

```

#include p18f2550.inc

; Various definitions

; 3210 chip select pin
#define tris__cs_3210      TRISB3
#define _cs_3210          LATB3

; 3210 reset and PCM bus pins

#define tris_pcm_3210_pclk  TRISA,TRISA4 ; has to be set to 0 for output
#define pcm_3210_pclk      LATA,LATA4    ; 1 is high, 0 is low

#define tris_pcm_3210_drx   TRISA,TRISA3 ; has to be set to 1 for input
#define pcm_3210_drx       PORTA,RA3     ; 1 is high, 0 is low

#define tris_pcm_3210_dtx   TRISA,TRISA2 ; has to be set to 0 for output
#define pcm_3210_dtx       LATA,LATA2    ; 1 is high, 0 is low

#define tris_pcm_3210_fsync TRISA,TRISA1 ; has to be set to 0 for output
#define pcm_3210_fsync     LATA,LATA1    ; 1 is high, 0 is low

#define tris__reset_3210   TRISA,TRISA0 ; has to be set to 0 for output
#define _reset_3210       LATA,LATA0    ; has to be set to 0 for no-rst

; Our exports
; code
GLOBAL      tmr1_isr_init
GLOBAL      tmr1_isr

; data
GLOBAL      rnginp
GLOBAL      rngout
GLOBAL      rinprp
GLOBAL      rinpwp
GLOBAL      routrp
GLOBAL      routwp
GLOBAL      elapsed

;
; Globals, all placed in GPR1 (0x100). It is important to have all
; of these in the same bank, because in the ISR we only do a single
; BANKSEL, assuming that all data lie in the same bank, in order to
; save us time. Thus, we ask the assembler to tell the linker to
; place our data at the beginning of GPR1.
gpr1      UDATA      0x100

; next six globals implement two read/write rings for buffering PCM
data
; between the timer1 ISR and the main code; NOTE: DON't put anything
; else before rnginp, otherwise the two rings will not be 64-byte
; boundary aligned and nothing will work as expected
rnginp    RES        64          ; stores ~8 ms worth of incoming data
rngout    RES        64          ; stores ~8 ms worth of outgoing data

```

```

rinprp      RES      1          ; pointer to read from rnginp (main)
rinpwp      RES      1          ; pointer to write into rnginp (ISR)
routrp      RES      1          ; pointer to read from rngout (ISR)
routwp      RES      1          ; pointer to write into rngout (main)
; next two values are one-byte buffers for the current bytes of PCM
data
; that are currently being read in and written out
byteinp     RES      1          ; stores the incoming 1-byte PCM data
byteout     RES      1          ; stores the outgoing 1-byte PCM data
; next come two timers that are maintained by the ISR in order to do
; different things at each invocation
cnt8        RES      1          ; counts 8 isr invocations
cnt4        RES      1          ; counts 4x8 invocations
; and a debugging return value
elapsed     RES      1          ; stores @'s value (for debugging only!)
; temporary storage for FSR1's value
sfsr1hu     RES      1          ; FSR1H, user's save space
sfsr1lu     RES      1          ; FSR1L, user's save space
sfsr1hi     RES      1          ; FSR1H, ISR's save space
sfsr1li     RES      1          ; FSR1L, ISR's save space
; temporary storage for read/write
rinprbu     RES      1          ; byte being read from rnginp (user)
routwbu     RES      1          ; byte being written into rngout (user)

```

```

; My TMR1 service interrupt routine

```

```

; I have modified the bootloader code to jump to 0x820 for the
; high-priority interrupt; thus, the PIC takes 3 instruction
; cycles (I use C to denote an instruction cycle) to process
; the interrupt and another 2 C to jump here, which sums up to
; 5 C.

```

```

timerl_isr CODE          0x820

```

```

tmr1_isr

```

```

; DOCNOTE:
; I use the @ symbol to denote the cycles already elapsed when we
; enter this routine; this is equal to 5 (see previous note).

```

```

; T:@ where @ >= 5, see above

```

```

#ifdef STORE_ELAPSED

```

```

; the following instruction saves @+1 into elapsed (this is a two-
; word/two-cycle instruction which actually gets executed at the
; second cycle, so an extra cycle is added to @ before we get its
; value into elapsed).

```

```

;;; MOVFF      TMR1L, elapsed          ; C:2 - comment me out!

```

```

; NOTE:

```

```

; use this ONLY for debugging! it adds two unnecessary clock cycles!
; plus, these cycles are NOT taken into account in calculations for
; the next value of the timer, so make sure to comment/if/ifdef this
; out on 'production' versions, otherwise the clock will drift.

