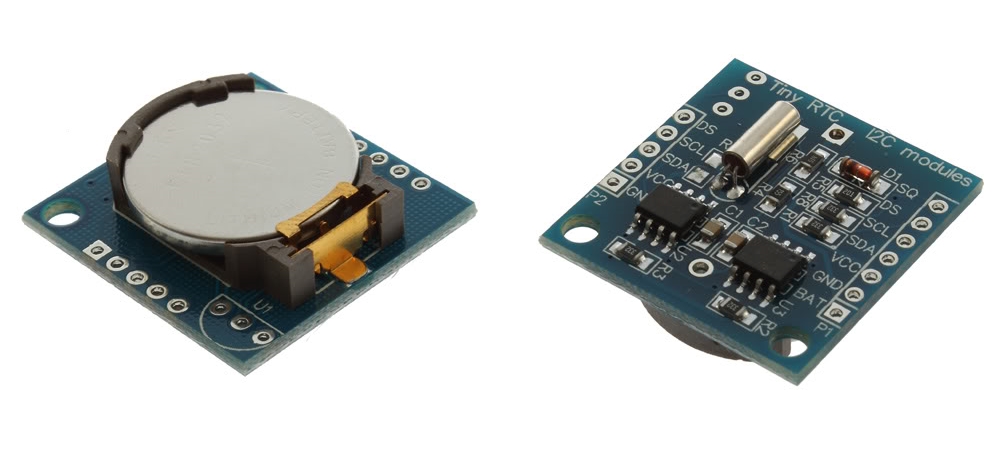


Relógio com o módulo RTC DS1307[315](https://www.filipeflop.com/blog/relogio-rtc-ds1307-arduino/#comments)

[Arduino](https://www.filipeflop.com/blog/category/arduino/), [Módulos](https://www.filipeflop.com/blog/category/modulos/)  [11 de junho de 2014](https://www.filipeflop.com/blog/relogio-rtc-ds1307-arduino/) [Adilson Thomsen](https://www.filipeflop.com/blog/author/adilsonth/)

Com o módulo [RTC DS1307](https://www.filipeflop.com/produto/real-time-clock-rtc-ds1307/) você tem um componente muito útil para montar algum tipo de relógio com o Arduino, setar alarmes e assim executar ações em horários predeterminados. Neste post vamos apresentar um tutorial de como conectar este módulo RTC DS1307 com um Arduino Uno, mostrando as informações de data e hora no *Serial Monitor*da IDE Arduino*.*

[](https://www.filipeflop.com/pd-6b854-real-time-clock-rtc-ds1307.html)

A sigla RTC significa Real Time Clock, ou seja, um Relógio de Tempo Real (RTC). Esse módulo tem 56 bytes de memória não-volátil disponível para uso, é capaz de armazenar e fornecer informações completas de data como dia da semana, dia do mês, mês, ano e além é claro, das funções de horas, minutos e segundos, nos formatos de 12 ou 24 horas. Meses com menos de 31 dias e anos bissextos são ajustados automaticamente.

Uma bateria de lítio garante que os dados sejam preservados mesmo sem alimentação externa, e é acionada automaticamente em caso de falta de energia no módulo.

