20140624 BCD Clock Part 2

LCD hooked up, and keypad can be used to input a date and time into the DS1307 RTC,
This is based off the code found on this website:
http://6502cpu.blogspot.com/2013/01/arduino-uno-and-ds1307-real-time-clock.html

On the site, the author says the DS1307 outputs BCD, I found this to not be the case, it does output Binary, and with a little coding was easy to change it to 4 Bit BCD.
Otherwise, what the author has on the site seems to be correct about the DS1307, and the DFRobot LCD with keypad.
One thing that I did notice while using the DS1307 is there doesn't seem to be any type of protection from the on board battery feeding back into the Arduino. I noticed that I had unpluged the power, and the 8x8 matrix, and the LCD screen tried to light up only briefly, also the clock seemed to become unstable and run fast. I've been unplugging the RTC from the arduino while it's just going to sit, and having noticed the LCD or 8x8 Matrix trying to stay powered on, the clock also seemed to become a little more stable (Thou it was still fast, it wasn't days fast)
My modified code, also includes the use of the 8x8 Matrix for a binary (BCD) style clock output.

```
/*
 * Date and time functions using a DS1307 RTC connected via I2C
 * and Wire lib
 * It is a simple clock application.
 * Author: Marek Karcz 2013. All rights reserved.
 * License: Freeware.
 * Disclaimer: Use at your own risk.
 * Hardware:
 *     1) MINI Arduino I2C RTC DS1307 AT24C32 module.
 *     eBay item# 180646747674 by seller: e_goto
 *     Serial: SKU 00100-049
 *     2) Keypad Shield 1602 LCD For Arduino MEGA 2560 1280 UNO
 *     R3 A005
 *     eBay item# 261039184894 by seller: womarts
 */

/*
  The LCD circuit:
 * LCD RS pin to digital pin 8
 * LCD Enable pin to digital pin 9
 * LCD D4 pin to digital pin 4
 * LCD D5 pin to digital pin 5
 * LCD D6 pin to digital pin 6
 * LCD D7 pin to digital pin 7
 * LCD BL pin to digital pin 10
 * KEY pin to analog pin 0
 */

#include <Wire.h>
#include "RTClib.h"
#include <LiquidCrystal.h>
#include "LedControl.h"

#define LOOP_DELAY  2000
```

```
LiquidCrystal lcd(8, 13, 9, 4, 5, 6, 7);
RTC_DS1307 RTC;
LedControl lc=LedControl(30,32,34,1);
boolean bBlink = true;
const char *appVer = "        RTC 1.4   ";
const char *modTxt = "Set clock ...   ";

// Global variables for time setup/displaying purposes.
uint16_t set_Year;
uint8_t set_Month;
uint8_t set_Day;
uint8_t set_Hour;
uint8_t set_Minute;

/*
 * Keypad shield uses resistors array and single analog input.
 * The values in adc_key_val array help to determine which
 * key on the shield was pressed by checking the analog input
 * read value.
 */
const int adc_key_val[5] ={50, 200, 400, 600, 800 };
int adc_key_in;

// keypad keys definitions
enum KP
{
  KEY_RIGHT = 0,
  KEY_UP,
  KEY_DOWN,
  KEY_LEFT,
  KEY_SELECT,
  KEY_NUMKEYS, // mark the end of key definitions
  KEY_NONE     // definition of none of the keys pressed
};

// Finite Machine States
enum FMS
{
  RUN = 0,
  SETCLOCK,
  SETYEAR,
  SETMONTH,
  SETDAY,
  SETHOUR,
  SETMINUTE
} ClockState;

// Finite Machine State Transitions Table.