```

```

#endif

```





```

RETFFIE          FAST          ; C:2

firstof4not32
; this is the first cnt8 round of a cnt4 round other than the first
one

; we check further if cnt4==1 or 2; on cnt4==1, we transfer one byte
; from the output ring into byteout, whereas on cnt4==2, we transfer
; bytein into the input ring (we don't do anything special if cnt4==3)

; T:@+12

DECF            cnt4, W, BANKED ; C:1 decrement cnt4, rslt in W
BNZ            notsecondof4    ; C:1/2 if cnt4 > 1, jump on

secondof4
; T:@+14
; this is the second cnt4 run just starting (cnt4==1); in this case,
; we read one byte from the output ring and write it into byteout

; save FSR1 to temp storage
MOVFF          FSR1H, sfsr1hi   ; C:2 save FSR1H
MOVFF          FSR1L, sfsr1li   ; C:2 save FSR1L
NOP            ; C:1

; assuming @==5, 24 cycles have elapsed since TMR1 fireup; it's time
; to lower PCLK and set the timer again, then we go on with reading
; the output ring

; T:@+19/&

; See NOTE on additional NOPs that might be needed for adjusting PCLK duty
cycle

BCF            pcm_3210_pclk, ACCESS ; C:1 lower PCLK

; T:&+1

SETF          TMR1H, ACCESS      ; C:1 set TMR1H to 0xFF
MOVLW        237                 ; C:1 next interrupt at &+23
MOVWF        TMR1L, ACCESS      ; C:1

; T:&+4

; fetch the read pointer, make sure it is not equal to the write ptr
MOVF         routrp, W, BANKED   ; C:1
SUBWF        routwp, W, BANKED   ; C:1
BZ          output_underrun     ; C:1/2

; initialize FSR1 with the base address of the output ring
LFSR         FSR1, rngout        ; C:2

; T:&+9

; get the next byte from the ring into byteout, incr. pointer % 64
MOVF         routrp, W, BANKED   ; C:1
ADDWF        FSR1L, F, ACCESS    ; C:1
MOVFF        POSTINCL, byteout   ; C:2
MOVLW        0x3f                ; C:1
ANDWF        FSR1L, W, ACCESS    ; C:1

```





```

; T:&+7

; TODO: we may want to flag the overrun condition to the user space
RETFIE          FAST          ; C:2

notthirdof4

; T:@+18

NOP             ; C:1

; assuming @==5, 24 cycles have elapsed since TMR1 fireup

; T:@+19/&

; See NOTE on additional NOPs that might be needed for adjusting PCLK duty
cycle

BCF             pcm_3210_pclk, ACCESS ; C:1 lower PCLK

; T:&+1

SETF           TMR1H, ACCESS ; C:1 set TMR1H to 0xFF
MOVLW         237 ; C:1 next interrupt at &+23
MOVWF         TMR1L, ACCESS ; C:1

; T:&+4

RETFIE          FAST          ; C:2

notfirstof8

; T:@+8

; if this is the first cnt4 round (cnt4==0), we will do PCM data I/O
; to and from byteout and byteinp, respectively

DECF          cnt4, W, BANKED ; C:1 result->W, set carry if 0
BNC           nfo32_8nops ; C:1/2

; T:@+10

; start by asserting the right DRX value (see previous notes on
timing)

RLCF          byteout, F, BANKED ; C:1 rotate left through carry
BNC           nfo32_clr_DRX ; C:1/2 check if carry is set
BSF          pcm_3210_drx, ACCESS ; C:1 it is set, so set DRX to 1
BRA          nfo32_DRX_ok ; C:2 done

nfo32_clr_DRX ; T:@+15

BCF          pcm_3210_drx, ACCESS ; C:1 carry not set, clear DRX
NOP ; C:1 make both paths' delay eq.

nfo32_DRX_ok ; T:@+15 via both paths

```

```

        BRA            nfo32_2nops            ; C:2

        ; we arrive here from branching if this is not the first cnt3 round
                                                ; T:@+11
nfo32_8nops

        NOP            ; C:1
        NOP            ; C:1
        NOP            ; C:1
        NOP            ; C:1
        NOP            ; C:1
        NOP            ; C:1

                                                ; T:@+17
nfo32_2nops
        NOP            ; C:1
        NOP            ; C:1

        ; assuming @==5, 24 cycles have elapsed since TMR1 fireup

                                                ; T:@+19/&+0

; See NOTE on additional NOPs that might be needed for adjusting PCLK duty
cycle

        BCF            pcm_3210_pclk, ACCESS ; C:1 lower PCLK

                                                ; T:&+1

        SETF           TMR1H, ACCESS        ; C:1 set TMR1H to 0xFF
        MOVLW          237                  ; C:1 next interrupt at &+23
        MOVWF          TMR1L, ACCESS        ; C:1

                                                ; T:&+4

        ; if this is our first cnt4 run, it's time to collect 3210's DTX
signal

        ; check if cnt4 is zero
        DECF           cnt4, W, BANKED      ; C:1 result->W, sets carry if 0
        BNC            nfo32_return         ; C:1/2

        RLNCF          byteinp, F, BANKED   ; C:1 rotate left byteinp
        BTFSC          pcm_3210_dtx, ACCESS ; C:1/2 skip next if DTX is 0
        BSF            byteinp, 0, BANKED   ; C:1 if 1, set bit 0 of byteinp

nfo32_return

        ; done, return
        RETFIE         FAST                 ; C:2

;      Following code section is relocatable
tmrlisrinit CODE

; The TMR1 ISR initialization function
tmrl_isr_init

        ; disable TMR1 interrupts

```



