DS130X Library Common Reference

Overview

This page contains information relevant to both the DS1302 and DS1307 real-time clock (RTC) chips. All information specific to one chip will be pointed out.

View additional chip-specific guides: DS1302, DS1307

This guide uses my **DS130X** library. Download the library in Zip format

All example code is under **File › Examples › DS130X › (your RTC)**.

Printing Date and Time

After installing the library, go to the **DateAndTime › PrintDateTime** example.

DS1302 example code

```
/*
   PrintDateTime.ino
   Sketch to print the date and time strings from the RTC.
   Created August 28, 2020

   DS1302 connections:
   CLK to 2
   DAT to 3
   CE  to 4
*/

#include <DS130X.h>

DS1302 rtc(3, 2, 4); // DAT, CLK, CE

void setup() {
  // Set time and date
  rtc.setWriteProtect(false);
  rtc.setDate(28, 8, 20, FRIDAY); // Friday, 28 August 2020
  rtc.setTime(10, 5, 50);         // 10:05:50
  Serial.begin(9600); // Open serial port
}

void loop() {
  // Read day of week, date, and time strings
  String dow  = rtc.dowAbbr(); // Day of week (abbreviated)
  String date = rtc.dateStr(); // Read date
  String time = rtc.timeStr(); // Read time
```

```
  // Print the data in format:
  // dow, date time
  Serial.println(dow + ", " + date + " " + time);
  delay(1000); // Delay one second
}
```

DS1307 example code

```
/*
   PrintDateTime.ino
   Sketch to print the date and time strings from the RTC.
   Created August 28, 2020
*/

#include <DS130X.h>
#include <Wire.h>

DS1307 rtc;

void setup() {
  // Set time and date
  Wire.begin();
  rtc.setDate(28, 8, 20, FRIDAY); // Friday, 28 August 2020
  rtc.setTime(10, 5, 50);         // 10:05:50
  Serial.begin(9600); // Open serial port
}

void loop() {
  // Read day of week, date, and time strings
  String dow  = rtc.dowAbbr(); // Day of week (abbreviated)
  String date = rtc.dateStr(); // Read date
  String time = rtc.timeStr(); // Read time

  // Print the data in format:
  // dow, date time
  Serial.println(dow + ", " + date + " " + time);
  delay(1000); // Delay one second
}
```

Serial monitor output

```
Fri, 28/08/20 10:05:50
Fri, 28/08/20 10:05:51
Fri, 28/08/20 10:05:52
Fri, 28/08/20 10:05:53
Fri, 28/08/20 10:05:54
Fri, 28/08/20 10:05:55
Fri, 28/08/20 10:05:56
Fri, 28/08/20 10:05:57
Fri, 28/08/20 10:05:58
```

```
Fri, 28/08/20 10:05:59
```

Creating the RTC Object

The DS1302 constructor takes parameters in the order:

1. `dataPin`: Arduino pin that the RTC's DAT pin is connected to
2. `clockPin`: Arduino pin that the RTC's CLK pin is connected to
3. `cePin`: Arduino pin that the RTC's CE pin is connected to

The DS1307 constructor takes no parameters. **`Wire.begin` must be called to initialize the I2C bus before using the RTC.**

Writing Date and Time

DS1302 only: Before writing to the RTC's registers, we need call `setWriteProtect(false);` which sets the write protect bit. When write protect is enabled, the `write` and `writeRaw` methods won't do anything, but it is disabled, these methods will write data to the chip.

`setTime` takes parameters in the order:

1. `hour`: Hour to set (0-23)
2. `minute`: Minute to set (0-59)
3. `second`: Second to set (0-59)

`setDate` takes parameters in the order:

1. `day`: Day of month to set (1-31)
2. `month`: Month to set (1-12)
3. `year`: Year to set (0-99)
4. `dayOfWk`: Day of week to set (use the weekday constants: `SUNDAY`, `MONDAY`, etc.)

Reading Date and Time

These functions return parts of the current date/time in predetermined formats:

- `dowAbbr`: Day of week abbreviation
- `dateStr`: Date as a string (dd/mm/yy format)
- `timeStr`: Time as a string (hh:mm:ss)

At the end of each loop iteration, the program pauses for one second so it doesn't read too quickly.

