Arduino Digital RTC

GitHub:

https://github.com/Psisco-yes/ArduinoDigitalRTC

List of contents:

Introduction

Objective: The goal of the project is to create a simple, accurate real time clock with a digital display that I had laying around. It had to have the element of user interaction in a form of buttons to set the clock and saving of the time set even when powered off.

How It Works:

- Initialization: Upon powering up, the Arduino reads time from the RTC module and sets up the display and buttons.
- Time Display: The Arduino, upon an interrupt, reads the current time from the RTC module and updates the seven-segment display to show the hours and minutes.
- Time Setting Mode: By pressing both buttons simultaneously, the clock enters a time-setting mode. In this mode, one button increments the hours while the other increments the minutes. The new time is then saved back to the RTC module by pressing both buttons again.
- Dot Blinking: The dot separating hours from minutes on the display blinks to indicate seconds, providing a visual representation of time progression.

Preparations

Bill of Materials:

The following components and materials are needed to build the clock:

- Arduino Nano ATmega328P (or other) microcontroller
- Dallas Semiconductor DS3231 RTC with auxiliary CR2032 battery
- Kingbright CA04-41SRWA Four-digit display
- 22x male-to-male jumper wires
- 15x 160Ω resistors
- 2x normally open push buttons
- Breadboard
- USB A to Mini B

An optional power bank can be installed to power the microcontroller.

Software:

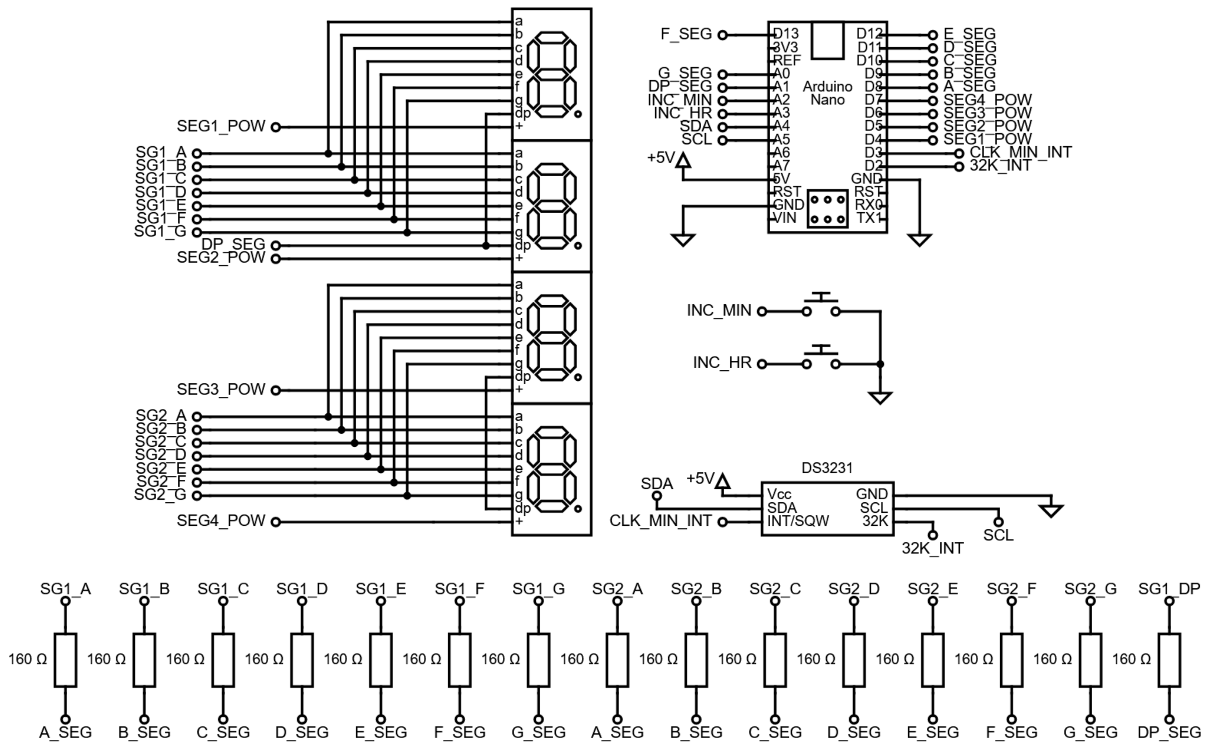
The project uses following libraries:

