JDC

Military Time Alarm Clock
2022-12 v1.0.0

Table of Contents

Introduction....3
Purpose....3
Usage....4
Setup The Clock....4
First Time Power On....6
Setting The Time....8
Sync Time....9
Setting The Alarm....10
Activating The Alarm....11
Alarm Duration....11
Technical Data....12
Specifications....12
Schematic....13
CAD....17
Code....21

Introduction

Purpose

The military time alarm clock is a standard alarm clock that uses military time for the display. The clock includes one alarm that is set in military time. The clock is designed using the state-of-the-80's hardware for user serviceability along with a full open source design under the MIT license.

Inside the clock is a 8051 micro-controller (ATMEL 89S51 variant) to power the device. This is a widely used part that is still being produced and used in multiple products as its core controller. The time base clock is a state-of-the-cmos 4060. The 4060 is a divider that has some of its unused clocks fed to a 4051 selection chip with an output to a LM386 amplifier. This gives the alarm its distinctive sound that cycles through all the tones being fed to it.

Usage

Setup The Clock

1) Remove the clock from its packaging.
2) Check that all the parts are included.
    - One clock
    - One 5 volt power adapter
3) Set the clock near a wall outlet that the power adapter can reach.
    - WARNING DO NOT PUT STRESS ON THE CORD.
4) Place the clock on a stable surface to prevent falling.

Figure 1.0

First Time Power On

1) After setting up the clock, plug the power adapter into the wall.
2) Plug the barrel jack of the power adapter into the rear connector of the clock.
    - This will insert about half an inch into the device, this is normal.
    - See figure 1.0 on the previous page.
3) The clock will take approximately 2 seconds to power on.
4) Once powered on it will flash 00:00 every other second.
5) This will continue till TIME SET is pressed.
    - See figure 1.1 on the next page.
6) The clock is now ready for operation.

Figure 1.1

Setting The Time

1) Once the clock is powered on, time set can be performed.
2) Press and hold the TIME SET button, while performing the following actions. See figure 1.1 on previous page.
    a) Press the minute button to increment the minutes till the desired number is displayed.
        - Can be pressed and held for speed increment, tapped for single increments.
        - Minutes will roll over from 59 to 0 and will not increment hours.
    b) Press the hour button to increment the hours till the desired number is displayed.
        - Can be pressed and held for speed increment, tapped for single increments.
        - Hours will roll over from 23 to 0.
3) Let go of the TIME SET button to complete time set.

Sync Time

To sync the time with another clock follow this procedure.

1) Set the time to one minute ahead of the clock you would like to sync to.
2) Hold the TIME SET button till the clock to sync to changes to that minute, and let go.
    - The clock will start from 0 seconds (seconds are reset) whenever TIME SET is pressed.

Setting The Alarm

1) With the time set, the alarm can be set in a similar manner.
2) Press and hold the ALARM SET button, while performing the following actions. See figure 1.1 on page 7.
    a) Press the minute button to increment the minutes till the desired number is displayed.
        - Can be pressed and held for speed increment, tapped for single increments.
        - Minutes will roll over from 59 to 0 and will not increment hours.
    b) Press the hour button to increment the hours till the desired number is displayed.
        - Can be pressed and held for speed increment, tapped for single increments.
        - Hours will roll over from 23 to 0.
3) Let go of the ALARM SET button to complete alarm set.

Activating The Alarm

The alarm is activated by pressing the ALARM ACT button.

- See figure 1.1 on page 7.
- Pressing and holding will only toggle the alarm once.
- When the alarm LED is on, the alarm is on.
- When the alarm LED is off, the alarm is off.

Alarm Duration

When the alarm is reached, it will stay active for one minute without intervention. Alarm can be shut off early by pressing the ALARM ACT button.

