A ATB Control Software

This appendix contains listings of the author's software for testing current-mode analog circuits on the ATB. §A.1 shows the low-level routines that manipulate the functional units of the ATB. §A.2 is `ftntest`, a (largish) program that offers a menu of functions for testing portions of the ATB hardware, and for basic testing of circuits via the ATB. §A.3 contains miscellaneous routines that `ftntest` uses; §A.4 is a makefile for `ftntest`.

A.1 Library Routines

These three files, `low-para.h`, `atbpara.h`, and `atbpara.c`, provide basic routines to manipulate the parallel interface from a C-language program.

```
/*-------------------------------------------------------------------*\
| low-para.h                                                          |
|                                                                     |
| Low-level constants and macros that define the parallel port and its  |
| control.  Here we take care of inverted bits, weird bit/nybble        |
| positions, and such arcana....                                      |
|                                                                     |
| Foo.                                                                |
|                                                                     |
| 92-11-10 Split out from atbpara.h, because only the atbpara.c file    |
|       really needs this stuff.                                      |
| 93/01/20 ... and ftntest.c ...                                      |
| 93-03-28 Rename "S_Reg_Nybble" to "S_Reg_Upper4" so I'll remember     |
|       which nybble has the data  :(                                 |
\*-------------------------------------------------------------------*/
#if !defined(_LOW_PARA_H)
#define _LOW_PARA_H
#include <dos.h>

#define D_Reg    0x0378         /*---------------------------------*/
#define S_Reg    0x0379         /* Parallel-port register addresses */
#define C_Reg    0x037a         /*---------------------------------*/

/*-------------------------------------------------------*\
| Bit 3 of the Status register is inverted.  This fixes it. |
```

```
\*-----------------------------------------------------------*/
#define STATUS_X_MASK   0x80    /* 1000_0000 */

/*-----------------------------------------------------------*\
| Extract a data nybble from (the high end of) the Status |
| register.  The 0x00f0 masking really is necessary ---   |
| it's cost me a week and a half to determine that :<     |
\*-----------------------------------------------------------*/
#define S_Reg_Upper4    (word)(0x00f0 & (STATUS_X_MASK ^ inportb(S_Reg)))

/*-------------------------------------------------------------*\
| Control-Port signals - ref. Lab Notebook #2, pp.13,14, etc.   |
|                                                               |
| Control bits (on the ATB interface board):                    |
|   7..4 - (no connection)                                      |
|      3 - DAC-addr enable ( ATB address line a20)              |
|      2 -  } Interface-latch selector                          |
|      1 -  }                                                   |
|      0 - '139 latch enable.L (interface addr buffer??)        |
|                                                               |
| The parallel-port Control Register inverts bits 3, 1, & 0     |
| (eat flaming death IBM), so these constants incorporate the   |
| necessary bit inversions to produce desired values at the     |
| interface board.                                              |
\*-------------------------------------------------------------*/
#define CTRL_X_MASK             0x0b    /* 0000_1011 */
#define LATCH_ENABLE_MASK       0x01

/*-------------------------------------------------------------*\
| Select one of the ATB-device-addressing latches (or nothing)  |
| to accept the byte from the Data register.  The latch isn't   |
| enabled until bit 0 goes TRUE (low at the '139 chip).         |
|                                                               |
| Usage:                                                        |
|     Send out the appropriate latch-select value.              |
|     Send it again, OR'd w/ LATCH_ENABLE_MASK, to actually     |
|       latch in the D_Reg value.                               |
|     Send it again w/o LATCH_ENABLE_MASK to disable the latch. |
|                                                               |
```

```
| This sequence avoids glitches when changing latch selects.    |
|                                                               |
| The desired value at the ATB interface board is XOR'd to      |
| convert it to what the C_Reg needs to see.                    |
\*-------------------------------------------------------------*/
#define ctrl_NULL       (0x01 ^ CTRL_X_MASK)    /* 0001 -> 1010 */
#define D_HI_LATCH      (0x03 ^ CTRL_X_MASK)    /* 0011 -> 1000 */
#define D_LO_LATCH      (0x05 ^ CTRL_X_MASK)    /* 0101 -> 1110 */
#define A_LATCH         (0x07 ^ CTRL_X_MASK)    /* 0111 -> 1100 */

/*-------------------------------------------------------------*\
| Select one of the "Blue Bus" latches.  These values           |
| incidentally select one of the device-addressing latches as   |
| well, but they are held DISabled by bit 0.  One of these      |
| latches is always unavoidably selected, and they have been    |
| hardwired to be ENabled; so there is always something on the  |
| Status-register lines.  We simply ignore the S_Reg until      |
| after we've deliberately latched a meaningful value.          |
|                                                               |
| As above, the desired value at the ATB interface board is     |
| XOR'd to convert it to what the C_Reg needs to see.           |
\*-------------------------------------------------------------*/
#define n3_READ         (0x01 ^ CTRL_X_MASK)    /* 0001 -> 1010 */
#define n2_READ         (0x07 ^ CTRL_X_MASK)    /* 0111 -> 1100 */
#define n1_READ         (0x05 ^ CTRL_X_MASK)    /* 0101 -> 1110 */
#define n0_READ         (0x03 ^ CTRL_X_MASK)    /* 0011 -> 1000 */

/*-------------------------------------------------------------*\
| Enable the outputs on the address-decoder board, so the       |
| addressed DAC will buffer a value from the Red bus.           |
\*-------------------------------------------------------------*/
#define DAC_ENABLE      (0x09 ^ CTRL_X_MASK)    /* 1001 -> 0010 */

/*-------------------------------------------------------------*\
| Produce the '139-enabling version of a ctrl-reg value.        |
|                                                               |
| The un-enabled ctrl-reg value sets up the '139 multiplexer    |
| to clock one of the '374 FF-registers on the interface board; |
| this enabling version causes the '139 to actually emit        |
```

```
| whichever "clock" pulse is set up.  The enabling protocol     |
| blocks any switching transients in the '139 from triggering   |
| one of the '374s inappropriately.                             |
\*-------------------------------------------------------------*/
#define enabled(Ctrl)           ((Ctrl) | LATCH_ENABLE_MASK)

#endif  /* _LOW_PARA_H */
```

```
/*-------------------------------------------------------------------*\
|  ATBPARA.H                                                          |
|                                                                     |
| This header file defines data types, constants, and function        |
| prototypes (or macros) for use by a program doing parallel I/O to   |
| the Analog Test Board (ATB).                                        |
|                                                                     |
| 93-05-25 ADC_delay gone again; I understand what I'm doing better...|
|       logic-analyzer timings provoke changes in atbpara.c           |
| 93-04-23 "lo(WORD)" and "hi(WORD)" moved from the newly-created     |
|          "local.h" to here.                                         |
| 93-03-20 ADC delay in MICROsecs instead of millisecs                |
| 93-03-17 read_data() replaced by ADC_bus().                         |
| 93-03-14 Vdd, Vss can now be set using 12-bit values.  Floating-point |
|          inputs use adjust_Vdd(), adjust_Vss() instead.             |
| 93-02-07 Debugged the comments (!) as part of the distribs package. |
| 93-02-02 Trying to figure out the post-Nov. changes... (ugh).       |
|          Cleaned up comments some.                                  |
| 92-11-10 Debug the internal changes.  Move some parallel-port stuff |
|          to low-para.h, "simplify" some more stuff.                 |
| 92-11-03 Internal changes for efficiency - some constants mutate too |
| 92-05-24                                                            |
\*-------------------------------------------------------------------*/
#if !defined(_ATBPARA_H)
#define _ATBPARA_H

typedef unsigned char byte;
typedef unsigned int  word;
```

```
#define lo(Byte_)  (Byte_ & 0x00ff)
#define hi(Byte_)  (Byte_ >> 8)

#define DAC_XFER    0x40                /*------------------*/
#define ADC_CONV    0x42                /* Device addresses */
#define Vss_DAC     0x44                /*------------------*/
#define Vdd_DAC     0x45
#define addr_NULL   0x3f   /*  An address that shouldn't enable anything  */

#define ADC0    0                       /*-------------*/
#define ADC1    1                       /* ADC indexes */
#define ADC2    2                       /*-------------*/
#define ADC3    3
#define ADC4    4
#define ADC5    5
#define no_ADC  0xffff

/*-----------------------------------------------*\
| Conversions between User units & device counts |
\*-----------------------------------------------*/
#define COUNTS_PER_VOLT (float)(MAX_COUNT/10.0)
#define MAX_COUNT       0xfff
#define NUM_COUNTS      (MAX_COUNT+1)

/*---------------------------------------*\
|  Bring the ATB to a known initial state.  |
\*---------------------------------------*/
void reset_ATB(void);

/*-------------------------------------------------------------------------*\
|  DAC conversions --- DAC_XFER device distributes a conversion signal  |
|  to all DACs.  Not used directly.                                     |
|                                                                       |
|  write_DAC() causes the selected DAC to latch the (12-bit) value      |
|  into its input buffer.                                               |
|                                                                       |
|  trigger_DACs() causes all the DACs to transfer their input buffers   |
|  to their conversion registers, thus generating new analog outputs.   |
```

```
|                                                                      |