- DS3231 by NorthernWidget on GitHub (available in Arduino Library Manager)
- Built in Wire library

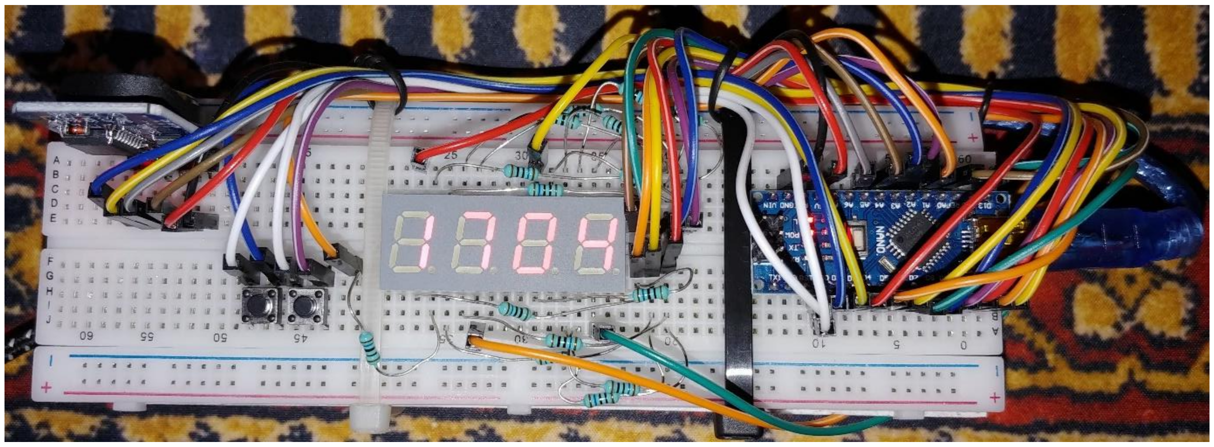
There are two Arduino sketches:

- INIT_CLOCK.ino is used to initialize the RTC module's registers and to start the crystal.
- CLOCK.ino is the main sketch to be used in the microcontroller after initializing the RTC module.

Schematic diagram:



Picture of the prototype:



Code breakdown

INIT_CLOCK.ino

The code begins by importing the Wire.h and DS3231.h libraries to communicate and configure the RTC module.

```
#include <DS3231.h>
#include <Wire.h>
```

Setup function creates a DS3231 RTC object and begins serial Wire communication. After that, it sets RTC's clock mode to 24-hour. Next step is to clear and set the modules built in registers to sensical values (00:00:00 01.01.2020), enable the 32K pin and set INT/SQW pin to be an interrupt pin.

```
void setup() {
  DS3231 RTC;
  Wire.begin();
  RTC.setClockMode(false);
  RTC.setHour(0);
  RTC.setMinute(0);
  RTC.setSecond(0);
  RTC.setDate(1);
  RTC.setMonth(1);
  RTC.setYear(20);
  RTC.enable32kHz(true);
  RTC.enableOscillator(false,false,0);
}
```

The void function is not used.

CLOCK.ino

The code begins by importing the Wire.h and DS3231.h libraries to communicate and configure the RTC module.

```
#include <DS3231.h>
#include <Wire.h>
```

Next, all pins are defined. These include pins for interrupts, each display segment, power to each digit and button pins.

```
// Pin definitions
const int led_int_pin = 2;
const int rtc_int_pin = 3;

// Segment pin definitions
const int a_pin = 8;
const int b_pin = 9;
const int c_pin = 10;
const int d_pin = 11;
const int e_pin = 12;
const int f_pin = 13;
const int g_pin = A0;
const int dot_pin = A1;

// Digit power pin definitions
const int d1_pin = 4;
const int d2_pin = 5;
const int d3_pin = 6;
const int d4_pin = 7;

// Button pin definitions
const int inc_hr_pin = A3;
const int inc_min_pin = A2;
```