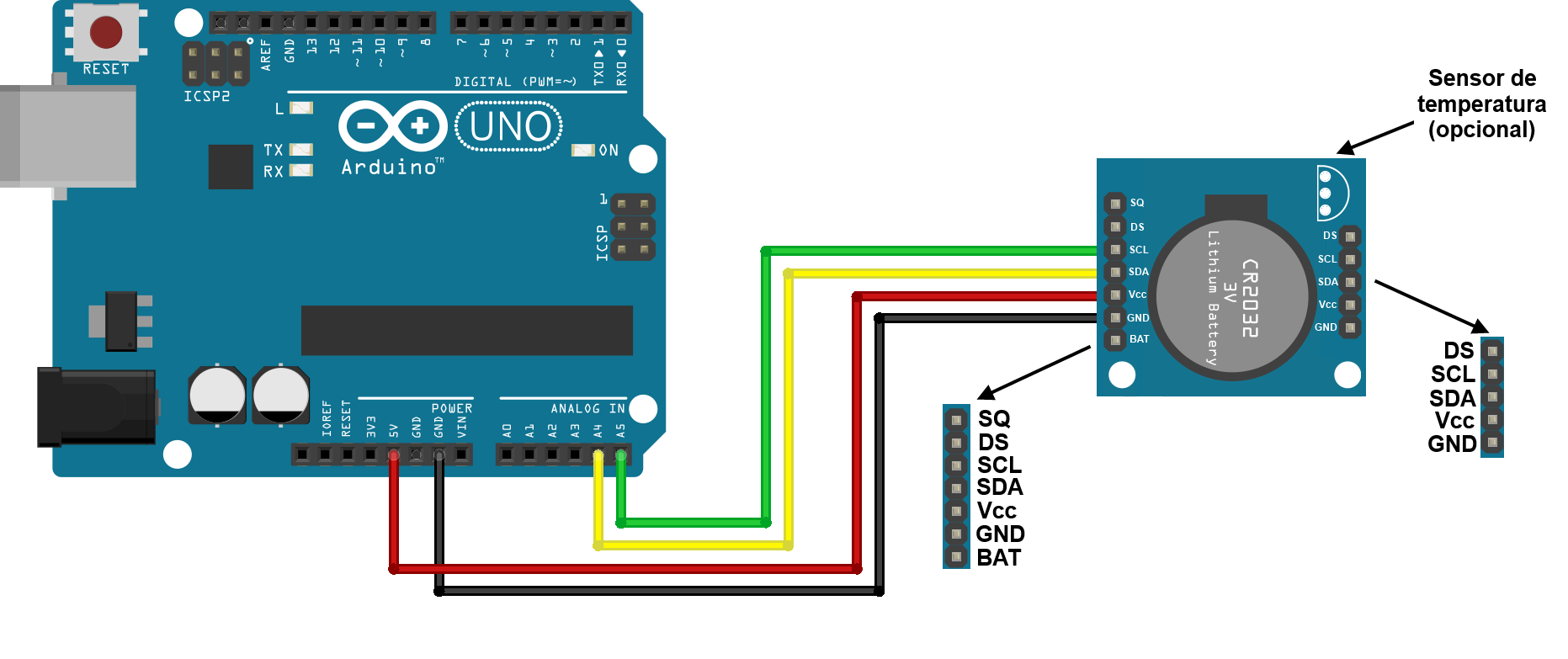
Uma outra característica desse módulo é que você pode utilizar um sensor de temperatura **DS18B20** (não incluso), e ler as informações do sensor à partir do pino DS do módulo, o que faz com que seja possível montar um relógio completo com data, hora, dia da semana e temperatura, sem a necessidade de outros componentes.

**CONECTANDO RTC DS1307 AO ARDUINO:**

Neste post vamos mostrar um breve tutorial de como ligar esse módulo RTC DS1307 ao Arduino e exibir todas essas informações no serial monitor.

Olhando o módulo pela parte inferior, podemos ver na parte esquerda os pinos**GND, Vcc, SDA e SCL**, utilizados para cascatear dispositivos I2C, e também o pino **DS**, que fornece os dados do sensor de temperatura, se o mesmo estiver instalado. No lado direito, vamos utilizar apenas os pinos **SCL, SDA, Vcc e GND** para ligação ao Arduino.

Conecte o módulo RTC DS1307 ao Arduino, tomando cuidado para não inverter as ligações, pois como vimos acima existem pinos com o mesmo nome dos dois lados do módulo :



Antes de carregar o programa, baixe a biblioteca RTC DS1307 [nesse link](https://github.com/filipeflop/DS1307), descompacte o arquivo e copie o conteúdo para a pasta **LIBRARIES** da **IDE** do seu Arduino.

No programa, a parte do código que contém os comandos

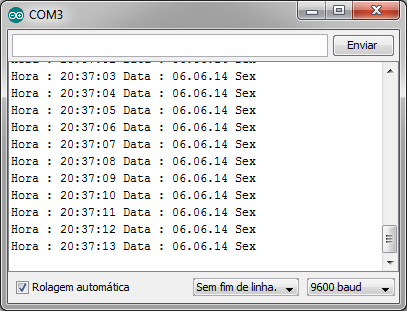
***rtc.setDOW(MONDAY);***  
***rtc.setTime(16, 30, 47);***  
***rtc.setDate(5, 6, 2014);***

só deve ser utilizada para setar a hora e data do RTC DS1307. Depois disso, essas linhas podem ser comentadas ou excluídas e o programa deve ser novamente carregado no Arduino.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | //Programa : Relogio com modulo RTC DS1307  //Autor : FILIPEFLOP    //Carrega a biblioteca do RTC DS1307  #include <DS1307.h>    //Modulo RTC DS1307 ligado as portas A4 e A5 do Arduino  DS1307 rtc(A4, A5);    void setup()  {    //Aciona o relogio    rtc.halt(false);      //As linhas abaixo setam a data e hora do modulo    //e podem ser comentada apos a primeira utilizacao    rtc.setDOW(FRIDAY);      //Define o dia da semana    rtc.setTime(20, 37, 0);     //Define o horario    rtc.setDate(6, 6, 2014);   //Define o dia, mes e ano      //Definicoes do pino SQW/Out    rtc.setSQWRate(SQW\_RATE\_1);    rtc.enableSQW(true);      Serial.begin(9600);  }    void loop()  {    //Mostra as informações no Serial Monitor    Serial.print("Hora : ");    Serial.print(rtc.getTimeStr());    Serial.print(" ");    Serial.print("Data : ");    Serial.print(rtc.getDateStr(FORMAT\_SHORT));    Serial.print(" ");    Serial.println(rtc.getDOWStr(FORMAT\_SHORT));      //Aguarda 1 segundo e repete o processo    delay (1000);  } |

