Introduction

In this activity, students will practice interfacing the Arduino with other devices over the I2C bus. For this, we will use a DS3231 Real Time Clock (RTC). We will cover the usage of the built-in Arduino library (Wire) as well as a device-specific library(RTClib).

Materials List

1. Adafruit DS3231 Breakout Board Kit
2. Soldering Iron and solder
3. Laptop and Programming Cable
4. Arduino

Assembling the RTC Breakout and Breadboard

1. Assembling the RTC
   1.1. Inside the bag for the RTC kit, you should find a small circuit board with the RTC chip and an 8-pin header for connecting that must be soldered to the board.
   1.2. The top side of the board has the RTC chip, a capacitor, and two pull-up resistors. [1]
   1.3. The bottom side has a socket for a coin cell battery to allow the clock to keep its time when it is not powered. You do not need to install a battery for this activity.
   1.4. Install the headers into the board and solder them in place. You should install it so that the plastic is on the bottom of the board and the short part of the header sticks through to the top.
   1.5. Solder the header pins to the breakout board. You should be applying the solder to the top of the circuit board. You can use your breadboard to hold

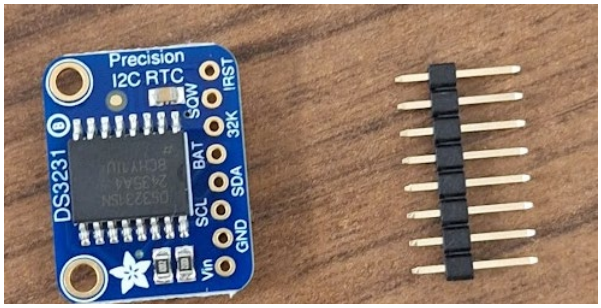

Figure 1: RTC Breakout board and header. You install the pins from the bottom side of the board.

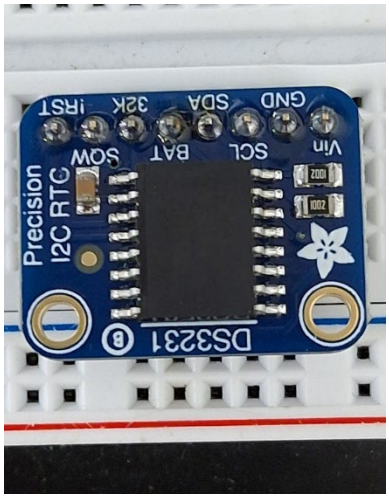

Figure :2 Soldered RTC. Notice the solder is applied to the top of the board.

the headpin in place and put the board on top while you are soldering.

1.6. When complete, it should look like Figure :2.

2. Setting up the Breadboard
   2.1. Now we need to set up the breadboard. Looking at the circuit board, locate the pins labeled Vin, GND, SCL, and SDA. These are the pins we need to connect to the Arduino.
   2.2. Vin should be connected to Arduino 5V.
   2.3. GND should connect to Arduino ground.
   2.4. SCL should connect Arduino pin 21 SCL
   2.5. SDA should connect to pin 20 SDA.
   2.6. Use your jumper kit and breadboard to make those connections. Remember that the columns of the breadboard are connected, so the pins should be arranged horizontally. A sample layout is shown in Figure 3. This uses the top + row for 5V and the bottom – to ground, with a jumper underneath the RTC bridging the chip gap. The blue and green jumpers are there to allow connection of an oscilloscope for troubleshooting if needed.

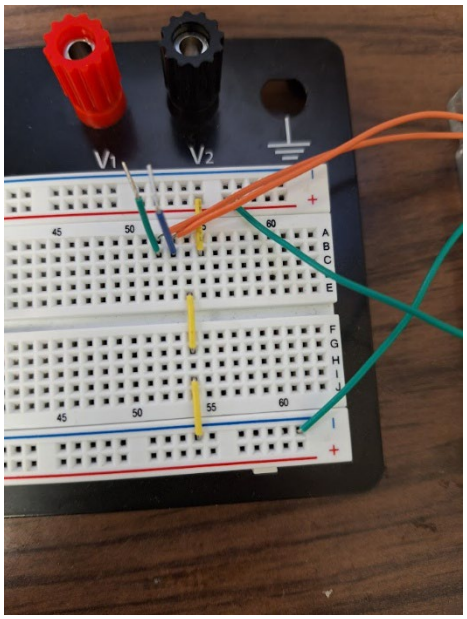

Figure 3: Sample RTC breadboard layout. The long green leads connect to 5V and ground, the orange leads connect to pins 20 and 21, the breakout board should be placed on top of these with the Vin pin on the left, lined up with the leftmost yellow jumper.

