Tiny Time 2 Watch

11th April 2017

This is a minimalist ATtiny85-based watch using 12 LEDs, arranged like a clock face, to show the time analogue-style. This new version of the watch uses a separate crystal-controlled low-power RTC chip to keep time to within a few seconds a month, and the battery will last for several years:

The Tiny Time 2 watch based on an ATtiny85 and a DS2417 RTC; it's ten to two.

To show the time you press the button on the watch face, and it lights up LEDs to show the time like the hour and minute hands on a clock. The LEDs flash to distinguish the hours and minutes, and show intermediate times.

Introduction

My original Tiny Time Watch had three serious shortcomings: First, because it used the ATtiny85's internal oscillator for the timing, it was only accurate to within a few minutes every 24 hours. Secondly, because the ATtiny85 had to run all the time to do the timing, the current consumption was significant, and a CR2032 battery would only last a month or so. Thirdly, it only showed the time to the nearest five minutes.

While puzzling about how to improve the watch I discovered Maxim Integrated's DS2417 RTC chip, a small 6-pin package which uses a 32.768 kHz crystal to keep accurate time [1]. It can communicate with the main ATtiny85 processor via a 1-wire interface, which uses just one I/O pin to transmit and receive data. Because the RTC chip is doing the timing the ATtiny85 can stay asleep in low-power mode when it's not actually displaying the time, dramatically reducing the power consumption.

I solved the third problem by flashing two LEDs to show the minutes. As before, to show the time you press the button on the watch face, and the time is then displayed for five seconds. It lights one LED to show the hour, and flashes one or two other LEDs to show the minutes to the nearest minute by altering the rate at which each of the LEDs flash. Here are some examples:

Search

Recent posts

▼ 2017
Four-Channel Thermometer
Flexible GPS Parser
Tiny Face Watch
Driving Four RGB LEDs from an ATtiny85
Big Text for Little Display
ATtiny85 Graphics Display
Tiny Time 2 Watch
10 or 12-bit DAC from the ATtiny85
Simple 1-Wire Interface
Audio Pitch Shifter
Tiny Lisp Computer 2 PCB
GameBone Simple Electronic Game
► 2016
► 2015
► 2014

Topics

► Games
► Sound & Music
► Clocks
► GPS
► Tools
► Tutorials

By processor

► ATtiny85
► ATtiny84
► ATtiny841
► ATtiny2313
► ATtiny861
► ATmega328
► ATmega1284

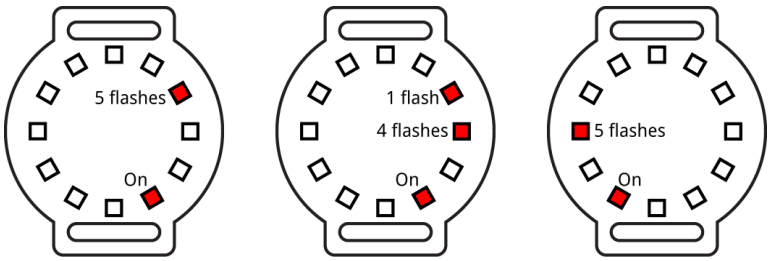
About me

About me
Contact me

Follow @technoblogy

Feeds

RSS feed



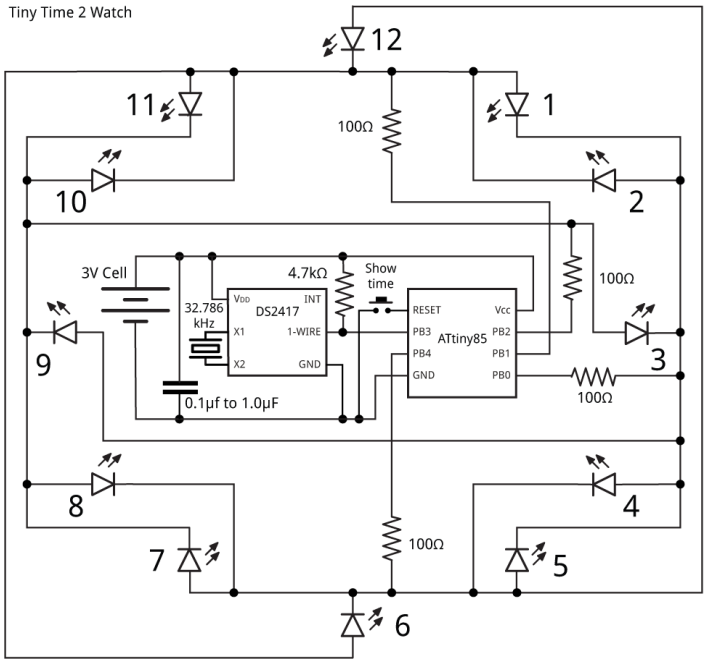
Examples of how the Time Time 2 watch displays the time.

For consistency with analogue watches, the Tiny Time Watch displays times later than half past the hour by lighting the next hour number. If only one LED lights up you know that both hands are pointing to the same hour mark.

