov R3, A         ; R3 <- R0

    mov A, R3
    mov R0, #Vm_led_argbuf0
    mov @R0, A
    mov R0, #Vm_led_argbuf1
    mov @R0, A

    mov A, R1
    mov R0, #Vm_led_argbuf0+1
    mov @R0, A
    mov R0, #Vm_led_argbuf1+1
    mov @R0, A

    mov A, R2
    mov R0, #Vm_led_argbuf0+2
    mov @R0, A
    mov R0, #Vm_led_argbuf1+2
    mov @R0, A
    ;---------------------------------------- }}}

    ; set meditation color {{{
    ; set meditation hue
    ; we want it to vary from cyan -> purple, which is about 0+128->85+128
    mov B, #085d      ; 120/360x255 = 85
    mov R0, #Vm_eeg_mdtn
    mov A, @R0
    add A, #080h      ; add
    mov R3, A
    ; set meditation saturation
    mov R1, #0FFh
    ; set meditation value
    mov R0, #Vm_eeg_mdtn
    mov A, @R0
    mov R0, #Vm_eeg_mdtn_lst
    mov B, @R0
    clr C
    subb A, B
    jnb OV, F_sp_sigokay_skipmdtncpl
        cpl A
        inc A
    F_sp_sigokay_skipmdtncpl:
    mov B, #04d
    mul AB
    mov R2, A

    mov A, R3
    mov R0, A         ; restore R0
    lcall F_cu_hsv2rgb        ; HSV -> RGB
    mov A, R0
    mov R3, A         ; R3 <- R0
```

```
    mov A, R3
    mov R0, #Vm_led_argbuf2
    mov @R0, A
    mov R0, #Vm_led_argbuf3
    mov @R0, A

    mov A, R1
    mov R0, #Vm_led_argbuf2+1
    mov @R0, A
    mov R0, #Vm_led_argbuf3+1
    mov @R0, A

    mov A, R2
    mov R0, #Vm_led_argbuf2+2
    mov @R0, A
    mov R0, #Vm_led_argbuf3+2
    mov @R0, A
    ;---------------------------------------------------------- }}}

    F_sp_algo1_done:
    ret
    ; }}}

F_sp_algo2: ; {{{
    ; pixel 0
    mov R1, #0FFh             ; full saturation
    mov R0, #Vm_eeg_thta+1    ; 2nd byte of theta
    mov A, @R0
    clr C
    rrc A
    mov R2, A
    mov R0, #000d             ; red
    lcall F_cu_hsv2rgb

    mov A, R0   ; set red
    mov R0, #Vm_led_argbuf0
    mov @R0, A
    mov A, R1   ; set green
    inc R0
    mov @R0, A
    mov A, R2   ; set blue
    inc R0
    mov @R0, A
    ; ----

    ; pixel 1
    mov R1, #0FFh             ; full saturation
    mov R0, #Vm_eeg_lalp+1    ; 2nd byte of low alpha
    mov A, @R0
    clr C
    rrc A
    mov R2, A
    mov R0, #064d             ; green
```

```
lcall F_cu_hsv2rgb

mov A, R0    ; set red
mov R0, #Vm_led_argbuf1
mov @R0, A
mov A, R1    ; set green
inc R0
mov @R0, A
mov A, R2    ; set blue
inc R0
mov @R0, A
; ---

; pixel 2
mov R1, #0FFh              ; full saturation
mov R0, #Vm_eeg_lbet+1     ; 2nd byte of low beta
mov A, @R0
clr C
rrc A
mov R2, A
mov R0, #0128d             ; blue
lcall F_cu_hsv2rgb

mov A, R0    ; set red
mov R0, #Vm_led_argbuf2
mov @R0, A
mov A, R1    ; set green
inc R0
mov @R0, A
mov A, R2    ; set blue
inc R0
mov @R0, A
; ---

; pixel 3
mov R1, #0FFh              ; full saturation
mov R0, #Vm_eeg_lgam+1     ; 2nd byte of low gamma
mov A, @R0
clr C
rrc A
mov R2, A
mov R0, #0196d             ; purple
lcall F_cu_hsv2rgb

mov A, R0    ; set red
mov R0, #Vm_led_argbuf3
mov @R0, A
mov A, R1    ; set green
inc R0
mov @R0, A
mov A, R2    ; set blue
inc R0
mov @R0, A
; ---
```

```
    F_sp_algo2_done:
        ret
    ; }}}

F_sp_algo3:
    ; pixel 0 {{{
        ; hue
        mov R0, #Vm_eeg_dlta      ; !
        mov A, @R0
        mov R1, A
        mov R0, #Vm_eeg_thta+1    ; !
        mov A, @R0
        clr C
        subb A, R1                ; lowfreq - highfreq
        jc F_sp_algo3_p0morelft ; ! if lowfreq > highfreq, shift hue left

        F_sp_algo3_p0morergt:     ; !
            mov R7, #042d         ; !
            ljmp F_sp_algo3_p0hue ; !
        F_sp_algo3_p0morelft:     ; !
            mov R7, #0d           ; !
            ljmp F_sp_algo3_p0hue ; !

        F_sp_algo3_p0hue:         ; !
            mov R1, #Vm_eqb_vals+3
            mov B, @R1                ; B = ewma
            mov A, #0FFh
            clr C
            subb A, B                 ; A = 1-ewma

            mov R0, #Vm_ewma_p0hue   ; !
            mov B, @R0                ; B = oldhue
            mul AB
            mov R3, B                 ; R3 = (1-ewma)*oldhue

            mov B, @R1                ; B = ewma
            mov A, R7                 ; R7 is newhue
            mul AB                    ; B = (ewma)*newhue
            mov A, R3
            add A, B
            mov R3, A                 ; R3 = ewma*newhue + (1-ewma)*oldhue

        ; value
        V_sp_algo3_p0val:         ; !
            mov R1, #Vm_eqb_vals+4
            mov B, @R1                ; B = ewma
            mov A, #0FFh
            clr C
            subb A, B                 ; A = 1-ewma

            mov R0, #Vm_ewma_p0val   ; !
            mov B, @R0                ; B = oldval
            mul AB
        mov R4, B                   ; R4 = (1-ewma)*oldval

        mov R0, #Vm_eeg_dlta        ; !
        mov A, @R0
        clr C
        rrc A
        mov B, A
        mov R0, #Vm_eeg_thta+1      ; !
        mov A, @R0
        clr C
        rrc A
        add A, B                    ; A = newval (average)
        mov B, @R1                  ; B = ewma
        mul AB
        mov A, R4
        add A, B
        mov R4, A                   ; R4 = ewma*newval + (1-ewma)*oldval

    ; update old values
    mov R0, #Vm_ewma_p0hue          ; !
    mov A, R3
    mov @R0, A
    mov R0, #Vm_ewma_p0val          ; !
    mov A, R4
    mov @R0, A

    ; set everything
    mov A, R3
    mov R0, A
    mov R1, #0FFh
    mov A, R4
    mov R2, A

    lcall F_cu_hsv2rgb

    mov A, R0    ; set red
    mov R0, #Vm_led_argbuf0         ; !
    mov @R0, A
    mov A, R1    ; set green
    inc R0
    mov @R0, A
    mov A, R2    ; set blue
    inc R0
    mov @R0, A
; }}}

; pixel 1 {{{
    ; hue
    mov R0, #Vm_eeg_lalp+1    ; !
    mov A, @R0
    mov R1, A
    mov R0, #Vm_eeg_halp+1    ; !
    mov A, @R0
    clr C
```

```
subb A, R1                 ; lowfreq - highfreq
jc F_sp_algo3_p1morelft ; ! if lowfreq > highfreq, shift hue left

F_sp_algo3_p1morergt:    ; !
    mov R7, #084d        ; !
    ljmp F_sp_algo3_p1hue ; !
F_sp_algo3_p1morelft:    ; !
    mov R7, #042d        ; !
    ljmp F_sp_algo3_p1hue ; !