// Defines the flow of the application modes from one to another.
enum FMS StateMachine[] =
{
   /* RUN      -> */ SETCLOCK,
   /* SETCLOCK   -> */ SETYEAR,
   /* SETYEAR  -> */ SETMONTH,
   /* SETMONTH   -> */ SETDAY,
   /* SETDAY   -> */ SETHOUR,
   /* SETHOUR  -> */ SETMINUTE,
   /* SETMINUTE  -> */ RUN
```

```
};

enum KP key = KEY_NONE;

// Array of the numbers of month days.
const unsigned int month_days [] =
{31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
const char *daysOfWeek[] =
{ "Su", "Mo", "Tu", "We", "Th", "Fr", "Sa" };

int nDelay = 0; // controlling loops/key press latency

void setup ()
{
       nDelay = 0;
       key = KEY_NONE;
  // initialize LCD keypad module
       lcd.clear();
       lcd.begin(16, 2);
       lcd.setCursor(0,0);
       lcd.print(appVer);
  // power to i2c_ds1307_at24c32 module provided via A2, A3 pins
       pinMode(A3, OUTPUT);
       digitalWrite(A3, HIGH);
       pinMode(A2, OUTPUT);
       digitalWrite(A2, LOW);
  // start communication, I2C and RTC
       Wire.begin();
       RTC.begin();
       lc.shutdown(0,false);
       lc.setIntensity(0,8);
       lc.clearDisplay(0);
       ClockState = RUN;

       readKey();
 // if key SELECT is held at RESET/Start up procedure
       if (key == KEY_SELECT)
       {
       setClock(RTC.isrunning());
       }
       else if (key == KEY_LEFT) // if key LEFT is held at RESET
       {
 // following line sets the RTC to the date & time this sketch
 // was compiled
       RTC.adjust(DateTime(__DATE__, __TIME__));
       }
       else if (! RTC.isrunning()) // if battery was changed
       {
       setClock(false);
       }
 }

void loop ()
{

       readKey();
       if (key == KEY_SELECT)
       {
        ClockState = StateMachine[ClockState]; // switch mode
        delay(330);
        }

        switch (ClockState)
        {
        case RUN:
        if (nDelay <= 0)
        {
                 matrixdis();
                DateTime now = RTC.now();

                dispTime(now.year(),
                 now.month(),
                        now.day(),

                        now.hour(),
                        now.minute(),
                        now.dayOfWeek());
                bBlink = ((bBlink) ? false : true);
        }
        break;

        case SETCLOCK:

        nDelay = 0;
        lcd.setCursor(0,0);
        lcd.print(modTxt);
        setClock(RTC.isrunning());
        lcd.setCursor(0,0);
        lcd.print(appVer);
        break;

        default: break;

        }
        if (nDelay <= 0)
        nDelay = LOOP_DELAY;
        else
        nDelay--;
}

/*
        else
        lcd.print('0');
  }
  if (ClockState == SETHOUR)
  {
        if (bBlink)
        lcd.print(hr, DEC);
        else
        {
        lcd.print(' ');
        if (hr >= 10)
        lcd.print(' ');
        }
  }
  else
```

```
       lcd.print(hr, DEC);
}

void dispMinute(uint8_t mn)
{
  if (mn < 10)
  {
       if (ClockState == SETMINUTE)
       {* Functions to aid displaying the date/time.