|  DAC_flow_through() causes all the DACs to pass their buffer values  |
|  straight through to their conversion registers.                     |
\*--------------------------------------------------------------------*/
#define DAC_convert_new()       write_device(DAC_XFER, 0x01)
#define hold_DAC_conversion()   write_device(DAC_XFER, 0x00)

#define write_DAC(addr, data)   write_device(addr, data)
#define trigger_DACs()          (DAC_convert_new(), hold_DAC_conversion())
#define DAC_flow_through()      DAC_convert_new()

/*-------------------------------------------*\
|  Reset all ADCs so they're ready to accept a  |
|  new conversion signal.  Used by reset_ATB(). |
\*-------------------------------------------*/
#define reset_ADCs()    write_device( ADC_CONV, 0 )

/*--------------------------------*\
|  Convert and read a single ADC.  |
\*--------------------------------*/
word ADC_sample(byte adc_num);

/*-----------------------------------------------------*\
|  Trigger ADC conversions, enable ADC output lines,    |
|  access the Blue output bus; reset ADC conversions.   |
|  ADC_sample() integrates these into one function.     |
|                                                       |
| 93-03-17 read_data() replaced by ADC_bus().           |
\*-----------------------------------------------------*/
#define trigger_ADC(adc)        adc_strobe( ADC_trgr(adc) )
#define trigger_all_ADCs()      adc_strobe(all_ADC_triggers)
#define latch_ADC_out(adc)      set_addr( ADC_addr(adc) )

word ADC_bus(void);             /* Read whatever's on the Blue bus. */

/*-----------------------------------------------------------*\
| High-level functions/macros that accept values in user units  |
| (eg, -10..10 volts) & control the ATB in byte-oriented terms. |
\*-----------------------------------------------------------*/
```

```
#define word_form(v)    (word)( (v) * COUNTS_PER_VOLT )
#define volt_form(c)    (float)( (c) / COUNTS_PER_VOLT )

#define set_Vdd_DAC(val)        write_DAC(Vdd_DAC, (val))
#define set_Vss_DAC(val)        write_DAC(Vss_DAC, (val))

void adjust_Vdd(float val);
void adjust_Vss(float val);

/*************************************************************************\
*  Low-level functions, constants, macros.  The high-level functions      *
*  above provide the interfaces to this stuff....                         *
*  Generally, User programs needn't refer to these directly.              *
\*************************************************************************/

void latch_in(byte val, byte ctrl);     /* Load byte to interface register */
void set_addr(byte addr);               /* load Addr interface register */
void set_data(word data);               /* load Data interface registers */

void write_device(byte addr, word data);        /* yer basic control ftn */

/*------------------------*\
| ADC addresses & triggers |
\*------------------------*/
#define ADC_addr(adc)    ( 0xf0 | adc )
#define ADC_trgr(adc)    ( 0x01 << adc )
#define all_ADC_triggers ( ADC_trgr(0) | ADC_trgr(1) | ADC_trgr(2) | \
                           ADC_trgr(3) | ADC_trgr(4) | ADC_trgr(5) )
void adc_strobe(byte adcbits);

#endif  /* _ATBPARA_H */
```

```
/*--------------------------------------------------------------------*\
|  ATBPARA.C                                                           |
|                                                                      |
| Subroutines that perform I/O to the Analog Test Board (ATB) via      |
| the parallel port.                                                   |
|                                                                      |
| 93-05-25: adc_strobe() doesn't need ADC_delay; it is slow enough to  |
|       let the single-pulser flipflops reset anyway.  ADC_sample()    |
|       gets changes based on logic-analyzer timing information.       |
| 93-04-29:(  latch_in() becomes a subroutine again, after evidence    |
|       that the inline code experiences more time jitter.  I wish I   |
|       understood why.  Make it externally visible while we're at it. |
| 93-03-20: Make use of u_time() microsecond timer.                    |
| 93-03-19: Change read_data() to ADC_bus() for aesthetic reasons.     |
| 93-03-14: Floating-point Vdd & Vss ftns renamed to "adjust_V.."      |
|       while "set_V.._DAC" become macros with integer inputs.         |
|       Fluke 97 indicates a 78.1KHz (12.8us) clock.                   |
|                                                                      |
| 92-11-16: After profiling:  latch_in() becomes a macro for more speed |
| 92-11-10: After a flakey initial attempt, clean up/speed up the      |
|       functions.  Move the parallel-port #defines into low-para.h    |
| 92-06-21: ADC-trigger patch. bobmon                                  |
| 92-06-18: touchup. bobmon                                            |
\*--------------------------------------------------------------------*/
#include <stdio.h>      /* For error messages in the Vdd/Vss routines. */
#include "low-para.h"
#include "atbpara.h"
#include "timing.h"

#undef DEBUG

#define C_reg_OFF()     outportb(C_Reg, ctrl_NULL)

void reset_ATB(void)
{
    C_reg_OFF();                /* Disable all interface latches.       */
    hold_DAC_conversion();      /* enable double-buffering.             */
    reset_ADCs();               /* ready ADCs for new conversion.       */
    C_reg_OFF();                /* Disable all latches (again).         */
```

```
    u_time_calibrate(1000000L); /* prepare for microsecond delays.      */
}

/*-------------------------------------------------------------------*\
|  Map one of 256 logical addresses to the physically-wired equivalent. |
|  (sigh... software hack for hardware quirk:  each pair of bits        |
|  (0&1, 2&3; 4&5, 6&7) is swapped.)                                    |
|  Each nybble translates as:                                           |
|       0000 : 0000     0100 : 1000     1000 : 0100     1100 : 1100     |
|       0001 : 0010     0101 : 1010     1001 : 0110     1101 : 1110     |
|       0010 : 0001     0110 : 1001     1010 : 0101     1110 : 1101     |
|       0011 : 0011     0111 : 1011     1011 : 0111     1111 : 1111     |
|  92-11-03 Okay, we make one big table, for speed.  The moot code just |
|       below shows the original mapping method.                        |
\*-------------------------------------------------------------------*/
#if 0
static const byte Addr_Map[16] = {
    0x0, 0x2,  0x1, 0x3,    0x8, 0xa,  0x9, 0xb,
    0x4, 0x6,  0x5, 0x7,    0xc, 0xe,  0xd, 0xf
};
#define address(addr)  \
    ( (Addr_Map[ (addr & 0xf0) >> 4 ] << 4)  |  Addr_Map[ addr & 0x0f ] )
#else   /*-----------------------------------------------------*/

static const byte near Addr_Map[] = {
    0x00, 0x02, 0x01, 0x03,  0x08, 0x0a, 0x09, 0x0b,
      0x04, 0x06, 0x05, 0x07,  0x0c, 0x0e, 0x0d, 0x0f,
    0x20, 0x22, 0x21, 0x23,  0x28, 0x2a, 0x29, 0x2b,
      0x24, 0x26, 0x25, 0x27,  0x2c, 0x2e, 0x2d, 0x2f,
    0x10, 0x12, 0x11, 0x13,  0x18, 0x1a, 0x19, 0x1b,
      0x14, 0x16, 0x15, 0x17,  0x1c, 0x1e, 0x1d, 0x1f,
    0x30, 0x32, 0x31, 0x33,  0x38, 0x3a, 0x39, 0x3b,
      0x34, 0x36, 0x35, 0x37,  0x3c, 0x3e, 0x3d, 0x3f,

    0x80, 0x82, 0x81, 0x83,  0x88, 0x8a, 0x89, 0x8b,
      0x84, 0x86, 0x85, 0x87,  0x8c, 0x8e, 0x8d, 0x8f,
    0xa0, 0xa2, 0xa1, 0xa3,  0xa8, 0xaa, 0xa9, 0xab,
      0xa4, 0xa6, 0xa5, 0xa7,  0xac, 0xae, 0xad, 0xaf,
    0x90, 0x92, 0x91, 0x93,  0x98, 0x9a, 0x99, 0x9b,
```

```
      0x94, 0x96, 0x95, 0x97,  0x9c, 0x9e, 0x9d, 0x9f,
    0xb0, 0xb2, 0xb1, 0xb3,  0xb8, 0xba, 0xb9, 0xbb,
      0xb4, 0xb6, 0xb5, 0xb7,  0xbc, 0xbe, 0xbd, 0xbf,

    0x40, 0x42, 0x41, 0x43,  0x48, 0x4a, 0x49, 0x4b,
      0x44, 0x46, 0x45, 0x47,  0x4c, 0x4e, 0x4d, 0x4f,
    0x60, 0x62, 0x61, 0x63,  0x68, 0x6a, 0x69, 0x6b,
      0x64, 0x66, 0x65, 0x67,  0x6c, 0x6e, 0x6d, 0x6f,
    0x50, 0x52, 0x51, 0x53,  0x58, 0x5a, 0x59, 0x5b,
      0x54, 0x56, 0x55, 0x57,  0x5c, 0x5e, 0x5d, 0x5f,
    0x70, 0x72, 0x71, 0x73,  0x78, 0x7a, 0x79, 0x7b,
      0x74, 0x76, 0x75, 0x77,  0x7c, 0x7e, 0x7d, 0x7f,

    0xc0, 0xc2, 0xc1, 0xc3,  0xc8, 0xca, 0xc9, 0xcb,
      0xc4, 0xc6, 0xc5, 0xc7,  0xcc, 0xce, 0xcd, 0xcf,
    0xe0, 0xe2, 0xe1, 0xe3,  0xe8, 0xea, 0xe9, 0xeb,
      0xe4, 0xe6, 0xe5, 0xe7,  0xec, 0xee, 0xed, 0xef,
    0xd0, 0xd2, 0xd1, 0xd3,  0xd8, 0xda, 0xd9, 0xdb,
      0xd4, 0xd6, 0xd5, 0xd7,  0xdc, 0xde, 0xdd, 0xdf,
    0xf0, 0xf2, 0xf1, 0xf3,  0xf8, 0xfa, 0xf9, 0xfb,
      0xf4, 0xf6, 0xf5, 0xf7,  0xfc, 0xfe, 0xfd, 0xff,
};
#define address(addr)  Addr_Map[addr]

#endif  /*-------------------------------------------------------*/

/*-------------------------------------------------------------*\
| Put a byte on the parallel-port data lines, then strobe the   |
| appropriate interface-board latch to accept it.               |
| This needs to be a function to try to keep timing clean...    |
| The final shutdown (ctrl_NULL) seems to help keep things      |
| cleaner on the interface board.                               |
|                                                               |
| We'll also make it visible "just in case".                    |
\*-------------------------------------------------------------*/
void latch_in(byte val, byte ctrl)
{
    outportb(D_Reg, val);               /* Value onto data bus  */
    outportb(C_Reg, ctrl);              /* Select desired latch */
```

```
    outportb(C_Reg, enabled(ctrl));     /* Enable the latch     */
    outportb(C_Reg, ctrl);              /* Drop the enable bit  */
}

/*-------------------------------------------------------------*\
|  Latch in an address.                                        |
\*-------------------------------------------------------------*/
#define s_set_addr(a)   latch_in(address(a), A_LATCH)

void set_addr(byte addr) { s_set_addr(addr); }

/*-------------------------------------------------------------*\
| Latch in both bytes of a data-bus value.  The macro is       |
| optimized after inspecting tcc's assembly output.            |
\*-------------------------------------------------------------*/
#define s_set_data(d)   ( latch_in( (d), D_LO_LATCH ), \
                          latch_in( ((d) >> 8), D_HI_LATCH ) )

void set_data(word data) { s_set_data(data); }

/*-------------------------------------------------------------*\
|  Latch a 12-bit datum, and an 8-bit address, to the board;   |
|  then strobe the address-enable so the addressed device      |
|  accepts the data value.  Finish by setting a null address.  |
\*-------------------------------------------------------------*/
void write_device(byte addr, word data)
{
    s_set_data(data);           /* Load the Red bus.                    */
    s_set_addr(addr);           /* Choose a device...                   */