F_sp_algo3_p1hue:        ; !
    mov R1, #Vm_eqb_vals+3
    mov B, @R1               ; B = ewma
    mov A, #0FFh
    clr C
    subb A, B                ; A = 1-ewma
    mov R0, #Vm_ewma_p1hue   ; !
    mov B, @R0               ; B = oldhue
    mul AB
    mov R3, B                ; R3 = (1-ewma)*oldhue

    mov B, @R1               ; B = ewma
    mov A, R7                ; R7 is newhue
    mul AB                   ; B = (ewma)*newhue
    mov A, R3
    add A, B
    mov R3, A                ; R3 = ewma*newhue + (1-ewma)*oldhue

; value
V_sp_algo3_p1val:        ; !
    mov R1, #Vm_eqb_vals+5   ; !
    mov B, @R1               ; B = ewma
    mov A, #0FFh
    clr C
    subb A, B                ; A = 1-ewma

    mov R0, #Vm_ewma_p1val   ; !
    mov B, @R0               ; B = oldval
    mul AB
    mov R4, B                ; R4 = (1-ewma)*oldval

    mov R0, #Vm_eeg_lalp+1   ; !
    mov A, @R0
    clr C
    rrc A
    mov B, A
    mov R0, #Vm_eeg_halp+1   ; !
    mov A, @R0
    clr C
    rrc A
    add A, B                 ; A = newval (average)
    mov B, @R1               ; B = ewma
    mul AB
    mov A, R4
```

```
        add A, B
        mov R4, A                   ; R4 = ewma*newval + (1-ewma)*oldval

    ; update old values
    mov R0, #Vm_ewma_p1hue          ; !
    mov A, R3
    mov @R0, A
    mov R0, #Vm_ewma_p1val          ; !
    mov A, R4
    mov @R0, A

    ; set everything
    mov A, R3
    mov R0, A
    mov R1, #0FFh
    mov A, R4
    mov R2, A

    lcall F_cu_hsv2rgb

    mov A, R0    ; set red
    mov R0, #Vm_led_argbuf1         ; !
    mov @R0, A
    mov A, R1    ; set green
    inc R0
    mov @R0, A
    mov A, R2    ; set blue
    inc R0
    mov @R0, A
; }}}

; pixel 2 {{{
    ; hue
    mov R0, #Vm_eeg_lbet+1  ; !
    mov A, @R0
    mov R1, A
    mov R0, #Vm_eeg_hbet+1  ; !
    mov A, @R0
    clr C
    subb A, R1              ; lowfreq - highfreq
    jc F_sp_algo3_p2morelft ; ! if lowfreq > highfreq, shift hue left

    F_sp_algo3_p2morergt:   ; !
        mov R7, #0126d      ; !
        ljmp F_sp_algo3_p2hue ; !
    F_sp_algo3_p2morelft:   ; !
        mov R7, #084d       ; !
        ljmp F_sp_algo3_p2hue ; !

    F_sp_algo3_p2hue:       ; !
        mov R1, #Vm_eqb_vals+3
        mov B, @R1                  ; B = ewma
        mov A, #0FFh
        clr C
    subb A, B                ; A = 1-ewma
    mov R0, #Vm_ewma_p2hue   ; !
    mov B, @R0               ; B = oldhue
    mul AB
    mov R3, B                ; R3 = (1-ewma)*oldhue

    mov B, @R1               ; B = ewma
    mov A, R7                ; R7 is newhue
    mul AB                   ; B = (ewma)*newhue
    mov A, R3
    add A, B
    mov R3, A                ; R3 = ewma*newhue + (1-ewma)*oldhue

; value
V_sp_algo3_p2val:        ; !
    mov R1, #Vm_eqb_vals+6   ; !
    mov B, @R1               ; B = ewma
    mov A, #0FFh
    clr C
    subb A, B                ; A = 1-ewma

    mov R0, #Vm_ewma_p2val   ; !
    mov B, @R0               ; B = oldval
    mul AB
    mov R4, B                ; R4 = (1-ewma)*oldval

    mov R0, #Vm_eeg_lbet+1   ; !
    mov A, @R0
    clr C
    rrc A
    mov B, A
    mov R0, #Vm_eeg_hbet+1   ; !
    mov A, @R0
    clr C
    rrc A
    add A, B                 ; A = newval (average)
    mov B, @R1               ; B = ewma
    mul AB
    mov A, R4
    add A, B
    mov R4, A                ; R4 = ewma*newval + (1-ewma)*oldval

; update old values
mov R0, #Vm_ewma_p2hue       ; !
mov A, R3
mov @R0, A
mov R0, #Vm_ewma_p2val       ; !
mov A, R4
mov @R0, A

; set everything
mov A, R3
mov R0, A
mov R1, #0FFh
```

```
    mov A, R4
    mov R2, A

    lcall F_cu_hsv2rgb

    mov A, R0    ; set red
    mov R0, #Vm_led_argbuf2       ; !
    mov @R0, A
    mov A, R1    ; set green
    inc R0
    mov @R0, A
    mov A, R2    ; set blue
    inc R0
    mov @R0, A
; }}}

; pixel 3 {{{
    ; hue
    mov R0, #Vm_eeg_lgam+1   ; !
    mov @R0, A
    mov R1, A
    mov R0, #Vm_eeg_mgam+1   ; !
    mov @R0, A
    clr C
    rlc A
    clr C
    subb A, R1               ; lowfreq - highfreq
    jc F_sp_algo3_p3morelft  ; ! if lowfreq > highfreq, shift hue left

    F_sp_algo3_p3morergt:    ; !
        mov R7, #0168d       ; !
        ljmp F_sp_algo3_p3hue ; !
    F_sp_algo3_p3morelft:    ; !
        mov R7, #0126d       ; !
        ljmp F_sp_algo3_p3hue ; !