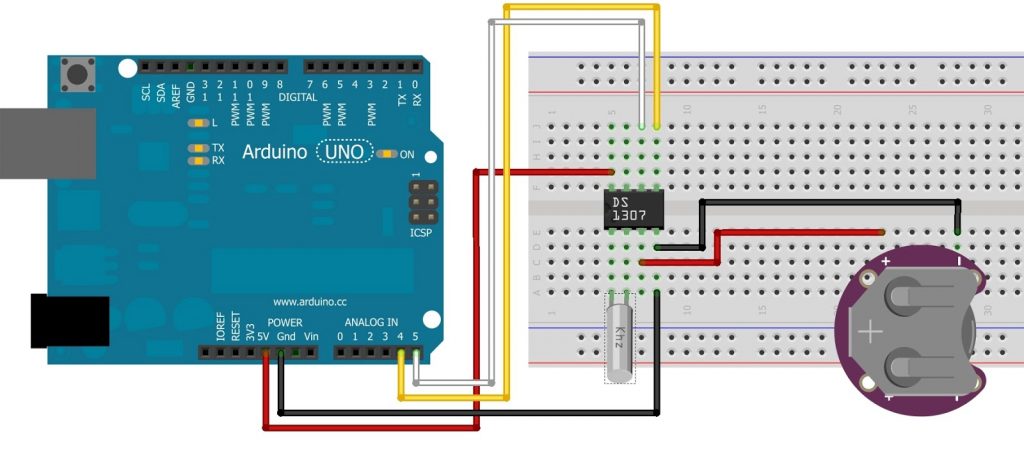
[?](https://www.filipeflop.com/blog/relogio-rtc-ds1307-arduino/)

Execute o programa, abra o *Serial Monitor* e você terá as informações do RTC DS1307 dessa maneira :



Essa biblioteca pode mostrar as informações da data em formato completo bastando retirar as informações de **FORMAT\_SHORT** (formato reduzido) do programa.

***Utilização directa do IC ds1307***



Um detalhe é a bateria, que é obrigatória nesse circuito. Sem ela, o CI gera alguns resultados bem estranhos. A bateria serve também para que, caso o Arduino seja desligado, as informações de data e hora sejam preservadas. O consumo do CI é baixíssimo (500 nano ampéres), o que significa que uma bateria de 3V ligada permanentemente ao circuito dura muito tempo.

No meu caso, usei uma CR2032, dessas que a gente encontra na motherboard do computador.

// Programa : Data e Hora com DS1307

// Autor : Arduino e Cia

#include "Wire.h"

#define DS1307\_ADDRESS 0x68

byte zero = 0x00;

void setup()

{

Wire.begin();

Serial.begin(9600);

//A linha abaixo pode ser retirada apos setar a data e hora

SelecionaDataeHora();

}

void loop()

{

Mostrarelogio();

delay(1000);

}

void SelecionaDataeHora() //Seta a data e a hora do DS1307

{

byte segundos = 40; //Valores de 0 a 59

byte minutos = 59; //Valores de 0 a 59

byte horas = 23; //Valores de 0 a 23

byte diadasemana = 1; //Valores de 0 a 6 - 0=Domingo, 1 = Segunda, etc.

byte diadomes = 17; //Valores de 1 a 31

byte mes = 4; //Valores de 1 a 12

byte ano = 13; //Valores de 0 a 99

Wire.beginTransmission(DS1307\_ADDRESS);

Wire.write(zero); //Stop no CI para que o mesmo possa receber os dados

//As linhas abaixo escrevem no CI os valores de

//data e hora que foram colocados nas variaveis acima

Wire.write(ConverteParaBCD(segundos));

Wire.write(ConverteParaBCD(minutos));

Wire.write(ConverteParaBCD(horas));

Wire.write(ConverteParaBCD(diadasemana));

Wire.write(ConverteParaBCD(diadomes));

Wire.write(ConverteParaBCD(mes));

Wire.write(ConverteParaBCD(ano));

Wire.write(zero); //Start no CI

Wire.endTransmission();

}

byte ConverteParaBCD(byte val){ //Converte o número de decimal para BCD

return ( (val/10\*16) + (val%10) );

}

byte ConverteparaDecimal(byte val) { //Converte de BCD para decimal

return ( (val/16\*10) + (val%16) );

}

void Mostrarelogio()

{

Wire.beginTransmission(DS1307\_ADDRESS);

Wire.write(zero);

Wire.endTransmission();

