Complex Embedded Applications on a COP8™ Device

You can use the COP8C Code Development System to create and manage a complex embedded application for a COP8™-based device. This Application Note explains how to create a user interface, play a sound file, and report on *COP8*™ ICU peripherals.

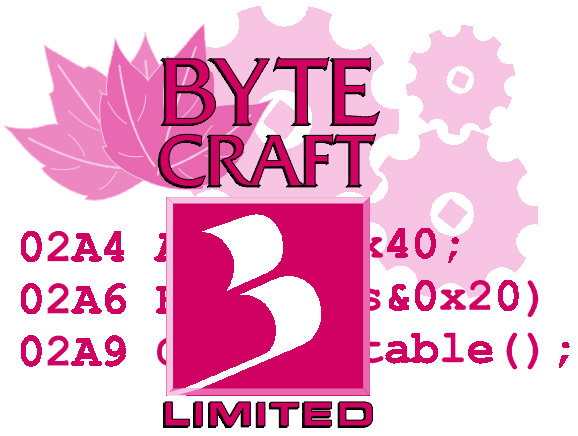
COP8C is a Code Development System from Byte Craft Limited. It includes an optimizing C compiler, the BCLink linker, device header files for all COP8™ devices, and libraries for many common embedded functions.

The new COP8FLASH device is suitable for complex control devices, with the memory to run a user-friendly LCD text-message interface. Programming such a user interface in assembly is an unnecessary chore: this application note includes a complex control application, written in C using Byte Craft's COP8C Code Development System.

For COP8C product information, see

http://www.bytecraft.com/icop.html

For COP8C technical support, please contact us at support@bytecraft.com



Byte Craft Limited

421 King Street North
Telephone: (519) 888-6911
Waterloo, Ontario
Fax: (519) 746-6751
Canada N2J 4E4

http://www.bytecraft.com

Sample Application

This sample application lets the user choose multiple functions from the two-line LCD display of the ICU, using two switches and a dial control. The functions include:

- Displaying the output of the *COP8*™ A/D converters.
- Playing sound samples through the ICU's piezo speaker.
- Displaying text messages.
- Toggles prompt sounds on and off.

It also communicates with the COP8FLASH Reference\Application Design console: you can make adjustments in the console, and then check them using the menu system.

Loading the Pre-compiled Binaries

This application note is accompanied in some distributions by

- `cop8capp.c`: the source code for the application described above.
- `cop8capp.cod`: the Byte Craft `.COD` file for use in the MetaLink in-system emulator.
- `cop8capp.hex`: a hex dump file for National's Betalite software.

If the file is not present, surf to **http://www.bytecraft.com** for more information and copies of the files.

If you are using a MetaLink in-system emulator:

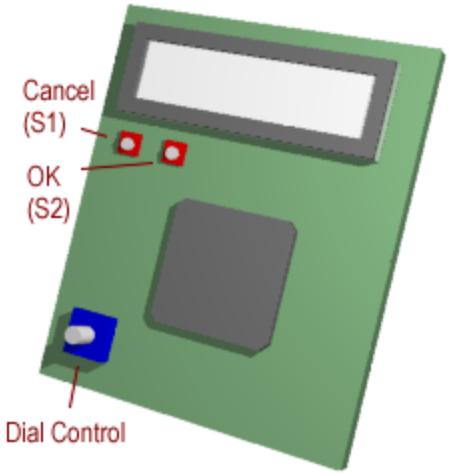
1. Load `cop8capp.cod`. This is a Byte Craft object file, which contains the executable and debugging information.
2. Click the `GO` button from the toolbar. The ICU should display a "splash screen" and begin playing a rhythm through the piezo speaker.

If you are using National's Betalite software, use `cop8capp.hex`.

This application was tested with the COP8FLASH Reference\Application Design console, version 1.0. Though later versions might be available, they are not guaranteed to work with this application. In online distributions of this application note and accompanying files, a tested version of the COP8FLASH console is available in the `Host_installation` subdirectory. Run the `setup.exe` program to install it.

Using the Application

The application acts much like a consumer control device: prompt messages appear on the LCD, and the user turns the dial control to scroll through options. The user can press switch 2 to select, and switch 1 to cancel.



The menu includes sections and subsections: the user can back out all the way by repeatedly pressing switch 1.

Preparing the sound files

A `.WAV` file is a stream of samples taken from an A/D converter in a sound capture device. The format of the file can vary, but this application uses 8kHz 8-bit mono `.WAV` files only.

The method we used to dump and rewrite the `.WAV` files as a C include file is in the comments of `cop8capp.c`. The wave data is obtained using a hex dump utility and an awk script. The sound sample is output by the awk script as a list of `db` statements in an inline assembly block. The awk script also writes a C `#define` statement that records the size of the sound table.

The main() Routine

`main()` contains the overall loop of the embedded application. After initialization, it enters an infinite loop that updates the display if necessary, services the two switches, performs sound effects for the dial control, and processes codes from the COP8FLASH console. Console commands are dispatched based on the command sent, but switch presses cause the program to pass control to `handle_menu_request()`.

The Menu System

The main loop updates the display based on the `refresh_display` flag. `__TIMERT0()` updates `item_selection`, using `menu_selection` to find the upper limit of any group of items.

In response to a switch press, `main()` calls `handle_menu_request()`. It uses the current values of `menu_selection` and `item_selection` to find the desired function from the selection tables (one for OK and one for CANCEL). Some entries are flagged with `ENTER_MENU` or `EXIT_MENU`: if the desired function is flagged, the value is actually a menu item; `menu_selection` and `item_selection` are reset from the table, effectively changing the menu level. If no flag is present, the value is a function; `handle_menu_request()` then calls the function.

Playing a sound sample

Playing a sound sample uses a timer-driven interrupt to iter through sound sample values, and a timer configured as a Pulse Width Modulator to drive the piezo speaker. The routine `enable_timer2()` configures timer 2 to invoke an interrupt at an 8kHz rate.