    F_sp_algo3_p3hue:        ; !
        mov R1, #Vm_eqb_vals+3
        mov B, @R1               ; B = ewma
        mov A, #0FFh
        clr C
        subb A, B                ; A = 1-ewma

        mov R0, #Vm_ewma_p3hue   ; !
        mov B, @R0               ; B = oldhue
        mul AB
        mov R3, B                ; R3 = (1-ewma)*oldhue
        mov A, B

        mov B, @R1               ; B = ewma
        mov A, R7                ; R7 is newhue
        mul AB                   ; B = (ewma)*newhue
        mov A, R3
        add A, B
    mov R3, A                   ; R3 = ewma*newhue + (1-ewma)*oldhue

; value
V_sp_algo3_p3val:         ; !
    mov R1, #Vm_eqb_vals+7  ; !
    mov B, @R1              ; B = ewma
    mov A, #0FFh
    clr C
    subb A, B               ; A = 1-ewma

    mov R0, #Vm_ewma_p3val  ; !
    mov B, @R0              ; B = oldval
    mul AB
    mov R4, B               ; R4 = (1-ewma)*oldval

    mov R0, #Vm_eeg_lgam+1  ; !
    mov A, @R0
    clr C
    rrc A
    mov B, A
    mov R0, #Vm_eeg_mgam+1  ; !
    mov A, @R0
    clr C
    rrc A
    add A, B                ; A = newval (average)
    mov B, @R1              ; B = ewma
    mul AB
    mov A, R4
    add A, B
    mov R4, A               ; R4 = ewma*newval + (1-ewma)*oldval

; update old values
mov R0, #Vm_ewma_p3hue      ; !
mov A, R3
mov @R0, A
mov R0, #Vm_ewma_p3val      ; !
mov A, R4
mov @R0, A

; set everything
mov A, R3
mov R0, A
mov R1, #0FFh
mov A, R4
mov R2, A

lcall F_cu_hsv2rgb

mov A, R0   ; set red
mov R0, #Vm_led_argbuf3     ; !
mov @R0, A
mov A, R1   ; set green
inc R0
mov @R0, A
```

```
    mov A, R2     ; set blue
    inc R0
    mov @R0, A
; }}}

F_sp_algo3_done:
    ret
```

Appendix A: Cerebro Code, Transition Library, Joseph Colosimo

```
; vim: ts=4 sw=4 fdm=marker
;------------------------------------------------------------------------------

; C E R E B R O
; J. Colosimo
; 6.115 Final Project
;
; TRANSITION HANDLER
; prefix: th
; handles transitions between two buffers
;------------------------------------------------------------------------------



;------------------------------------------------------------------------------

; CONSTANTS
;------------------------------------------------------------------------------


;------------------------------------------------------------------------------

; FUNCTIONS
;------------------------------------------------------------------------------

F_th_copy:   ; copy LED information to the free transition buffer {{{
    ; --- accounting ---
    ; R0 : buffer pointer
    ; R1 : LED args pointer
    ; R2 : loop counter

    ; save A, B, and registers {{{
    push acc
    push B
    push 00h     ; R0
    push 01h     ; R1
    push 02h     ; R2
    ; }}}

    mov R0, #Vm_led_rgbargs_cur
    mov A, @R0
    cjne A, #Vm_led_rgbargsB, F_th_copy_setB

    F_th_copy_setA:
        mov @R0, #Vm_led_rgbargsA
        ljmp F_th_copy_do

    F_th_copy_setB:
        mov @R0, #Vm_led_rgbargsB
        ljmp F_th_copy_do

    F_th_copy_do:
        mov A, @R0
        mov R0, A               ; R0 has address of current buffer
        mov R1, #Vm_led_argbuf  ; R1 has address of LED args
        mov R2, #012d
        F_th_copy_do_loop:
            mov A, @R1
            mov @R0, A
            inc R0
            inc R1
            djnz R2, F_th_copy_do_loop

    F_th_copy_done:
        ; restore A, B, and registers {{{
        pop 02h ; R2
        pop 01h ; R1
        pop 00h ; R0
        pop B
        pop acc
        ; }}}

        ret
    ; }}}

F_th_start: ; initiate transition {{{
    ; save registers {{{
    push 00h ; R0
    ; }}}

    ; reset crossfade
    mov R0, #Vm_led_xfade
    mov @R0, #0h

    ; enable timer
    setb TR0

    ; restore registers {{{
    pop 00h ; R0
    ; }}}

    ret
    ; }}}

F_th_int:   ; interrupt handler
    ; --- accounting ---
    ; R0 : start transition pointer
    ; R1 : end transition pointer
    ; R2 : LED rgb pointer
    ; R3 : crossfade value
    ; R4 : loop counter
    ; R5 : temp value storage
    ; R6 : transition speed

    ; save A, B, and registers {{{
    push acc
    push B
push 00h     ; R0
push 01h     ; R1
push 02h     ; R2
push 03h     ; R3
push 04h     ; R4
push 05h     ; R5
push 06h     ; R6
; }}}

; get some values from memory
mov R0, #Vm_led_xfade
mov A, @R0
mov R3, A

mov R0, #Vm_led_rgbargs_cur
mov A, @R0
cjne A, #Vm_led_rgbargsB, F_th_int_Bstarts

F_th_int_Astarts:
    mov R0, #Vm_led_rgbargsA
    mov R1, #Vm_led_rgbargsB
    ljmp F_th_int_do
F_th_int_Bstarts:
    mov R0, #Vm_led_rgbargsB
    mov R1, #Vm_led_rgbargsA
    ljmp F_th_int_do

F_th_int_do:
    mov R2, #Vm_led_rgbargs
    mov R4, #012d
    F_th_int_do_loop:
        mov A, #0FFh
        clr C
        subb A, R3
        mov B, A     ; B = 255-xfade
        mov A, @R0   ; A = start value
        mul AB       ; A*B
        mov R5, B    ; R5 holds result

        mov A, @R1   ; A = end value
        mov B, R3    ; B = xfade
        mul AB       ; A*B
        mov A, B     ; A holds result
        add A, R5    ; sum the two values
        mov B, A     ; store to B

        mov A, R2    ; A has output pointer
        xch A, R0    ; R0 now has output pointer
                     ; A has start pointer
        mov @R0, B   ; store output
        xch A, R0    ; A has output pointer
                     ; R0 has start pointer
        mov R2, A    ; R2 has output pointer
```

```
            inc R0      ; go to next start addr
            inc R1      ; go to next end addr
            inc R2      ; go to next output addr

            djnz R4, F_th_int_do_loop

    F_th_int_done:
        ; set panel
        lcall F_lp_setpixels
        ; increment xfade and store it to memory
        ; get increment amount
        mov R0, #Vm_eqb_vals+1
        mov A, @R0
        add A, R3
        ;mov A, R3
        mov R0, #Vm_led_xfade
        mov @R0, A
        ; if the result was 0, then the crossfader is done
        ; so stop the transition interrupt
        ; jnz F_th_int_done_skipstop
        jnc F_th_int_done_skipstop

        F_th_int_done_stop:
            mov @R0, #0FFh  ; set the crossfade to 255 so we stay at the last
                            ; color
            clr TR0         ; disable interrupt

        F_th_int_done_skipstop:
        ; restore A, B, and registers {{{
        pop 06h ; R6
        pop 05h ; R5
        pop 04h ; R4
        pop 03h ; R3
        pop 02h ; R2
        pop 01h ; R1
        pop 00h ; R0
        pop B
        pop acc
        ; }}}

        reti
    ; }}}

F_th_inittimer:
    mov TMOD, #022h
    mov TH0, #0128d
    setb ET0
    ret
```

Appendix A: Cerebro Code, LED Panel Controller Library, Joseph Colosimo

```
; vim: ts=4 sw=4 fdm=marker
;-----------------------------------------------------------------------------

; C E R E B R O
; J. Colosimo
; 6.115 Final Project
;
; LED PANEL UTILITIES
; prefix: lp
; functions for assembling and sending packets for the LED panel
;-----------------------------------------------------------------------------



;-----------------------------------------------------------------------------

; NEEDED VALUES
;-----------------------------------------------------------------------------

; UART:   Va_led_dll, Va_led_dlm, Va_led_ier, Va_led_lcr, Va_led_lsr,
    Va_led_thr
; Vm_led_rgbargs - base address for LED panel arguments