    outportb(C_Reg, DAC_ENABLE); /* All dev's enabled, only 1 addr'd.   */
    s_set_addr(addr_NULL);      /* Un-choose the device.                */
    C_reg_OFF();                /* Disable all interface latches.       */
}

/*-------------------------------------------------------------*\
| Trigger an ADC to convert, give the conversion some time,    |
| then  enable its output buffer and read the value one nybble |
| at a time.  The Nat'l Semi. ADC1210 needs 100us from the     |
| Start-Conversion trigger to produce a valid 12-bit output.   |
```

```
|                                                                  |
| 93-05-30: Do a correct ADC-reset at the end.  This function      |
|       CANNOT support reading more than one ADC after a common    |
|       triggering event.  To do that, all the ADCs should be      |
|       triggered, each ADC latched and bussed out, then           |
|       they should all be reset.                                  |
| 93-05-25: 200usec -> ~40usec error margin for conversion.        |
| 93-03-20: u_time() 4X margin of error --- delay(1) gives 10X     |
\*-----------------------------------------------------------------*/
word ADC_sample(byte adc_num)
{
    word tmp;

    trigger_ADC( adc_num );     /* Start the conversion and...  */
    u_time(200);                /*   give it time to complete.  */

    latch_ADC_out( adc_num );   /* Put result on the Blue...    */
    tmp = ADC_bus();            /*   bus, and read it out.      */

    reset_ADCs();               /* Prepare for another reading. */
    return tmp;
}

/*-----------------------------------------------------------------*\
| Reset & start conversion on ADC(s).                              |
| "trigger_ADC()" is a macro that converts from an ADC number      |
| to the "adcbits" byte needed here.                               |
|                                                                  |
| 93-05-25: No delay needed between reset & pulse --- ftn-call     |
|       overhead gives the single-pulsers plenty of time,          |
|       given the 78.1KHz/12.8us ADCclock  (meas. by Fluke 97)     |
\*-----------------------------------------------------------------*/
void adc_strobe(byte adcbits)
{
    write_device(ADC_CONV, 0);          /* Reset the ADC single-pulsers */
    write_device(ADC_CONV, adcbits);    /* Pulse all desired /SC lines  */
}

/*-----------------------------------------------------------------*\
```

```
| Read the ADC output via Status register, nybble-by-nybble.    |
| Data nybbles come in on bits 7..3 of the Status register, so  |
| they are shifted into position as they arrive.                |
|                                                               |
| 93-04-23: Delays out; they aren't the problem?                |
| 93-03-20: Add delays to increase the chance of reading the    |
|       parallel port correctly.  (A shot in the dark, this.)   |
| 93-03-17: read_data() becomes ADC_bus(), w/ change in         |
|       argument-passing style.                                 |
| 93-03-16: Read each nybble separately, OR them together.      |
| 92-11-10: I went too far... Stuff now buried in low-para.h    |
| 92-11-03: Speed things up as much as possible by coding       |
|       only what's needed (elegance & regularity lose to       |
|       efficiency).                                            |
| Original: This routine reads all four nybbles, even though    |
|       the high nybble (n3) is hardwired to 0 on the ATB.      |
\*--------------------------------------------------------------*/
#if 0
#   define Wait_usec    utime(1)
#else
#   define Wait_usec    {}
#endif

word ADC_bus(void)
{
    word hi, mid, lo;

    outportb(C_Reg, n2_READ);           /* bits 11..8 */
    Wait_usec;                          /* wait 1 usec (yeah right) */
    hi = S_Reg_Upper4;

    outportb(C_Reg, n1_READ);           /* bits  7..4 */
    Wait_usec;                          /* wait 1 usec (yeah right) */
    mid = S_Reg_Upper4;

    outportb(C_Reg, n0_READ);           /* bits  3..0 */
    Wait_usec;                          /* wait 1 usec (yeah right) */
    lo = S_Reg_Upper4;
```

```
    return ( (hi<<4) | mid | (lo>>4) );
}

/*-------------------------------------------------------------*\
|  Set the constant-V 'power-supply' DACs to a desired voltage. |
\*-------------------------------------------------------------*/
void adjust_Vdd(float val)
{
    if ( (val < 0.0) || (10.0 < val) )
        fprintf(stderr,"Invalid Vdd:  0 < %f < 10 fails.\n", val);
    else {
#ifdef DEBUG
        printf("Vdd: word_form(%f) = $%x ( %s )\n",
                 val, word_form(val), dec_bin(word_form(val),16));
#endif
        write_DAC( Vdd_DAC, word_form(val) );
    }
}

void adjust_Vss(float val)
{
    if ( (val < -10.0) || (0.0 < val) )
        fprintf(stderr,"Invalid Vss:  -10 < %f < 0 fails.\n", val);
    else {
        val = -val;
#ifdef DEBUG
        printf("(-): word_form(%f) = $%x ( %s )\n",
                 val, word_form(val), dec_bin(word_form(val),16));
#endif
        write_DAC( Vss_DAC, word_form(val) );
    }
}

/*-----------------------------  END  -------------------------------*/
```

A.2 `ftntest` Testing Program

This file contains the program `ftntest`, which provides a menu of testing options. It uses some of the ATB library routines, duplicates some of those routines for verification purposes, and also requires some utility functions which are listed in Appendix A.3 along with the matching ".h" files.

```
/*-------------------------------------------------------------------*\
| ftntest.c --- Trying to control the ATB via the parallel port...    |
|                                                                     |
|       Copyright 1994 R A Montante                                   |
|       All rights reserved.                                          |
|                                                                     |
| 94-08-18 Delete the hardcoded Itref option (i).                     |
|       Modify con_read(), because keyboard-macro programs affect the |
|       console input --- 50 and 25 couldn't be entered, with my      |
|       personal macros, for example....                              |
|       Add some copyright notice per Tech Transfer's request.        |
|       Make text graphics compatible with ASCII listing, shorten     |
|       frame-drawing code.  Also, cosmetic changes.                  |
| 93-10-18 Modify "write_each_DAC()" so it's escapable.               |
| 93-10-08 Subshell supported via do_system() choice.                 |
| 93-10-07 Make noises switchable.  Update some functions?            |
| 93-07-09 Discard empty output files; prettier menu.                 |
| 93-07-05 Hacking for output-to-files, windows                       |
| 93-05-05 Prettier menu.                                             |
| 93-04-26 Conversion to "locals.lib" and .h file.  Mods to the       |
|       wr_loop() routine to support debugging of the Red Bus.        |
| 93-03-23  Try it again without the nybble-hesitation (connector     |
|       was fixed, so bits all appear again).  More menu hacking.     |
| 93-03-20  Lotsa minor changes... "void read_data(wordp)" replaced   |
|       by "word ADC_bus(void)" for aesthetic reasons; ADC_bus()      |
|       hesitates while reading nybbles, to ensure stable values.     |
| 93-03-15 Ramp_DAC() added, screen-clearing w/ menu window.          |
| 93-03-08 Settable I_tref.  write-read loop added earlier            |
|       (probably other changes too)                                  |
| 92-07-03 fold in the null-loop timing options (tidiness)            |
| 92-06-20 ftn-pointer table implemented                              |
| 92-05-23                                                            |
\*-------------------------------------------------------------------*/
```

```
#include <stdio.h>
#include <string.h>
#include <dos.h>
#include <stdlib.h>
#include <process.h>
#include <conio.h>

#include "low-para.h"
#include "atbpara.h"
#include "converts.h"
#include "local.h"
#include "timing.h"

#define FALSE   0
#define TRUE    !FALSE

#define EOL     "\r\n"
/*-------------------------------------*\
| Text-graphics   0xde  0xdd  0xb3      |
\*-------------------------------------*/
#define RBLOK   "\xde"
#define LBLOK   "\xdd"
#define VBAR    "\xb3"


/*---------------------------------------------------------------------*/

static float Vdd, Vss;
static int noisy = 1;
static int std_out = -1;
static FILE *outfile;
const char out_fmt_msg[] = "File outputs in multiple bases [ny]?";
const char step_msg[] = "\r<SPACE> to step; 'C' to complete";


/*---< Menu-option Function Prototypes >-------------------------------*/

static void set_busses(void);
static void set_Addr_bus(void);
```

```
static void do_bases(void);
static void do_c(void);
static void do_d(void);
static void do_D(void);
static void do_g(void);
static void wr_loop(void);
static void ramp_1_DAC(void);
static void ramp_multiple_DACs(void);
static void draw_menu(void);
static void do_null(void);
static void do_readADC(void);
static void do_readblue(void);
static void do_reset(void);
static void do_t(void);
static void do_T(void);
static void do_V(void);
static void wrt_a_DAC(void);
static void w_t_DAC(void);
static void wrt_each_DAC(void);
static void wrt_all_DACs(void);
static void do_1(void);
static void do__(void);
static void do_escape(void);
static void do_redirect(void);
static void fix_noise(void);
static void do_system(void);

/*-----------------------------------------------------------------------*/

typedef void (*ftn_p)(void);

typedef struct _tagged_ftn {
    char tag;
    ftn_p f;
} tagged_ftn;

const tagged_ftn choice[] = {
    {'a', set_busses}   , {'A', set_Addr_bus}
    , {'B', do_bases}   , {'c', do_c}
```

```
    , {'d', do_d}       , {'D', do_D}
    , {'g', do_g}       , {'G', do_reset}
    , {'l', ramp_1_DAC} , {'L', wr_loop}
    , {'r', do_readADC} , {'R', do_readblue}
    , {'t', do_t}       , {'T', do_T}
    , {'V', do_V}
    , {'w', wrt_a_DAC}  , {'W', w_t_DAC}
    , {'z', wrt_all_DACs}, {'Z', wrt_each_DAC}
    , {'3', ramp_multiple_DACs}, {'1', do_1}
    , {'m', draw_menu}  , {'?', draw_menu}
    , {'n', fix_noise}  , {'N', do_null}
    , {'>', do_redirect}, {'!', do_system}
    , {'_', do__}       , {0x1b,do_escape}
};
#define n_choices  ( sizeof(choice) / sizeof(tagged_ftn) )


/*-------------------------------------------------------------------------*/

const char title[] =
 "\n"
 EOL"    ftntest:  portmanteau Analog Test Board exerciser"
 EOL"    version ["__DATE__", "__TIME__"]"
 "\n"
 EOL"    Copyright 1994 R A Montante"
 EOL"    All rights reserved."