The function `__TIMERT2A()` services the interrupt. It uses the FLASH support routine `readbf()` to read a byte from ROM. It loads this value into the A and B autoload registers of timer 3.

Timer 3 counts down from A until it reaches 0, toggles the output pin connected to the amplifier and the piezo speaker, and repeats with B. Since each sample is a byte-sized unsigned integer, we load A with the sample and B with its complement.

The global variable `sample_index` tracks the position of the current sound output value, and the local variable `last_sample` keeps the position of the last sample in the playback routines.

Note that the sound coming directly from the piezo is quite harsh, but that the audio being played back is of higher quality. Connect an appropriate small speaker across the terminals of the piezo device, and you will hear a better quality audio output.

The Real-Time Clock and ID bytes

When first programmed, the real time clock will initialize to the time that the program was compiled. If the internal ID string has changed (this occurs when the clock is set), the clock will be reinitialized from the internally-stored time.

The real-time clock is driven by Timer 0. The `__TIMERT0()` function checks the dial encoder, and increments `tick_counter`. `update_clock()`, which is called by switch checks and sound playback, carries ticks over to the seconds, minutes, and hours counts.

To set the real-time clock from the dial encoder, the program preempts the menu item selection mechanism. An infinite loop watches for the cancel switch. Each time the dial encoder moves another position (updated in the `__TIMERT0()` interrupt), the program adds a second to the clock.

`store_id_clock()` uses the FLASH support routines to read, update, and rewrite a block of flash.