Using the Arduino Wire Library

1. Basic I2C Read operations
   1.1. First, we want to write a simple program that just requests a 1 byte of data from the RTC. We want to print the number of bytes we received (so we can tell if we got something). Then we want to print the data byte and wait one second (since the clock only updates every 1 second.) The flowchart for this code is shown in Figure 5,
   1.2. To perform I2C operations, we need to include the **Wire** library. This can be done by selecting the library from the **Sketch**>**Include Library** menu. You can also just manually type the include statement.
   1.3. We will need to know the 7-bit address of the RTC to communicate with it. If you look at the datasheet [2] For the DS3231 on page 17, this address is 110 1000, or 68 in hexadecimal(104 decimal). We want to create a variable to store that address, and if we use and 0x, that will tell the compiler we mean the number is in hex, so we do not need to convert to decimal.

1.4. Just like the Serial, we need to initialize the I2C. This is done by calling the **Wire.begin()** function in **setup(). Wire.begin()** can be used in two ways. If we give no argument, that means the Arduino will act as the I2C Master.
The less common option is give an address in **Wire.begin()** and that will make Arduino act as a slave device with that address.

1.5. To request data, we use the **Wire.requestFrom()**, giving the function the address of the device and the number of bytes of data to read. When called, the function starts an I2C read operation and copies the received data into a buffer. The function returns the number of bytes that were actually received. If the Arduino does not get any data (maybe there was a connection issue or the device address was wrong) **Wire.requestFrom()** will return 0.

1.6. After **Wire.requestFrom()**, you still need to access the received data. This is done by calling **Wire.read()**. When called, it returns the first available byte in the buffer and then removes it from the buffer. So if multiple bytes were requested you will need to call **Wire.read()** multiple times. **Wire.read()** will still return a value even if nothing is received. **Wire.available()** can be used to tell how many bytes there are are in the buffer.

1.7. Write a program to perform the operations shown in Figure 5. Your code should look like Figure 4.

```
#include <Wire.h>
byte RTC_address=0x68;
//This is the I2C address of the RTC
void setup() {
  // put your setup code here, to run once:
  Wire.begin();
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  int value_received;
  int bytes_received;

  bytes_received=Wire.requestFrom(RTC_address,1);
  //Request one byte from the RTC
  //returns the number of bytes received or 0 if no response
  Serial.print("Received Bytes: ");
  Serial.println(bytes_received);

  value_received=Wire.read();
  Serial.print("Value was: ");
  Serial.println(value_received);
  //prints out seconds
  delay(1000);

}

```

Figure 4 Basic RTC readout code, requests a single byte and prints the result to the serial port.

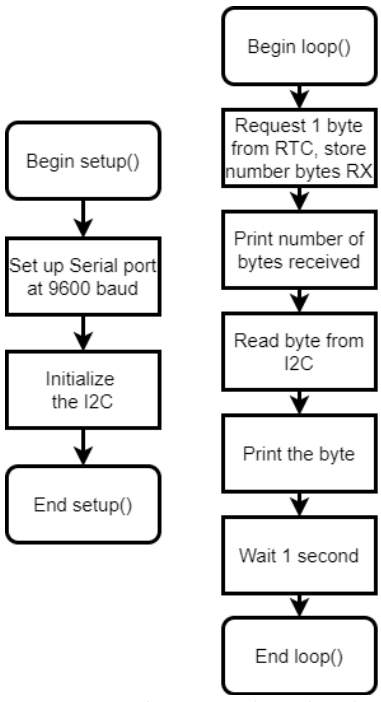

Figure 5: Simple RTC Readout Flowchart

1.8. Open the serial monitor, and you should see something similar to the Serial monitor in Figure 7. If your received bytes are 0, the Arduino is not communicating with the RTC correctly. Check your connections and Address. You may need to connect a scope to the SDA and SCL pins to verify you see traffic on both the clock (SCL) and data (SDA) lines. It should look like Figure 6.