Wire.requestFrom(DS1307\_ADDRESS, 7);

int segundos = ConverteparaDecimal(Wire.read());

int minutos = ConverteparaDecimal(Wire.read());

int horas = ConverteparaDecimal(Wire.read() & 0b111111);

int diadasemana = ConverteparaDecimal(Wire.read());

int diadomes = ConverteparaDecimal(Wire.read());

int mes = ConverteparaDecimal(Wire.read());

int ano = ConverteparaDecimal(Wire.read());

//Mostra a data no Serial Monitor

Serial.print("Data: ");

Serial.print(diadomes);

Serial.print("/");

Serial.print(mes);

Serial.print("/");

Serial.print(ano);

Serial.print(" ");

Serial.print("Hora : ");

Serial.print(horas);

Serial.print(":");

Serial.print(minutos);

Serial.print(":");

Serial.print(segundos);

switch(diadasemana)

{

case 0:Serial.println(", Domingo");

break;

case 1:Serial.println(", Segunda");

break;

case 2:Serial.println(", Terca");

break;

case 3:Serial.println(", Quarta");

break;

case 4:Serial.println(", Quinta");

break;

case 5:Serial.println(", Sexta");

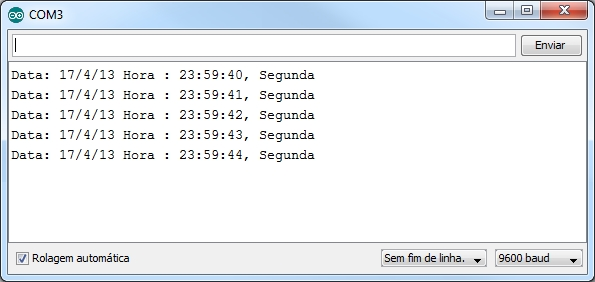
break;

case 6:Serial.println(", Sabado");

}

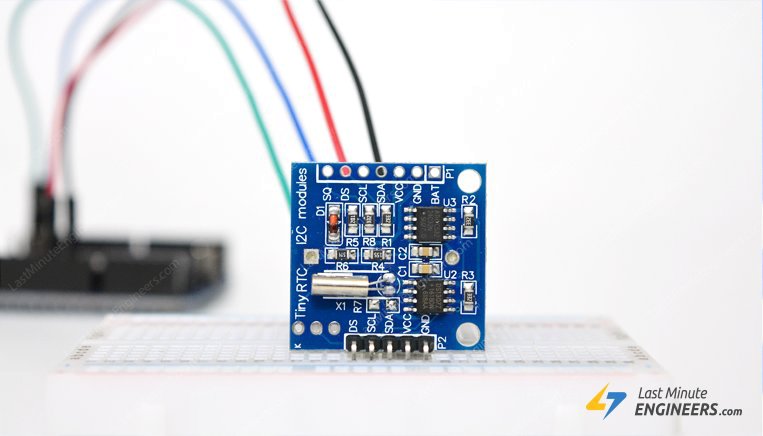
}

O que ele faz nada mais é do que setar a data e a hora, como indicado no programa, e depois mostrar esses valores no serial monitor, como na figura abaixo:



A **linha 15** do programa, com a indicação ***SelecionaDataeHora()***, é utilizada para setar os dados iniciais do relógio e pode ser comentada posteriormente, já que o DS1307 estará sendo alimentado pela bateria.

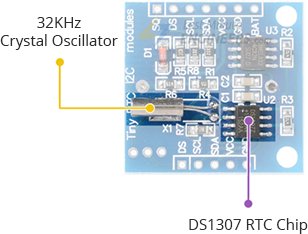
# Interface DS1307 RTC Module with Arduino



We all know that most MCUs we use for our projects are time-agnostic; simply put they are unaware of the time around them. It’s OK for most of our projects but once in a while when you come across an idea where keeping time is a prime concern, DS1307 RTC module is a savior. It’s perfect for projects containing data-logging, clock-building, time stamping, timers and alarms.

## DS1307 RTC chip

At the heart of the module is a low-cost, quite accurate RTC chip from Maxim – **DS1307**. It manages all timekeeping functions and features a simple two-wire I2C interface which can be easily interfaced with any microcontroller of your choice.



The chip maintains seconds, minutes, hours, day, date, month, and year information. The date at the end of the month is automatically adjusted for months with fewer than 31 days, including corrections for leap year (valid up to 2100). The clock operates in either the 24-hour or 12-hour format with an AM/PM indicator.

The other cool feature of this board comes with SQW pin, which outputs one of four square-wave frequencies 1Hz, 4kHz, 8kHz or 32kHz and can be enabled programmatically.

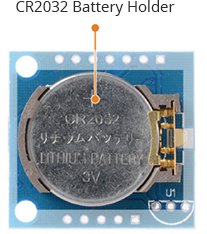
DS1307 come with an external 32kHz crystal for time-keeping. The problem with these crystals is that external temperature can affect their oscillation frequency. This change in frequency is negligible but it surely adds up.