The total power consumption with no display is now just 1µA, giving an estimated battery life of over 10 years from a single CR2016 battery!

The circuit

Here's the circuit of the Tiny Time 2 watch, laid out like the circuit board:



Circuit of the ATtiny85-based Time Time 2 watch.

The crystal is a standard 32.768 kHz quartz watch crystal; Maxell specify a 6pf load capacitance, and I chose a MS1V-T1K micro crystal which had the advantage that its case could be soldered to the board [2], but I expect any watch crystal would be suitable.

The LEDs are 0805 size, and for this version of the watch I chose orange LEDs to match Ragworm's nice circuit board. I got mine from Bright Components in the UK, who sell them for under £1 for ten [3].

I arranged the LEDs to simplify the PCB layout which explains the rather haphazard order, but this is easily accommodated in the software. The following table shows which LED lights up when you take one I/O line high and the other I/O line low:

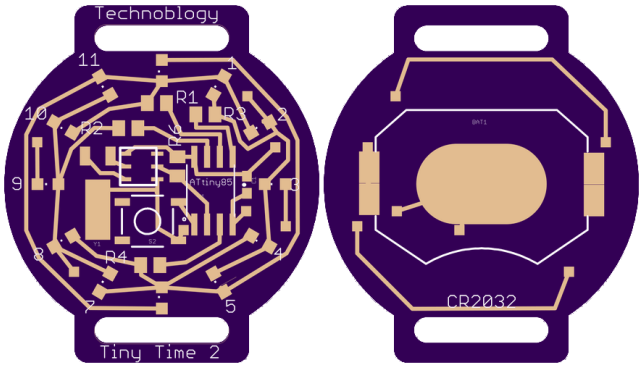
Cathodes \ Anodes	PB0	PB1	PB2	PB4
PB0		1	3	5
PB1	2		10	12
PB2	9	11		7
PB4	4	6	8	

The button is a miniature SMD push button available from Sparkfun [4], available from Proto-PIC in the UK [5].

The battery is a 20mm coin cell. Given the low current consumption I decided to use the slimmer CR2016 cell, and found a suitable SMD battery holder on Mouser [6]. Alternatively you could use a CR2032 battery, with an SMD 20mm coin cell holder available from Sparkfun [7], or from Proto-PIC in the UK [8].

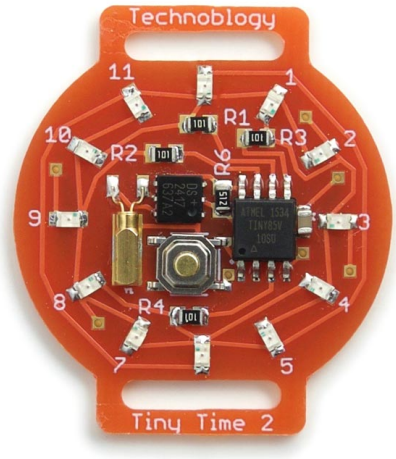
Construction

I designed the board in Eagle and sent it to Ragworm in the UK for fabrication [9]. Here's the layout (from the OSH Park preview):



There are only two tracks on the reverse of the board, so it should be straightforward to use a single-sided board with wire links replacing these tracks. There's a link to the Eagle files at the end of the article if you want to make yourself a board.

I built the watch on a small printed circuit board, using SMD components, with all the components apart from the battery holder soldered to one side of the board. I used an SOIC ATtiny85 and 0805 resistors and LEDs, so they should be relatively easy to solder by hand:



The completed Tiny Time 2 watch circuit board.

The DS2417 RTC chip is in a TSOC package, and is probably the trickiest item to solder as its legs are tucked under the package. The LEDs should be soldered with the same orientation, with the negative sides facing the centre of the board.

I used a Youyue 858D+ hot air gun at 250°C to solder the SMD components onto the front of the board, and then finally soldered the battery holder onto the back of the board using a conventional soldering iron. If you

don't have a hot air gun you should be able to solder the SMD components with a bit of care using a fine-tipped soldering iron.

The program

This section explains the various sections of the Tiny Time Watch program.

1-Wire interface

To communicate with the RTC I used the 1-Wire interface from my earlier article Simple 1-Wire Interface. It reads the five bytes of data from the RTC into an array **DataBytes[5]**. The first byte is a configuration byte, and the last four bytes give the number of seconds as a long integer; to make it easier I defined a union, so the time bytes can be accessed as **rtc.seconds**:

```
static union {
  uint8_t DataBytes[5];
  struct {
    uint8_t control;
    long seconds;
  } rtc;
};
```

The display

The 12 LEDs are driven by the four I/O lines PB0, PB1, PB2, and PB4. PB3 is used for the 1-Wire interface, because using it on PB0-PB2 interfered with programming the ATtiny85.