SOURCE CODE: cop8capp.c

```
/*      COP8C Code Development System
        Sample Application for National Semiconductor:
                Flash Device
        This code may be adapted for any purpose
        when used with the COP8C Code Development
        System.  No warranty is implied or given
        as to their usability for any purpose.

        (c) Copyright 2000 Byte Craft Limited
        421 King St.N., Waterloo, ON, Canada, N2J 4E4
        VOICE: 1 (519) 888 6911
        FAX  : 1 (519) 746 6751
        email: support@bytecraft.com

REVISION HISTORY
V1.00 AL 06/00 Initial version
*/
#pragma option f0 /* no page breaks in listing file */
#pragma option CALLMAP /* add call map to listing file */
#define __NO16BIT_NOMATH_ISR /* use simplified versions of SaveContext() */
#define __SHOW_LIBRARY
/*
 This sample application plays a .wav audio file on a speaker
 using the COP8's high speed PWM.

 The wave data is obtained using a hex dump utility and an awk script
   $ hexdump -x beat.wav > beak.dump
   $ awk -f wave.awk beat.dump > beat.c

#### wave.awk ####
        BEGIN { star_count = 0 }
        BEGIN {print "#asm"}
        BEGIN {print "beat_wave_table"}
        BEGIN {byte_count=0}
        BEGIN {start_skip_count=0}

        {
          if (star_count==0)
          {
            ++start_skip_count
          }
        }

        { # add three '*'s to the dump to mark
          # where you want to start playing
          # add one more '*' to mark the end
        if ($1 == "*")
        {
          ++star_count
        }
        }
        {
          if ( (star_count == 3) || (start_skip_count > 64) )
          {
            for (i=2;i<=NF;i++)
            {
              print "    db 0x" substr($i,1,2)
              print "    db 0x" substr($i,3,2)
              ++byte_count
              ++byte_count
            }
          }
        }

        END { print "#endasm"}
        END { print "#define BEAT_WAVE_TABLE_SIZE "  byte_count }
#### wave.awk ####

  The .WAV file used must be 8khz 8bit mono.

  The sound is synthesized by applying a new 8bit sample voltage every 125us
  (8khz).  Timer 2 is configured to generate an interrupt every 125us.
```

```
  calculations:

        The sample voltage needs to be changed at a rate of 8khz.
                1/8khz = 125us
        At 10Mhz every cycly requires 1us.  125us/1us = 125 = 0x007d

  Inside the timer 2 interrupt, timer 3 is set up to generate a PWM voltage on
  the speaker corresponding the the current wave sample.  The PWM generates a
  40khz carrier frequency.  The sample determines the duty cycle.

  calculations:

        The square wave period is 0x100 cycles.  At 10Mhz in high speed mode
        every cycle is 100ns.  This gives a 100ns * 0x100 = 25.6us ~= 40khz
        carrier frequency.  If the sample is 0x40.  Time on is 0x40*100ns.
        Time off is (0x100-0x40)*100ns.

*/
#include <dev\flashcop.h> /* Device Header File */
#include <lcd.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <startup.h>
#include <port.h>
#include <flash.h>
#include <cop8_isr.h>

/* library function aliases */
#define LCD_BUSY_CHECK() lcd_delay()
#define LCD_WRITE_DATA(BYTE) mwire_write(BYTE)
#define LCD_DATA_IN_CONTROL_OUT() DDR_WAIT()
#define LCD_DATA_OUT_CONTROL_OUT() DDR_WAIT()
#define LCD_CONTROL_SET_READ_MODE()
#define LCD_CONTROL_SET_WRITE_MODE()
#define LCD_E_PORT  PORTE
#define LCD_E_PIN   6
#define LCD_RS_PORT PORTE
#define LCD_RS_PIN  7
#define getch() uart_getch()
#define putch(CHAR) lcd_putch(CHAR)
#define itoa(VAL,DEST,RADIX) ui8toa(VAL,DEST,RADIX) /* unsigned 8 bit integer */
#define uart_clrscr() uart_putch('\f');
#define lcd_clrscr() lcd_send_control(LCDCLR)
#define clrscr() lcd_clrscr()

/* startup initialization */
#define CNTRL_STARTUP_VAL  0x08        /* init microwire */
#define PORTA_STARTUP_DATA 0b00000000  /* DEFAULT DATA */
#define PORTA_STARTUP_CONF   IIIIIIII  /* ALL WEEK PULL-UP INP */
#define PORTB_STARTUP_DATA 0b00000000  /* DEFAULT DATA */
#define PORTB_STARTUP_CONF   IIIOOOOO  /* SET TO LOW (OP AS VOLTAGE FOLLOWER) */
#define PORTC_STARTUP_DATA 0b00101011  /* DEFAULT DATA */
#define PORTC_STARTUP_CONF   IOIOIIOI  /* ROTARY ENCODER INP WKP,ANALOG POWER OUTPUT ON */
#define PORTD_STARTUP_DATA 0b00000000  /* DEFAULT DATA ALL LOW */
#define PORTE_STARTUP_DATA 0b11000000  /* DEFAULT DATA */
#define PORTE_STARTUP_CONF   OOIIOOII  /* SCL AND SDA OUTPUT LOW ALL OTHER HZ INP */
#define PORTF_STARTUP_DATA 0b00000011  /* DEFAULT DATA */
#define PORTF_STARTUP_CONF   IIIIOOII  /* PB INP WP, RADIO/IRDA SD LOW (OFF) */
#define PORTG_STARTUP_DATA 0b00000101
#define PORTG_STARTUP_CONF   IIOOOOIO  /* DEFAULT DATA */
#define PORTL_STARTUP_DATA 0b00101000  /* DEFAULT DATA */
#define PORTL_STARTUP_CONF   IOIOIOOO  /* LOW SPEED OSC MUST BE HZ TO WORK */
#define LIGHT_SWITCH_PORT    PORTLP
#define LIGHT_SWITCH_PIN     7

#include "beat.c"     /* beat wave table */
#include "greeting.c" /* greeting wave table */

enum wave_table_enum            { BEAT_WAVE           ,            GREETING_WAVE };
const long wave_tables[] =      { beat_wave_table     ,      greeting_wave_table };
const long wave_table_sizes[] = { BEAT_WAVE_TABLE_SIZE, GREETING_WAVE_TABLE_SIZE };
bit sound_enabled;

/* PWM reload value to get 8k interrupts
```

```
 * per second */
#define TIMER_RELOAD_8khz_LO  0x7D
#define TIMER_RELOAD_8khz_HI  0x00
unsigned long register sample_index;

#define TICKS_PER_SECOND 488
unsigned long register tick_counter;
const char id_clock[] = "ZBCL 56789012345";
char new_id_clock[sizeof(id_clock)];
char clock_hours;
char clock_minutes;
char clock_seconds;
char id1,id2,id3,id4; /* ID bytes */

/* hardware UART ring buffer */
char uart_receive_buffer[20];
char far * uart_receive_pointer_in;
char far * uart_receive_pointer_out;

#define BUTTON1_PIN   0
#define BUTTON2_PIN   1
#define BUTTON1_PORT  PORTFP
#define BUTTON2_PORT  PORTFP
#define BUTTON1       1
#define BUTTON2       2
#define BOTH_BUTTONS  3
#define CANCEL_BUTTON BUTTON1
#define OK_BUTTON     BUTTON2

/* host commands */
#define HOST_SET_TIME   0x01
#define HOST_GET_ADC    0x03
#define HOST_PLAY_AUDIO 0x05
#define HOST_SET_ID     0x0d