Right after that, there's the display class definition. It handles displaying digits on the display by using a binary mask.

```
// Segment display class
class SegmentedDisplay {
  public:
    int segment_pins[8];

    SegmentedDisplay(int segment_a_pin, int segment_b_pin, int segment_c_pin,
int segment_d_pin, int segment_e_pin, int segment_f_pin, int segment_g_pin,
int dot_pin) {
      segment_pins[0] = segment_a_pin;
      segment_pins[1] = segment_b_pin;
      segment_pins[2] = segment_c_pin;
      segment_pins[3] = segment_d_pin;
      segment_pins[4] = segment_e_pin;
      segment_pins[5] = segment_f_pin;
      segment_pins[6] = segment_g_pin;
      segment_pins[7] = dot_pin;
    }
    void displayDigit(int num, int digit_pin, bool dot) {
      unsigned char segments;
      switch (num) {
        case 0: segments = B01111110; break;
        case 1: segments = B00110000; break;
        case 2: segments = B01101101; break;
        case 3: segments = B01111001; break;
        case 4: segments = B00110011; break;
        case 5: segments = B01011011; break;
        case 6: segments = B01011111; break;
        case 7: segments = B01110000; break;
        case 8: segments = B01111111; break;
        case 9: segments = B01111011; break;
        default: segments = B00000000;
      }

      for (int i = 6; i >= 0; i--) {
        digitalWrite(segment_pins[i], (segments & 1) ? LOW : HIGH);
        segments >>= 1;
      }
      digitalWrite(segment_pins[7], dot ? LOW : HIGH);

      digitalWrite(digit_pin, HIGH);
      delay(1);
      digitalWrite(digit_pin, LOW);
    }
};
```

This section contains function declarations, definitions of flags, button timeout, display digits cache and initialization of RTC and Display objects.

```
// Function declarations
void dotBlink();
void updateTimeInt();
void updateTime();
void setMode();

// Dot blink state
bool dot_blink = true;

// Update flag
bool update = true;

// Button timeout
const unsigned char timeout = 100;
```

```
// Variables for caching display digits
unsigned char display_hr1;
unsigned char display_hr2;
unsigned char display_min1;
unsigned char display_min2;

// RTC object
DS3231 RTC;

// Display object
SegmentedDisplay display(a_pin, b_pin, c_pin, d_pin, e_pin, f_pin, g_pin,
dot_pin);
```

Setup function handles all pins initialization and configuring RTC's alarm function to generate interrupts for the microcontroller.

```
void setup() {
  // Initialize digit pins
  pinMode(d1_pin, OUTPUT);
  pinMode(d2_pin, OUTPUT);
  pinMode(d3_pin, OUTPUT);
  pinMode(d4_pin, OUTPUT);

  digitalWrite(d1_pin, LOW);
  digitalWrite(d2_pin, LOW);
  digitalWrite(d3_pin, LOW);
  digitalWrite(d4_pin, LOW);

  // Initialize segment pins
  pinMode(a_pin, OUTPUT);
  pinMode(b_pin, OUTPUT);
  pinMode(c_pin, OUTPUT);
  pinMode(d_pin, OUTPUT);
  pinMode(e_pin, OUTPUT);
  pinMode(f_pin, OUTPUT);
  pinMode(g_pin, OUTPUT);
  pinMode(dot_pin, OUTPUT);

  digitalWrite(a_pin, HIGH);
  digitalWrite(b_pin, HIGH);
  digitalWrite(c_pin, HIGH);
  digitalWrite(d_pin, HIGH);
  digitalWrite(e_pin, HIGH);
  digitalWrite(f_pin, HIGH);
  digitalWrite(g_pin, HIGH);
  digitalWrite(dot_pin, HIGH);
  // Initialize button pins
  pinMode(inc_min_pin, INPUT_PULLUP);
  pinMode(inc_hr_pin, INPUT_PULLUP);

  // Initialize interrupt pins
  pinMode(led_int_pin, INPUT);
  pinMode(rtc_int_pin, INPUT_PULLUP);

  // Initialize DS3231 RTC's alarm interrupts
  Wire.begin();

  RTC.turnOffAlarm(1);
  RTC.setA1Time(0, 0, 0, 0, B00001110, false, false, false);
  RTC.turnOnAlarm(1);

  RTC.setA2Time(0, 0, 255, B01100000, false, false, false);
  RTC.turnOffAlarm(2);
  RTC.checkIfAlarm(2);

  attachInterrupt(digitalPinToInterrupt(rtc_int_pin), updateTimeInt, FALLING);
  attachInterrupt(digitalPinToInterrupt(led_int_pin), dotBlink, CHANGE);

  RTC.enable32kHz(true);
}
```