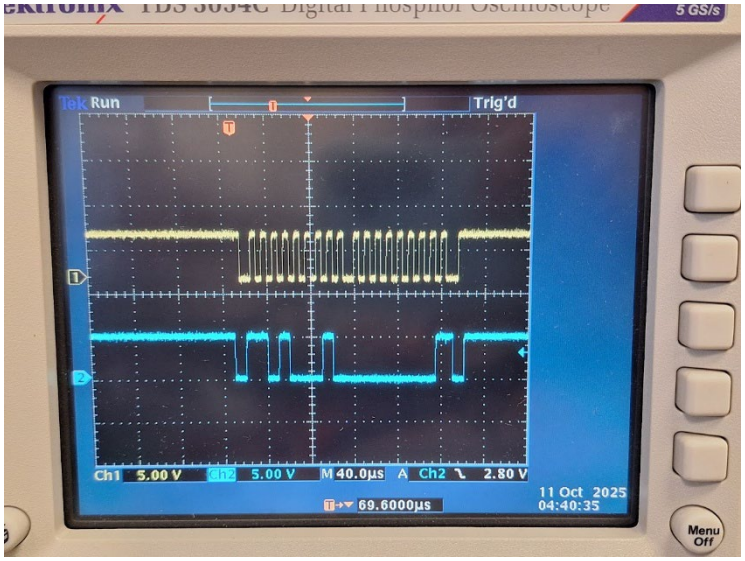

Figure 6: The waveform of the initial RTC read. The clock line is shown at the top in yellow, and the data line is shown below in blue. If you look closely you can see a slightly larger gap in the clock between the first byte (address) and second (data from the RTC).

```
Value was: 22
Received Bytes: 1
Value was: 5
Received Bytes: 1
Value was: 16
Received Bytes: 1
Value was: 16
Received Bytes: 1
Value was: 37
Received Bytes: 1
Value was: 0
Received Bytes: 1
Value was: 0
Received Bytes: 1
Value was: 0
Received Bytes: 1
Value was: 0
```

Figure 7: Output of the serial monitor when reading the RTC.

2. Reading a specific register from the RTC

2.1. Once you are sure you are receiving data, look at the actual values. The numbers appear to be jumping around randomly. How is this a clock?

2.2. If you look at the table on page 11 of the datasheet [2] We see the clock has several different data registers storing different pieces of data like Seconds, Minutes, Hours, etc. So, what data are we actually getting when we request a byte of data?

2.3. If you read page 17 of the datasheet [2] we get the explanation. The clock has an internal pointer that moves from register to register. When a read occurs, the clock sends the byte value of the register the pointer has selected, and then the pointer moves to the next register in the order listed in the table.

When we were reading, values were cycling through seconds, minutes, hours, days, etc.

2.4. How do we select a specific register so we can read a particular piece of data? Again, if we look at page 17 of the datasheet [2] it tells. If we do an I2C write operation, the first

byte of data after the device address will be the internal address of the data register, and the pointer will move to that register.

2.5. So to read seconds, we need to start a write, send 0x00, request 1 byte of data, and then read the received byte.

2.6. I2C writes are accomplished with 3 functions. First, **Wire.beginTransmission()** is used to set the address of the device to which the data will be written. Then **Wire.write()** is used to add bytes to the transmit buffer on the Arduino. Finally, **Wire.endTransmission()** is called. That function actually sends the data in the transmit buffer to the target address.

2.7. So to read the seconds from the clock, we need a program that will do the steps shown in Figure 8.

2.8. Add the code to read off the seconds from the RTC. You program should look like *Figure 9*.

2.9. Now look at the output on your serial monitor. You should see numbers counting up as expected, but occasionally, it will skip some numbers.

