

```

//*****
/// @file      main.c
/// @author    Jay Convertino (electrobs@gmail.com)
/// @brief     Military Time Clock program, a 24 hour clock.
/// @details   This program uses ifs for its time keeping. This is done to reduce
///            the time needed to execute and instruction. Divides and by
///            extension mod, need many instruction cycles to complete.
///            Ifs and compares are usually faster but not as clean. For such
///            a low resource micro-controller a bit more code space was preferred
///            vs longer execution time. In addition, the decision to to have so
///            much code in the ISRs is ill-advised. In this case with careful
///            testing this operates well and doesn't present a problem.
///
/// @copyright Copyright 2022 Johnathan Convertino
///
/// license: MIT
///
/// Permission is hereby granted, free of charge, to any person obtaining a copy
/// of this software and associated documentation files (the "Software"), to deal
/// in the Software without restriction, including without limitation the rights
/// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
/// copies of the Software, and to permit persons to whom the Software is
/// furnished to do so, subject to the following conditions:
///
/// The above copyright notice and this permission notice shall be included in
/// all copies or substantial portions of the Software.
///
/// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
/// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
/// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
/// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
/// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
/// FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
/// IN THE SOFTWARE.
//*****

/// @brief ATMEGA 89s51 specific header, has a 3rd timer.
#include <at89x51.h>
/// @brief standard int for uints
#include <stdint.h>

/// @def Timer 0 high reg for 12 MHz milliseconds count
#define TH0_START 0xFC
/// @def Timer 1 low reg for 12 MHz milliseconds count
#define TL0_START 0x18

/// @def Timer 1 high reg for 2 Hz clock divide by 2 for seconds.
#define TH1_START 0xFF
/// @def Timer 1 low reg for 2 Hz clock divide by 2 for seconds.
#define TL1_START 0xFE

/// @def ON is binary 1
#define ON 1
/// @def OFF is binary 0
#define OFF 0

/// @def binary position for one minutes segment transistor input.
#define SEG_ONE_MINUTE 1
/// @def binary position for ten minutes segment transistor input.
#define SEG_TEN_MINUTE 2

```

```

/// @def binary position for one hours segment transistor input.
#define SEG_ONE_HOUR 4
/// @def binary position for ten minutes segment transistor input.
#define SEG_TEN_HOUR 8

/// @def Clock DOT LED transistor input.
#define DOT_LED P1_6
/// @def Clock display LED for alarm on/off transistor input.
#define ALARM_LED P1_7

/// @def Switch alarm set location.
#define SET_A_SWITCH P3_4
/// @def Switch time set location.
#define SET_T_SWITCH P3_3
/// @def Switch Hour increment location.
#define HOUR_SWITCH P3_0
/// @def Switch Minute increment location.
#define MINUTE_SWITCH P3_1
/// @def Switch alarm on/off location.
#define ALARM_SWITCH P3_2

/// @def MIN_DELAY minimum delay for switch press.
#define MIN_DELAY 75
/// @def INIT_DELAY initial delay for switch press when setting time.
#define INIT_DELAY 125
/// @def RAMP_DELAY ramp for initial delay to decrease with time to speed up time set when held.
#define RAMP_DELAY 5
/// @def TONE_TIME time for tone to stay activated in milliseconds before next tone.
#define TONE_TIME 250

/// @brief 7 segment lookup table A=0,B=1,C=2,D=3,E=4,F=5,G=6
const uint8_t segmentArray[] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F};

/// @def Sturct to hold time elements for alarm and current time.
struct time
{
    uint8_t one_minutes;
    uint8_t ten_minutes;
    uint8_t one_hours;
    uint8_t ten_hours;
};

/// @brief Global variable for digit selection.
volatile uint8_t digitSelect = 1;
/// @brief Global variable to keep count of the number of milliseconds a switch is pressed.
volatile uint8_t switchTimeout = 0;
/// @brief Global variable to hold the initial time that is reduced by ramp_delay.
volatile uint8_t initTimeout = INIT_DELAY;
/// @brief Global variable to hold the number of milliseconds passed.
volatile uint16_t milliseconds = 0;
/// @brief Global variable to hold the number of previous milliseconds passed.
volatile uint16_t prev_milliseconds = 0;
/// @brief Global variable to hold the number of seconds passed.
volatile uint8_t seconds = 0;
/// @brief Global struct to hold the current time.
volatile struct time gs_timeKeeper = {0,0,0,0};
/// @brief Global struct to hold the current alarm set time
volatile struct time gs_alarmKeeper = {0,0,0,0};
/// @brief Global variable to tell if the alarm is on.
volatile uint8_t alarm_on_off = OFF;