 */
void dispYear(uint16_t yr)
{
  if (ClockState == SETYEAR)
  {
       if (bBlink)
       lcd.print(yr, DEC);
       else
       lcd.print("    ");
  }
  else
       lcd.print(yr, DEC);
}

void dispMonth(uint8_t mo)
{
  if (ClockState == SETMONTH)
  {
       if (bBlink)
       lcd.print(mo, DEC);
       else
       {
       lcd.print(' ');
       if (mo >= 10)
       lcd.print(' ');
       }
  }
  else
       lcd.print(mo, DEC);
}

void dispDay(uint8_t dy)
{
  if (ClockState == SETDAY)
  {
       if (bBlink)
       lcd.print(dy, DEC);
       else
       {
       lcd.print(' ');
       if (dy >= 10)
       lcd.print(' ');
       }
  }
  else
       lcd.print(dy, DEC);
}

void dispHour(uint8_t hr)
{
  if (hr < 10)
  {
       if (ClockState == SETHOUR)
       {
       if (bBlink)
       lcd.print('0');
       else
       lcd.print(' ');
       }

       if (bBlink)
       lcd.print('0');
       else
       lcd.print(' ');
       }
       else
       lcd.print('0');
  }
  if (ClockState == SETMINUTE)
  {
       if (bBlink)
       lcd.print(mn, DEC);
       else
       {
       lcd.print(' ');
       if (mn >= 10)
       lcd.print(' ');
       }
  }
  else
       lcd.print(mn, DEC);
}

void dispTime(uint16_t  yr,
              uint8_t   mo,
              uint8_t   dy,
              uint8_t   hr,
              uint8_t   mn,
              uint8_t   dow)
{
  lcd.setCursor(0,1);
  dispYear(yr);
  lcd.print('/');
  dispMonth(mo);
  lcd.print('/');
  dispDay(dy);
  lcd.print(' ');
  lcd.print(' ');
  lcd.setCursor(11,1);
  dispHour(hr);
  if (ClockState == RUN)
  {
       if (bBlink)
       lcd.print(':');
       else
       lcd.print(' ');
  }
  else
       lcd.print(':');
  dispMinute(mn);
  lcd.setCursor (14, 0);
  lcd.print(daysOfWeek[dow]);
}

/*
 * Functions to aid setting date/time.
 */
void setYear(boolean incdec)
{
  if (incdec)
  {
        if (set_Year < 2100)
        set_Year++;
  }
  else
  {
        if (set_Year > 2000)
        set_Year--;
  }
}

void setMonth(boolean incdec)
{
  if (incdec)
  {
        if (set_Month < 12)
        set_Month++;
  }
  else
  {
        if (set_Month > 1)
        set_Month--;
  }
}

void setDay(boolean incdec)
{
  if (incdec)
  {
        if ((set_Month != 2 && set_Day < month_days[set_Month])
        ||
        (set_Month == 2 && set_Day < 28)
        ||
        (set_Month == 2 && set_Day == 28 && isLeapYear(set_Year))
        )
        set_Day++;
  }
  else
  {
        if (set_Day > 1)
        set_Day--;
  }
}

void setHour(boolean incdec)
{
  if (incdec)
  {
        if (set_Hour < 23)
        set_Hour++;
        else
        set_Hour = 0;
  }
  else
  {
        if (set_Hour > 1)
        set_Hour--;
        else
        set_Hour = 23;
  }
}

void setMinute(boolean incdec)
{
  if (incdec)
  {
        if (set_Minute < 59)
        set_Minute++;
        else
        set_Minute = 0;
  }
  else
  {
        if (set_Minute > 1)
        set_Minute--;
        else
        set_Minute = 59;
  }
}

void setDateTime(boolean incdec)
{
  if (incdec)
  {
        switch (ClockState)
        {
        case SETYEAR:

        setYear(true); // increment year
        break;

        case SETMONTH:

        setMonth(true);   // increment month
        break;

        case SETDAY:

        setDay(true);  // increment day
        break;

        case SETHOUR:

        setHour(true); // increment hour
        break;
```

```
       case SETMINUTE:

       setMinute(true);  // increment minute
       break;

       default: break;

       }
  }
  else
  {
       switch (ClockState)
       {
       case SETYEAR:

       setYear(false);       // decrement year
       break;

       case SETMONTH:

       setMonth(false);   // decrement month
       break;

       case SETDAY:

       setDay(false);        // decrement day
       break;

       case SETHOUR:

       setHour(false);       // decrement hour
       break;