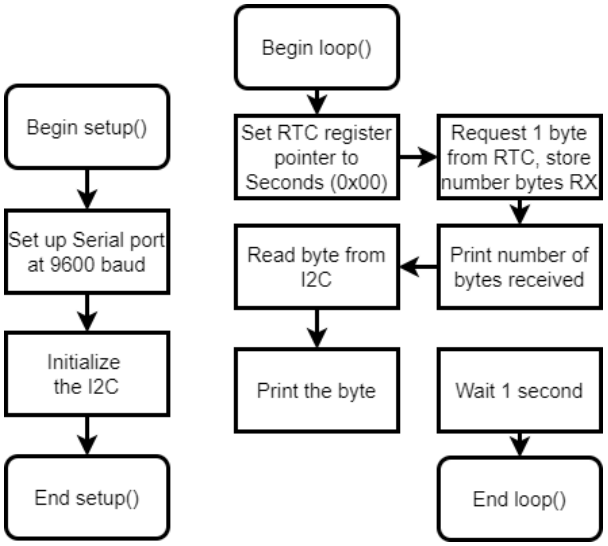

Figure 8 I2C Seconds readout flowchart, with the additional step to set the pointer to the Seconds register.

```
10  void loop() {
11    // put your main code here, to run repeatedly:
12    int value_received;
13    int bytes_received;
14    int converted_value;
15
16    Wire.beginTransmission(RTC_address);
17    //Queues the Address + Write bit in the transmit queue
18    Wire.write(0x00);
19    //Adds the pointer address to the queue
20    Wire.endTransmission();
21    //Close the queue and transmit the bytes on the data line
22
23    bytes_received=Wire.requestFrom(RTC_address,1);
24    //Request one byte from the RTC
25    //returns the number of bytes received or 0 if no response
26    Serial.print("Received Bytes: ");
27    Serial.println(bytes_received);
28
29    value_received=Wire.read();
30    Serial.print("Value was: ");
31    Serial.println(value_received);
32    //prints out seconds
33    delay(1000);
34
```

Figure 9 I2C Seconds Readout Code, lines 16 to 21 perform an I2C write operation to move the RTC pointer.

3. Converting Binary Coded Decimal Numbers

3.1. Let's add a print function to show the raw bits received to see if we can figure it out. We can do this by adding (BIN) as an argument to the Serial.print() function. This tells the Serial to print out the series of 0 and 1 characters matching the byte.

3.2. Now we can see that when the seconds go from 9 to 10, the binary goes from 1001 (9) to 10000 (16). Looking again at the table on page 11 of the datasheet, we see that the numbers are not stored in binary format. Instead, Bits 0-3 are used for the decimal 1s digit, and bits 4-6 are used for the 10s digit. This format is called Binary Coded Decimal.

```
29  value_received=Wire.read();
30  Serial.print("Value was: ");
31  Serial.print(value_received, BIN);
32  //Outputs the received byte in binary
33  Serial.print(" ");
34  Serial.println(value_received);
35  //prints out seconds
36  delay(1000);
```

Figure 10 Adding a second print to output the received byte in binary

3.3. We want to convert these BCD numbers to a regular integer. Let's write a function to do so, so that we can use it for the minutes as well.

3.4. We need to separate the first 4 bits upper 3 bits into two numbers, then multiply the upper 3 bits by 10 and add them together.

```
Value was: 1000 8
Received Bytes: 1
Value was: 1001 9
Received Bytes: 1
Value was: 10000 16
Received Bytes: 1
Value was: 10001 17
Received Bytes: 1
Value was: 10010 18
```

Figure 11: BCD rollover from 9 to 16, notice how once it gets to 9, the next bit is added in the fourth place. Normally, the next binary should be 1010 for 10.

3.5. Since every bit is a power of 2, we can use division by powers of two to split the bits. If we divide a byte by 16 (2^4), we will get bits 4-7, bits 0-3 are the remainder and will be discarded since this is integer division. We can use the modulus operator **%** to get bits 0-3.