;-----------------------------------------------------------------------------

; CONSTANTS
;-----------------------------------------------------------------------------

V_lp_lpid equ 05h

V_lp_x_sync equ 0AAh
V_lp_x_fpanl equ 0252d
V_lp_x_fdisp equ 0247d

;-----------------------------------------------------------------------------

; FUNCTIONS
;-----------------------------------------------------------------------------

F_lp_setwholepanel:      ; set all 4 pixels to same color {{{
    ; args: R0:red, R1:grn, R2:blu
    ; --- accounting --- {{{
    ; R0 : red
    ; R1 : grn
    ; R2 : blu
    ; }}}

    ; save A, B, and registers {{{
    push acc
    ; }}}
    mov A, #V_lp_x_sync
    lcall F_lp_sendbyte      ; [sync]

    mov A, #V_lp_x_fpanl
    lcall F_lp_sendbyte      ; [fpanl]

    mov A, R0
    lcall F_lp_sendbyte      ; red

    mov A, R1
    lcall F_lp_sendbyte      ; green

    mov A, R2
    lcall F_lp_sendbyte      ; blue

    F_lp_setwholepanel_end:
        ; restore A, B, and registers {{{
        pop acc
        ; }}}

        ret
    ; }}}

F_lp_setpixels:          ; set each pixel individually {{{
    ; --- accounting --- {{{
    ; R0 : rgb array pointer
    ; R1 : gain correction
    ; R2 : arg counter
    ; }}}

    ; save A, B, and registers {{{
    push acc
    push 00h     ; R0
    push 01h     ; R1
    push 02h     ; R2
    ; }}}

    mov A, #V_lp_x_sync
    lcall F_lp_sendbyte      ; [sync]

    mov A, #V_lp_x_fdisp
    lcall F_lp_sendbyte      ; [fdisp]

    mov R1, #Vm_eqb_vals
    mov A, @R1
    mov R1, A

    mov R0, #Vm_led_rgbargs
    mov R2, #012d    ; 12 arguments
    F_lp_setpixels_argloop:
        mov A, @R0

        mov B, R1    ; apply gain correction
        mul AB
        mov A, B

        lcall F_lp_sendbyte
        inc R0
        djnz R2, F_lp_setpixels_argloop

    F_lp_setpixels_end:
        ; restore A, B, and registers {{{
        pop 02h ; R2
        pop 01h ; R1
        pop 00h ; R0
        pop acc
        ; }}}

       ret
    ; }}}

F_lp_initled:           ; initialize LED Panel UART {{{
    push acc

    ; set line control register
    ;   0b10011011
    ;   set-dlab no-break stick-parity even-parity
    ;   parity-enable 1-stop-bit 8-bit-words[2]
    mov A, #09Bh
    mov DPTR, #Va_led_lcr
    movx @DPTR, A

    ; set divisor
    ;   1.8432e6/16/38400 = 3
    mov A, #03h
    mov DPTR, #Va_led_dll
    movx @DPTR, A
    mov A, #000h
    mov DPTR, #Va_led_dlm
    movx @DPTR, A

    ; clear divisor latch
    mov A, #01Bh
    mov DPTR, #Va_led_lcr
    movx @DPTR, A

    ; disable interrupts
    ;mov A, #000h
    ;mov DPTR, #Va_led_ier
    ;movx @DPTR, A

    pop acc
    ret
    ; }}}

F_lp_sendbyte:          ; send a byte to the LED Panel UART {{{
    ; args: A:byte
    ; send byte to 16C450
```

```
push 0h ; R0
push DPH
push DPL

mov DPTR, #Va_led_thr
movx @DPTR, A

mov R0, #0h
; wait until it has been tranmitted
F_lp_sendbyte_wait:
    ;movx A, @DPTR
    ;jnz F_lp_sendbyte_wait
    djnz R0, F_lp_sendbyte_wait

pop DPL
pop DPH
pop 0h  ; R0
ret
; }}}
```

Appendix A: Cerebro Code, Serial Communication Library, Joseph Colosimo

```
; vim: ts=4 sw=4 fdm=marker
;------------------------------------------------------------------------------

; C E R E B R O
; J. Colosimo
; 6.115 Final Project
;
; SERIAL COMMUNICATIONS LIBRARY
; prefix: sc
; provides functions for communicating with a computer over serial
; some stuff taken from minmon
;------------------------------------------------------------------------------


;------------------------------------------------------------------------------

; FUNCTIONS
;------------------------------------------------------------------------------

F_sc_initserial:        ; initialize serial communication {{{
    mov  TMOD, #020h     ; sw-controlled gate, timer mode, 8-bit ar timer 1
    setb TR1
    ;mov  TH1, #0FFh      ; TH1 = 256 - 2x11.0592e6/384/57600 = 255 with PCON
        on
    ;orl  PCON, #080h     ; set PCON on
    mov TH1, #0FDh
    mov  SCON, #050h     ; enable 8-bit UART, REN on

    ret
    ; }}}

F_sc_tx:
    clr TI              ; clear the receiver interrupt so we can send data
    mov SBUF, A         ; push the last character we had stored (and stripped) out
                        ; to the serial buffer
    F_sc_tx_loop:
        jnb TI, F_sc_tx_loop    ; wait for the transmitter to finish
    ret
    ; }}}

F_sc_crlf:              ; send a CR, LF over serial {{{
    mov A, #010d        ; issue CR
    lcall F_sc_tx       ; send over serial
    mov A, #013d        ; issue LF
    lcall F_sc_tx       ; send that over serial too
    ret
    ; }}}

F_sc_prthex:
   push acc
   lcall F_sc_binasc          ; convert acc to ascii
   lcall F_sc_tx              ; print first ascii hex digit
   mov   a,  r2               ; get second ascii hex digit
   lcall F_sc_tx              ; print it
   pop acc
   ret

F_sc_binasc:
    mov   r2, a              ; save in r2
    anl   a,  #0fh           ; convert least sig digit.
    add   a,  #0f6h          ; adjust it
    jnc   F_sc_binasc_noadj1 ; if a-f then readjust
    add   a,  #07h
    F_sc_binasc_noadj1:
        add   a,  #3ah           ; make ascii
        xch   a,  r2             ; put result in reg 2
        swap  a                  ; convert most sig digit
        anl   a,  #0fh           ; look at least sig half of acc
        add   a,  #0f6h          ; adjust it
        jnc   F_sc_binasc_noadj2     ; if a-f then re-adjust
        add   a,  #07h
    F_sc_binasc_noadj2:
        add   a,  #3ah           ; make ascii
        ret
```

Appendix A: Cerebro Code, Color Utilities, Joseph Colosimo

```
; vim: ts=4 sw=4 fdm=marker
;------------------------------------------------------------------------------

; C E R E B R O
; J. Colosimo
; 6.115 Final Project
;
; COLOR UTILITIES
; prefix: cu
; algorithms for processing color information
;------------------------------------------------------------------------------



;------------------------------------------------------------------------------

; CONSTANTS
;------------------------------------------------------------------------------


;------------------------------------------------------------------------------

; FUNCTIONS
;------------------------------------------------------------------------------


F_cu_hsv2rgb:    ; converts hsv values into rgb {{{
    ; args: R0:hue, R1:sat, R2:val
    ; out : R0:red, R1:grn, R2:blu
    ; --- accounting --- {{{
    ; R0 : hue -> red
    ; R1 : sat -> grn
    ; R2 : val -> blu
    ;
    ; R3 : h*
    ; R4 : f
    ; R5 : p
    ; R6 : q
    ; R7 : t
    ; }}}

    ; save A, B, and registers {{{
    push acc
    push B
    push 03h     ; R3
    push 04h     ; R4
    push 05h     ; R5
    push 06h     ; R6
    push 07h     ; R7
    ; }}}

    ; notes:
    ;   - hue ranges from 0 -> 255 and wraps around
    ;     that means that each increment in value corresponds to a shift in
;      360/255 = 1.412 degrees
;    - sat and val both range from 0 -> 255
;    - this is not a very accurate calculation -- you need more bits of
;      precision to get an accurate calculation. However, it's very close
;      (running a simple program that cycles through hues proved this) and
;      therefore fine for this application.