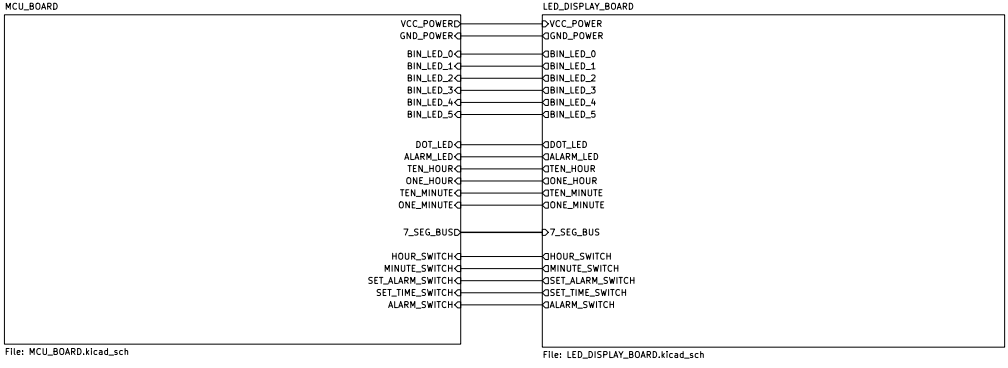
Technical Data

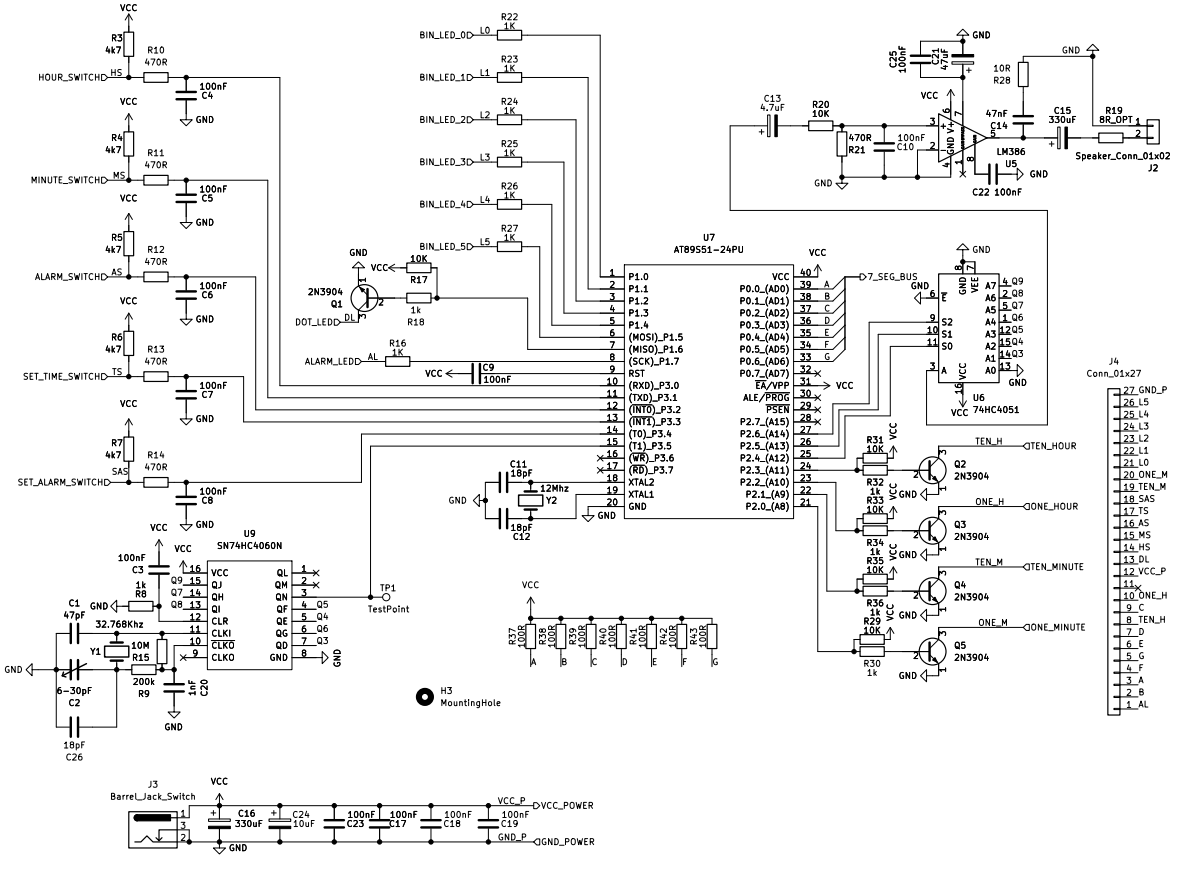
Specifications

- POWER: 5 volt DC at 500 mA max. Center Positive.
- Audio output: 325 mW Max
- Time Display: Green 7 segment LED, with 5 Binary seconds LED (0 to 59)
- Micro-controller: 89S51
- Case: ABS Plastic