3.6. Now write a function to take the received BCD. Do those calculations and return the converted number. Your code should look like Figure 12.

```
42  int BCDtoDEC(int BCD){
43    int ones;
44    int tens;
45    int decimal;
46
47    ones=BCD%16;
48    //Gives bits 0-3
49    tens=BCD/16;
50    /*Drops Bits 0-3, shifting 4-7
51    to the right */
52    decimal=tens*10+ones;
53    return decimal;
54  }
```

Figure 12: BCD Conversion function.

3.7. Now we just need to call our conversion function in the loop and print the converted number. Now our code should perform as Figure 14. At this point, your code should look like Figure 13.

3.8. Upload your code, and it should now properly be counting the seconds, and you should also see it roll over from 59 back to 0.

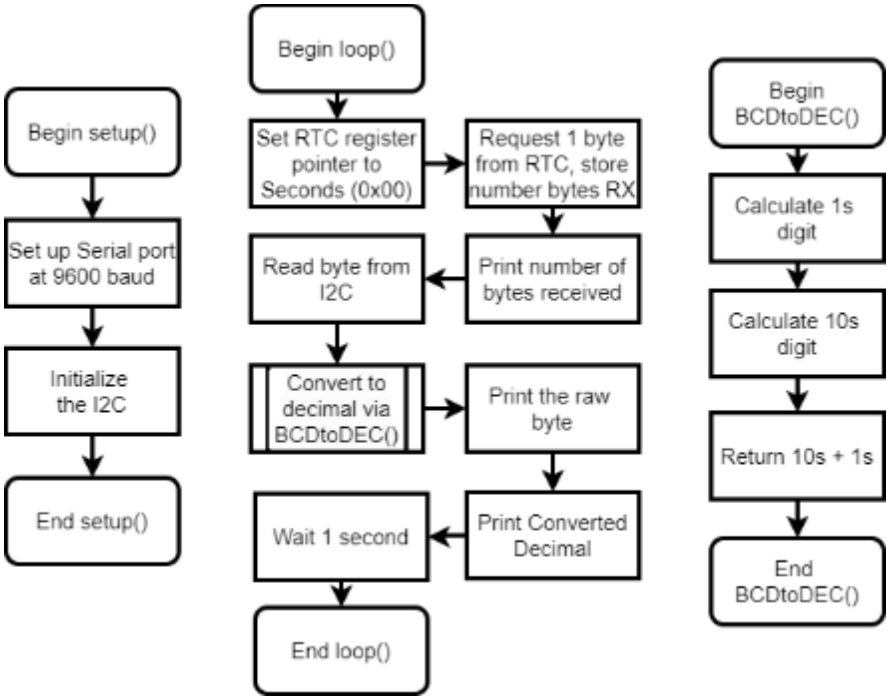

Figure 13 I2C Seconds readout with the addition of the BCDtoDEC() function. Notice how the function has its own separate subroutine.

```
10  void loop() {
11    // put your main code here, to run repeatedly:
12    int value_received;
13    int bytes_received;
14    int converted_value;
15
16    Wire.beginTransmission(RTC_address);
17    //Queues the Address + Write bit in the transmit queue
18    Wire.write(0x00);
19    //Adds the pointer address to the queue
20    Wire.endTransmission();
21    //Close the queue and transmit the bytes on the data line
22
23    bytes_received=Wire.requestFrom(RTC_address,1);
24    //Request one byte from the RTC
25    //returns the number of bytes received or 0 if no response
26    Serial.print("Received Bytes: ");
27    Serial.println(bytes_received);
28
29    value_received=Wire.read();
30    Serial.print("Value was: ");
31    Serial.print(value_received, BIN);
32    //prints the raw byte in binary
33    converted_value=BCDtoDEC(value_received);
34    //converts the BCD value to regular integer format
35    Serial.print(" ");
36    Serial.println(converted_value);
37    //prints out seconds
38    delay(1000);
39
40  }
```

Figure 14: Main loop now modified to convert the received byte using the BCDtoDEC() function.