; first , we want h*, the hex hue chunk of a value
;    60 / (360/255) = 42.5, so each hexagonal hue chunk is roughly spaced
;    42.5 vals away. okay, so that's a bit annoying, but we can make some
;    spaced at 42 and others at 43 as follows:
;    |   42   |   43   |   42   |   43   |   42   |   43   |
;    |   0    |   1    |   2    |   3    |   4    |   5    |
;    |  0: 41| 42: 84| 85:126|127:169|170:211|212:255|

mov A, R0
mov B, #042d
div AB

cjne A, #06d, F_cu_hsv2rgb_continue
    mov R3, 0
    mov R4, 0
    sjmp F_cu_hsv2rgb_skip
F_cu_hsv2rgb_continue:
    ; the quotient is the unfair h*
    mov R3, A
    ; the remainder is called "f"
    mov R4, B

F_cu_hsv2rgb_skip:
; now to even things out a bit, look at unfair h* and f

; next compute p = v * ( 1 - s )
mov A, #0FFh
clr C
subb A, R1        ; A = 255 - s
mov B, R2         ; B = val
mul AB            ; p = val * (255 - s)
mov R5, B         ; p = top byte (range shift 256*256 -> 256)

; and q = v * ( 1 - f*s )
mov A, R4         ; A = f
mov B, R1         ; B = s
mul AB            ; compute f*s   (range 0->255*41)
;------------- TODO: subroutine-ify
; now we need a range shift from 0->255*41 to 0->255
; we do this by multiplying the value by 255/42 = 6 and taking the top
   byte
push 00h ; R0
push 01h ; R1

mov R0, B         ; save high byte of multiplication
mov B, #06d
mul AB            ; multiply low byte
```

```
mov R1, B         ; we only care about high byte of that result
mov A, R0
mov B, #06d
mul AB            ; multiply previous high byte by 6
add A, R1         ; add low byte of the result to the high byte of the
    previous result

pop 01h ; R1
pop 00h ; R0
;---------------
mov B, A          ; move quotient to B
mov A, #0FFh
clr C
subb A, B         ; A = 255 - f*s
mov B, R2         ; B = val
mul AB            ; q = val * (255 - f*s)
mov R6, B         ; q = top byte

; and finally t = v * ( 1 - ( 1 - f ) * s )
mov A, #042d
clr C
subb A, R4        ; A = 42 - f
mov B, R1         ; B = sat
mul AB            ; AB = (42-f)*s
;--------------- TODO: subroutine-ify
; now we need a range shift from 0->255*41 to 0->255
; we do this by multiplying the value by 255/42 = 6 and taking the top
    byte
push 00h ; R0
push 01h ; R1

mov R0, B         ; save high byte of multiplication
mov B, #06d
mul AB            ; multiply low byte
mov R1, B         ; we only care about high byte of that result
mov A, R0
mov B, #06d
mul AB            ; multiply previous high byte by 6
add A, R1         ; add low byte of the result to the high byte of the
    previous result

pop 01h ; R1
pop 00h ; R0
;---------------
mov B, A          ; move quotient to B
mov A, #0FFh
clr C
subb A, B         ; A = 255 - (42-f)*s
mov B, R2         ; B = val
mul AB            ; t = val * (255 - (42-f)*s)
mov R7, B         ; t = top byte

; lastly, figure out how to set rgb based on the value of h*:
F_cu_hsv2rgb_h0:      ; rgb = vtp
```

```
    cjne R3, #00h, F_cu_hsv2rgb_h1
    mov A, R2          ; v
    mov R0, A
    mov A, R7          ; t
    mov R1, A
    mov A, R5          ; p
    mov R2, A
    ljmp F_cu_hsv2rgb_end

F_cu_hsv2rgb_h1:       ; rgb = qvp
    cjne R3, #01h, F_cu_hsv2rgb_h2
    mov A, R6          ; q
    mov R0, A
    mov A, R2          ; v
    mov R1, A
    mov A, R5          ; p
    mov R2, A
    ljmp F_cu_hsv2rgb_end

F_cu_hsv2rgb_h2:       ; rgb = pvt
    cjne R3, #02h, F_cu_hsv2rgb_h3
    mov A, R5          ; p
    mov R0, A
    mov A, R2          ; v
    mov R1, A
    mov A, R7          ; t
    mov R2, A
    ljmp F_cu_hsv2rgb_end

F_cu_hsv2rgb_h3:       ; rgb = pqv
    cjne R3, #03h, F_cu_hsv2rgb_h4
    mov A, R5          ; p
    mov R0, A
    mov A, R6          ; q
    mov R1, A
    mov A, R2          ; v
    mov R2, A
    ljmp F_cu_hsv2rgb_end

F_cu_hsv2rgb_h4:       ; rgb = tpv
    cjne R3, #04h, F_cu_hsv2rgb_h5
    mov A, R7          ; t
    mov R0, A
    mov A, R5          ; p
    mov R1, A
    mov A, R2          ; v
    mov R2, A
    ljmp F_cu_hsv2rgb_end

F_cu_hsv2rgb_h5:       ; rgb = vpq
    mov A, R2          ; v
    mov R0, A
    mov A, R5          ; p
    mov R1, A
    mov A, R6          ; q
    mov R2, A
    ljmp F_cu_hsv2rgb_end

F_cu_hsv2rgb_end:
    ; restore A, B, and registers {{{
    pop 07h       ; R7
    pop 06h       ; R6
    pop 05h       ; R5
    pop 04h       ; R4
    pop 03h       ; R3
    pop B
    pop acc
    ; }}}

    ret
; }}}
```

Appendix A: Cerebro Code, EEG Data Capture Utility, Joseph Colosimo

```
;------------------------------------------------------------------------------

; C E R E B R O
; J. Colosimo
; 6.115 Final Project
;
; eegtest.asm
;
; test file that directly forwards data from the EEG chip to the serial port
;------------------------------------------------------------------------------


;------------------------------------------------------------------------------

; CONSTANTS
;------------------------------------------------------------------------------

V_addr_eeg equ 0FE00h
V_addr_eeg_rxb equ (V_addr_eeg + 00h)    ; receiver buffer reg (dlab = 0)
V_addr_eeg_ier equ (V_addr_eeg + 01h)    ; interrupt enable reg (dlab = 0)
V_addr_eeg_dll equ (V_addr_eeg + 00h)    ; divisor latch (lsb) (dlab = 1)
V_addr_eeg_dlm equ (V_addr_eeg + 01h)    ; divisor latch (msb) (dlab = 1)
V_addr_eeg_lcr equ (V_addr_eeg + 03h)    ; line control reg
V_addr_eeg_lsr equ (V_addr_eeg + 05h)    ; line status reg