#define HOST_SEND_ID    0x0b

#define ENTER_MENU 0x80
#define EXIT_MENU 0x40
int register menu_selection;
int register item_selection;
bit refresh_display;
bit do_click;

#define MAIN_MENU_SIZE  4
#define ADC_MENU_SIZE   16
#define AUDIO_MENU_SIZE 3
#define TIME_MENU_SIZE  3
#define ABOUT_MENU_SIZE 3
const int menu_sizes[] = { MAIN_MENU_SIZE-1, ADC_MENU_SIZE-1, AUDIO_MENU_SIZE-1, TIME_MENU_SIZE-1,
ABOUT_MENU_SIZE-1 };
enum menu_function_index
{
        show_adc_index, play_wave_loop_index, play_wave_index, toggle_sound_index,
        show_time_index, store_time_index, set_time_index,
        show_compile_time_index, show_compile_date_index, show_ROM_id_index,
        not_handled_index
};
enum MENUS
{
        MAIN_MENU=0, ADC_MENU, AUDIO_MENU, TIME_MENU, ABOUT_MENU,
        NUMBER_OF_MENUES
};
char current_menu_item[NUMBER_OF_MENUES];
enum ITEMS
{
        SELECT_ADC_MENU=0,
        SELECT_AUDIO_MENU,
        SELECT_TIME_MENU,
        SELECT_ABOUT_MENU,
        VIEW_CHANNEL_0=0,
        VIEW_CHANNEL_1,
        VIEW_CHANNEL_2,
        VIEW_CHANNEL_3,
        VIEW_CHANNEL_4,
        VIEW_CHANNEL_5,
        VIEW_CHANNEL_6,
```

```
        VIEW_CHANNEL_7,
        VIEW_CHANNEL_8,
        VIEW_CHANNEL_9,
        VIEW_CHANNEL_10,
        VIEW_CHANNEL_11,
        VIEW_CHANNEL_12,
        VIEW_CHANNEL_13,
        VIEW_CHANNEL_14,
        VIEW_CHANNEL_15,
        PLAY_BEAT=0,
        PLAY_GREETING,
        TOGGLE_AUDIO,
        SHOW_TIME=0,
        STORE_TIME,
        SET_TIME,
        COMPILE_TIME=0,
        COMPILE_DATE,
        SHOW_ROM_ID
};
const char far * menu_strings[] = {
        "Main Menu",
        "View ADC",
        "Audio",
        "Time",
        "About",
        "View ADC",
        "Channel 0",
        "Channel 1",
        "Channel 2",
        "Channel 3",
        "Channel 4",
        "Channel 5",
        "Channel 6",
        "Channel 7",
        "Channel 8",
        "Channel 9",
        "Channel 10",
        "Channel 11",
        "Channel 12",
        "Channel 13",
        "Channel 14",
        "Channel 15",
        "Audio",
        "Play a Beat",
        "Play Greeting",
        "Toggle Sound",
        "Time",
        "Show Time",
        "Store Time",
        "Set Time",
        "About",
        "Compile Time",
        "Compile Date",
        "ID"
};
const char ok_menu_selection_table[] =
{
/*              MENU,             ITEM,                   FUNCTION HANDLER, */
            MAIN_MENU,    SELECT_ADC_MENU, ENTER_MENU|             ADC_MENU,
            MAIN_MENU,  SELECT_AUDIO_MENU, ENTER_MENU|           AUDIO_MENU,
            MAIN_MENU,   SELECT_TIME_MENU, ENTER_MENU|            TIME_MENU,
            MAIN_MENU,  SELECT_ABOUT_MENU, ENTER_MENU|           ABOUT_MENU,
             ADC_MENU,     VIEW_CHANNEL_0,                   show_adc_index,
             ADC_MENU,     VIEW_CHANNEL_1,                   show_adc_index,
             ADC_MENU,     VIEW_CHANNEL_2,                   show_adc_index,
             ADC_MENU,     VIEW_CHANNEL_3,                   show_adc_index,
             ADC_MENU,     VIEW_CHANNEL_3,                   show_adc_index,
             ADC_MENU,     VIEW_CHANNEL_4,                   show_adc_index,
             ADC_MENU,     VIEW_CHANNEL_5,                   show_adc_index,
             ADC_MENU,     VIEW_CHANNEL_6,                   show_adc_index,
             ADC_MENU,     VIEW_CHANNEL_7,                   show_adc_index,
             ADC_MENU,     VIEW_CHANNEL_8,                   show_adc_index,
             ADC_MENU,     VIEW_CHANNEL_9,                   show_adc_index,
             ADC_MENU,    VIEW_CHANNEL_10,                   show_adc_index,
             ADC_MENU,    VIEW_CHANNEL_11,                   show_adc_index,
             ADC_MENU,    VIEW_CHANNEL_12,                   show_adc_index,
```

```
            ADC_MENU,    VIEW_CHANNEL_13,                   show_adc_index,
            ADC_MENU,    VIEW_CHANNEL_14,                   show_adc_index,
            ADC_MENU,    VIEW_CHANNEL_15,                   show_adc_index,
          AUDIO_MENU,          PLAY_BEAT,              play_wave_loop_index,
          AUDIO_MENU,      PLAY_GREETING,                   play_wave_index,
          AUDIO_MENU,       TOGGLE_AUDIO,                toggle_sound_index,
           TIME_MENU,          SHOW_TIME,                   show_time_index,
           TIME_MENU,         STORE_TIME,                  store_time_index,
           TIME_MENU,           SET_TIME,                    set_time_index,
          ABOUT_MENU,       COMPILE_TIME,           show_compile_time_index,
          ABOUT_MENU,       COMPILE_DATE,           show_compile_date_index,
          ABOUT_MENU,        SHOW_ROM_ID,                 show_ROM_id_index
};

const char cancel_menu_selection_table[] =
{
/*               MENU,             ITEM,                  FUNCTION HANDLER, */
           MAIN_MENU,    SELECT_ADC_MENU,                  not_handled_index,
           MAIN_MENU,  SELECT_AUDIO_MENU,                  not_handled_index,
           MAIN_MENU,   SELECT_TIME_MENU,                  not_handled_index,
           MAIN_MENU,  SELECT_ABOUT_MENU,                  not_handled_index,
            ADC_MENU,     VIEW_CHANNEL_0, EXIT_MENU|              MAIN_MENU,
            ADC_MENU,     VIEW_CHANNEL_1, EXIT_MENU|              MAIN_MENU,
            ADC_MENU,     VIEW_CHANNEL_2, EXIT_MENU|              MAIN_MENU,
            ADC_MENU,     VIEW_CHANNEL_3, EXIT_MENU|              MAIN_MENU,
            ADC_MENU,     VIEW_CHANNEL_4, EXIT_MENU|              MAIN_MENU,
            ADC_MENU,     VIEW_CHANNEL_5, EXIT_MENU|              MAIN_MENU,
            ADC_MENU,     VIEW_CHANNEL_6, EXIT_MENU|              MAIN_MENU,
            ADC_MENU,     VIEW_CHANNEL_7, EXIT_MENU|              MAIN_MENU,
            ADC_MENU,     VIEW_CHANNEL_8, EXIT_MENU|              MAIN_MENU,
            ADC_MENU,     VIEW_CHANNEL_9, EXIT_MENU|              MAIN_MENU,
            ADC_MENU,    VIEW_CHANNEL_10, EXIT_MENU|              MAIN_MENU,
            ADC_MENU,    VIEW_CHANNEL_11, EXIT_MENU|              MAIN_MENU,
            ADC_MENU,    VIEW_CHANNEL_12, EXIT_MENU|              MAIN_MENU,