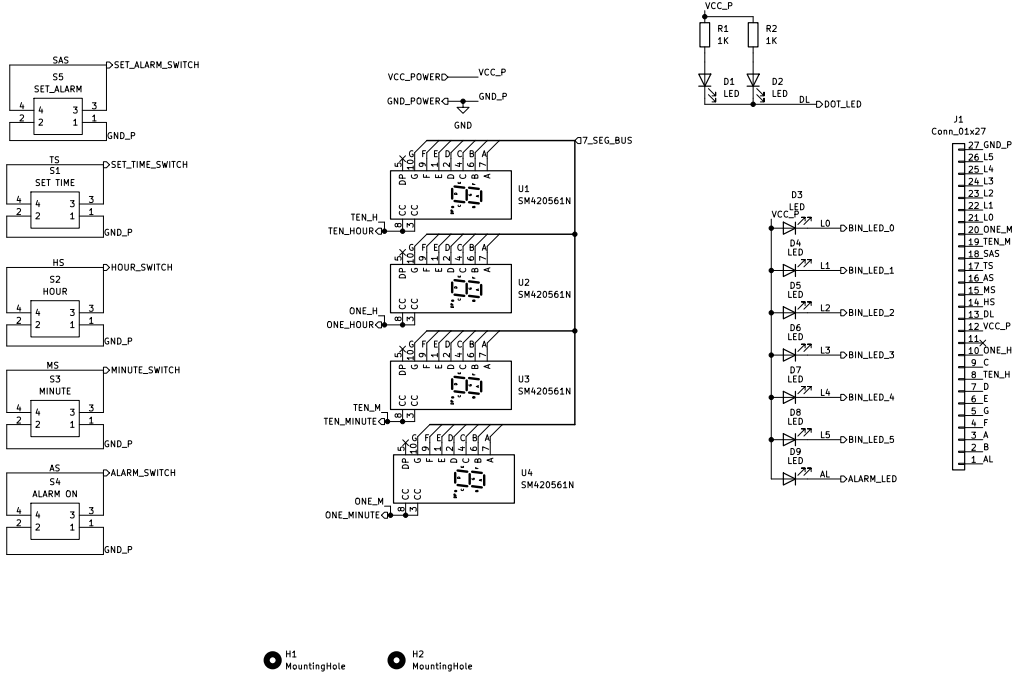
Schematic

The following pages contain the schematic for the device.


MCU_BOARD
VCC_POWERD
GND_POWER
BIN_LED_0
BIN_LED_1
BIN_LED_2
BIN_LED_3
BIN_LED_4
BIN_LED_5
DOT_LED
ALARM_LED
TEN_HOUR
ONE_HOUR
TEN_MINUTE
ONE_MINUTE
7_SEG_BUSD
HOUR_SWITCH
MINUTE_SWITCH
SET_ALARM_SWITCH
SET_TIME_SWITCH
ALARM_SWITCH
File: MCU_BOARD.kicad_sch
LED_DISPLAY_BOARD
VCC_POWER
GND_POWER
BIN_LED_0
BIN_LED_1
BIN_LED_2
BIN_LED_3
BIN_LED_4
BIN_LED_5
DOT_LED
ALARM_LED
TEN_HOUR
ONE_HOUR
TEN_MINUTE
ONE_MINUTE
7_SEG_BUS
HOUR_SWITCH
MINUTE_SWITCH
SET_ALARM_SWITCH
SET_TIME_SWITCH
ALARM_SWITCH
File: LED_DISPLAY_BOARD.kicad_sch