Reading and Writing Registers

An example of reading registers is at **DateAndTime › ReadData**.

DS1302 example code

```
/*
   ReadData.ino
   Sketch to read unformatted time data from the RTC.
   Created August 28, 2020

   DS1302 connections:
   CLK to 2
   DAT to 3
   CE  to 4
*/

#include <DS130X.h>

DS1302 rtc(3, 2, 4); // DAT, CLK, CE

void setup() {
  // Set time and date
  rtc.setWriteProtect(false);
  rtc.setDate(28, 8, 20, FRIDAY); // Friday, 28 August 2020
  rtc.setTime(10, 5, 50);         // 10:05:50
  Serial.begin(9600); // Open serial port
}

void loop() {
  // Read time from RTC
  uint8_t hours   = rtc.read(HOURS);
  uint8_t minutes = rtc.read(MINUTES);
  uint8_t seconds = rtc.read(SECONDS);

  Serial.println("Hour: "   + String(hours));
  Serial.println("Minute: " + String(minutes));
  Serial.println("Second: " + String(seconds));

  Serial.println();

  // Read date from RTC
  uint8_t dayOfWeek  = rtc.read(DAY_OF_WEEK);
  uint8_t dayOfMonth = rtc.read(DAY_OF_MONTH);
  uint8_t month      = rtc.read(MONTHS);
  uint8_t year       = rtc.read(YEARS);

  Serial.println("Day of week: " + String(dayOfWeek));
  Serial.println("Day: "         + String(dayOfMonth));
  Serial.println("Month: "       + String(month));
```

```
  Serial.println("Year: "        + String(year));

  Serial.println("--------------------"); // Print a separator
  delay(1000); // 1 second pause
}
```

DS1307 example code

```
/*
   ReadData.ino
   Sketch to read unformatted time data from the RTC.
   Created August 28, 2020
*/

#include <DS130X.h>
#include <Wire.h>

DS1307 rtc;

void setup() {
  // Set time and date
  Wire.begin();
  rtc.setDate(28, 8, 20, FRIDAY); // Friday, 28 August 2020
  rtc.setTime(10, 5, 50);         // 10:05:50
  Serial.begin(9600); // Open serial port
}

void loop() {
  // Read time from RTC
  uint8_t hours   = rtc.read(HOURS);
  uint8_t minutes = rtc.read(MINUTES);
  uint8_t seconds = rtc.read(SECONDS);

  Serial.println("Hour: "   + String(hours));
  Serial.println("Minute: " + String(minutes));
  Serial.println("Second: " + String(seconds));

  Serial.println();

  // Read date from RTC
  uint8_t dayOfWeek  = rtc.read(DAY_OF_WEEK);
  uint8_t dayOfMonth = rtc.read(DAY_OF_MONTH);
  uint8_t month      = rtc.read(MONTHS);
  uint8_t year       = rtc.read(YEARS);

  Serial.println("Day of week: " + String(dayOfWeek));
  Serial.println("Day: "         + String(dayOfMonth));
  Serial.println("Month: "       + String(month));
  Serial.println("Year: "        + String(year));
```

```
  Serial.println("--------------------"); // Print a separator
  delay(1000); // 1 second pause
}
```

Serial monitor output

```
Hour: 10
Minute: 5
Second: 50

Day of week: 6
Day: 28
Month: 8
Year: 20
--------------------
Hour: 10
Minute: 5
Second: 51

Day of week: 6
Day: 28
Month: 8
Year: 20
--------------------
Hour: 10
Minute: 5
Second: 52

Day of week: 6
Day: 28
Month: 8
Year: 20
--------------------
```

Available Registers

The RTC classes have four methods to read and write to registers:

- `read`
- `readRaw`
- `write`
- `writeRaw`

All four methods will require you to specify a register to perform operations on. There are two ways to do this. The first way is to specify a register constant:

- `SECONDS` (range 0-59)

- `MINUTES` (0-59)
- `HOURS` (0-23)
- `DAY_OF_WEEK` (1-7)
- `DAY_OF_MONTH` (1-31)
- `MONTHS` (1-12)
- `YEARS` (0-99)
- `TCR_GET` (**DS1302 only**, read-only)
- `TCR_SET` (**DS1302 only**, write-only)
- `SQW_OUT` (**DS1307 only**)

You can also specify a register address. For example, with a DS1302, `write(0x80)` writes to the register 0x80 (the seconds register). (On the DS1307, this register is 0x00.)