            ADC_MENU,    VIEW_CHANNEL_13, EXIT_MENU|              MAIN_MENU,
            ADC_MENU,    VIEW_CHANNEL_14, EXIT_MENU|              MAIN_MENU,
            ADC_MENU,    VIEW_CHANNEL_15, EXIT_MENU|              MAIN_MENU,
          AUDIO_MENU,          PLAY_BEAT, EXIT_MENU|              MAIN_MENU,
          AUDIO_MENU,      PLAY_GREETING, EXIT_MENU|              MAIN_MENU,
          AUDIO_MENU,       TOGGLE_AUDIO, EXIT_MENU|              MAIN_MENU,
           TIME_MENU,          SHOW_TIME, EXIT_MENU|              MAIN_MENU,
           TIME_MENU,         STORE_TIME, EXIT_MENU|              MAIN_MENU,
           TIME_MENU,           SET_TIME, EXIT_MENU|              MAIN_MENU,
          ABOUT_MENU,       COMPILE_TIME, EXIT_MENU|              MAIN_MENU,
          ABOUT_MENU,       COMPILE_DATE, EXIT_MENU|              MAIN_MENU,
          ABOUT_MENU,        SHOW_ROM_ID, EXIT_MENU|              MAIN_MENU
};

char str[128]; /* temporary string buffer */
/* This interrupt handling routine samples the dial position
 * and updates a tick counter for the real time clock
 */
void __TIMERT0(void)
{
        int register dial_position; /* position reading of dial */
        int register dial_previous; /* previous position reading */
        SaveContext();
        const char dial_position_states[] = { 0x0,0x1,0x9,0x8 };
        ICNTRL.T0PND=0;
        dial_position=PORTCP&0b1001; /* read dial */
        if( dial_position_states[dial_previous&3]!=dial_position)
        {
                if(dial_position_states[(dial_previous+1)&3]==dial_position)
                {
                        if(item_selection>0)
                        {
                                item_selection--;
                                refresh_display=1;
                                do_click=1;
                        }
                        dial_previous++;
                }
                else
                {
                        if(item_selection<menu_sizes[menu_selection])
                        {
```

```
                                item_selection++;
                                refresh_display=1;
                                do_click=1;
                        }
                        dial_previous--;
                }
        }
        tick_counter++;
        RestoreContext();
}

/* this interrupt handler changes the PWM output at a rate
 * of 8khz */
void __TIMERT2A(void) /* interrupt every 1/8khz = 125us */
{
        SaveContext();
        T2CNTRL.T2PNDA=0; /* clean pending flag */
        readbf(sample_index++); /* get a byte from ROM */
        T3RALO = AC; /* get a byte from ROM */
        T3RBLO = -T3RALO;
        RestoreContext();
}

/* UART receiver interrupt */
void __UARTR(void)
{
        SaveContext(); /* save compiler temps */

        *uart_receive_pointer_in=RBUF; /* get byte from receiver */

        /* add byte to buffer */
        if( uart_receive_pointer_in == (uart_receive_buffer+sizeof(uart_receive_buffer)-1) )
                uart_receive_pointer_in=uart_receive_buffer;
        else
                uart_receive_pointer_in++;

        /* check for overrun */
        if( uart_receive_pointer_in == uart_receive_pointer_out )
        {
                if( uart_receive_pointer_out == (uart_receive_buffer+sizeof(uart_receive_buffer)-1) )
                        uart_receive_pointer_out=uart_receive_buffer;
                else
                        uart_receive_pointer_out++;
        }
        RestoreContext(); /* restore compiler temps */
}

/* ignore all unexpected interrupts */
void __SWI(void){}
void __EXT(void){}
void __TIMER1A(void){}
void __TIMER1B(void){}
void __MICRO(void){}
void __UARTT(void){}
void __TIMERT2B(void){}
void __TIMERT3A(void){}
void __TIMERT3B(void){}
void __PORTL(void){}
void __VIS(void){}

void update_clock(void)
{
        unsigned long temp;
        PSW.GIE=0;
        temp = tick_counter;
        PSW.GIE=1;
        if(temp>TICKS_PER_SECOND)
        {
                PSW.GIE=0;
                tick_counter-=TICKS_PER_SECOND;
                PSW.GIE=1;
                clock_seconds++;
        }
        else
        if(clock_seconds>=60)
        {
```

```
                clock_seconds-=60;
                ++clock_minutes;
        }
        else
        if(clock_minutes>=60)
        {
                clock_minutes-=60;
                ++clock_hours;
        }
        else
        if(clock_hours>=24)
                clock_hours-=24;
}

void uart_putch(char ch)
{
        while(ENU.TBMT==0); /* wait while transmission in progress */
        TBUF=ch;
}

char kbhit(void)
{
        /* check for chacter in buffer */
        if(uart_receive_pointer_in!=uart_receive_pointer_out)
                return(1);
        else
                return(0);
}

char uart_getch(void)
{
        char ch;
        while(uart_receive_pointer_in==uart_receive_pointer_out); /* wait for a character */
        ch=*uart_receive_pointer_out; /* get character from buffer */

        /* remove character from buffer */
        if( uart_receive_pointer_out == (uart_receive_buffer+sizeof(uart_receive_buffer)-1) )
                uart_receive_pointer_out=uart_receive_buffer;
        else
                uart_receive_pointer_out++;
        return(ch)
}

/* read_button's <button> parameter can be BUTTON1, BUTTON2 or BOTH_BUTTONS
   if <button> is not pressed 0 is returned
   if <button> is pressed 1 is returned when it is released */
char read_button(char button)
{
        char result=0;
        char debounce_temp=0x80;
        update_clock();
        if(!BUTTON1_PORT.BUTTON1_PIN || !BUTTON2_PORT.BUTTON2_PIN) /* if a button is pressed */
        {
                while(--debounce_temp);
                switch(button) /* switch on which button state to check */
                {
                        case BUTTON1: /* return 1 if button 1 is pressed */
                                while(!BUTTON1_PORT.BUTTON1_PIN && BUTTON2_PORT.BUTTON2_PIN)
                                        result=1;
                                break;
                        case BUTTON2: /* return 1 if button 2 is pressed */