Using the Adafruit RTC Library

1. Installing the RTClib Library
   1.1. As you can see, manually handling the I2C communications is possible, but often there will be a library that can handle the low-level functions. Such a library exists for this RTC breakout board.
   1.2. Open your library manager, and search for “rtclib”. You should see a library called RTClib written by Adafruit (the company that makes the breakout board). Install that library and the required dependencies.
   1.3. Now, let's look at the library's documentation to see if we can figure out how to use it. From the File menu, select Examples. Near the bottom of the listed libraries, you should see RTClib, and inside of that, you will see several examples for different RTCs.

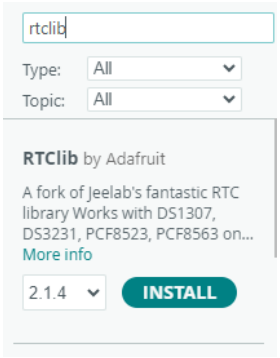

Figure 15: This is the RTClib library we want to install.

   1.4. Select the DS3231 example to open the example sketch.
   1.5. If we look at the code, we will see a short program that demonstrates how to use the code. You should be able to compile and upload the code; do so now.
   1.6. When you open the serial monitor, you will probably need to change the baud rate. Looking at line 9 of the example, we can see they set the baud rate to 57600.
   1.7. Now you should see the time, date, and some other information printed out every 3 seconds.
   1.8. Let's look back at the code.
   1.9. On line 4, we can see there is a new variable type for RTC. We can also see a new type called DateTime that can be used for storing dates and times.

```
1  // Date and time functions u
2  #include "RTClib.h"
3
4  RTC_DS3231 rtc;
```

```
24   // following line sets the RTC to the date & time this sketch was compiled
25   rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
26   // This line sets the RTC with an explicit date & time, for example to set
27   // January 21, 2014 at 3am you would call:
28   // rtc.adjust(DateTime(2014, 1, 21, 3, 0, 0));
29 }
40   DateTime now = rtc.now();
41
42   Serial.print(now.year(), DEC);
43   Serial.print('/');
44   Serial.print(now.month(), DEC);
45   Serial.print('/');
46   Serial.print(now.day(), DEC);
47   Serial.print(" (");
48   Serial.print(daysOfTheWeek[now.dayOfTheWeek()]);
49   Serial.print(") ");
```

Figure 16: Snippets from the DS3231 example. These lines give us an idea of how to use the new variable types and functions added by the library.

   1.10. We can see a few functions for the RTC **begin()** (line 15), **adjust()** (line 25), and **now()** (line 40). From the code and the comments, we can guess that **adjust()** is used to set the clock and **now()** reads it. We can also see some functions for getting the hour, minutes, seconds, etc., out of the DateTime variables.
2. Navigating the library documentation

2.1. While we can infer some information from the example, we may need more detailed documentation from the library so we are not just guessing.

2.2. Open the library manager again and search for RTClib. Click on the “more info” link. This will open a link to github where the library is hosted. We can see this has copies of the library code itself and if we scroll down on the page, some links and documentation.

2.3. If you scroll to the bottom of the page you will find a “Documentation and doxygen” section. If you click on the first link in this section, you will go to the detailed documentation for the library.