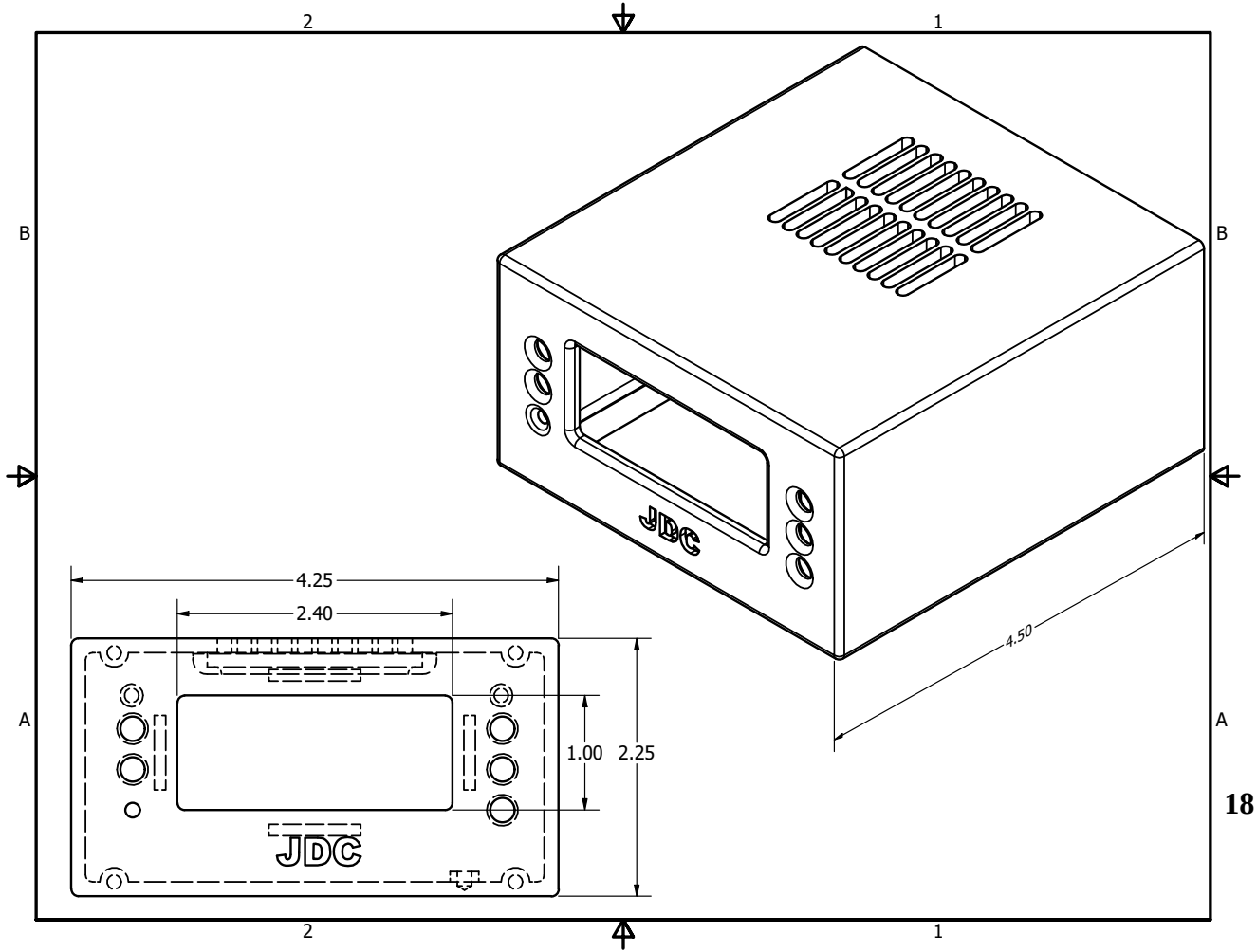


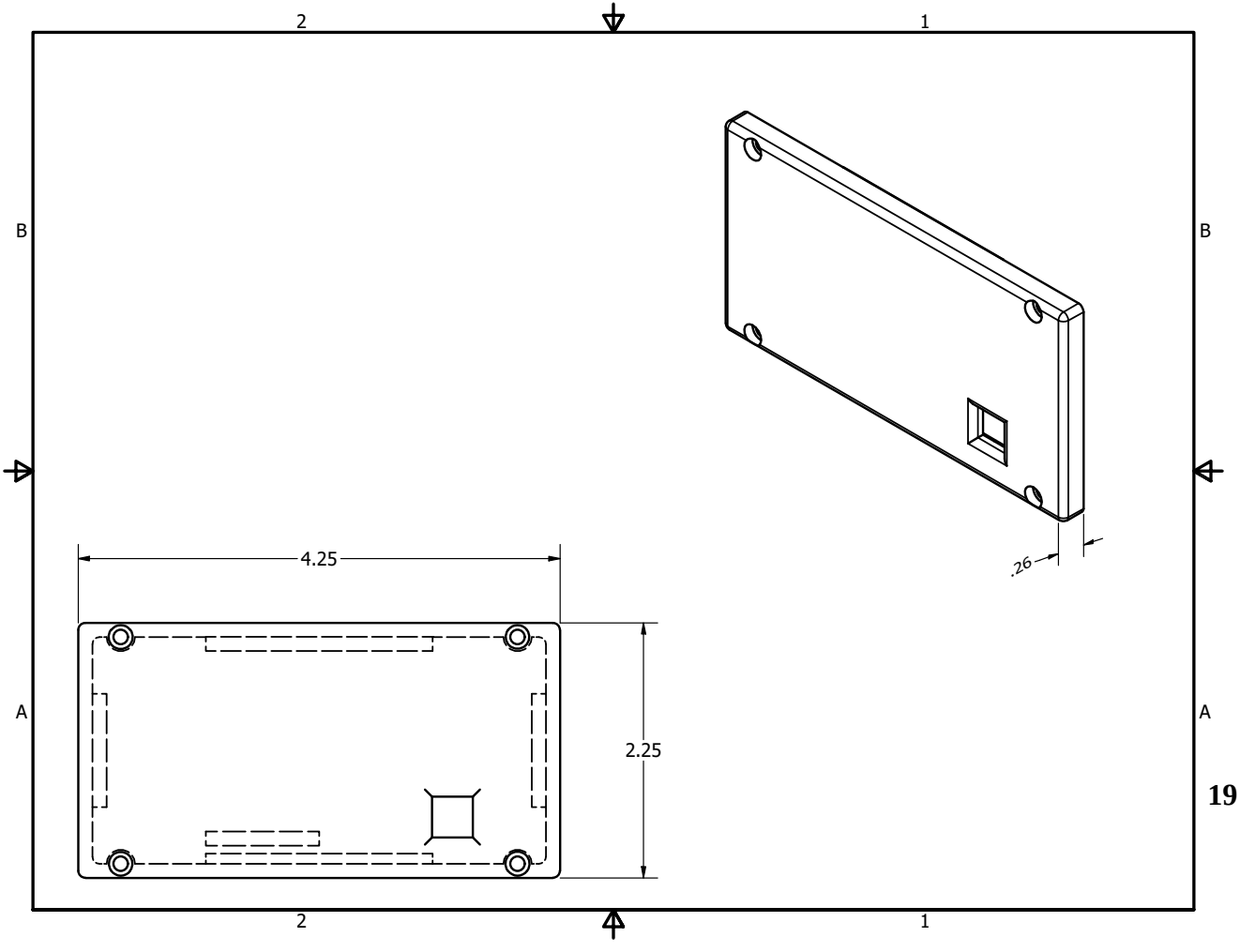
U7
AT89S51-24PU
U9
SN74HC4060N
U6
74HC4051
LM386
U5
Speaker_Conn_01x02
J2
J4
Conn_01x27
J3
Barrel_Jack_Switch
H3
MountingHole
TP1
TestPoint
HOUR_SWITCH
MINUTE_SWITCH
ALARM_SWITCH
SET_TIME_SWITCH
SET_ALARM_SWITCH
BIN_LED_0
BIN_LED_1
BIN_LED_2
BIN_LED_3
BIN_LED_4
BIN_LED_5
DOT_LED
ALARM_LED
7_SEG_BUS
TEN_HOUR
ONE_HOUR
TEN_MINUTE
ONE_MINUTE
VCC_POWER
GND_POWER