                                while(!BUTTON2_PORT.BUTTON2_PIN && BUTTON1_PORT.BUTTON1_PIN)
                                        result=1;
                                break;
                        case BOTH_BUTTONS: /* return 1 if both buttons are pressed */
                                while(!BUTTON1_PORT.BUTTON1_PIN && !BUTTON2_PORT.BUTTON2_PIN)
                                        result=1;
                                break;
                }
                if(result!=0)
                        while(--debounce_temp);
        }
        return(result);
}
```

```
/* menu actions that do nothing call this function */
void not_handled(void)
{
}

/* turn on or off the click when menu items are scrolled */
void toggle_sound(void)
{
        sound_enabled=!sound_enabled;
}

void enable_timer2(void)
{
        /* interrupt at a rate of 8khz */
        T2RBLO=T2RALO = TIMER_RELOAD_8khz_LO;
        T2RBHI=T2RAHI = TIMER_RELOAD_8khz_HI;
        T2CNTRL.T2C1 = 0; /* autoreload */
        T2CNTRL.T2C2 = 0;
        T2CNTRL.T2C3 = 1; /* don't toggle a pin */
        T2CNTRL.T2PNDA = 0; /* clean pending flag */
        T2CNTRL.T2C0 = 1; /* start counting */
        T2CNTRL.T2ENA = 1; /* enable timer 1 interrupt */
}

void disable_timer2(void)
{
        T2CNTRL.T2ENA = 0; /* disable timer 1 interrupt */
        T2CNTRL.T2C0 = 0; /* stop counting */
}

void enable_pwm(void)
{
        T3RAHI = T3RBHI = 0;
        HSTCR.T3HS = 1; /* high speed */
        T3CNTRL = 0xB0; /* start PWM */
}

void enable_low_speed_pwm(void)
{
        T3RAHI = T3RBHI = 0;
        HSTCR.T3HS = 0; /* low speed */
        T3CNTRL = 0xB0; /* start PWM */
}


void disable_pwm(void)
{
        T3CNTRL = 0x00; /* stop PWM */
        HSTCR.T3HS = 1;
}

void play_wave(void)
{
        unsigned long last_sample;
        sample_index=wave_tables[item_selection];
        last_sample = wave_tables[item_selection]+wave_table_sizes[item_selection];
        enable_timer2(); /* interrupt for sample rate timing */
        enable_pwm(); /* output sample timing */
        while(sample_index<last_sample) /* wait until all samples are played */
                update_clock();
        disable_pwm();
        disable_timer2();
}

void play_a_wave(char wave_to_play)
{
        unsigned int temp_item;
        temp_item=item_selection;
        item_selection=wave_to_play;
        play_wave();
        item_selection=temp_item;
}
```

```
void play_wave_loop(void)
{
        unsigned long last_sample;
        enable_timer2(); /* interrupt for sample rate timing */
        enable_pwm(); /* interrupt for sample output timing */
        sample_index=wave_tables[BEAT_WAVE];
        last_sample = wave_tables[BEAT_WAVE]+wave_table_sizes[BEAT_WAVE];
        while(1)
        {
                if(sample_index>last_sample)
                        sample_index=wave_tables[BEAT_WAVE];
                if(read_button(CANCEL_BUTTON)) /* keep playing until cancel button is pressed */
                {
                        disable_pwm();
                        disable_timer2();
                        return;
                }
        }
}

void show_adc(void)
{
        const char progress_table[] =  { '/','-','|','-' };
        unsigned int progress=0;
        clrscr();
        ENAD.PSC = 1;
        ENAD&=0x0f;
        ENAD|=item_selection<<4;
        puts("channel: ");
        itoa(item_selection,str,10);
        puts(str);
        while(1)
        {
                ENAD.ADBSY=1;
                while(ENAD.ADBSY);
                ENAD.ADBSY=1;
                lcd_gotoXY(10,1);
                putch(progress_table[progress]);
                progress=(progress+1)&3;
                int i=0;
                int j=10;
                while(--j)
                        while(--i)
                                if(read_button(CANCEL_BUTTON)) /* cancel button exits */
                                        return;
                lcd_gotoXY(0,1);
                puts("   ");
                while(ENAD.ADBSY);
                itoa(ADRSLTH,str,0x10);
                puts(str);
                itoa(ADRSLTL,str,0x10);
                puts(str);
        }
}

void show_menu(void)
{
        char i;
        char string_index;
        if(refresh_display)
        {
                clrscr();
                string_index=0;
                for(i=0;i<menu_selection;i++)
                {
                        string_index+=menu_sizes[i]+2;
                }
                puts(menu_strings[string_index]);
                lcd_gotoXY(0,1);
                string_index+=item_selection+1;
                puts(menu_strings[string_index]);
                refresh_display=0;
        }
}
```

```
char light_switch(void) /* return 1 if switch changed */
{
        static int previous = 0;
        if(previous != LIGHT_SWITCH_PORT.LIGHT_SWITCH_PIN) /* light switch toggle? */
        {
                previous = LIGHT_SWITCH_PORT.LIGHT_SWITCH_PIN;
                return(1);
        }
        return(0);
}


void store_time(void)
{
        PGMTIM=0x7B;
        block_readf(id_clock&0xff80,128,str);
        page_erase(id_clock);
        str[(id_clock&0x7f)+0]='Q';
        str[(id_clock&0x7f)+1]=id1;
        str[(id_clock&0x7f)+2]=id2;
        str[(id_clock&0x7f)+3]=id3;
        str[(id_clock&0x7f)+4]=id4;
        str[(id_clock&0x7f)+5]=clock_hours;
        str[(id_clock&0x7f)+6]=clock_minutes;
        str[(id_clock&0x7f)+7]=clock_seconds;
        block_writef(str+0x00,16,(id_clock&0xff80)+0x00);
        block_writef(str+0x10,16,(id_clock&0xff80)+0x10);
        block_writef(str+0x20,16,(id_clock&0xff80)+0x20);
        block_writef(str+0x30,16,(id_clock&0xff80)+0x30);
        block_writef(str+0x40,16,(id_clock&0xff80)+0x40);
        block_writef(str+0x50,16,(id_clock&0xff80)+0x50);
        block_writef(str+0x60,16,(id_clock&0xff80)+0x60);
        block_writef(str+0x70,16,(id_clock&0xff80)+0x70);
}

void host_set_clock(void)
{
        (void)uart_getch(); /* ignore size byte */
        clock_hours=uart_getch();
        clock_minutes=uart_getch();
        clock_seconds=uart_getch();
        uart_putch(0x02); /* ack */
        uart_putch(0x06); /* number of bytes */