This may sound like a problem, but it’s not. It actually results with the clock being off by around five or so minutes per month.

## Battery Backup

The DS1307 incorporates a battery input, and maintains accurate timekeeping when main power to the device is interrupted.

The built-in power-sense circuit continuously monitors the status of VCC to detect power failures and automatically switches to the backup supply. So, you need not worry about power outages, your MCU can still keep track of time.



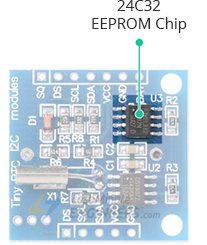
The bottom side of the board holds a battery holder for 20mm 3V lithium coincells. Any CR2032 battery can fit well.

Assuming a fully charged CR2032 battery with capacity 47mAh is used and chip consumes its minimum 300nA, the battey can keep the RTC running for a minimum of 17.87 years without an external 5V power supply.

**47mAh/300nA = 156666.67 hours = 6527.78 days = 17.87 years**

## Onboard 24C32 EEPROM

DS1307 RTC module also comes with a 32 bytes 24C32 EEPROM chip from Atmel having limited read-write cycles. It can be used to save settings or really anything.



The 24C32 EEPROM uses I2C interface for communication and shares the same I2C bus as DS1307.

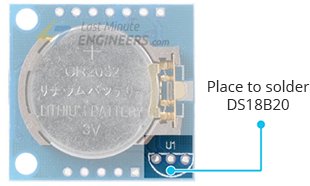
The onboard 24C32 EEPROM has a hardwired I2C address and is set to 0x50HEX

Code for reading/writing onboard 24C32 EEPROM is given at the [end of the tutorial](https://lastminuteengineers.com/ds1307-rtc-arduino-tutorial/#arduino-code-reading-writing-in-24c32-eeprom).

## Module’s Hidden Feature – DS18B20

There’s a provision on our DS1307 RTC module that often goes unnoticed. It allows us to install DS18B20 temperature sensor.

The 3 holes in the top corner right next to the battery holder(labled as U1) is where the DS18B20 is installed.

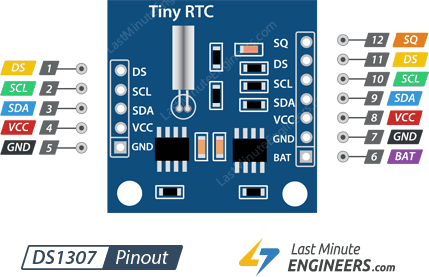


Once you install the DS18B20, you will be able to get temperature readings from **DS** pin. These readings can further be used to compensate for temperature based time drift in code.

Follow the silk-screen to solder DS18B20. You might also need a 4.7K resistor between VCC and DS.

## DS1307 RTC Module Pinout

The DS1307 RTC module has total 7 pins that interface it to the outside world. The connections are as follows:



SQW pin outputs one of four square-wave frequencies 1Hz, 4kHz, 8kHz or 32kHz and can be enabled programmatically.

DS pin is supposed output temperature readings if your module has a DS18B20 temperature sensor installed right next to the battery holder(labled as U1).

SCL is the clock input for the I2C interface and is used to synchronize data movement on the serial interface.

SDA is the data input/output for the I2C serial interface.

VCC pin supplies power for the module. It can be anywhere between 3.3V to 5.5V.

GND is a ground pin.

BAT is a backup supply input for any standard 3V lithium cell or other energy source to maintain accurate timekeeping when main power to the device is interrupted.

## Wiring DS1307 RTC module to Arduino UNO

Let’s hook the RTC up to the Arduino.

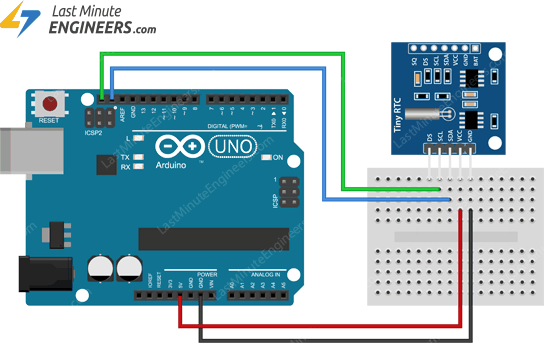
Connections are fairly simple. Start by connecting VCC pin to the 5V output on the Arduino and connect GND to ground.

Now we are remaining with the pins that are used for I2C communication. Note that each Arduino Board has different I2C pins which should be connected accordingly. On the Arduino boards with the R3 layout, the SDA (data line) and SCL (clock line) are on the pin headers close to the AREF pin. They are also known as A5 (SCL) and A4 (SDA).