Declared functions are defined here. The functions updateTimeInt and updateTime update microcontroller's cache variables which are displayed on the display.

```
void updateTimeInt() {
  update = true;
}

void updateTime() {
  update = false;
  bool h12, hPM;
  byte hour = RTC.getHour(h12, hPM);
  byte minutes = RTC.getMinute();
  display_hr1 = hour / 10;
  display_hr2 = hour % 10;
  display_min1 = minutes / 10;
  display_min2 = minutes % 10;
  RTC.checkIfAlarm(1);
}
```

Function setMode allows the user to configure the current time using buttons and saves it in the RTC module.

```
void setMode() {
  delay(500);
  bool button1 = HIGH;
  bool button2 = HIGH;
  bool lastStateB1 = HIGH;
  bool lastStateB2 = HIGH;
  unsigned long timestampB1 = 0;
  unsigned long timestampB2 = 0;
  byte hours = display_hr1 * 10 + display_hr2;
  byte minutes = display_min1 * 10 + display_min2;

  while (true) {
    button1 = digitalRead(inc_hr_pin);
    button2 = digitalRead(inc_min_pin);

    if ((millis() - timestampB1 >= timeout) && button1 == HIGH && lastStateB1
== LOW && button2 == HIGH) {
      hours = (hours > 22) ? 0 : ++hours;
      timestampB1 = millis();
    }

    if ((millis() - timestampB2 >= timeout) && button2 == HIGH && lastStateB2
== LOW && button1 == HIGH) {
      minutes = (minutes > 58) ? 0 : ++minutes;
      timestampB2 = millis();
    }

    display.displayDigit(hours / 10, d1_pin, false);
    display.displayDigit(hours % 10, d2_pin, true);
    display.displayDigit(minutes / 10, d3_pin, false);
    display.displayDigit(minutes % 10, d4_pin, false);

    if (button1 == LOW && button2 == LOW) break;

    lastStateB1 = button1;
    lastStateB2 = button2;
  }

  RTC.setHour(hours);
  RTC.setMinute(minutes);
  RTC.setSecond(0);
  delay(500);
}
```

Function dotBlink counts interrupts from the 32K pin and toggles dot_blink flag every half a second.

```
void dotBlink() {
  static uint16_t counter = 0;
  if (++counter % 32768 == 0) {
    dot_blink = !dot_blink;
  }
}
```

Main loop's job is to constantly display cached time and monitor button presses to enter setMode.

```
void loop() {
  display.displayDigit(display_hr1, d1_pin, false);
  display.displayDigit(display_hr2, d2_pin, dot_blink);
  display.displayDigit(display_min1, d3_pin, false);
  display.displayDigit(display_min2, d4_pin, false);

  bool btn1 = digitalRead(inc_hr_pin);
  bool btn2 = digitalRead(inc_min_pin);

  if (btn1 == LOW && btn2 == LOW) {
    setMode();
  }

  if (update) updateTime();
}
```

References

1. Arduino Nano Documentation:
   https://docs.arduino.cc/hardware/nano/
2. Dallas Semiconductor DS3231 Documentation:
   https://www.alldatasheet.pl/datasheet-pdf/pdf/112132/DALLAS/DS3231.html
3. DS3231 Library Documentation:
   https://github.com/NorthernWidget/DS3231/tree/master/Documentation
4. Kingbright CA04-41SRWA Documentation:
   https://www.kingbrightusa.com/images/catalog/SPEC/CA04-41SRWA.pdf

Future Improvements

The code has flaws. Firstly, the usage of delay functions is suboptimal for the reason that internal Arduino clock overflows after around 50 days, after which buttons and display may misbehave.

Secondly, the structure of the code is not satisfactory to me, I believe it can be improved to be shorter and not so hacky (update functions).

Thirdly, one of the best improvements to be done is to reduce the power usage. In its current state my clock can only run for around 1,5 days from a 5000mAh power bank (it might be faulty).