 "\n\n\n"
 EOL"  Press 'm' for a menu of commands..."
;

const char menu[] =
 EOL
 " R  Read the blue ADC bus            c  Set Control-register" EOL
 " a  Set Lo/Hi data, Addr             A  Set Address bus (verbose)" EOL
 " d  Set Data bus (hi,lo)             D  Lo,Hi Data (verbose)" EOL
 " T  Verbose DAC Transfer (Fire)      G  Reset all ADCs" EOL
 "\n"
 " w  Write a DAC                      t  Fire all DACs" EOL
 " W  Write a DAC & fire DACs          g  Trigger all ADCs" EOL
```

```
 " l  Ramp a DAC, read an ADC          r  Read an ADC" EOL
 " L  Write/Fire/Read until keypress   1  Write/Fire a DAC, read an ADC" EOL
 " z  Write all DACs & fire            3  Ramp multiple DACs, read an ADC" EOL
 " Z  Write each DAC & fire            V  Show & reset Vss, Vdd" EOL
 "\n"
 " N  Timing loop [null-ftn"VBAR"ADC_bus-read"VBAR"empty]"EOL
 " n  Toggle noises/no-noises          B  Convert value to other bases"EOL
 "\n"
 " m  Clear & redisplay Menu           >  Toggle outputs-to-a-file"EOL
 " q  Quit                             !  DOS subshell"
;
/*....:....1....:....2....:....3....:....4....:....5....:....6....:....7....:.*\
\*   Menu-spacing ruler: digits mark decades, colons mark half-decades.       */


#define TOP     1
#define LEFT    3
#define frame_depth     23                      /* output lines, & borders */
#define frame_width     76                      /* menu width, borders */
#define BOTTOM          (TOP+frame_depth)       /* incl. "gutter" row  */
#define RIGHT           (LEFT+frame_width-1)

#define NoColor -1      /* invalid color number, used in show_prompt() */

/*---------------------------------------------------------------------*\
| Keep the sizes, colors, and current positions of each text window     |
| in a convenient data structure.                                       |
| WINDOWS:                                                              |
|       0 - frame;  1 - menu;  2 - output;   3 - time;  4 - "prompt"    |
\*---------------------------------------------------------------------*/
typedef struct winpos_ {
    int x, y;           /* cursor position      */
    int l, t, r, b;     /* window coordinates   */
    int bg, fg;         /* colors               */
} winpos;

winpos win_pos[] = {
      {1,1, LEFT,   TOP,   RIGHT,  BOTTOM,  BLACK,    LIGHTCYAN }
    , {0,0, LEFT+1, TOP+1, RIGHT-2,BOTTOM-2,CYAN,     WHITE     }
```

```
    , {0,0, LEFT+1, TOP+1, RIGHT-2,BOTTOM-2,LIGHTGRAY,BLACK    }
    , {0,0, LEFT+1, BOTTOM,LEFT+37,BOTTOM,  BLACK,    WHITE    }
    , {0,0, LEFT+38,BOTTOM,RIGHT,  BOTTOM,  BLACK,    WHITE    }
};
static int current_window = 0;

void hop_window(int new_window, int clear)
{
    winpos *wp;
    wp = &(win_pos[current_window]);
    wp->x = wherex();
    wp->y = wherey();
    current_window = new_window;
    wp = &(win_pos[current_window]);
    window(wp->l, wp->t, wp->r, wp->b);
    textbackground(wp->bg); textcolor(wp->fg);
    if (clear)
        clrscr();       /* Clear out the window? */
    else {
        gotoxy(wp->x,wp->y);
    }
}

#define Output_Window(CLEAR)    hop_window(2,(CLEAR))
#define Time_Window()           hop_window(3, TRUE)
#define Prompt_Window()         hop_window(4, TRUE)

/*-------------------------------------------------------------------*\
| The "output-to-file" indicator isn't worth a window of its own,     |
| although the syntax is similar.                                     |
\*-------------------------------------------------------------------*/
void File_Window(void)
{
    static int x, y;
    const  char flag[2] = { '>', YELLOW };
    static char locn[2];

    switch (std_out) {
      case 1:
```

```
        puttext(x,y, x,y, flag);
        break;

      case 0:
        puttext(x,y, x,y, locn);
        break;

      case -1:
        x = win_pos[0].l + (frame_width>>1);    /* middle of frame... */
        y = win_pos[0].b - 1;                   /* ...bottom edge     */
        gettext(x,y, x,y, locn);
        std_out = 0;
    }
}
/*-------------------------------------------------------------------------*\
| The window frame... formed of text-graphics characters:                   |
|    0xba  0xcd  0xc9  0xbb  0xc8  0xbc  0xb5  0xc6                         |
\*-------------------------------------------------------------------------*/
#define PrgNm "\xb5 ftntest \xc6"
#define draw_frame() {                                  \
    winpos *wp;                                         \
    int i;                                              \
    wp = &(win_pos[0]);                                 \
    window(wp->l, wp->t, wp->r, wp->b);                 \
    textbackground(wp->bg); textcolor(wp->fg);          \
    gotoxy(wp->x, wp->y);                  /* Top */    \
    putch(0xc9);                                        \
    for (i = 2; i < frame_width-1; ++i)                 \
        putch(0xcd);                                    \
    putch(0xbb);                                        \
    gotoxy(wp->x+(sizeof(PrgNm)>>1)-1, wp->y);          \
    cputs(PrgNm);                                       \
    for (i = 2; i < frame_depth; ) {       /* Sides */  \
        gotoxy(wp->x, i);                               \
        putch(0xba);                                    \
        gotoxy(wp->x + frame_width-2, i++);             \
        putch(0xba); putch(' ');                        \
    }                                                   \
    gotoxy(wp->x, wp->y + frame_depth-1);  /* Bottom */ \
```

```
    putch(0xc8);                                        \
    for (i = 2; i < frame_width - 1; ++i)               \
        putch(0xcd);                                    \
    putch(0xbc); putch(' ');                            \
}

/*--------------------------------------------------------------------*/

void show_prompt(const char *prmpt, const int color)
{
    Prompt_Window();
    if (color != NoColor)
        textcolor(color);
    cputs(prmpt);
}

/*--------------------------------------------------------------------*/

static void cpause(void)
{
    cputs("\r <SPACE BAR> to continue");
    getch();
}

/*--------------------------------------------------------------------*\
| Text-graphics char cycles with each call, provides animated display.  |
\*--------------------------------------------------------------------*/
static char mark = 0x10;
char marker(void)
{
    if (!(mark & 0x01)) {
        mark ^= 0x0f;
    } else {
        mark &= 0xfe;
    }
    return mark;
}

/*--------------------------------------------------------------------*/
```

```
static void audible(unsigned freq)
{
    if (noisy)
        chirp(freq);
}

/*-------------------------------------------------------------------------*\
| Display a prompt, accept a character string for return, report the       |
| string's length.  Make sure a 0-length string is 0-terminated.          |
\*-------------------------------------------------------------------------*/
int con_read(char *prompt, char *bfr, int len)
{
    int x, y;
    char *b = bfr + 2;

    cputs(prompt); clreol();
    x = wherex(); y = wherey();
    fgets(b, len-2, stdin);
    gotoxy(x, y); cputs(EOL); clreol();

    b[ lookup('\n',b) ] = (char)0;
    b[ lookup('\r',b) ] = (char)0;
    return stringlen(b);
}

/*-------------------------------------------------------------------------*/

word get_word_num(char *prompt)
{
    char ans[81];

    sprintf(ans, "%s ", prompt);
    audible(660);
    Prompt_Window();
    con_read(ans, ans, 81);
    return (word)numeric(ans+2);
}

/*-------------------------------------------------------------------------*/
```

```
void con_log_string(char *bfr)
{
    cputs(bfr);
    if (std_out) {
        fprintf(outfile, "# %s", bfr+2);
        flushall();
    }
}
        /*-----------------------------------*/

void rslt_log(word data, word result, int verbose)
{
    if (!std_out)
        return;

    fprintf(outfile, "%4u  %4u  #  $%3x  $%3x", data,result, data,result);
    if (verbose) {
        fprintf(outfile, "  ##  %s  ", dec_bin(data,12) );
        fputs(dec_bin(result,12), outfile);
    }
    fputc('\n', outfile);
    flushall();
}
        /*-----------------------------------*/

void newline(void)
{
    putch('\n');
    if (std_out) {
        putc('\n', outfile);
        flushall();
    }
}

/*-------------------------------------------------------------------------*/

/*
| Ctrl-signal to inverting-register map, following the original logic.