If you have a Mega, the pins are different! You’ll want to use digital 21 (SCL) and 20 (SDA). Refer below table for quick understanding.

|  |  |  |
| --- | --- | --- |
|  | SCL | SDA |
| Arduino Uno | A5 | A4 |
| Arduino Nano | A5 | A4 |
| Arduino Mega | 21 | 20 |
| Leonardo/Micro | 3 | 2 |

The following diagram shows you how to wire everything.

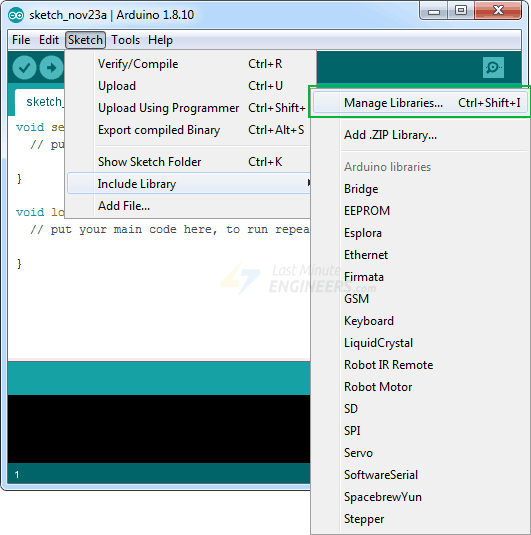


Wiring DS1307 RTC module with Arduino

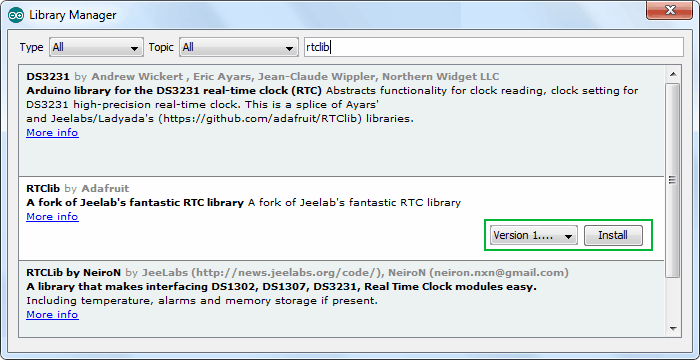
## Installing RTClib library

Communicating with a RTC module is a bunch of work. Fortunately, [RTClib library](https://github.com/adafruit/RTClib) was written to hide away all the complexities so that we can issue simple commands to read the RTC data.

To install the library navigate to the Sketch > Include Library > Manage Libraries…Wait for Library Manager to download libraries index and update list of installed libraries.



Filter your search by typing ‘**rtclib**’. There should be a couple entries. Look for RTClib by Adafruit. Click on that entry, and then select Install.



## Arduino Code – Reading Date & Time

The following sketch will give you complete understanding on how to set/read date & time on DS1307 RTC module and can serve as the basis for more practical experiments and projects.

#include <Wire.h>

#include "RTClib.h"

RTC\_DS1307 rtc;

char daysOfTheWeek[7][12] = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"};

void setup ()

{

Serial.begin(9600);

delay(3000); // wait for console opening

if (! rtc.begin()) {

Serial.println("Couldn't find RTC");

while (1);

}

if (!rtc.isrunning()) {

Serial.println("RTC lost power, lets set the time!");

// Comment out below lines once you set the date & time.

// Following line sets the RTC to the date & time this sketch was compiled

rtc.adjust(DateTime(F(\_\_DATE\_\_), F(\_\_TIME\_\_)));

// Following line sets the RTC with an explicit date & time

// for example to set January 27 2017 at 12:56 you would call:

// rtc.adjust(DateTime(2017, 1, 27, 12, 56, 0));

}

}

void loop ()

{

DateTime now = rtc.now();

Serial.println("Current Date & Time: ");

Serial.print(now.year(), DEC);

Serial.print('/');

Serial.print(now.month(), DEC);

Serial.print('/');

Serial.print(now.day(), DEC);

Serial.print(" (");

Serial.print(daysOfTheWeek[now.dayOfTheWeek()]);

Serial.print(") ");

Serial.print(now.hour(), DEC);

Serial.print(':');

Serial.print(now.minute(), DEC);

Serial.print(':');

Serial.print(now.second(), DEC);

Serial.println();

Serial.println("Unix Time: ");

Serial.print("elapsed ");

Serial.print(now.unixtime());

Serial.print(" seconds/");

Serial.print(now.unixtime() / 86400L);

Serial.println(" days since 1/1/1970");

// calculate a date which is 7 days & 30 seconds into the future

DateTime future (now + TimeSpan(7,0,0,30));

Serial.println("Future Date & Time (Now + 7days & 30s): ");

Serial.print(future.year(), DEC);

Serial.print('/');

Serial.print(future.month(), DEC);