        uart_putch(0x00); /* ok */
        uart_putch(clock_hours);
        uart_putch(clock_minutes);
        uart_putch(0); /* always send 0 seconds */
        store_time();
}

void host_read_ad(void)
{
        {
                char channel;
                (void)uart_getch(); /* ignore size byte */
                channel = uart_getch(); /* get the channel */
                uart_putch(0x04); /* ack */
                uart_putch(0x06); /* size */
                uart_putch(0x00); /* ok */
                ENAD.PSC = 1;
                ENAD&=0x0f;
                ENAD|=channel<<4;
                ENAD.ADBSY=1;
                while(ENAD.ADBSY);
                ENAD.ADBSY=1;
                while(ENAD.ADBSY);
        }
        {
                char temp;
                temp = (ADRSLTH>>6);
                uart_putch(temp); /* high result */
                temp = (ADRSLTH<<2) | (ADRSLTL>>6);
                uart_putch(temp); /* low result */
                uart_putch(1); /* gain */
        }
}
```

```
const unsigned int tones[] = /* table of frequencies for PWM */
        {
              0x16,0x31,  //5681,   /* 440Hz A  */
              0x14,0xf4,  //5364,   /* 466Hz B  */
              0x13,0xc4,  //5060,   /* 494Hz C  */
              0x12,0xac,  //4780,   /* 523Hz D  */
              0x11,0xa0,  //4512,   /* 554Hz E  */
              0x10,0xa2,  //4258,   /* 587Hz F  */
              0x0f,0xb3,  //4019,   /* 622Hz G  */
              0x0e,0xd1,  //3793,   /* 659Hz A1 */
              0x0d,0xfd,  //3581,   /* 698Hz B1 */
              0x0d,0x32,  //3378,   /* 740Hz C1 */
              0x0c,0x74,  //3188,   /* 784Hz D1 */
              0x0b,0xc0,  //3008,   /* 831Hz E1 */
              0x0b,0x18,  //2840    /* 880Hz F1 */
              0x16,0x31,  //5681,   /* 440Hz A  */
              0x14,0xf4,  //5364,   /* 466Hz B  */
              0x13,0xc4   //5060,   /* 494Hz C  */
        };

void play_tone(char tone, char tone_length)
{
        unsigned int i,j;
        enable_low_speed_pwm();
        j=0x80;
        i=0;
        if(tone>(sizeof(tones)/2))
        {
                T3RBLO=T3RALO=0x31;
                T3RBHI=T3RAHI=0x16;
                j=10;
                while(--j)
                        while(--i);
        }
        else
        {
                T3RBLO=T3RALO=tones[tone+tone+1]; /* second byte of tone */
                T3RBHI=T3RAHI=tones[tone+tone]; /* first byte of tone */
                do
                {
                        while(--j)
                                while(--i);
                        if(read_button(CANCEL_BUTTON)) break;
                }
                while(tone_length--);
        }
        disable_pwm();
}

void host_play_sound(void)
{
        char number_of_tones;
        char tone;
        char tone_length;
        uart_putch(0x06); /* ack */
        uart_putch(0x03); /* size */
        uart_putch(0x00); /* ok */
        number_of_tones = uart_getch()-2;
        while(number_of_tones)
        {
                tone_length=uart_getch();
                tone=uart_getch();
                number_of_tones-=2;
                play_tone(tone,tone_length);
        }
}

void host_set_id(void)
{
        (void)uart_getch(); /* ignore size byte */
        id1=uart_getch();
        id2=uart_getch();
        id3=uart_getch();
        id4=uart_getch();
        uart_putch(0x0e); /* ack */
```

```
        uart_putch(0x07); /* size */
        uart_putch(0x00); /* ok */
        uart_putch(id1);
        uart_putch(id2);
        uart_putch(id3);
        uart_putch(id4);
        store_time();
}

void host_send_id(void)
{
        (void)uart_getch(); /* ignore size byte */
        uart_putch(0x0E); /* ack */
        uart_putch(0x07); /* size */
        uart_putch(0x00); /* ok */
        uart_putch(id1); uart_putch(id2); uart_putch(id3); uart_putch(id4);
}

void show_time(void)
{
        clrscr();
        while(1)
        {
                if(read_button(CANCEL_BUTTON))
                        return;
                itoa(clock_hours,str,10);
                puts(str);
                putch(':');
                itoa(clock_minutes,str,10);
                puts(str);
                putch(':');
                itoa(clock_seconds,str,10);
                puts(str);
                lcd_gotoXY(0,0);
        }
}

void set_time(void)
{
        char save_item_selection;
        save_item_selection=item_selection;
        item_selection=0;
        clrscr();
        while(1)
        {
                if(read_button(CANCEL_BUTTON))
                {
                        item_selection=save_item_selection;
                        return;
                }
                if(item_selection!=0)
                {
                        item_selection=0;
                        clock_seconds+=60;
                }
                itoa(clock_hours,str,10);
                puts(str);
                putch(':');
                itoa(clock_minutes,str,10);
                puts(str);
                putch(':');
                itoa(clock_seconds,str,10);
                puts(str);
                lcd_gotoXY(0,0);
        }
}

void show_compile_time(void)
{
        clrscr();
        puts("Compile Time");
        lcd_gotoXY(0,1);
        puts(__TIME__);
        while(read_button(CANCEL_BUTTON)==0);
}
```

```
void show_compile_date(void)
{
        clrscr();
        puts("Compile Date");
        lcd_gotoXY(0,1);
        puts(__DATE__);
        while(read_button(CANCEL_BUTTON)==0);
}

void show_ROM_id(void)
{
        clrscr();
        puts("ROM ID");
        lcd_gotoXY(0,1);
        memcpy(str,id_clock,sizeof(id_clock));
        putch(str[1]); putch(str[2]); putch(str[3]); putch(str[4]);
        while(read_button(CANCEL_BUTTON)==0);
}

void mwire_write(char out_byte)
{
                SIOR=out_byte;
                PSW.BUSY=1;
                while(PSW.BUSY);
}

void lcd_delay(void)
{
        char j=0;
        while(--j);
}

const void (* menu_function[])(void) =
{
        show_adc, play_wave_loop, play_wave, toggle_sound, show_time,
        store_time, set_time, show_compile_time, show_compile_date,
        show_ROM_id, not_handled