;------------------------------------------------------------------------------

; INCLUDES
;------------------------------------------------------------------------------


org 00h
ljmp main

;------------------------------------------------------------------------------

; INTERRUPTS
;------------------------------------------------------------------------------


;------------------------------------------------------------------------------

; MAIN LOOP
;------------------------------------------------------------------------------

org 0100h
main:
    lcall F_init_serial
    lcall F_init_eeg

    ; test
    ;Lloop:
    ;     mov A, #'T'
    ;     lcall F_ser_tx
    ;     sjmp Lloop

    L_main_loop:
        mov DPTR, #V_addr_eeg_lsr    ; check line status reg
        movx A, @DPTR
        anl A, #01h                  ; bit 0 is data ready
        jz L_main_loop

        mov DPTR, #V_addr_eeg_rxb    ; read byte
        movx A, @DPTR
        lcall F_ser_tx               ; send it over serial

        sjmp L_main_loop

;-------------------------------------------------------------------------------

; FUNCTIONS
;-------------------------------------------------------------------------------

F_init_eeg:      ; initialize EEG UART {{{
    ; set line control register
    ;   0b10000011
    ;   set-dlab no-break stick-parity odd-parity
    ;   no-parity 1-stop-bit 8-bit-words[2]
    mov A, #083h
    mov DPTR, #V_addr_eeg_lcr
    movx @DPTR, A

    ; set divisor
    ;   1.842e6/16/9600 ~= 12
    mov A, #0Ch
    mov DPTR, #V_addr_eeg_dll
    movx @DPTR, A
    mov A, #000h
    mov DPTR, #V_addr_eeg_dlm
    movx @DPTR, A

    ; clear divisor latch
    mov A, #003h
    mov DPTR, #V_addr_eeg_lcr
    movx @DPTR, A

    ; set interrupts
    ;   disable all for now, but eventually switch to using 1 since that will
    ;   enable the interrupt on received data available
    mov A, #000h     ; TODO: change to #001h
    mov DPTR, #V_addr_eeg_ier
    movx @DPTR, A

    ; setb EX0
    ; setb EA
    ret
    ; }}}

F_init_serial:        ; initialize serial communication {{{
    mov  TMOD, #020h      ; sw-controlled gate, timer mode, 8-bit ar timer 1
    setb TR1
    ;mov  TH1, #0FFh       ; TH1 = 256 - 2x11.0592e6/384/57600 = 255 with PCON
        on
    ;orl  PCON, #080h      ; set PCON on
    mov TH1, #0FDh
    mov  SCON, #050h      ; enable 8-bit UART, REN on

    ret
    ; }}}

F_ser_tx:             ; send a character over serial {{{
    clr TI            ; clear the receiver interrupt so we can send data
    mov SBUF, A       ; push the last character we had stored (and stripped) out
                      ; to the serial buffer
    L_ser_tx_loop:
        jnb TI, L_ser_tx_loop   ; wait for the transmitter to finish
    ret
    ; }}}

F_ser_crlf:           ; send a CR, LF over serial {{{
    mov A, #010d      ; issue CR
    lcall F_ser_tx    ; send over serial
    mov A, #013d      ; issue LF
    lcall F_ser_tx    ; send that over serial too
    ret
    ; }}}

; --- --- ---
; vim: et ts=4 sw=4 fdm=marker
```

Appendix A: Cerebro Code, Payload Test, Joseph Colosimo

```
; vim: ts=4 sw=4 fdm=marker
;————————————————————————————————————————————————————————

; C E R E B R O
; J. Colosimo
; 6.115 Final Project
;
; EEG Payload Capture Test
; Gets payload, does basic processing, prints to serial
;————————————————————————————————————————————————————————


;————————————————————————————————————————————————————————

; CONSTANTS
;————————————————————————————————————————————————————————

Va_eeg equ 0FE00h
Va_eeg_rxb equ (Va_eeg + 00h)   ; receiver buffer reg (dlab = 0)
Va_eeg_ier equ (Va_eeg + 01h)   ; interrupt enable reg (dlab = 0)
Va_eeg_dll equ (Va_eeg + 00h)   ; divisor latch (lsb) (dlab = 1)
Va_eeg_dlm equ (Va_eeg + 01h)   ; divisor latch (msb) (dlab = 1)
Va_eeg_lcr equ (Va_eeg + 03h)   ; line control reg
Va_eeg_lsr equ (Va_eeg + 05h)   ; line status reg

Vm_eeg_pptr_d equ 050h  ; 0x50 -> 0x50+32=0x6F
Vm_eeg_smst   equ 030h
Vm_eeg_plen   equ 031h
Vm_eeg_pptr   equ 032h
Vm_eeg_csum   equ 033h
Vm_eeg_sgnl   equ 034h
Vm_eeg_dlta   equ 038h  ; 0x38 -> 0x38+23=0x4F
Vm_eeg_attn   equ 035h
Vm_eeg_mdtn   equ 036h
Vm_eeg_drdy   equ 037h
Vm_eeg_csum_got equ 072h

;Vm_eeg_iptr    equ 070h
;Vm_eeg_optr    equ 071h

org 00h
ljmp main

;————————————————————————————————————————————————————————

; INTERRUPTS
;————————————————————————————————————————————————————————

org 03h
    ljmp F_ec_int
;    implementation of a circular buffer
;     TODO: add buffer counter
;     push acc
;     push dph
;     push dpl
;     push 00h ; R0
;
;     mov R0, #Vm_eeg_iptr
;     mov A, @R0               ; get iptr
;     mov R0, A                ; R0 <- iptr
;
;     mov DPTR, #Va_eeg_rxb
;     movx A, @DPTR            ; get byte from serial
;
;     mov @R0, A               ; store byte to @iptr
;
;     inc R0                   ; increment pointer
;     cjne R0, #Vm_eeg_buf+064d, IE0_end  ; if we hit the end of the buffer...
;         mov R0, #Vm_eeg_buf ; then wrap
;
;     IE0_end:
;         mov A, R0
;         mov R0, #Vm_eeg_iptr
;         mov @R0, A
;         pop 00h ; R0
;         pop dpl
;         pop dph
;         pop acc
;
;     reti

;------------------------------------------------------------------------------

; MAIN LOOP
;------------------------------------------------------------------------------

org 0100h
main:
    lcall F_sc_initserial
    lcall F_ec_initeeg
    setb EA

    mov R0, #Vm_eeg_drdy
    mov @R0, #00h

    L_main_loop:
        mov R0, #Vm_eeg_drdy
        mov A, @R0
        jz L_main_loop    ; is value ready?