Serial.print('/');

Serial.print(future.day(), DEC);

Serial.print(' ');

Serial.print(future.hour(), DEC);

Serial.print(':');

Serial.print(future.minute(), DEC);

Serial.print(':');

Serial.print(future.second(), DEC);

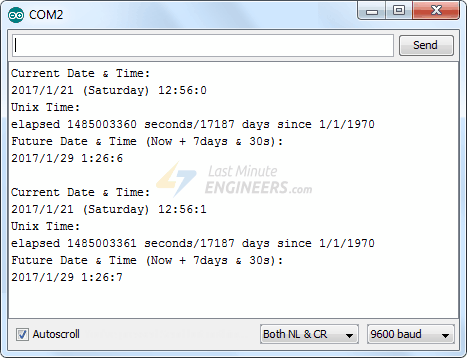
Serial.println();

Serial.println();

delay(1000);

}

Here’s how the output looks like in the serial monitor.

DS1307 Output On Serial Monitor

### Code Explanation:

The sketch starts with including wire.h & RTClib.h libraries for communicating with the module. We then create an object of RTClib library and define daysOfTheWeek 2D character array to store days information.

In setup and loop sections of the code, we use following functions to interact with the RTC module.

**begin()** function ensures that the RTC module is connected.

**isrunning()** function reads the DS1307’s internal I2C registers to check if the chip has lost track of time. If the function returns false, we can then set the date & time.

**adjust()** function sets the date & time. This is an overloaded function.

* One overloaded method DateTime(F(\_\_DATE\_\_), F(\_\_TIME\_\_)) sets the date & time at which the sketch was compiled.
* Second overloaded method DateTime(YYYY, M, D, H, M, s) sets the RTC with an explicit date & time. For example to set January 27 2017 at 12:56 you would call: rtc.adjust(DateTime(2017, 1, 27, 12, 56, 0));

**now()** function returns current date & time. Its return value is usually stored in the variable of datatype DateTime.

**year()** function returns current year.

**month()** function returns current month.

**day()** function returns current day.

**dayOfTheWeek()** function returns current day of the week. This function is usually used as an index of a 2D character array that stores days information like the one defined in above program daysOfTheWeek

**hour()** function returns current hour.

**minute()** function returns current minute.

**second()** function returns current seconds.

**unixtime()** function returns unix time in seconds. Unix time is a system for describing a point in time. It is the number of seconds that have elapsed since 00:00:00(known as Coordinated Universal Time – Thursday, 1 January 1970).

**TimeSpan()** function is used to add/subtract time to/from current time. You can add/subtract days, hours, minutes & seconds. It’s also an overloaded function.

* now() + TimeSpan(seconds) returns the future time with seconds added into current time.
* now() - TimeSpan(days,hours, minutes, seconds) returns the past time.

## Arduino Code – Reading/Writing in 24C32 EEPROM

With DS1307 RTC module, as a bonus, you get 32 bytes of Electrically Erasable ROM. Its contents will not be erased even if main power to the device is interrupted.

The following program writes and then reads a message from the 24C32 EEPROM. You can use this program to save settings or passwords or really anything.

#include <Wire.h>

void setup()

{

char somedata[] = "lastminuteengineers.com"; // data to write

Wire.begin(); // initialise the connection

Serial.begin(9600);

Serial.println("Writing into memory...");

// write to EEPROM

i2c\_eeprom\_write\_page(0x57, 0, (byte \*)somedata, sizeof(somedata));

delay(100); //add a small delay

Serial.println("Memory written");

}

void loop()

{

Serial.print("Reading memory: ");

int addr=0; //first address

// access the first address from the memory

byte b = i2c\_eeprom\_read\_byte(0x57, 0);

while (b!=0)

{

Serial.print((char)b); //print content to serial port

addr++; //increase address

b = i2c\_eeprom\_read\_byte(0x57, addr); //access an address from the memory

}

Serial.println(" ");

delay(2000);

}

void i2c\_eeprom\_write\_byte( int deviceaddress, unsigned int eeaddress, byte data ) {

int rdata = data;

Wire.beginTransmission(deviceaddress);

Wire.write((int)(eeaddress >> 8)); // MSB

Wire.write((int)(eeaddress & 0xFF)); // LSB

Wire.write(rdata);

Wire.endTransmission();

}

// WARNING: address is a page address, 6-bit end will wrap around

// also, data can be maximum of about 30 bytes, because the Wire library has a buffer of 32 bytes

void i2c\_eeprom\_write\_page( int deviceaddress, unsigned int eeaddresspage, byte\* data, byte length ) {

Wire.beginTransmission(deviceaddress);

Wire.write((int)(eeaddresspage >> 8)); // MSB

Wire.write((int)(eeaddresspage & 0xFF)); // LSB

byte c;

for ( c = 0; c < length; c++)