};

void handle_menu_request(char far * table_ptr)
{
        enum menu_function_index function_index;
        function_index=0; /* default to first item */
        do
        {
                if( *table_ptr == menu_selection )
                {
                    if( *(table_ptr+1) == item_selection )
                    {
                        function_index = *(table_ptr+2);
                        break;
                    }
                }
                table_ptr += 3;
        }
        while(1);
                /* this code will never be executed
                 * the purpose is to make sure these functions
                 * are added to the program call tree */
                char imposible=0;
                if(imposible)
                {
                        show_adc();
                        play_wave();
                        play_wave_loop();
                        toggle_sound();
                        show_time();
                        store_time();
                        set_time();
                        show_compile_time();
                        show_compile_date();
                        show_ROM_id();
                        not_handled();
                }
        if(function_index&ENTER_MENU)
        {
```

```
                current_menu_item[menu_selection]=item_selection;
                menu_selection = function_index&(~ENTER_MENU);
                item_selection = current_menu_item[menu_selection];
        }
        else if( function_index&EXIT_MENU )
        {
                current_menu_item[menu_selection]=item_selection;
                menu_selection = function_index&(~EXIT_MENU);
                item_selection = current_menu_item[menu_selection];
        }
        else
                (*menu_function[function_index])();
}

void uart_init(void)
{
        ENU  = 0b10100000; /* even parity 8 bit frame */
        ENUR = 0b00000000; /* clear just in case */
        ENUI = 0b00100010; /* one stop async, baud/psw as clock */
        PSR  = 0b11001000; /* baud=9600, p=13, N=9 */
        BAUD = 0b00001001;
        uart_receive_pointer_in=uart_receive_buffer;
        uart_receive_pointer_out=uart_receive_buffer;
}

void dial_init(void)
{
        ITMR.ITSEL2 = 0;
        ITMR.ITSEL2 = 0;
        ITMR.ITSEL2 = 0;
        ICNTRL.T0PND = 0;
        ICNTRL.T0EN = 1;
        do_click = 0;
        /* select the first menu item in every menu */
        menu_selection=NUMBER_OF_MENUES;
        do
                current_menu_item[--menu_selection]=0;
        while(menu_selection!=0);
        refresh_display = 1;
        /* wait for timer 0 interrupt to find dial position */
        int i=0;
        int j=0;
        while(--i)
                while(--j);
        /* make current dial position item_selection 0 */
        item_selection = 0;
}

void clock_init(void)
{
        readbf(id_clock+1);
        id1=AC;
        readbf(id_clock+2);
        id2=AC;
        readbf(id_clock+3);
        id3=AC;
        readbf(id_clock+4);
        id4=AC;
        if(strcmp(id_clock,"ZBCL 56789012345\0")==0)
        {
                /* initialize with compile time */
                clock_hours=((__TIME__[0]-'0')*10)+(__TIME__[1]-'0');
                clock_minutes=((__TIME__[3]-'0')*10)+(__TIME__[4]-'0');
                clock_seconds=((__TIME__[6]-'0')*10)+(__TIME__[7]-'0');
        }
        else
        {
                readbf(id_clock+5);
                clock_hours=AC;
                readbf(id_clock+6);
                clock_minutes=AC;
                readbf(id_clock+7);
                clock_seconds=AC;
        }
}
```

```
void show_intro(void)
{
        puts("   BYTE CRAFT");
        lcd_gotoXY(0,1);
        puts("Hello & Welcome");
        sound_enabled=1;
        play_a_wave(PLAY_GREETING);
        char i,j;
        i=0; j=0;
        while(--i)
        {
                while(--j); while(--j);
                if(read_button(OK_BUTTON)||read_button(CANCEL_BUTTON))
                        break;
        }
}

void main(void)
{
        PSW.GIE=1; /* enable interrupts */
        clock_init();
        lcd_init();
        show_intro();
        dial_init();
        uart_init();
        while(1)
        {
                show_menu();
                if(read_button(OK_BUTTON))
                {
                        handle_menu_request(ok_menu_selection_table);
                        refresh_display=1;
                }
                if(read_button(CANCEL_BUTTON))
                {
                        handle_menu_request(cancel_menu_selection_table);
                        refresh_display=1;
                }
                if(do_click)
                {
                        if(sound_enabled)
                                play_tone(30,0);
                        do_click=0;
                }
                if(kbhit()) /* if a command has been received */
                {
                        lcd_gotoXY(11,0);
                        puts("Host");
                        switch(uart_getch()) /* get the command */
                        {
                                case HOST_SET_TIME:
                                        host_set_clock(); break;
                                case HOST_GET_ADC:
                                        host_read_ad(); break;
                                case HOST_PLAY_AUDIO:
                                        host_play_sound(); break;
                                case HOST_SET_ID:
                                        host_set_id(); break;
                                case HOST_SEND_ID:
                                        host_send_id(); break;
                                default: /* invalid command */
                                        while(kbhit()) /* empty buffer */
                                                (void)uart_getch();
                        }
                        lcd_gotoXY(11,0);
                        puts("    ");
                }
        }
}
```

421 King Street North • Waterloo, Ontario • Canada N2J 4E4
Telephone: (519) 888-6911 • Fax: (519) 746-6751
Printed in Canada • July, 2000

Byte Craft Limited