DS1302 only:

The read registers are always one higher than the write registers. For example, to write to the minutes register, use `write(0x82, x);` to write and `read(0x83);` to read.

If you are using a register constant (like `MINUTES`), this difference is already handled, and you can use `write(MINUTES, x);` and `read(MINUTES);`

Register Operations

`read` and `readRaw` both take a single parameter: the register to read from.

`read` will convert the register data from BCD to base-10 and apply a bitmask, if necessary. `readRaw` will directly return the register's data without converting or masking.

`write` and `writeRaw` both take two parameters in the order:

1. `reg`: Register address to write to
2. `value`: Data to write to the register

`write` expects the value to be in base-10, then converts it to BCD to write. `writeRaw` expects the value to be in BCD, and directly writes to the register without converting.

When writing to the seconds register with `writeRaw`, you can explicitly set the halt bit, which is the MSB, in the seconds register).

Reading and Writing RAM

Register operations can be used for operating on the RTC's RAM. An example is at **Other › RAMReadWrite**.

DS1302 example code

```
/*
   RAMReadWrite.ino
   Sketch to demonstrate reading and writing the DS1307's RAM.
   Created August 30, 2020

   DS1302 connections:
   CLK to 2
   DAT to 3
   CE  to 4
*/

#include <DS130X.h>

DS1302 rtc(3, 2, 4); // DAT, CLK, CE

void setup() {
  rtc.setWriteProtect(false);
  Serial.begin(9600);

  // Write to all bytes of RAM
  Serial.println("Writing to RAM");
  int value = 0; // Value to hold in RAM
  for (int i = rtc.ramWStart(); i <= rtc.ramWEnd(); i += 2) {
    // Add 2 to counter to skip every other address (the read address)
    // ramWStart() returns the starting address for writing RAM.
    // ramWEnd() returns the ending address for writing RAM.
    rtc.write(i, value); // Write to RAM
    value++; // New value for next address
  }

  // Read from all bytes of RAM
  Serial.println("Reading from RAM");
  for (int i = rtc.ramRStart(); i <= rtc.ramREnd(); i += 2) {
    // Add 2 to counter to skip every other address (the write address)
    // ramRStart() returns the starting address for reading RAM.
    // ramREnd() returns the ending address for reading RAM.
    uint8_t ramValue = rtc.read(i); // Read from RAM
    // Print the value in RAM
    Serial.println("Address 0x" + String(i, HEX) + " has value " + String(ramValue));
  }
}

void loop() {} // No loop
```

DS1307 example code

```
/*
   RAMReadWrite.ino
```

```
   Sketch to demonstrate reading and writing the DS1307's RAM.
   Created August 29, 2020
*/

#include <DS130X.h>
#include <Wire.h>

DS1307 rtc;

void setup() {
  Wire.begin();
  Serial.begin(9600);

  // Write to all bytes of RAM
  Serial.println("Writing to RAM");
  int value = 0; // Value to hold in RAM
  for (int i = rtc.ramStart(); i <= rtc.ramEnd(); i++) {
    // ramStart() returns the starting address of RAM.
    // ramEnd() returns the ending address of RAM.
    rtc.write(i, value); // Write to RAM
    value++; // New value for next address
  }

  // Read from all bytes of RAM
  Serial.println("Reading from RAM");
  for (int i = rtc.ramStart(); i <= rtc.ramEnd(); i++) {
    uint8_t ramValue = rtc.read(i); // Read from RAM
    // Print the value in RAM
    Serial.println("Address 0x" + String(i, HEX) + " has value " + String(ramValue));
  }
}

void loop() {} // No loop
```