```

```

/// @brief Global variable to store the current tone set from clock divider to 4051 router.
volatile uint8_t alarm_tone = 0;

/// @brief function to flash clock at 00:00 on/off per second till time set pressed. Indicates power
outage and the clock needs to be set.
inline void waitForTimeSet();

/// @brief main entry point for program.
int main(void)
{
    /// @brief local variable to store previous digitSelect value. Only set ports when it changes to
    keep application from resetting values needlessly.
    uint8_t prev_digitSelect = 1;

    // Setup 89s51 for timer 0, counter 1, and interrupt enable.
    TMOD = 0x51;
    TH0 = TH0_START;
    TL0 = TL0_START;
    TH1 = TH1_START;
    TL1 = TL1_START;
    // enable interrupts
    ET0 = 1;
    ET1 = 1;
    EA = 1;
    TR0 = 1;
    TR1 = 1;
    // change priorities so timer 1 is highest.
    PS = 0;
    PT1 = 1;
    PX1 = 0;
    PT0 = 0;
    PX0 = 0;
    /// @brief P0 is 7 segment LED driver
    P0 = segmentArray[0];
    /// @brief P1 is the seconds binary leds, DOT LED, and alarm led outputs
    P1 = 0xBF;
    /// @brief P2 is the digit select control
    P2 = 0x00;
    /// @brief P3 is the switch input, and counter input for the seconds clock (2 Hz).
    P3 = 0x3F;

    waitForTimeSet();

    // loop forever
    for(;;)
    {
        // if the previous digit select is not equal to the current digit select, update display.
        if(prev_digitSelect != digitSelect)
        {
            // Turn off the LED's for a moment, this reduces flicker issues.
            P0 = 0;

            // seconds, complimented since 0 is 1 or on.
            P1 = (P1 & 0xC0) | (!SET_A_SWITCH ? 0x00 : (~seconds & 0x3F));

            // update previous digit select
            prev_digitSelect = digitSelect;

            // assert digit select and set alarm tone every other seconds.
            P2 = (alarm_tone << 4) | (digitSelect & 0x0F);
        }
    }
}

```

```

// turn the DOT LED on when seconds is 1, off when 0.
DOT_LED = ((!SET_T_SWITCH || !SET_A_SWITCH) ? 0 : seconds & 0x01);

// based on selected digit, send out the digit to the proper 7 segment led. if alarm switch is
held, show the alarm set time.
switch(digitSelect)
{
    case SEG_ONE_MINUTE:
        P0 = segmentArray[(SET_A_SWITCH ? gs_timeKeeper.one_minutes :
gs_alarmKeeper.one_minutes)];
        break;
    case SEG_TEN_MINUTE:
        P0 = segmentArray[(SET_A_SWITCH ? gs_timeKeeper.ten_minutes :
gs_alarmKeeper.ten_minutes)];
        break;
    case SEG_ONE_HOUR:
        P0 = segmentArray[(SET_A_SWITCH ? gs_timeKeeper.one_hours : gs_alarmKeeper.one_hours)];
        break;
    case SEG_TEN_HOUR:
        P0 = segmentArray[(SET_A_SWITCH ? gs_timeKeeper.ten_hours : gs_alarmKeeper.ten_hours)];
        break;
}
}
}

return 0;
}

// function to flash clock at 00:00 on/off per second till time set pressed. Indicates power outage
and the clock needs to be set.
inline void waitForTimeSet()
{
    // wait for a second till 2 Hz clock stabilizes.
    while(milliseconds < 1000);

    // reset 2 Hz clock
    TH1 = TH1_START;
    TL1 = TL1_START;
    seconds = 0;

    // wait till set switch is pressed
    while(SET_T_SWITCH)
    {
        // flash all digits at once with 0
        P2 = ((seconds & 0x01) ? 0x0F : 0x00);
        // flash dot LED's in sync
        DOT_LED = seconds & 0x01;
        // roll seconds from 0,1,0,1... so that the clock doesn't start incrementing time.
        seconds = seconds & 0x01;
    }

    // reset seconds when time is set to 0 just cause, not really needed.
    seconds = 0;
}

/// @brief control_isr is a interrupt function for timer 0 when a over flow occurs. This also
processed button presses.
void control_isr (void) __interrupt (TF0_VECTOR)
{