       case SETMINUTE:

       setMinute(false);  // decrement minute

       default: break;

       }
  }
}

void setClock(boolean readrtc)
{
       DateTime now = DateTime(2013,1,1,0,0,0);

       if (readrtc)
       now = RTC.now();

       delay(500);
       set_Year = now.year();
       set_Month = now.month();
       set_Day = now.day();
       set_Hour = now.hour();
       set_Minute = now.minute();
```

```
        ClockState = SETYEAR;

        while (ClockState >= SETCLOCK)
        {
        if (nDelay <= 0)
        {
        dispTime(set_Year,
                set_Month,
                set_Day,
                set_Hour,
                set_Minute,
                now.dayOfWeek());
        bBlink = ((bBlink) ? false : true);
        }

        readKey();
        if (key == KEY_UP || key == KEY_DOWN)
        bBlink = true;
        if (key == KEY_SELECT)
        {
        ClockState = StateMachine[ClockState];
        delay(330);
        }
        else
        {
        if (nDelay <= 0)
        {
        if (key == KEY_UP)
        {
                setDateTime(true);  // increment
        }
        else if (key == KEY_DOWN)
        {
                setDateTime(false);  // decrement
        }
        else if (key == KEY_LEFT)
        {
                ClockState = RUN;     // exit set clock mode
        }
        nDelay = LOOP_DELAY;
        }
        }
        if (nDelay <= 0)
        nDelay = LOOP_DELAY;
        else
        nDelay--;
        //delay(330);
        }

        RTC.adjust(DateTime(set_Year,
                        set_Month,
                        set_Day,
                        set_Hour,
                        set_Minute,
                        0));
}

// Determine the leap year.
```

```
boolean isLeapYear(uint16_t yr)
{
  if ((yr%400)==0)
       return true;
  else if ((yr%100)==0)
       return false;
  else if ((yr%4)==0)
       return true;

  return false;
}

// Get key code from analog input.
unsigned int get_key(unsigned int input)
{
   unsigned int k;
   for (k = KEY_RIGHT; k < KEY_NUMKEYS; k++)
   {
       if (input < adc_key_val[k])
       {
       return k;
       }
   }
   if (k >= KEY_NUMKEYS) k = KEY_NONE; // No valid key pressed

   return k;
}

// Read analog input 0 to obtain key code in global variable: key
void readKey(void)
{
       key = KEY_NONE;
       adc_key_in = analogRead(0); // read the value from the sensor
       // convert into key press
       key = (enum KP) get_key(adc_key_in);
}


void matrixdis() {
DateTime now = RTC.now();
       //Serial.print(now.year(), DEC);
       //Serial.print('/');
       //Serial.print(now.month(), DEC);
       //Serial.print('/');
       //Serial.print(now.day(), DEC);
       //Serial.print(' ');
       //Serial.print(now.hour(), DEC);
       lc.setColumn(0,0, ((now.hour()/10)%10));
       lc.setColumn(0,1, (now.hour()%10));
       colon();
       //Serial.print(':');
       //Serial.print(now.minute(), DEC);
       lc.setColumn(0,3,((now.minute()/10)%10));
       lc.setColumn(0,4,(now.minute()%10));
       colon();
       //Serial.print(':');
       //Serial.print(now.second(), DEC);
       lc.setColumn(0,6,((now.second()/10)%10));
       lc.setColumn(0,7,(now.second()%10));
```

```
        colon();
        //Serial.println();
        }

void colon() {
   //    lc.setLed(0,4,2, true);
   lc.setLed(0,6,2, true);
   // lc.setLed(0,4,5, true);
   lc.setLed(0,6,5, true);
       delay (250);

   // lc.setLed(0,4,2, false);
       lc.setLed(0,6,2, false);
   // lc.setLed(0,4,5, false);
       lc.setLed(0,6,5, false);
       delay(250);
       }
```

Some more information about the DS1307, some of it's information is also incorrect, and some of it is better information then the above site: http://bildr.org/2011/03/ds1307-arduino/

This was/is a fun project to do, and doesn't require much to do it. The library for the DS1307 is easy to use, and works quite well.