```

```
*/
static const byte Ctrl_Val[16] = {
    0xb, 0xa, 0x9, 0x8, 0xf, 0xe, 0xd, 0xc,
    0x3, 0x2, 0x1, 0x0, 0x7, 0x6, 0x5, 0x4
};

void verbose_latch(byte val, byte ctrl)
{
    unsigned ctrl_enable = ctrl | LATCH_ENABLE_MASK;

    Output_Window(FALSE);
    outportb(D_Reg, val);                       /* Value onto data bus. */
    cprintf( EOL"Write %s data.", dec_bin(val, 8) );
    cpause();

    outportb(C_Reg, Ctrl_Val[ctrl] );           /* Select desired latch */
    cprintf("  ctrl: %s", dec_bin(ctrl, 4));
    cpause();

    outportb(C_Reg, Ctrl_Val[ctrl_enable] );    /* Enable the latch;    */
    cprintf("    : %s", dec_bin(ctrl_enable, 4));
    cpause();

    outportb(C_Reg, Ctrl_Val[ctrl] );           /* Drop the enable bit. */
    cprintf("    : %s\n", dec_bin(ctrl, 4));
}


/*-------------------------------------------------------------*\
| Reset & start conversion on ADC(s).                           |
| 93-05-25: No delay needed between reset & pulse --- ftn-call  |
|       overhead gives the single-pulsers plenty of time,       |
|       given the 78.1KHz/12.8us ADCclock  (meas. by Fluke 97)  |
\*-------------------------------------------------------------*/
void local_adc_strobe(byte adcbits)
{
    write_device(ADC_CONV, 0);          /* Reset the ADC single-pulsers */
    write_device(ADC_CONV, adcbits);    /* Pulse all desired /SC lines  */
}
```

```
/*-------------------------------------------------------------------------*\
| Expanded local version of ADC_sample...                                   |
\*-------------------------------------------------------------------------*/
word verbose_ADC_sample(byte adc_num)
{
    word tmp;

    adc_strobe( ADC_trgr(adc_num) );    /* Trigger the ADC sync/cvt/SC  */
    cpause();

    latch_ADC_out( adc_num );   /* Latch the ADC onto the Blue Bus...   */
    cpause();

    tmp = ADC_bus();            /* read 3 nybbles thru Control Port;    */
    cpause();

    reset_ADCs();

    return tmp;
}


/*---------------------------*\
| Intercept ctrl_C interrupts |
\*---------------------------*/
static int ctrl_C(void)
{
    flushall();
    audible(1100);
    show_prompt("  ARGghhhh...  ", LIGHTRED);
    gotoxy(1,25);
    return 0;
}


/*********************************************************\
*                                                         *
```

```
*       utility functions above, menu options below     *
*                                                       *
\*******************************************************/


/*********************\
* ATB-control options *
\*********************/

static const byte Addr_Map[16] = {
    0x0, 0x2, 0x1, 0x3, 0x8, 0xa, 0x9, 0xb,
    0x4, 0x6, 0x5, 0x7, 0xc, 0xe, 0xd, 0xf
};

#define address(addr)  \
    ( (Addr_Map[ (addr & 0xf0) >> 4 ] << 4)  |  Addr_Map[ addr & 0x0f ] )

static void set_Addr_bus(void)
{
    word addr;
    addr = get_word_num("device address?");
    Output_Window(FALSE);
    cprintf(EOL"%02x maps to ", addr);
    addr = address(addr);
    cprintf("%02x; addr: ", addr);
    verbose_latch(addr, A_LATCH);
    putch('\n');
}

/*-------------------------------------------------------------------------*/
static void set_busses(void)
{
    word addr, data;

    addr = get_word_num("device address?");
    data = get_word_num("(Low-1st) Data-bus value?");
    Output_Window(FALSE);
    cprintf( EOL"%s low", dec_bin(lo(data), 8) );
    cprintf( ", %s high", dec_bin(hi(data), 8) );
```

```
    cprintf( ";  Addr %s\n", dec_bin(addr, 8) );
    set_data( data );
    set_addr( addr );
}

/*-------------------------------------------------------------------------*/
static void do_c(void)
{
    word data;

    data = get_word_num("Control-register value?");
    Output_Window(FALSE);
    cprintf( EOL"%s maps to Ctrl pattern: ", dec_bin(data, 4) );
    data = Ctrl_Val[data];
    cputs(dec_bin(data, 4));
    putch('\n');
    outportb( C_Reg, data );
}

/*-------------------------------------------------------------------------*/
static void do_d(void)
{
    word data;
    data = get_word_num("Data-bus value?");
    set_data( data );
}

/*-------------------------------------------------------------------------*/
static void do_D(void)
{
    word data;
    data = get_word_num("Data-bus value to write low-1st?");
    Output_Window(FALSE);
    cputs(EOL"low:  "); verbose_latch(lo(data), D_LO_LATCH); cpause();
    cputs(EOL"high: "); verbose_latch(hi(data), D_HI_LATCH);
    putch('\n');
}

/*-------------------------------------------------------------------------*/
```

```
static void do_g(void)
{
    trigger_all_ADCs();
    show_prompt("trigger_all_ADCs() to convert.", NoColor);
}

/*-------------------------------------------------------------------*\
| Repetitively send values to the chip, and read a result.           |
\*-------------------------------------------------------------------*/
static void wr_loop(void)
{
    word src, rslt, dac_addr, adc_addr;
    char bfr[81];
    int ans;

    /*--------------------------*\
    | Set up for a particular DAC |
    \*--------------------------*/
    dac_addr = get_word_num("Input DAC?");
    src = get_word_num("Data to write?");

    /*----------------------*\
    | Identify the output ADC |
    \*----------------------*/
    adc_addr = get_word_num("ADC [0..5]?");

    if (std_out) {
        cputs(out_fmt_msg);
        ans = ('y' == getche());
    } else
        ans = 0;

    sprintf(bfr, EOL"DAC $%x  to  ADC $%x\n", dac_addr, adc_addr);
    Output_Window(TRUE);
    con_log_string(bfr);

    do {
        set_data( ~src );               /* flip the bits on the Red bus */
        write_device(dac_addr, src);
```

```
        trigger_DACs();                 /* trigger the DACs */
        u_time(1000);                   /* wait 1 milliseconds... */
        rslt = ADC_sample(adc_addr);

        cprintf("\r%c  %s --> ", marker(), dec_bin(src,12));
        cprintf( "%s  (%4u --> %4u)", dec_bin(rslt,12),  src, rslt );
        rslt_log(src, rslt, ans);
    } while (!kbhit());
    getch();
    newline();
}


/*-------------------------------------------------------------------------*\
| Output a succession of signals to one or more DACs, and read an ADC's |
| responses.                                                              |
\*-------------------------------------------------------------------------*/
typedef struct _addr_node {
    word addr;
    struct _addr_node *next;
} addr_node;

static void ramp_DACs(addr_node *addrs)
{
    word rampout, data, result, step;
    addr_node *this;
    char bfr[256], c;
    int ans, pos;

    rampout = get_word_num("output ADC [0..5]?");
    step = get_word_num("Stepsize ( >100 suggested )?");
    if (std_out) {
        cputs(out_fmt_msg);
        ans = ('y' == getche());
    } else
        ans = 0;

    this = addrs;
    pos = sprintf(bfr, EOL"signal DAC: $%x", this->addr);
```

```
    this = this->next;
    while ( (this != 0) && (pos < 243) ) {
        pos += sprintf(bfr+pos, ", $%x", this->addr);
        this = this->next;
    }
    sprintf(bfr+pos, "; Response ADC: $%x\n", rampout);

    show_prompt(step_msg, LIGHTBLUE);
    Output_Window(TRUE);
    con_log_string(bfr);
    c = getch();

    for (data = 0; data < NUM_COUNTS; data += step) {
        this =addrs;
        do {
            write_DAC(this->addr, data);
            this = this->next;
        } while (this != 0);
        trigger_DACs();
        u_time(1000);           /* Let the test circuit stabilize.. */
        result = ADC_sample( rampout );

        rslt_log(data, result, ans);
        if ( (1 != std_out) || ('C' != c) ) {
            cprintf(EOL"input: $%3x / %s " LBLOK " output: $%3x / ",
                        data, dec_bin(data,12), result);
            cputs(dec_bin(result,12));
        } else {
            cprintf("\r%c", marker());
        }
        if ('C' != c) {
            c = getch();
        }
    }
    newline();
    Prompt_Window();
}

/*-------------------------------------------------------------------------*\
```

```
| Ramp one DAC only.                                                    |
\*--------------------------------------------------------------------*/
static void ramp_1_DAC(void)
{
    addr_node one;
    one.next = 0;
    one.addr = get_word_num("Ramp DAC #?");
    ramp_DACs(&one);
}

/*--------------------------------------------------------------------*\
| Ramp multiple DACs.                                                   |
/*--------------------------------------------------------------------*/
static void ramp_multiple_DACs(void)
{
    addr_node head, *this, *last;
    int addr;

    this = &head;
    addr = get_word_num("Ramp DAC #?");
    do {
        this->addr = addr;
        addr = get_word_num("Another DAC #? [-1 to quit]");
        if (addr == -1) {
            break;
        }
        this->next = (addr_node *)malloc(sizeof(addr_node));
        this = this->next;
    } while (1);
    this->next = 0;

    ramp_DACs(&head);
    this = head.next;
    while (this != 0) {
        last = this;
        this = this->next;
        free(last);
    }
}
```

```
/*-----------------------------------------------------------------------*/
static void do_readADC(void)
{
    word addr, data;

    addr = get_word_num("ADC [0..5 or 10..15]?");
    if (addr <= 5) {
        data = ADC_sample( addr );
    } else {
        addr -= 10;
        data = verbose_ADC_sample( addr );
    }
    Output_Window(FALSE);
    cprintf(EOL"ADC %d reports $%03x (%s)\n",addr,data,dec_bin(data,12));
}

/*-----------------------------------------------------------------------*/
static void do_readblue(void)
{
    word blue;
    blue = ADC_bus();
    Output_Window(FALSE);
    cprintf( "Blue (ADC) bus reports %d/$%s/%%",
             blue,val_to_ASCII(blue,16,3) );
    cputs(val_to_ASCII(blue,2,12));
}

/*-----------------------------------------------------------------------*/
