Introduction

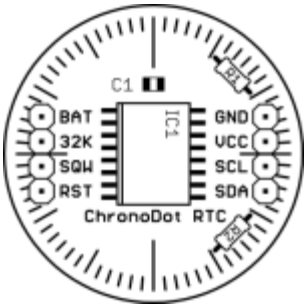
The ChronoDot RTC is an extremely accurate real time clock module, based on the DS3231 temperature compensated RTC (TCXO). It includes a CR2016 battery which should last at least 8 years if the I2C interface is only used while the device has 5V power available. No external crystal or tuning capacitors are required.

The DS3231 has an internal crystal and a switched bank of tuning capacitors. The temperature of the crystal is continously monitored, and the capacitors are adjusted to maintain a stable frequency. Other RTC solutions may drift minutes per month, especially in extreme temperature ranges...the ChronoDot will drift less than a minute per year. This makes the ChronoDot very well suited for time critical applications that cannot be regularly synchronized to an external clock. Here is a live demo of a DS3231 chip that has been keeping time since it was last synchronized in 2005: DS3231 Web Demo

The ChronoDot will plug into a standard solderless breadboard and also has mounting holes for chassis installation.

The I2C interface is very straightforward and virtually identical to the register addresses of the popular DS1337 and DS1307 RTCs, which means that existing code for the Arduino, Basic Stamp, Cubloc, and other controllers should work with no modification.

Features



Controller:	Maxim DS3231
Function:	Temperature-compensated RTC
Accuracy:	± 2ppm at 0C to 40C (~1 minute per year)
Power Supply:	2.3 to 5.5 V
Current:	200uA (active), 840nA (timekeep)
PCB Size:	1.2 inches diameter
Pin Spacing:	0.1 inches

Assembly

The ChronoDot is typically shipped without the lithium CR2016 battery soldered in place. To allow timekeeping in the absence of external power, you will need to attach the battery. The two solder pads on the PCB are marked + and -. The battery provided with the ChronoDot has a metal tab that runs from the top (positive, +) of the battery and down to the same level as the bottom (negative, -) side of the battery. You will need to solder the tab leading from the top of the battery to the + pad on the ChronoDot. Please refer to the bottom-view image above if necessary.

Power Connections

The **VCC** and **GND** pins are used to power the ChronoDot when it is connected to a microcontroller. The ChronoDot *can* be used just with the internal battery power, but it could quickly run the battery down and require replacement. **VCC** can be anywhere between 2.3 and 5.5 volts, making it possible to use the ChronoDot with 3.3V and 5V systems.

Battery

The battery is a 3V lithium CR2016 with solder tabs, typically provided with the ChronoDot but not soldered in place. You will need to solder the battery before using it. Since the battery should supply the ChronoDot with enough power to run for 8 years, future versions of the ChronoDot may be shipped with the battery pre-soldered. The battery voltage is also connected directly to the **BAT** pin. This could be used to attach an external battery instead of the CR2016, or to monitor the health of the battery. Make sure not to short this pin accidentally, or the battery could be drained quickly.

SDA/SCL

The **SDA** and **SCL** pins are used to communicate with the ChronoDot, using the I2C standard interface. The I2C bus requires pullup resistors from **SDA** and **SCL** to **VCC**. Since many devices that have I2C buses already have the resistors in place, these are not provided with the ChronoDot. But there are two locations for 4.7K or 10K resistors to be soldered, in case your controlling device does not have its own pullup resistors.

RST

The **RST** pin is an active-low signal that can be used in some microcontroller applications. In most cases you will not need to use it. If the **VCC** supply is less than ~2.5V, the **RST** line will activate. Additionally, the **RST** line will watch for the signal to be pulled low by an external signal. If that happens, it will hold the **RST** line low for 250ms beyond the point the external signal release the line. This can be useful for resetting a microcontroller in the event of low power, or an external condition like a reset pushbutton. The **RST** line does not affect timekeeping functions.

SQW

The **SQW** pin can be configured to output a square wave signal at 1.000 Hz, 1024 Hz, 4096 Hz, or 8192 Hz. This is controlled by the RS2 and RS1 bits in the control register (address 0x0E). The pin can also be used as an alarm trigger. If the INTCN bit in the control register is set, then the output will go low when the current time matches the time set in either of the two alarm registers. This pin needs to be pulled up to **VCC** with a resistor if it is used, 10K would be a good choice.

32K

The **32K** pin is open-drain (needs a pullup resistor) and outputs a 32768 Hz clock signal. This can be enabled using the EN32kHz bit in the Status register (address 0x0F). This pin could be used as an accurate oscillator reference for some external device.

Datasheet

The DS3231 datasheet provides all other necessary information such as registers, maximum voltages, power consumption, and special features.

Examples

Very simple Arduino code to read and print the hours, minutes, and seconds from the ChronoDot:

```
#include <Wire.h>

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

  // clear /EOSC bit
  // Sometimes necessary to ensure that the clock
  // keeps running on just battery power. Once set,
  // it shouldn't need to be reset but it's a good
  // idea to make sure.
  Wire.beginTransmission(0x68); // address DS3231
  Wire.send(0x0E); // select register
  Wire.send(0b00011100); // write register bitmap, bit 7 is /EOSC
  Wire.endTransmission();
}

void loop()
{
  // send request to receive data starting at register 0
  Wire.beginTransmission(0x68); // 0x68 is DS3231 device address
  Wire.send(0); // start at register 0
  Wire.endTransmission();
  Wire.requestFrom(0x68, 3); // request three bytes (seconds, minutes, hours)

  while(Wire.available())
  {
    int seconds = Wire.receive(); // get seconds
    int minutes = Wire.receive(); // get minutes
    int hours = Wire.receive();   // get hours

    seconds = (((seconds & 0b11110000)>>4)*10 + (seconds & 0b00001111)); // convert BCD to
decimal
    minutes = (((minutes & 0b11110000)>>4)*10 + (minutes & 0b00001111)); // convert BCD to
decimal
    hours = (((hours & 0b00110000)>>4)*10 + (hours & 0b00001111)); // convert BCD to decimal
(assume 24 hour mode)

    Serial.print(hours); Serial.print(":"); Serial.print(minutes); Serial.print(":");
Serial.println(seconds);
  }

  delay(1000);
}
```