2.4. We can see the new variable types the library adds (called Classes, a type of user-defined variable).

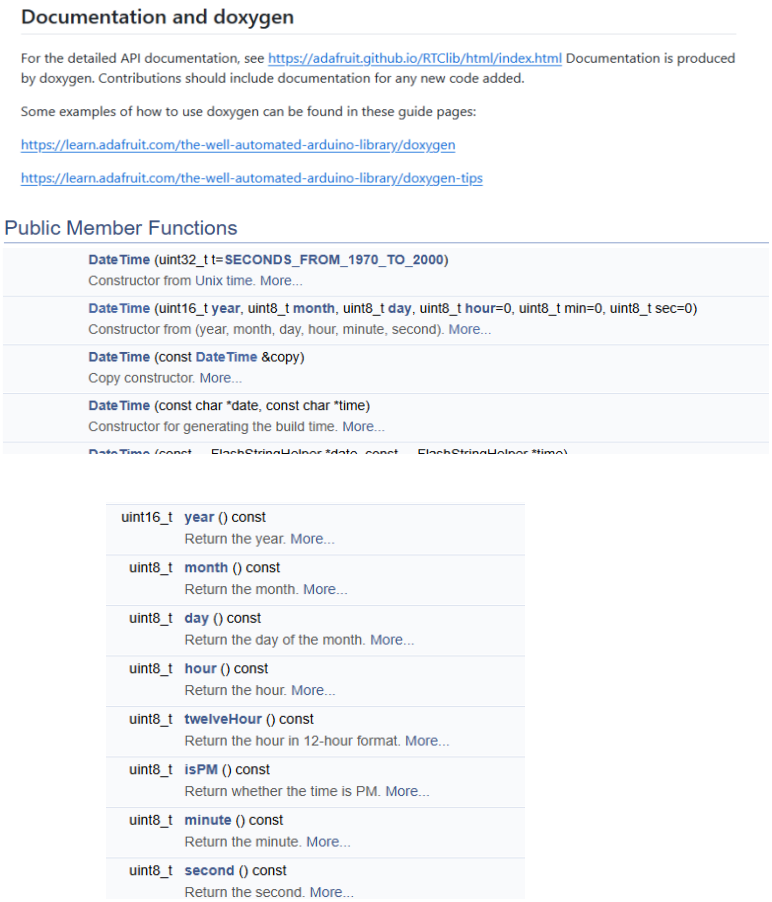

Figure 17: By following the links from the library repository, we can find the more detailed documentation. This documentation was generated by an automated tool called doxygen from the code and comments.

2.5. You can click on the DateTime and DS3231, and you will find detailed documentation of the variable types and functions. Read through the documentation and example until you have an understanding of the basic library functions.

3. Set and read the clock using RTClib
   3.1. Using the documentation from the example and the documentation website to write a program that does the following tasks, as shown in Figure 18.
      3.1.1. Sets the RTC to the given initial date and time.
      3.1.2. Reads the clock every second
      3.1.3. Prints the data date time out in the given format

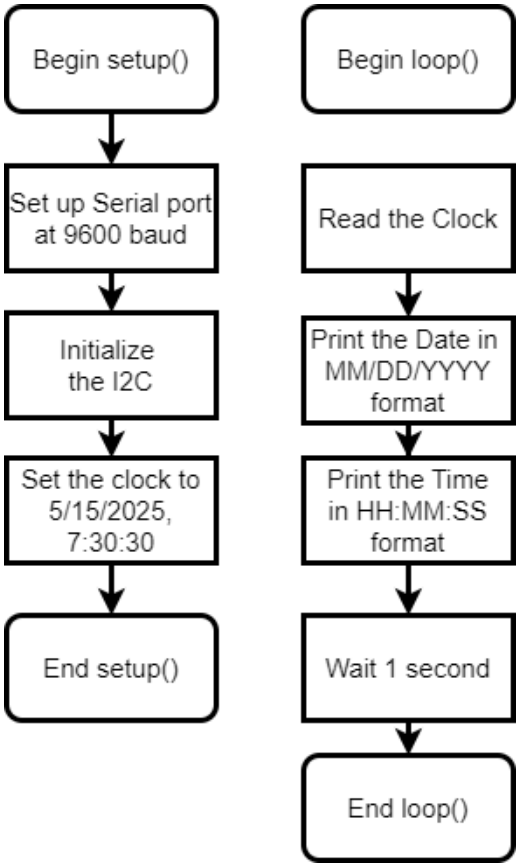

Figure 18: Programming tasks for reading the clock using RTClib.

References

[1] Adafruit, "Adafruit DS3231 Precision RTC Breakou," [Online]. Available: https://learn.adafruit.com/adafruit-ds3231-precision-rtc-breakout/.

[2] Analog Devices, "DS3231 Real time clock datasheet," [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/DS3231.pdf.