static void do_reset(void)
{
    reset_ADCs();
    show_prompt("Reset'd all ADCs.", NoColor);
}

/*-----------------------------------------------------------------------*/
static void do_t(void)
{
    show_prompt("issuing XFER/trigger to all DACs", NoColor);
```

```
    trigger_DACs();
}

/*-------------------------------------------------------------------------*/
static void do_T(void)
{
    DAC_convert_new();
    Prompt_Window();
    cprintf( "$%02x gets new-convert signal;  ", DAC_XFER );
    cpause();

    hold_DAC_conversion();
    Prompt_Window();
    cprintf( "$%02x gets hold signal.\r", DAC_XFER );
}

/*-------------------------------------------------------------------------*/
static void do_V(void)
{
    char bfr[82];
    float v;
    Output_Window(FALSE);
    audible(550);
    sprintf(bfr, "\rCurrently:  Vss=%.3f, Vdd=%.3f.\r\nNew Vss? ", Vss, Vdd);
    if ( con_read(bfr, bfr, 82) ) {
        putch('\r');
        v = atof(bfr+2);
        if (v <= 0) {
            Vss = v;
            adjust_Vss(Vss);
        } else {
            audible(1200);
            cputs("Bad Value!"EOL);
        }
    }
    audible(550);
    if ( con_read("\rNew Vdd? ", bfr, 82) ) {
        putch('\r');
        v = atof(bfr+2);
```

```
        if ( (10 >= v) && (v >= 0) ) {
            Vdd = v;
            adjust_Vdd( Vdd );
        } else {
            audible(1200);
            cputs("Bad Value!"EOL);
        }
    }
    sprintf(bfr, EOL"Now:  Vss=%.3f, Vdd=%.3f.\n", Vss, Vdd);
    con_log_string(bfr);
}

/*-------------------------------------------------------------------------*/
static void wrt_a_DAC(void)
{
    word addr, data;
    char bfr[81];

    addr = get_word_num("DAC address?");
    data = get_word_num("Data to write?");
    write_device(addr, data);
    Output_Window(FALSE);
    sprintf(bfr, EOL"DAC $%02x gets $%03x  (%s)\n",
            addr, data, dec_bin(data, 12));
    con_log_string(bfr);
}

/*-------------------------------------------------------------------------*/
static void w_t_DAC(void)
{
    wrt_a_DAC();
    do_t();
}

/*-------------------------------------------------------------------------*/
static void wrt_each_DAC(void)
{
    word addr, data;
    char bfr[82], dacstr[32];
```

```
    Output_Window(FALSE);
    cputs(EOL"Write each DAC.  Nil entry to stop."EOL);
    for (addr = 0x00; addr <= 0x23; ++addr) {
        sprintf(dacstr, "DAC $%02x value? ", addr);
        if (!con_read(dacstr, bfr, 82)) {
            return;
        }
        data = atoi(bfr+2);
        write_DAC( addr, data );

        sprintf( bfr, " $%02x gets %u (%04x, %s)"EOL,
                addr, data, data, dec_bin(data, 12) );
        con_log_string(bfr);
    }
}

/*---------------------------------------------------------------------*/
static void wrt_all_DACs(void)
{
    word addr, data;
    char bfr[82];

    Output_Window(FALSE);
    data = get_word_num("global DAC value?");
    for (addr = 0x00; addr <= 0x23; ++addr) {
        write_DAC( addr, data );               /* Set to uniform value */
    }
    sprintf( bfr, " DACS $00..$23 get %u (%04x, %s)"EOL,
                data, data, dec_bin(data, 12) );
    con_log_string(bfr);
}


/*---------------------------------------------------------------------*/
static void do_1(void)
{
    wrt_a_DAC();                /* Write a specified DAC */
    do_t();                     /* (trigger the DACs)    */
```

```
    do_readADC();               /* Read a specified ADC  */
}


/*******************************\
* Program-control, misc. options *
\*******************************/

/*---------------------------------------------------------------------*/
static void null_ftn(void) {}

static void do_null(void)
{
    char bfr[82], ans[6];
    unsigned long tym;
    unsigned long v;
    float utymf, vf;
    word blue;

    Output_Window(FALSE);
    if (!con_read("\rftn/bus/empty loop [fbe]?", ans, 6))
        return;

    con_read("How many loops ( >= 1 ) ? ", bfr, 82);
    vf = atof(bfr+2);
    v = atol(bfr+2);
    switch (ans[2]) {
      case 'f':
        cputs("Null_ftn():");
        tym = millisecs();
        do {
            null_ftn();
        } while (--v != 0);
        tym = millisecs() - tym;
        break;
      case 'b':
        cputs("Blue bus read:");
        tym = millisecs();
        do {
```

```
            blue = ADC_bus();
        } while (--v != 0);
        tym = millisecs() - tym;
        cprintf( "  Blue (ADC) bus reports %d/$%s/%%",
                blue, val_to_ASCII(blue,16,3) );
        cprintf("%s", val_to_ASCII(blue,2,12));
        break;
      case 'e':
        cputs("Empty loop:");
        tym = millisecs();
        do {
        } while (--v != 0);
        tym = millisecs() - tym;
        break;
    }
    utymf = 1000.0 * (float)tym;
    sprintf(bfr, EOL" %lu msec, %f usec/iteration\n", tym, (utymf/vf) );
    con_log_string(bfr);
}

/*-------------------------------------------------------------------*/
static void do_bases(void)
{
    char str[40], bfr[127];
    int pos;
    float val;
    long unsigned lua;

    Output_Window(FALSE);
    con_read("\rValue? ", str, 38);
    val = cvt_val(str+2);
    lua = (long unsigned)val;

    pos = sprintf(bfr, "%s = %#g " VBAR " %lu " VBAR " %%%s",
                str+2, val, lua, val_to_ASCII(lua, 2, 0));
    pos += sprintf((bfr+pos), " " VBAR " @%s", val_to_ASCII(lua, 8, 0));
    sprintf((bfr+pos), " " VBAR " $%s\n", val_to_ASCII(lua, 16, 0));
    clreol(); con_log_string(bfr);
}
```

```
/*---------------------------------------------------------------------*/
static void do_system(void)
{
    show_prompt("DOS shell...", LIGHTGRAY);
    textcolor(BLACK);
    system("");
    Prompt_Window();
}

/*---------------------------------------------------------------------*\
| Toy commands.                                                         |
\*---------------------------------------------------------------------*/
static void do_escape(void)
{
    hop_window(1, TRUE);        /* copyright notice... */
    cputs(title);
    audible(150);
    show_prompt("no escape.", BROWN);
}

static void do__(void)
{
    show_prompt("PLEASE DON'T DO THAT.", LIGHTRED);
}

/*---------------------------------------------------------------------*\
| Toggle the echo-to-output-file status.  The global variable "std_out" |
| keeps track of this status, for reference by all routines             |
\*---------------------------------------------------------------------*/
static void do_redirect(void)
{
    static char name_buf[81];
    char *name, *m;
    fpos_t len;

    name = name_buf + 2;
    Prompt_Window();
```

```
    if (std_out) {

        audible(220); audible(220);
        len = ftell(outfile);
        if (len) {
            fprintf(outfile, "\n# %s\n", time_now());
            fclose(outfile);
            m = "Closed";
        } else {
            fclose(outfile);
            unlink(name);
            m = "Discarded";
        }
        std_out = 0;

    } else {

        audible(700);
        if ( 0 == con_read("output filename? ", name_buf, 81) ) {
            name[0] = 0;
            m = "No";
        } else {
            if ( NULL == (outfile = fopen(name, "w")) ) {
                audible(990);
                m = "Can't fopen()";
            } else {
                std_out = 1;
                m = "Echoing to";
            }
        }
    }
    cprintf(EOL"%s file %s", m, name);
    File_Window();
}

/*---------------------------------------------------------------------*/
static void fix_noise(void)
{
    noisy = !noisy;
```

```
    audible(660);
}

/*---------------------------------------------------------------------*/

void draw_menu(void)
{
    hop_window(1, TRUE);        /* draw the menu... */
    cputs(menu);
}

/*********************************************************************/

int main(int argc, char **argv)
{
    unsigned i;
    char sec, ans;
    char *oldscreen;
    int ctrlbrk_setting = getcbrk();

    (void)argv;(void)argc;      /* suppress "never used" messages */

    setcbrk(1);
    ctrlbrk(ctrl_C);
    chirp(0);

    oldscreen = (char *)malloc(2 + 2*( (RIGHT-LEFT+1) * (BOTTOM-TOP+1) ));
    oldscreen[0] = (char)wherex();
    oldscreen[1] = (char)wherey();
    gettext(LEFT,TOP, RIGHT,BOTTOM, oldscreen+2);

    draw_frame();
    File_Window();
    hop_window(1, TRUE);        /* copyright notice... */
    cputs(title);
    Prompt_Window();

LOOP: {
        flushkeys();
```

```
        audible(440);
        sec = -1;       /* unlikely second (for a few years, anyway) */

        Time_Window();
        do {
            if (sec != (time_now())[24]) {
                gotoxy(2,1);
                textcolor(LIGHTGREEN); cputs(time_now());
                textcolor(YELLOW); cputs(" choice? ");
                textcolor(WHITE); sec = time_now()[24];
            }
        } while (!kbhit());
        ans = getche();
        Prompt_Window();

        if ( ('q'== ans) )
            goto QUIT;

        for (i = 0; i < n_choices; ++i) {
            if (ans == choice[i].tag) {
                choice[i].f();
                i += n_choices;
            }
        }
        if (n_choices == i) {
            textcolor(LIGHTRED);
            cprintf("'%c' is unknown.", ans);
            audible(1760); audible(440); audible(860); audible(1660);
        }

    } goto LOOP;

QUIT:
    if (std_out)
        do_redirect();

    audible(1000);
    window(1,1,80,25);
    puttext(LEFT,TOP, RIGHT,BOTTOM, oldscreen+2);
```

```
    gotoxy((int)oldscreen[0], (int)oldscreen[1]);
    free(oldscreen);
    setcbrk(ctrlbrk_setting);
    return 0;
}
/*------------------------------  END  --------------------------------*/
```

A.3 Utility Functions

These files contain functions unrelated to ATB operation, but used by the `ftntest` program for timekeeping, numeric conversions, and other "bells and whistles".