Wire.write(data[c]);

Wire.endTransmission();

}

byte i2c\_eeprom\_read\_byte( int deviceaddress, unsigned int eeaddress ) {

byte rdata = 0xFF;

Wire.beginTransmission(deviceaddress);

Wire.write((int)(eeaddress >> 8)); // MSB

Wire.write((int)(eeaddress & 0xFF)); // LSB

Wire.endTransmission();

Wire.requestFrom(deviceaddress,1);

if (Wire.available()) rdata = Wire.read();

return rdata;

}

// maybe let's not read more than 30 or 32 bytes at a time!

void i2c\_eeprom\_read\_buffer( int deviceaddress, unsigned int eeaddress, byte \*buffer, int length ) {

Wire.beginTransmission(deviceaddress);

Wire.write((int)(eeaddress >> 8)); // MSB

Wire.write((int)(eeaddress & 0xFF)); // LSB

Wire.endTransmission();

Wire.requestFrom(deviceaddress,length);

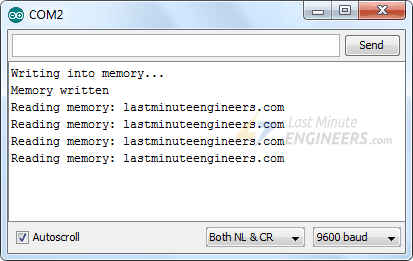
int c = 0;

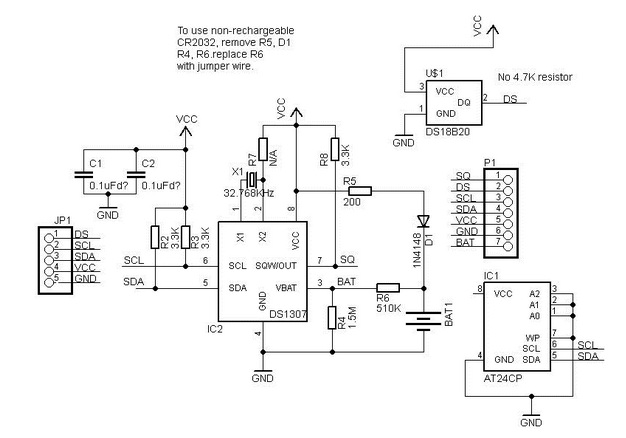
for ( c = 0; c < length; c++ )

if (Wire.available()) buffer[c] = Wire.read();

}

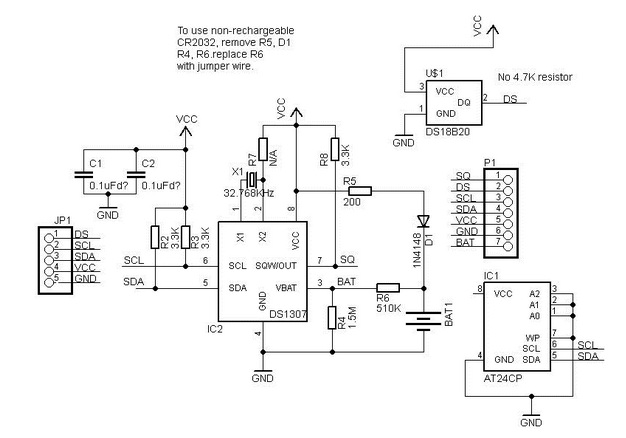
Here’s how the output looks like in the serial monitor.





## pcb[Tiny\_RTC](https://os.mbed.com/teams/Oxford-CWM-Team/wiki/Tiny_RTC)

A page collecting stuff relating to the Tiny RTC arduino module.

This board has two i2c devices on board.. a maximum DS1307 realtime clock and an amtel A24C32 32kb Eeprom. Some apparetnly alos come with a DS18B20 1 wire thermometer as well but most of the ebay variety don't, having three holes in the pcb to wire them to.. if you have one..

The Ds1307 runs at 100kbps whilst the A24C32 an do 400kbps.

Both are on the same bus lines with common pullup resistors on board of 3.3k.

The address for the DS1307 (RTC) is 0x1101000r whilst the A24C32 Eeprom is 0x1010xxxr where xxx are three bits set by address lines A2-A1. r is the read or write bit where r=0 to write to the slave (mbed is the master, peripherals are the slaves) and r=1 to read from it.

On the tiny RTC these address lines are all set to 0 so we have the EEprom address as 0x1010000r where r is the read / write bit determining the operation.

To use this device one needs to connect as follows (pin names for the KL25Z)

|  |  |  |
| --- | --- | --- |
| **Tiny RTC pin** | **mbed pin** | **function** |
| DS | n/c | data for 1 wire thermometer if present |
| SCL | I2C SCL | I2C clock |
| SDA | I2C SDA | I2C Data |
| VCC | 5V | must be 5V |
| GND | 0V | any gnd |