        ; print some crap
        mov R0, #Vm_eeg_sgnl     ; signal quality
        mov A, @R0
        lcall F_sc_prthex

            mov A, #' '
            lcall F_sc_tx
        mov R0, #Vm_eeg_attn      ; attention
        mov A, @R0
        lcall F_sc_prthex

            mov A, #' '
            lcall F_sc_tx

        mov R0, #Vm_eeg_mdtn      ; meditation
        mov A, @R0
        lcall F_sc_prthex

            mov A, #' '
            lcall F_sc_tx

        mov R0, #Vm_eeg_dlta      ; FFT values values
        mov R1, #024d
        L_main_loop_fft:
            mov A, #' '
            lcall F_sc_tx
            mov A, @R0
            inc R0
            lcall F_sc_prthex
            djnz R1, L_main_loop_fft

        lcall F_sc_crlf

        ; reset drdy
        mov R0, #Vm_eeg_drdy
        mov @R0, #00h

        sjmp L_main_loop

;————————————————————————————————————————————————————————————————

; FUNCTIONS
;————————————————————————————————————————————————————————————————


;————————————————————————————————————————————————————————————————

; INCLUDES
;————————————————————————————————————————————————————————————————

#include eegctrl.asm
#include sercom.asm
```

Appendix A: Cerebro Code, LED Panel Test 1, Joseph Colosimo

```
; vim: ts=4 sw=4 fdm=marker
;------------------------------------------------------------------------------

; C E R E B R O
; J. Colosimo
; 6.115 Final Project
;
; LED Panel Serial Output Test
; test file that outputs a color pattern to the LED panel
;------------------------------------------------------------------------------



;------------------------------------------------------------------------------

; CONSTANTS
;------------------------------------------------------------------------------

V_addr_led equ 0FE20h
V_addr_led_thr equ (V_addr_led + 00h)    ; transmitter buffer reg (dlab = 0)
V_addr_led_ier equ (V_addr_led + 01h)    ; interrupt enable reg (dlab = 0)
V_addr_led_dll equ (V_addr_led + 00h)    ; divisor latch (lsb) (dlab = 1)
V_addr_led_dlm equ (V_addr_led + 01h)    ; divisor latch (msb) (dlab = 1)
V_addr_led_lcr equ (V_addr_led + 03h)    ; line control reg
V_addr_led_lsr equ (V_addr_led + 05h)    ; line status reg

;------------------------------------------------------------------------------

; INCLUDES
;------------------------------------------------------------------------------


org 00h
ljmp main

;------------------------------------------------------------------------------

; INTERRUPTS
;------------------------------------------------------------------------------


;------------------------------------------------------------------------------

; MAIN LOOP
;------------------------------------------------------------------------------

org 0100h
main:
    lcall F_init_led

    L_main_loop:
        mov DPTR, #V_addr_led_thr
        mov A, #0aah
        movx @DPTR, A

            Lwait0:
                movx A, @DPTR
                jnz A, Lwait0

        mov A, #0252d
        movx @DPTR, A

            Lwait1:
                movx A, @DPTR
                jnz A, Lwait1

        mov A, #080d
        movx @DPTR, A

            Lwait2:
                movx A, @DPTR
                jnz A, Lwait2

        mov A, #010d
        movx @DPTR, A

            Lwait3:
                movx A, @DPTR
                jnz A, Lwait3

        mov A, #040d
        movx @DPTR, A

        mov R0, #0h
        mov R1, #0h
        LwaitA:
            LwaitB:
                ;djnz R0, LwaitB
            djnz R1, LwaitA

        sjmp L_main_loop

;------------------------------------------------------------------------------

; FUNCTIONS
;------------------------------------------------------------------------------

F_init_led:     ; initialize LED Panel UART {{{
    ; set line control register
    ;   0b10011011
    ;   set-dlab no-break stick-parity even-parity
    ;   parity-enable 1-stop-bit 8-bit-words[2]
    mov A, #09Bh
    mov DPTR, #V_addr_led_lcr
    movx @DPTR, A
    ; set divisor
    ;    1.8432e6/16/38400 = 3
    mov A, #03h
    mov DPTR, #V_addr_led_dll
    movx @DPTR, A
    mov A, #000h
    mov DPTR, #V_addr_led_dlm
    movx @DPTR, A

    ; clear divisor latch
    mov A, #01Bh
    mov DPTR, #V_addr_led_lcr
    movx @DPTR, A

    ; disable interrupts
    ;mov A, #000h
    ;mov DPTR, #V_addr_led_ier
    ;movx @DPTR, A

    ret
    ; }}}

; --- --- ---
; vim: et ts=4 sw=4 fdm=marker
```

Appendix A: Cerebro Code, LED Panel Test 2, Joseph Colosimo

```
; vim: ts=4 sw=4 fdm=marker
;------------------------------------------------------------------------------

; C E R E B R O
; J. Colosimo
; 6.115 Final Project
;
; LED Panel Serial Output Test
; test file that outputs a color pattern to the LED panel
;------------------------------------------------------------------------------



;------------------------------------------------------------------------------

; CONSTANTS
;------------------------------------------------------------------------------

V_addr_led equ 0FE20h
V_addr_led_thr equ (V_addr_led + 00h)    ; transmitter buffer reg (dlab = 0)
V_addr_led_ier equ (V_addr_led + 01h)    ; interrupt enable reg (dlab = 0)
V_addr_led_dll equ (V_addr_led + 00h)    ; divisor latch (lsb) (dlab = 1)
V_addr_led_dlm equ (V_addr_led + 01h)    ; divisor latch (msb) (dlab = 1)
V_addr_led_lcr equ (V_addr_led + 03h)    ; line control reg
V_addr_led_lsr equ (V_addr_led + 05h)    ; line status reg

V_addr_eeg equ 0FE00h
V_addr_eeg_rxb equ (V_addr_eeg + 00h)    ; receiver buffer reg (dlab = 0)
V_addr_eeg_ier equ (V_addr_eeg + 01h)    ; interrupt enable reg (dlab = 0)
V_addr_eeg_dll equ (V_addr_eeg + 00h)    ; divisor latch (lsb) (dlab = 1)
V_addr_eeg_dlm equ (V_addr_eeg + 01h)    ; divisor latch (msb) (dlab = 1)
V_addr_eeg_lcr equ (V_addr_eeg + 03h)    ; line control reg
V_addr_eeg_lsr equ (V_addr_eeg + 05h)    ; line status reg

;------------------------------------------------------------------------------

; INCLUDES
;------------------------------------------------------------------------------


org 00h
ljmp main

;------------------------------------------------------------------------------

; INTERRUPTS
;------------------------------------------------------------------------------


;------------------------------------------------------------------------------

; MAIN LOOP
;------------------------------------------------------------------------------
```

```
org 0100h
main:
    lcall F_init_led
    lcall F_init_eeg
    lcall F_init_serial

    L_main_loop:
        mov DPTR, #V_addr_led_thr
        mov A, #'a'
        movx @DPTR, A

        Lwait:
            mov DPTR, #V_addr_eeg_lsr    ; check line status reg
            movx A, @DPTR
            anl A, #01h                  ; bit 0 is data ready
            jz Lwait

            mov DPTR, #V_addr_eeg_rxb    ; read byte
            movx A, @DPTR
            lcall F_ser_tx               ; send it over serial

        ; pause
        Lwait1:
            Lwait2:
                ;djnz R0, Lwait2
            djnz R1, Lwait1

        sjmp L_main_loop

;------------------------------------------------------------------------------

; FUNCTIONS
;------------------------------------------------------------------------------

F_init_led:        ; initialize LED Panel UART {{{
    ; set line control register
    ;   0b10011011
    ;   set-dlab no-break stick-parity even-parity
    ;   parity-enable 1-stop-bit 8-bit-words[2]
    mov A, #083h
    mov DPTR, #V_addr_led_lcr
    movx @DPTR, A