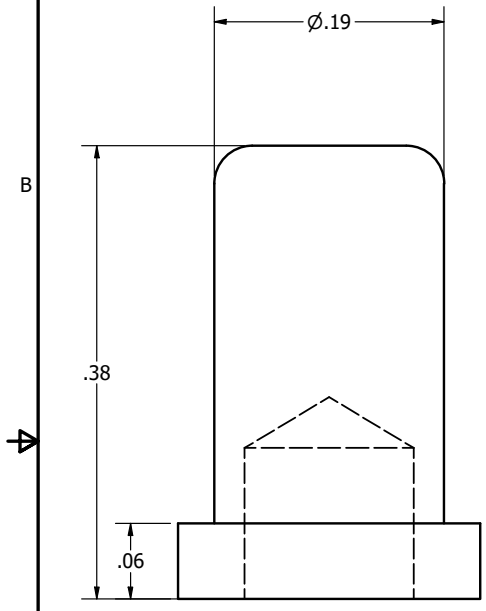

SAS
S5
SET_ALARM
SET_ALARM_SWITCH
GND_P
TS
S1
SET TIME
SET_TIME_SWITCH
GND_P
HS
S2
HOUR
HOUR_SWITCH
GND_P
MS
S3
MINUTE
MINUTE_SWITCH
GND_P
AS
S4
ALARM ON
ALARM_SWITCH
GND_P
VCC_POWER
VCC_P
GND_POWER
GND_P
GND
7_SEG_BUS
U1
SM420561N
U2
SM420561N
U3
SM420561N
U4
SM420561N
TEN_H
TEN_HOUR
ONE_H
ONE_HOUR
TEN_M
TEN_MINUTE
ONE_M
ONE_MINUTE
VCC_P
R1
1K
R2
1K
D1
LED
D2
LED
DL
DOT_LED
D3
LED
D4
LED
D5
LED
D6
LED
D7
LED
D8
LED
D9
LED
L0
L1
L2
L3
L4
L5
AL
BIN_LED_0
BIN_LED_1
BIN_LED_2
BIN_LED_3
BIN_LED_4
BIN_LED_5
ALARM_LED
J1
Conn_01x27
27 GND_P
26 L5
25 L4
24 L3
23 L2
22 L1
21 L0
20 ONE_M
19 TEN_M
18 SAS
17 TS
16 AS
15 MS
14 HS
13 DL
12 VCC_P
11
10 ONE_H
9 C
8 TEN_H
7 D
6 E
5 G
4 F
3 A
2 B
1 AL
H1
MountingHole
H2
MountingHole

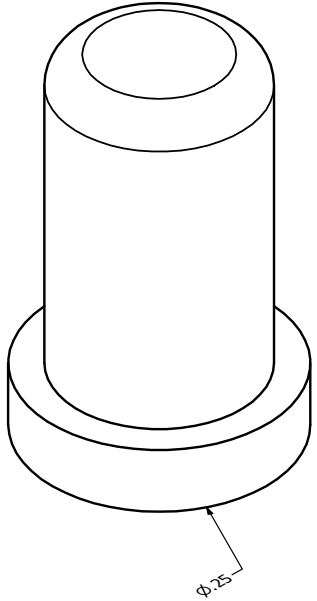
CAD

The following pages contain the CAD for the device. Dimensions are not complete, and all are in inches.


2
1
B
A
4.25
2.40
1.00
2.25
JDC
4.50


2
1
B
B
A
A
4.25
2.25
.26


Ø.19
.38
.06


Ø.25

Code

The following pages contain the C code for the device. They are sideways for proper formatting.

```
//*****************************************************************************
/// @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)
    {
```

```
    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;
      }
    }
  }
}
```

JDC

Military Time Alarm Clock
2022-12
v1.0.0