```

```

// reset timer overflow, though it does this anyways.
TF0 = 0;

// reset timer counters start point.
TH0 = TH0_START;
TL0 = TL0_START;

// its been a millisecond, increment
milliseconds++;

// check if the alarm on/off switch is being pressed.
if(!ALARM_SWITCH)
{
    // if the switch is below the the min delay, increment it till it is greater
    switchTimeout = (switchTimeout > MIN_DELAY ? switchTimeout : switchTimeout + 1);

    // once the switch timeout is equal to the min delay, allow a button press.
    if(switchTimeout == MIN_DELAY)
    {
        // toggle the alarm on or off
        alarm_on_off = ((alarm_on_off == ON) ? OFF : ON);

        ALARM_LED = !alarm_on_off;

        // make sure to turn off the tone if the alarm is turned off.
        if(alarm_on_off == OFF)
        {
            prev_milliseconds = 0;
            alarm_tone = 0;
        }
    }
}

// check if the alarm set switch is being pressed.
else if(!SET_A_SWITCH)
{
    // increment switch timeout
    switchTimeout++;

    // when both switches are not pressed, reset initial delay.
    if(MINUTE_SWITCH && HOUR_SWITCH)
    {
        initTimeout = INIT_DELAY;
    }

    // when either switch is pressed, and the press as exceeded the current timeout allow a button
    // press
    if((!MINUTE_SWITCH || !HOUR_SWITCH) && (switchTimeout > initTimeout))
    {
        // when minute is pressed add one
        gs_alarmKeeper.one_minutes += (MINUTE_SWITCH ? 0 : 1);

        // when hour is pressed add one
        gs_alarmKeeper.one_hours += (HOUR_SWITCH ? 0 : 1);

        // the below is the same code used in timer ISR. copy pasta with tweaks
        if(gs_alarmKeeper.one_minutes > 9)
        {
            gs_alarmKeeper.ten_minutes++;
            gs_alarmKeeper.one_minutes = 0;
        }
    }
}

```

```

    if(gs_alarmKeeper.ten_minutes > 5)
    {
        gs_alarmKeeper.ten_minutes = 0;
    }

    if(gs_alarmKeeper.one_hours > 9)
    {
        gs_alarmKeeper.ten_hours++;
        gs_alarmKeeper.one_hours = 0;
    }

    if((gs_alarmKeeper.ten_hours >= 2) && (gs_alarmKeeper.one_hours >= 4))
    {
        gs_alarmKeeper.ten_hours = 0;
        gs_alarmKeeper.one_hours = 0;
    }

    // clear switch timeout since press has happened
    switchTimeout = 0;

    // if the initial timeout is greater then the minimal delay, ramp it down so holding the
    button will get faster.
    if(initTimeout > MIN_DELAY)
    {
        initTimeout = initTimeout - RAMP_DELAY;
    }
}

// check if the time set switch is being pressed.
else if(!SET_T_SWITCH)
{
    // increment switch timeout
    switchTimeout++;

    // when both switches are not pressed, reset initial delay.
    if(MINUTE_SWITCH && HOUR_SWITCH)
    {
        initTimeout = INIT_DELAY;
    }

    // when either switch is pressed, and the press as exceeded the current timeout allow a button
    press
    if((!MINUTE_SWITCH || !HOUR_SWITCH) && (switchTimeout > initTimeout))
    {
        // when minute is pressed add one
        gs_timeKeeper.one_minutes += (MINUTE_SWITCH ? 0 : 1);

        // when hour is pressed add one
        gs_timeKeeper.one_hours += (HOUR_SWITCH ? 0 : 1);

        // the below is the same code used in timer ISR. copy pasta with tweaks
        if(gs_timeKeeper.one_minutes > 9)
        {
            gs_timeKeeper.ten_minutes++;
            gs_timeKeeper.one_minutes = 0;
        }

        if(gs_timeKeeper.ten_minutes > 5)
        {