    ; set divisor
    ;   2e6/16/38400 ~= 3, but with a 7% error rate -- maybe too high
    ;mov A, #03h
    mov A, #03h
    mov DPTR, #V_addr_led_dll
    movx @DPTR, A
    mov A, #000h
    mov DPTR, #V_addr_led_dlm
    movx @DPTR, A
    ; clear divisor latch
    mov A, #003h
    mov DPTR, #V_addr_led_lcr
    movx @DPTR, A

    ; disable interrupts
    ;mov A, #000h
    ;mov DPTR, #V_addr_led_ier
    ;movx @DPTR, A

    ret
    ; }}}

F_init_eeg:      ; initialize EEG UART {{{
    ; set line control register
    ;   0b10000011
    ;   set-dlab no-break stick-parity odd-parity
    ;   no-parity 1-stop-bit 8-bit-words[2]
    mov A, #083h
    mov DPTR, #V_addr_eeg_lcr
    movx @DPTR, A

    ; set divisor
    ;   2e6/16/9600 ~= 13
    mov A, #03h
    mov DPTR, #V_addr_eeg_dll
    movx @DPTR, A
    mov A, #000h
    mov DPTR, #V_addr_eeg_dlm
    movx @DPTR, A

    ; clear divisor latch
    mov A, #003h
    mov DPTR, #V_addr_eeg_lcr
    movx @DPTR, A

    ; set interrupts
    ;   disable all for now, but eventually switch to using 1 since that will
    ;   enable the interrupt on received data available
    mov A, #000h     ; TODO: change to #001h
    mov DPTR, #V_addr_eeg_ier
    movx @DPTR, A

    ; setb EX0
    ; setb EA

    ret
    ; }}}

F_init_serial:       ; initialize serial communication {{{
    mov  TMOD, #020h      ; sw-controlled gate, timer mode, 8-bit ar timer 1
    setb TR1
    ;mov  TH1, #0FFh       ; TH1 = 256 - 2x11.0592e6/384/57600 = 255 with PCON
       on
    ;orl  PCON, #080h      ; set PCON on
    mov TH1, #0FDh
    mov  SCON, #050h      ; enable 8-bit UART, REN on

    ret
    ; }}}

F_ser_tx:              ; send a character over serial {{{
    clr TI             ; clear the receiver interrupt so we can send data
    mov SBUF, A        ; push the last character we had stored (and stripped) out
                       ; to the serial buffer
    L_ser_tx_loop:
        jnb TI, L_ser_tx_loop    ; wait for the transmitter to finish
    ret
    ; }}}

F_ser_crlf:            ; send a CR, LF over serial {{{
    mov A, #010d       ; issue CR
    lcall F_ser_tx     ; send over serial
    mov A, #013d       ; issue LF
    lcall F_ser_tx     ; send that over serial too
    ret
    ; }}}


; --- --- ---
; vim: et ts=4 sw=4 fdm=marker
```

Appendix A: Cerebro Code, LED Panel Test 3, Joseph Colosimo

```
; vim: ts=4 sw=4 fdm=marker
;------------------------------------------------------------------------------

; C E R E B R O
; J. Colosimo
; 6.115 Final Project
;
; LED Panel Color Shifting Test
; shows a hue shifting output on the LED panel
;------------------------------------------------------------------------------



;------------------------------------------------------------------------------

; CONSTANTS
;------------------------------------------------------------------------------

Va_led equ 0FE20h
Va_led_thr equ (Va_led + 00h)   ; transmitter buffer reg (dlab = 0)
Va_led_ier equ (Va_led + 01h)   ; interrupt enable reg (dlab = 0)
Va_led_dll equ (Va_led + 00h)   ; divisor latch (lsb) (dlab = 1)
Va_led_dlm equ (Va_led + 01h)   ; divisor latch (msb) (dlab = 1)
Va_led_lcr equ (Va_led + 03h)   ; line control reg
Va_led_lsr equ (Va_led + 05h)   ; line status reg

org 00h
ljmp main

;------------------------------------------------------------------------------

; INTERRUPTS
;------------------------------------------------------------------------------


;------------------------------------------------------------------------------

; MAIN LOOP
;------------------------------------------------------------------------------

org 0100h
main:
    lcall F_lp_initled

    mov R3, #020h
    mov R4, #0FFh
    mov R5, #030h

    L_main_loop:
        ; set stuff manually {{{
        ;mov A, #0aah
        ;lcall F_lp_sendbyte
        ;mov A, #0252d
        ;lcall F_lp_sendbyte

        ;mov A, #080d
        ;lcall F_lp_sendbyte

        ;mov A, #010d
        ;lcall F_lp_sendbyte

        ;mov A, #040d
        ;lcall F_lp_sendbyte
        ; }}}

        mov A, R3
        mov R0, A

        mov A, R4
        mov R1, A

        mov A, R5
        mov R2, A

        lcall F_cu_hsv2rgb
        lcall F_lp_setwholepanel

        ;mov A, R3
        ;add A, #05h
        ;mov R3, A
        inc R3

        ; -- pause and loop --
        mov R6, #0h
        mov R7, #0h
        LwaitA:
            LwaitB:
                ;djnz R6, LwaitB
            djnz R7, LwaitA

        ljmp L_main_loop

;------------------------------------------------------------------------------
; FUNCTIONS
;------------------------------------------------------------------------------


;------------------------------------------------------------------------------
; INCLUDES
;------------------------------------------------------------------------------

#include colutils.asm
#include ledpanel.asm
```

```
; --- --- ---
; vim: et ts=4 sw=4 fdm=marker
```

Appendix A: Cerebro Code, Equalizer Board Test, Joseph Colosimo

```
; vim: ts=4 sw=4 fdm=marker
;---------------------------------------------------------------------------

; C E R E B R O
; J. Colosimo
; 6.115 Final Project
;
; equalizer test
; tests the functionality of the equalizer board by scanning through the
    values
; and printing out the results.
;---------------------------------------------------------------------------


;---------------------------------------------------------------------------

; CONSTANTS
;---------------------------------------------------------------------------

Va_eqb_mux equ P1
Va_eqb_adc equ 0FE10h

org 00h
ljmp main

;---------------------------------------------------------------------------

; MAIN LOOP
;---------------------------------------------------------------------------

org 0100h
main:
    lcall F_sc_initserial
    setb EA
    mov DPTR, #Va_eqb_adc     ; set DPTR to the ADC

    L_main_loop:
    mov Va_eqb_mux, #0F0h     ; set first mux address
    mov R0, #08d              ; counter
    F_eq_scan_loop:
        ; read value
        mov  DPTR, #Va_eqb_adc
        movx @DPTR, A    ; initiate conversion
        ; wait until conversion is done
        F_eq_scan_loop_wait: jb P3.3, F_eq_scan_loop_wait
        movx A, @DPTR    ; read result

        lcall F_sc_prthex

        mov A, #' '
        lcall F_sc_tx

        inc Va_eqb_mux   ; go to next address on the mux
```

```
          djnz R0, F_eq_scan_loop

     lcall F_sc_crlf
     mov R0, #0d
     L_wait: djnz R0, L_wait
     mov R0, #0d
     L_wait2: djnz R0, L_wait2
     sjmp L_main_loop

;------------------------------------------------------------------------------------------------

; INCLUDES
;------------------------------------------------------------------------------------------------

#include sercom.asm
```