```
//Holden Sandlar
//Multidisciplinary Senior Design

//Reads from RTC over I2C and displays the current time on four seven-segment displays

#include "Wire.h"
#include <stdio.h>
#include <stdlib.h>
#define DS1307_I2C_ADDRESS 0x68 //RTC I2C address

//SDA is on Arduino Pin A4
//SCL is on Arduino Pin A5

#define clk 12 //CLK out to shift registers
#define rst 8 //RST out to shift registers


#define MI_LO 5 //Minute Low digit on Shift register attached to arduino pin 5
#define MI_HI 4 //Minite high digit on shift register attached to arduino pin 4
#define HR_LO 3 //Hour low digit on shift register attached to arudino pin 3
#define HR_HI 2 //Hour high digit on shift register attached to arduino pin 2

// 1               2 :              3                 0
// ^ hour high     ^ hour low       ^ minute high     ^ minute low


//Arduino Wire library was updated.. define I2C_READ/WRITE correctly for the version being used:
#if defined(ARDUINO) && ARDUINO >= 100    // Arduino v1.0 and newer
  #define I2C_WRITE Wire.write
  #define I2C_READ Wire.read
#else                                     // Arduino Prior to v1.0
  #define I2C_WRITE Wire.send
  #define I2C_READ Wire.receive
#endif

//Variables for storing R/W values to/from RTC:
byte second, minute, hour, dayOfWeek, dayOfMonth, month, year;
byte zero;


//Digit map -- this is a map of which bits need to be turned on to display a given number on
the 7-segment
//Order of bits (which represent segments) is A, B, C, D, E, F, G
int digit_map [10][7] = {
  { 1, 1, 1, 1, 1, 1, 0 }, //0
  { 0, 1, 1, 0, 0, 0, 0 }, //1
  { 1, 1, 0, 1, 1, 0, 1 }, //2
  { 1, 1, 1, 1, 0, 0, 1 }, //3
  { 0, 1, 1, 0, 0, 1, 1 }, //4
  { 1, 0, 1, 1, 0, 1, 1 }, //5
  { 1, 0, 1, 1, 1, 1, 1 }, //6
  { 1, 1, 1, 0, 0, 0, 0 }, //7
  { 1, 1, 1, 1, 1, 1, 1 }, //8
```

```
  { 1, 1, 1, 0, 0, 1, 1 }, //9
};

void setup()
{

  Serial.begin(57600); // For DEBUG

  Wire.begin(); //Start I2C connection (2-wire interface)

  //Define outputs:
  //CLK, RST, Input to each shift register
  pinMode(clk, OUTPUT); //active high
  pinMode(rst, OUTPUT); //active low

  pinMode(HR_HI, OUTPUT); //Hour high digit
  pinMode(HR_LO, OUTPUT); //Hour low digit
  pinMode(MI_HI, OUTPUT); //Minute high digit
  pinMode(MI_LO, OUTPUT); //Minute low digit

  digitalWrite(clk, LOW); //Set clock low
  zero = 0x00; //Will be used to reset the RTC internal pointer
  clear_shift_regs(); //Clear any unwanted data contained in the shift registers


}

void loop()
{
  getDateDs1307(); //Get current time from RTC
  clear_shift_regs(); //Clear anything currently in the shift registers
  if(hour > 12) hour -= 12; //If decimal hour returned > 12, subtract 12 to only display
  12-hour time format
  if(hour == 0) hour = 12; //If decimal hour returned == 0, it is 12 AM


  //Serial.println(hour,BIN); // DEBUG
  hour = decToBcd(hour); //Convert the decimal hour to BCD -- Ex. 12 (dec) = 0x12 (hex-BCD) =
  0001 0010 (bin-BCD)

  shift_number_in(hour>>4, hour&0x0F, minute>>4, minute&0x0F); //Parallel shift of current time
  digits into shift registers

  delay(30000); //Wait 30 seconds before updating again to prevent flicker...
}


//Function: shift_number_in ( hour_high_digit, hour_low_digit, minute_high_digit,
minute_low_digit )
void shift_number_in(int hour_hi, int hour_lo, int min_hi, int min_lo)
{
  //Shift in seven bits to each shift register. Output of shift registers is hooked up to 7-segs.
  for(int i = 0; i<7; i++)
```

```
  {
    digitalWrite(HR_HI, digit_map[hour_hi][i]);
    digitalWrite(HR_LO, digit_map[hour_lo][i]);
    digitalWrite(MI_HI, digit_map[min_hi][i]);
    digitalWrite(MI_LO, digit_map[min_lo][i]);

    //Clock this data in
    digitalWrite(clk, HIGH);
    delayMicroseconds(3);
    digitalWrite(clk, LOW);
  }

}

//Function: clear_shift_regs()
//Send a reset to the shift registers to eliminate any registered data
void clear_shift_regs()
{
  digitalWrite(rst,LOW); //active low reset
  delayMicroseconds(1); //The required pulse width for reset to work is in 10's to 100's of ns
  range. 1-3 uS should be plenty
  digitalWrite(rst,HIGH); //disable reset
}

// Gets the date and time from the ds1307
void getDateDs1307()
{
  // Reset the register pointer
  Wire.beginTransmission(DS1307_I2C_ADDRESS); //Start transmision
  I2C_WRITE(zero); //Write 0x00 to register pointer (points to register to read in RTC memory
  space)
  Wire.endTransmission(); //End transmission

  Wire.requestFrom(DS1307_I2C_ADDRESS, 3); //Start a transaction to read 7 bytes from the RTC
  starting at RTC register 0x00 (set in previous command)


  second  = bcdToDec(I2C_READ() & 0x7f); //MSB is a Clock Halt bit. Mask this off.
                                         //Seconds in format S7 S6 S5 S4 S3 S2 S1
                                         //S7-S5 represents 10's of seconds (ex. 40 seconds
                                         would be 010)
                                         //S4-S1 represents 1's of seconds (ex. 5 seconds would
                                         be 0101)
  minute = I2C_READ(); //Minutes in format M8 M7 M6 M5 M4 M3 M2 M1
                       //M8 is always 0
                       //M7-M5 represents 10's of minutes
                       //M4-M1 represents 1's of minutes

  hour = bcdToDec(I2C_READ() & 0x3f); //Hours in format H8 H7 H6 H5 H4 H3 H2 H1
                                      //H8 is always 0 -- masked here
                                      //H7 Indicates 12 hour or 24 hour format -- configure
                                      this for 24 hour - This bit set LOW
                                      //H6-H5 Indicates 10's of hours -- (ex. 12:00 would be
```

```
                                    represented as 01 (bin), 24:00 would be represented as 10
                                    (bin))
                                    //H4-H1 Indicates 1's of hours -- (ex. 5:00 would be 0101
                                    (bin), 13:00 would be 0011)
}

//Converts a decimal number to a BCD number
byte decToBcd(byte val)
{
  return ( (val/10*16) + (val%10) );
}

// Convert binary coded decimal to normal decimal numbers
byte bcdToDec(byte val)
{
  return ( (val/16*10) + (val%16) );
}
```