```
/*--------------------------------------------------------------------*\
|  CONVERTS.H --- Functions to convert values back and forth.          |
| 93-11-10  cvt_val() added; allows for conversion to floating point.  |
|           Calls numeric() if floating-point fails.                   |
| 93-04-22  legal_digits[] moved to here, as part of generalized       |
|           "spacers" support and use of lookup().  to_ascii() and     |
|           to_numer() also modified.  Set up machinery for maintaining |
|           allowable spacers as used by to_ascii() and to_numer().    |
| 93-03-25  Support conversion of "negative" strings to negative values |
\*--------------------------------------------------------------------*/
#if !defined(_CONVERTS_H)
#define _CONVERTS_H

#if !defined(NULL)
#   define NULL  0
#endif

#define ASCII_MAX_LEN   81    /* Longest possible string, +1 */
#define legal_digits    "0123456789abcdef"
#define legal_size      (sizeof(legal_digits) - 1)

        /* Convert a string to a floating-point value if possible;   *\
        \* if not, pass the string on to numeric().                  */
double cvt_val(char *strng);

        /* Determine the base of a # string, and convert it to an    *\
        \* integer.  '$', '@', '%' prefixes accepted.                */
long numeric(char *strng);

        /* Convert ASCII-string to integer, in the given base.       *\
        \* Tolerate underscores or commas, e.g., "a_4317", "123,456" */
long ASCII_val(char *strng, int base);

        /* Convert unsigned long integer to ASCII string with a      *\
        |* spacer char separating groups of digits:  viz., "1_0010", *|
```

```
        |* "7,324".  The first character in the "spacers" string is  *|
        |* used.  set_spacers() controls this; set a null string to  *|
        |* get no spacer.  'base' specifies the desired base of the  *|
        |* converted string.  The 'len' specifies *minimum* string   *|
        \* size.  Char string changes w/ each use.                   */
char *val_to_ASCII(unsigned long value, int base, unsigned len);

#define bin_dec(bits)           ASCII_val((bits), 2)
#define dec_bin(value, len)     val_to_ASCII(value, 2, len)
#define to_ascii(value, base)   val_to_ASCII(value, base, 0)

        /* Define acceptable spacer characters in inputs. */
char *set_spacers(char *list);

#define DEFAULT_SPACERS "_,'"
#if defined(_SPACER_C)
    char *spacers = DEFAULT_SPACERS;
    int   spacer_ct = sizeof(DEFAULT_SPACERS)-1;
#else
    extern char *spacers;
    extern int   spacer_ct;
#endif

/*------------------------------*\
| My version of index()/strpos() |
\*------------------------------*/
int lookup(int c, const char *list);
#define stringlen(STRING)  lookup(0,STRING)

void chirp(unsigned freq);

#define flushkeys() {while (kbhit()) getch();}
/*--------------------------------------------------------------------*/
#endif  /* _CONVERTS_H */
#if 0 /* LaTeX formatting commands here... */
```

```
#endif
/*--------------------------------------------------------------------*\
| timing.h --- microsecond and millisecond functions.                  |
| 93-03-20                                                             |
\*--------------------------------------------------------------------*/
#if !defined(_TIMING_H)
#define _TIMING_H

/*-------------------------------------------*\
| Day-date string in my preferred format, and   |
| timing functions in units of milliseconds.    |
\*-------------------------------------------*/
char *time_now(void);             /*  Current time-&-date char-string   */
unsigned long millisecs(void);    /*  Number of msec since 1970-01-01   */
char *duration(unsigned long ms); /*  Millisecs-as-minutes char-string  */

extern const char *day[7];
extern const char *month[12];

/*-------------------------------------------------------------*\
| u_time():  Burn microseconds.  Accuracy is limited by the     |
| machine speed; a 25MHz '386 may be good to one or a few       |
| microseconds.  However, times under a dozen usec or so will   |
| be dominated by the function-call overhead anyway.  (Still |
| better than the delay() function's millisecond resolution).   |
| Default calibration is for my generic 25MHz no-cache 386box;  |
|generally a recalibration should be done before use.   |
|                                                               |
| u_time_calibrate() allows recalibrating for a fixed time      |
| interval; u_calibrate() gives a 5-second recalibration run.   |
\*-------------------------------------------------------------*/
void u_time(float usec);
void u_time_calibrate(long unsigned cal_usecs);
#define u_calibrate()   u_time_calibrate(5000000L)

/*--------------------------------------------------------------------*/
#endif  /* _TIMING_H */
#if 0 /* LaTeX formatting commands here... */
```

```
#endif
/*-------------------------------------------------------------------------*\
| spacer.c --- Set up spacer characters for to_ascii() and to_numer()       |
|                                                                           |
| 93-04-22                                                                  |
\*-------------------------------------------------------------------------*/
#include <string.h>
#define _SPACER_C
#include "converts.h"

char *set_spacers(char *list)
{
    if (NULL != list)
        spacers = list;
    spacer_ct = strlen( spacers );
    return spacers;
}
#if 0 /* LaTeX formatting commands here... */
```

```
#endif
/*-------------------------------------------------------------------------*\
| TO_ASCII.C --- convert numeric values to ASCII strings.                   |
|                                                                           |
| Generate the char-string form of a supplied value, using the supplied     |
| base.  To improve readability, strings contain a supplied character       |
| (ex.: underscore) every 3 characters (bases 8, 10) or 4 characters        |
| (all other bases).  This can be suppressed by supplying a NULL            |
| character (0).                                                            |
|                                                                           |
| 'len' specifies the *minimum* length of the returned string;              |
| ASCII_MAX_LEN (in coverts.h) defines the maximum length.                  |
| NOTE:  The returned char string is changed on every call.                 |
|                                                                           |
| Example:  binary strings of the form "1_0101_0000_1001_0000"              |
|                                                                           |
| 92-11-03 - Replace dec_bin() with a call to val_to_ASCII                  |
| 93-02-02 - Jazz up the comments, discard the ifdef'd dec_bin(),           |
|            move ASCII_MAX_LEN to converts.h                               |
\*-------------------------------------------------------------------------*/
#include <stdlib.h>
#include <string.h>
#include "converts.h"

static char near S_String[ASCII_MAX_LEN];

char *val_to_ASCII(unsigned long value, int base, unsigned len)
{
    char near *s, spacer;
    int near spacing, chunk;

    if (base >= sizeof(legal_digits))   /* Error-check: requested */
        return NULL;                    /*   base out of range?   */

    spacer = spacers[0];
    if (spacer) {
        switch (base) {                 /* If a spacer character */
          case 8:                       /* is requested, how     */
          case 10:                      /* frequently should it  */
```

```
            chunk = 3;                  /* be added?            */
            break;
          default:
            chunk = 4;
        }
        spacing = chunk;
    }

    *(s = S_String + ASCII_MAX_LEN - 1) = 0;    /* NULL the end-of-string */
    do {
        if (spacer) {
            if (0 == spacing) {                 /* Time to add a */
                *(--s) = spacer;                /* spacer char.? */
                spacing = chunk;
            }
            --spacing;
        }

        *(--s) = legal_digits[(unsigned)(value%base)];  /* each digit... */
        value /= base;
        if (len)                                        /* At requested */
            --len;                                      /* length yet?  */
    } while ( (value || len) && (s > S_String) );

    return s;
}
/*------------------------------  END  --------------------------------*/
#if 0 /* LaTeX formatting commands here... */
```

```
#endif
/*-------------------------------------------------------------------*\
|  TO_NUMER.C --- convert ASCII strings into values.                  |
|       This code supports underscores or commas (for readability) in |
|       number strings; these separators are (the most common?) options |
|       for visual formatting that to_ascii() does.                   |
|                                                                     |
| 93-11-10 - cvt_val() for floating- & general integer-format operands. |
| 93-07-16 - numeric() strips leading whitespace.  ASCII_val() stops  |
|       when it hits nondigits, so these can parse a line.            |
| 93-04-22 - Remove strtol() usage, for no very good reason.          |
|       Hack up the treatment of number-string spacers, for mucho     |
|       generality via the new lookup() function.  Also no very       |
|       good reason.  Test code removed to a separate file.           |
| 93-03-25 - Handle negative numbers in the form                      |
|              [<sign>][<base-marker>]<digits>                        |
|            where <sign> and <base-marker> are optional.             |
| 92-11-03 - Abandon the redundant special cases.                     |
\*-------------------------------------------------------------------*/
#include "converts.h"

double cvt_val(char *strng)
{
    double v;
    if ( 0 == (v = atof(strng)) ) /* Does atof() return 0 on failure?? */
        v = numeric(strng);
    return v;
}

/*-------------------------------------------------------------------*\
| Determine the base of a # string, and convert it to an integer.     |
| 93-07-16 Strip leading blanks the UGLY way.  This is a good example |
|       of tail recursion, which is optimized into a "goto", yes that's |
| right, a "goto".  (cf. '-', which isn't tail recursive.) |
\*-------------------------------------------------------------------*/
long numeric(char *strng)
{
What_Kind:
    switch (*strng) {
```

```
      case '%':                                 /* binary w/underscores */
        return ASCII_val( strng+1, 2 );

      case '@':                                 /* octal w/underscores  */
        return ASCII_val( strng+1, 8 );

      case '$':                                 /* hex w/underscores    */
        return ASCII_val( strng+1, 16 );

      case '-':                                 /*  Recurses!           */
        return ( - numeric(strng+1) );          /*   Negative value...  */

      case '0':                                 /* C syntax:            */
        if ('x' == strng[1])
            return ASCII_val( strng+2, 16 );    /* 0x-hex, underscores  */
        else
            return ASCII_val( strng+1, 8 );     /* 0-octal, underscores */

      case '\t':
      case ',':
      case ' ':                                 /* Tail Recurses!       */
        ++strng;                                /* skip a whitespace,   */
        goto What_Kind;                         /* try again...         */

      default:                                  /* assume it's decimal! */
        return ASCII_val( strng, 10 );
    }
}

/*-------------------------------------------------------------------*\
| Convert an ASCII string to a long integer, in a given base.         |
| Bases through hexadecimal are accepted, using [0..9a..f] (or [A..F]). |
| A leading '-' indicates a negative value.  Strings may contain      |
| spacing characters; other characters terminate the conversion.      |
\*-------------------------------------------------------------------*/
long ASCII_val(char *strng, int base)
{
    unsigned long near value;
    register char near c;