```

```

    gs_timeKeeper.ten_minutes = 0;
}

if(gs_timeKeeper.one_hours > 9)
{
    gs_timeKeeper.ten_hours++;
    gs_timeKeeper.one_hours = 0;
}

if((gs_timeKeeper.ten_hours >= 2) && (gs_timeKeeper.one_hours >= 4))
{
    gs_timeKeeper.ten_hours = 0;
    gs_timeKeeper.one_hours = 0;
}

// clear switch timeout since press has happened
switchTimeout = 0;

// if the initial timeout is greater then the minimal delay, ramp it down so holding the
button will get faster.
if(initTimeout > MIN_DELAY)
{
    initTimeout = initTimeout - RAMP_DELAY;
}
}

// if no switch is pressed, timeout is cleared and initial timeout is set to initial value.
else
{
    switchTimeout = 0;
    initTimeout = INIT_DELAY;
}

// when alarm tone is not 0, the alarm is on, decrement alarm tone to change the current tone
played.
if(alarm_tone != 0)
{
    if((milliseconds - prev_milliseconds) > TONE_TIME)
    {
        prev_milliseconds = milliseconds;
        alarm_tone = ((alarm_tone <= 1) ? 7 : alarm_tone - 1);
    }
}

// move digit selection by one on each millisecond.
digitSelect = (digitSelect < (1 << 3) ? digitSelect << 1 : 1);
}

/// @brief Keep track of time in seconds as precisely as possible.
void timer_isr (void) __interrupt (TF1_VECTOR)
{
    // reset timer overflow, though it does this anyways.
    TF1 = 0;

    // reset timer counters start point.
    TH1 = TH1_START;
    TL1 = TL1_START;

    // check if the time set switch is pressed. If so keep seconds at and hold.

```

```

if(!SET_T_SWITCH)
{
    seconds = 0;
    return;
}

// increment seconds on each timer overflow.
seconds++;

// once over 59 seconds, increment minutes and reset seconds
if(seconds > 59)
{
    gs_timeKeeper.one_minutes++;
    seconds = 0;
}

// once over 9 minutes, increment ten minutes and reset minutes
if(gs_timeKeeper.one_minutes > 9)
{
    gs_timeKeeper.ten_minutes++;
    gs_timeKeeper.one_minutes = 0;
}

// once over 5 ten minutes, increment hours and reset ten minutes.
if(gs_timeKeeper.ten_minutes > 5)
{
    gs_timeKeeper.one_hours++;
    gs_timeKeeper.ten_minutes = 0;
}

// once over 9 one hours, increment ten hours and reset one hours.
if(gs_timeKeeper.one_hours > 9)
{
    gs_timeKeeper.ten_hours++;
    gs_timeKeeper.one_hours = 0;
}

// once ten hours is at or above 2, and one hours is at or above 4, reset both to 0.
if((gs_timeKeeper.ten_hours >= 2) && (gs_timeKeeper.one_hours >= 4))
{
    gs_timeKeeper.ten_hours = 0;
    gs_timeKeeper.one_hours = 0;
}

// if alarm is on, compare the elements to see if we have hit the correct time.
if(alarm_on_off == ON)
{
    if((gs_alarmKeeper.ten_hours == gs_timeKeeper.ten_hours) && (gs_alarmKeeper.one_hours ==
gs_timeKeeper.one_hours) && (gs_alarmKeeper.ten_minutes == gs_timeKeeper.ten_minutes) &&
(gs_alarmKeeper.one_minutes == gs_timeKeeper.one_minutes))
    {

        if(seconds == 0)
        {
            prev_milliseconds = milliseconds;
            alarm_tone = 7;
        }

        if(seconds >= 59)
        {

```



```
        prev_milliseconds = 0;  
        alarm_tone = 0;  
    }  
}  
}  
}
```