DS1302 serial monitor output

```
Writing to RAM
Reading from RAM
Address 0xc1 has value 0
Address 0xc3 has value 1
Address 0xc5 has value 2
Address 0xc7 has value 3
Address 0xc9 has value 4
Address 0xcb has value 5
Address 0xcd has value 6
Address 0xcf has value 7
Address 0xd1 has value 8
Address 0xd3 has value 9
Address 0xd5 has value 10
Address 0xd7 has value 11
```

```
Address 0xd9 has value 12
Address 0xdb has value 13
Address 0xdd has value 14
Address 0xdf has value 15
Address 0xe1 has value 16
Address 0xe3 has value 17
Address 0xe5 has value 18
Address 0xe7 has value 19
Address 0xe9 has value 20
Address 0xeb has value 21
Address 0xed has value 22
Address 0xef has value 23
Address 0xf1 has value 24
Address 0xf3 has value 25
Address 0xf5 has value 26
Address 0xf7 has value 27
Address 0xf9 has value 28
Address 0xfb has value 29
Address 0xfd has value 30
```

DS1307 serial monitor output

```
Writing to RAM
Reading from RAM
Address 0x8 has value 0
Address 0x9 has value 1
Address 0xa has value 2
Address 0xb has value 3
Address 0xc has value 4
Address 0xd has value 5
Address 0xe has value 6
Address 0xf has value 7
Address 0x10 has value 8
Address 0x11 has value 9
Address 0x12 has value 10
Address 0x13 has value 11
Address 0x14 has value 12
Address 0x15 has value 13
Address 0x16 has value 14
Address 0x17 has value 15
Address 0x18 has value 16
Address 0x19 has value 17
Address 0x1a has value 18
Address 0x1b has value 19
Address 0x1c has value 20
Address 0x1d has value 21
Address 0x1e has value 22
Address 0x1f has value 23
Address 0x20 has value 24
Address 0x21 has value 25
```

```
Address 0x22 has value 26
Address 0x23 has value 27
Address 0x24 has value 28
Address 0x25 has value 29
Address 0x26 has value 30
Address 0x27 has value 31
Address 0x28 has value 32
Address 0x29 has value 33
Address 0x2a has value 34
Address 0x2b has value 35
Address 0x2c has value 36
Address 0x2d has value 37
Address 0x2e has value 38
Address 0x2f has value 39
Address 0x30 has value 40
Address 0x31 has value 41
Address 0x32 has value 42
Address 0x33 has value 43
Address 0x34 has value 44
Address 0x35 has value 45
Address 0x36 has value 46
Address 0x37 has value 47
Address 0x38 has value 48
Address 0x39 has value 49
Address 0x3a has value 50
Address 0x3b has value 51
Address 0x3c has value 52
Address 0x3d has value 53
Address 0x3e has value 54
Address 0x3f has value 55
```

Addresses and Bounds

Using the below functions as the lower and upper bounds of a for loop, we can iterate through each address.

To operate on a single byte, use `read` or `write` and set the register value to the desired address.

DS1302 Specifics

The DS1302 class has four methods to get the lowest and highest RAM addresses, split into read and write:

- `ramWStart, ramWEnd` for write start and write end, respectively
- `ramRStart, ramREnd` for read start and read end, respectively

In each loop, the increment statement is `i += 2`. Since the addresses alternate between read and write, we have to skip every other one.

RAM byte	Address	Function
0	0xC0	Write
	0xC1	Read
1	0xC2	Write
	0xC3	Read
2	0xC4	Write
	0xC5	Read

DS1307 Specifics

The DS1307 class has two methods to get the lowest and highest RAM addresses:

- `ramStart` for the start
- `ramEnd` for the end

All addresses can be written to and read from.

Additional Functions

Halt Bit

When the RTC is halted (determined by the halt bit), its oscillator is disabled and goes into low-power mode. This means that it won't keep track of time until you set it back to normal mode.

The RTC's `setHalt` method writes to this halt bit. It takes a Boolean parameter: `true` to enable the halt bit, `false` to disable.

DS1307 only: The SQW output won't work when the RTC is halted.