```

```
    register int near v;
    int near neg;

    if ('-' == *strng) {
        ++strng;
        neg = 1;
    } else {
        neg = 0;
    }
    value = 0;
    while ( 0 != (c = *(strng++)) ) {
      /*-------------------------------------------*\
      | Look for legal digits, possibly uppercase... |
      \*-------------------------------------------*/
        if ( legal_size > (v = lookup(c, legal_digits))
             ||  legal_size > (v = lookup(c|0x20, legal_digits)) )
        {
            value = value*base + v;
            continue;
        }
      /*----------------*\
      | Cosmetic spacer? |
      \*----------------*/
        if (spacer_ct > lookup(c, spacers))
            continue;

      /*-----------------------------------------------------*\
      | If we got here, we must've hit the ASCII string's end. |
      \*-----------------------------------------------------*/
        break;
    }

    return (neg  ?  -value  :  value);
}
/*-----------------------------  END  -------------------------------*/
#if 0 /* LaTeX formatting commands here... */
```

```
#endif
/*-------------------------------------------------------------------------*\
| timeftns.c --- reinventing the wheel...                                   |
\*-------------------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include <time.h>
#include "timing.h"

/*-----------------------------------------------------*\
|  Return the number of milliseconds since 1970-01-01   |
\*-----------------------------------------------------*/
#include <sys/timeb.h>

unsigned long millisecs(void)
{
    struct timeb t;
    ftime(&t);
    return ( (unsigned long)1000*t.time + (unsigned long)t.millitm );
}

/*-----------------------------------------------------*\
|  convert milliseconds to minutes-etc. char. string.   |
|  Overwritten by each call...                          |
\*-----------------------------------------------------*/
#define MS_PER_HR  3600000L
#define MS_PER_MIN   60000L
#define MS_PER_SEC    1000L

static char minute[40];

char *duration(unsigned long ms)
{
    unsigned long near i;

    minute[0] = 0;
    if ( 0 != (i = ms / MS_PER_HR) ) {
        sprintf(minute, "%uh", i);
        ms %= MS_PER_HR;
```

```
    }
    if ( 0 != (i = ms / MS_PER_MIN) ) {
        sprintf(minute+strlen(minute), "%um", i);
        ms %= MS_PER_MIN;
    }
    i = ms / MS_PER_SEC;
    ms %= MS_PER_SEC;
    sprintf(minute+strlen(minute), "%lu.%03lus", i, ms);
    return minute;
}

/*-------------------------------------------------------*\
|  Return current-time-and-date character string.       |
|  Overwritten by each call...                          |
\*-------------------------------------------------------*/
const char *day[7] = {
    "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat",
};
const char *month[12] = {
    "Jan", "Feb", "Mar", "Apr", "May", "Jun",
    "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
};

static char time_string[26];

char *time_now(void)
{
    time_t tymt;
    struct tm *tym;
    tymt = time(NULL);
    tym = localtime(&tymt);
    sprintf(time_string,
                "%04u %3s %2d, %3s %2u:%02u:%02u",
                1900+tym->tm_year, month[tym->tm_mon], tym->tm_mday,
                day[tym->tm_wday], tym->tm_hour, tym->tm_min, tym->tm_sec);
    return time_string;
}

/*-------------------------------------------------------------------*/
```

```
#ifdef TEST_MILLI
void test_milli(void)
{
        long i;

        printf("TIME==>%s<==\n", time_now());
        puts(duration(millisecs()));
        for (i=0; i<15000; i +=1271) {
                puts(duration(i));
        }
}
#endif
/*------------------------------  END  ---------------------------------*/
#if 0 /* LaTeX formatting commands here... */
```

```
#endif
/*-------------------------------------------------------------------------*\
| utimer.c --- provide sub-millisecond delays via software loops.           |
| 93-03-20                                                                  |
\*-------------------------------------------------------------------------*/
#include <dos.h>
#include "timing.h"

static float near loops_per_usec = 0.562;  /* An informed starting guess... */

/*-----------------------------------------------------------------*\
| Burn the requested microseconds by spinning a null loop           |
| "enough" times, as determined by the "loops_per_usec"             |
| calibration factor.                                               |
\*-----------------------------------------------------------------*/
void u_time(float usec)
{
    register long unsigned near loops;
    loops = loops_per_usec * usec;
    if (!loops)
        return;
    do {} while (--loops);
}

/*-----------------------------------------------------------------*\
| Run a fixed number of loops, count the number of reported         |
| centiseconds between start and finish, and convert to a           |
| floating-point loops/microsecond calibration factor.              |
\*-----------------------------------------------------------------*/
void u_time_calibrate(long unsigned cal_usecs)
{
    struct time ts, tf;
    float near usecs;

    do {
        gettime(&ts);
        u_time(cal_usecs);
        gettime(&tf);
    } while (ts.ti_hour > tf.ti_hour);
```

```
    usecs = 1.0e4 * (float)(
        100L * ((3600L * ((long)tf.ti_hour - (long)ts.ti_hour)) +
                (  60L * ((long)tf.ti_min  - (long)ts.ti_min )) +
                (        ((long)tf.ti_sec  - (long)ts.ti_sec ))  )
        + (long)tf.ti_hund - (long)ts.ti_hund );
    loops_per_usec *= (float)cal_usecs / usecs;
}

/*-------------------------------------------------------------------------*/
#ifdef TEST_MICRO
#include <stdio.h>

void test_micro(void)
{
    struct time ts, tf;
    unsigned long sample;
    fprintf(stderr,"\nUTIMER starting calibration: %f\n", loops_per_usec);
    u_time_calibrate(20000000L);
    fprintf(stderr,"New loops per usec: %f\n", loops_per_usec);

    for (sample = 1.0e4; sample <= 1.0e7; sample *= 10.0) {
        gettime(&ts);
        u_time(sample);
        gettime(&tf);
        fprintf(stderr,
                "%lu microseconds ---\n"
                " start: %2u:%02u:%02u.%02u\n"
                "finish: %2u:%02u:%02u.%02u\n",
                sample,
                ts.ti_hour, ts.ti_min, ts.ti_sec, ts.ti_hund,
                tf.ti_hour, tf.ti_min, tf.ti_sec, tf.ti_hund);
    }
}
#endif
/*-----------------------------  END  ------------------------------*/
#if 0 /* LaTeX formatting commands here... */
```

```
#endif
#include <stdlib.h>
#include <dos.h>

/*-------------------------------------------------------------------------*\
| chirp --- make a controlled noise.                                        |
\*-------------------------------------------------------------------------*/
void chirp(unsigned freq)
{
    sound(0);  nosound();  sound(freq); delay(10); nosound();
}

/*-------------------------------------------------------------------------*\
| lookup                                                                    |
|       Find the position of a character in a string.                       |
|       Returns: 0-based position of character within string; or            |
|                Length of string if no match or character is '\0'; or      |
|                -1 if a null string is supplied.                           |
|                                                                           |
|       stringlen() can be implemented in terms of lookup, as:              |
|                                                                           |
|       #define stringlen(foo)  lookup( 0, foo )                           |
|                                                                           |
| 93-04-22-bob,mon.                                                         |
\*-------------------------------------------------------------------------*/
int lookup(int c, const char *list)
{
    int i, lpos;
    i = -1;

    if (NULL != list) {
        do {
            lpos = list[ ++i ];
            if (c == lpos)
                return i;
        } while (NULL != lpos);
    }
    return i;
}
```

```
/*---------------------------------------------------------------------*/
#if defined(TEST)
#include <stdio.h>
const char alphabet[] =
    "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";

int main(int argc, char **argv)
{
    printf("alphabet: %s\nsizeof alphabet = %d, stringlen(alphabet) = %d\n",
        alphabet, sizeof(alphabet), stringlen(alphabet));
    while (--argc) {
        printf("arg %d: '%s' is pos'n %d\n",
                argc, argv[argc], lookup(argv[argc][0], alphabet));
    }
    printf("'%c' & null string:  pos'n %d\n",  'A', lookup('A', 0));
    return 0;
}
#endif
/*---------------------------------------------------------------------*/
```

A.4 Makefile

This makefile expects the Borland "make v3.5" program. It makes the library of ATB routines, and the `ftntest` testing program.

```
#-----------------------------------------------------------------#
# makefile for ftntest.exe, atbs.lib, and ftntest.zip archive.
#
# 94-07-20 Distribution version. -bob,mon.
#-----------------------------------------------------------------#

#---------------------------------#
# Paths to the files under consideration...
# Fix these up as needed to find the tcc, tlink, & tlib programs
# and the \include and \lib subdirectories.
#
### ATBLIB = .\libatb
ATBLIB = .
WHERE = .
CPATH = c:\tc20

#---------------------------------#
# If a 'x87 coprocessor is available, change these to
# FPLIB = fp87
# FPFLAG = f87
# Change MDL only if a different memory model is really needed for
# some sick-puppy reason.
#
FPLIB = emu
FPFLAG = f
MDL = s

#-----------------------------------------------------------------#
# In theory, nothing beyond this point needs to be customized.
#-----------------------------------------------------------------#

CC = $(CPATH)\bin\tcc
TLINK = tlink
TLIB = tlib
```

```
INC = $(CPATH)\include
LIB = $(CPATH)\lib

CCFLAGS = -1 -O -Z -d -w -r -$(FPFLAG)
CFLAGS = $(CCFLAGS) -c

#---------------------------------#

FTNOBJS = ftntest.obj utility.obj
FTNLIBS = atbs.lib
FTNFLAGS = -N -w-stk

#---------------------------------#

ATBOBJS = $(ATBLIB)\atbpara.obj
ATBADDS =  -+atbpara

#---------------------------------#

ARCER = zip -u9rD
### ARCER = pkzip -ex -uorp
SRC_ARC = _ftntest.zip

#-----------------------------------------------------------------#

no_args:
    @echo make targets:
    @echo - "make clean" ......removes .obj files
    @echo - "make veryclean" ..removes .obj, .bak, .lst, .exe, .lib files
    @echo - "make archive" ....updates $(SRC_ARC)
    @echo - "make atb" ........make a library
    @echo - "make ftntest" ....make the portmanteau exercise program
    @echo .

#---------------------------------#

clean:
    -rm -f $(WHERE)\*.obj
    -rm $(WHERE)\*.bak
```

```
veryclean: clean
    -rm -i *.exe *.lib ../bin
    -rm -f *.obj *.bak *.lst *.lnk
    -rm -f $(WHERE)\*.obj $(WHERE)\*.bak $(WHERE)\*.lst

#--------------------------------#

archive:
    -rm *.obj *.bak *.map *.lst foo*.*
    -$(ARCER) $(SRC_ARC) mak* *.c *.h *.tc *.prj *.lib *.bat *.lst lib*.*
    @echo $(SRC_ARC) is updated.
    @echo .
    -@ls -l $(SRC_ARC)

#--------------------------------#

atb: atb$(MDL).lib

atb$(MDL).lib: $(ATBOBJS)
    $(TLIB) $(WHERE)\atb$(MDL) /E $(ATBADDS), $(WHERE)\atb$(MDL).lst

atbpara.obj: atbpara.h low-para.h timing.h converts.h

#--------------------------------#

ftntest: ftntest.exe

ftntest.exe: $(FTNOBJS) $(FTNLIBS)
    @echo Using $(FPLIB) f.p. library...
    $(TLINK) @&&!
/L$(LIB) c0s.obj $(FTNOBJS)
$&.exe
$&.map
$(FTNLIBS) $(FPLIB) maths cs
!
    @type $&.map
    @ls -l $<
    @-rm -f $&.map
```

```
ftntest.obj: ftntest.c low-para.h atbpara.h converts.h timing.h local.h
    $(CC) -m$(MDL) $(CFLAGS) $(FTNFLAGS) -I$(INC) $&

#--------------------------------#

utility.obj: utility.c local.h converts.h timing.h

#----------------------------------------------------------------#

.c.obj:
    $(CC) -m$(MDL) $(CFLAGS) -I$(INC) {$< }

#----------------------------------------------------------------#
```