The array Pin[5][5] specifies how the LEDs are connected to the five I/O lines:

```
int Pins[5][5] = {{ -1,  1,  3, -1,  5 },
                  {  2, -1, 10, -1,  0 },
                  {  9, 11, -1, -1,  7 },
                  { -1, -1, -1, -1, -1 },
                  {  4,  6,  8, -1, -1 } };
```

The first row of the array specifies which LEDs have their cathodes connected to PB0: the LED at 1 o'clock has its anode connected to PB1, the LED at 3 o'clock has its anode connected to PB2, and the LED at 5 o'clock has its anode connected to PB5. The values in the array corresponding to PB3 are set to -1.

I used Timer/Counter0 running at 250 Hz to multiplex the display. This is set up by the routine **DisplaySetup()** as follows:

```
void DisplaySetup () {
  // Set up Timer/Counter0 to multiplex the display
  TCCR0A = 2<<WGM00;            // CTC mode; count up to OCR0A
  TCCR0B = 0<<WGM02 | 4<<CS00;  // Divide by 256
  OCR0A = 124;                  // Divide by 125 -> 250Hz
}
```

The interrupt service routine simply calls **DisplayNextRow()** and decrements the global variable **Timeout**:

```
ISR(TIM0_COMPA_vect) {
  DisplayNextRow();
  Timeout--;
}
```

Two additional routines are used to turn the display on and off:

```
void DisplayOn () {
  TIMSK = 1<<OCIE0A;    // Enable compare match interrupt
}
```

```
void DisplayOff () {
  TIMSK = 0;            // Disable compare match interrupt
  DDRB = 0;             // Blank display - all inputs
  PORTB = 0x17;         // All pullups on except PB3
}
```

When the display is turned off the compare match interrupt is off, to avoid interfering with the 1-Wire interface.

Display multiplexing

The routine **DisplayNextRow()** works as follows:

```
void DisplayNextRow() {
  Cycle++;
  byte row = Cycle & 0x03;
  if (row == 3) row = 4;    // Skip PB3
  byte bits = 0;
  for (int i=0; i<5; i++) {
    if (Hours == Pins[row][i]) bits = bits | 1<<i;
    if (Fivemins == Pins[row][i]) bits = bits | 1<<i;
  }
  DDRB = 1<<row | bits;
  PORTB = bits;
}
```

Up to two LEDs can be lit at once. The LED specified by the variable **Hours** is used to show the hours, and is displayed continuously. The LED specified by the variable **Fivemins** is used for the minutes.

The bottom two bits of the variable **Cycle** determine which row is being displayed. For a given row the array **Pins[row][i]** is checked to see if any of the LEDs in that row need to be displayed. If so, the appropriate bits are set in the variable **bits**. This is then written to the port.

Setting the time

When you first apply power to the watch it checks the MCUSR register to detect a power-on reset, and runs **SetTime()** which allows you to set the time to the nearest second. It works as follows: wait until the current time is an exact multiple of five minutes, and insert the battery. The watch will then start from 12:00, stepping through the display five minutes at a time. When the watch shows the time you inserted the battery, press the reset button. The watch will account for the additional time it took you to set the time, and then go to sleep. It's now ready for use.

The **SetTime()** routine keeps incrementing the variable **secs** by 300, equivalent to five minutes, displaying the time for a second between steps. At each step it writes the value of secs to the RTC, with an offset added to account for the elapsed time since starting the procedure.

```
void SetTime () {
  unsigned long secs = 0;
  for (;;) {
    Fivemins = (unsigned long)(secs/300)%12;
    Hours = (unsigned long)((secs+1799)/3600)%12;
    // Write time to RTC
    rtc.control = 0x0C;
    rtc.seconds = secs + (Offset/Tickspersec);
    OneWireReset();
    OneWireWrite(SkipROM);
    OneWireWrite(WriteClock);
    OneWireWriteBytes(5);
    DisplayOn();
    Delay(Tickspersec);
    DisplayOff();
    secs = secs + 300;
```

```
  }
}
```

Displaying the time

The ATtiny85 is normally in sleep mode, which draws negligible current. The push button is connected to the reset input, which wakes the processor and generates a reset.

The main routine **loop()** first reads the time from the RTC into the variable secs:

```
  OneWireReset();
  OneWireWrite(SkipROM);
  OneWireWrite(ReadClock);
  OneWireReadBytes(5);
  OneWireReset();
  secs = rtc.seconds;
```

It then calculates the values of the variables **Hours** and **Mins**:

```
  Hours = (unsigned long)((secs+1859)/3600)%12;
  Fivemins = 12;
  int Mins = (unsigned long)(secs/60)%60;
  int From = Mins/5;
  int Count = Mins%5;
  DisplayOn();
  for (int i=0; i<5-Count; i++) {
    Fivemins = From; Delay(Tickspersec/5);
    Fivemins = 12; Delay(Tickspersec/5);
  }
  for (int i=0; i<Count; i++) {
    Fivemins = (1+From)%12; Delay(Tickspersec/5);
    Fivemins = 12; Delay(Tickspersec/5);
  }
  DisplayOff();
```

The first **for** loop flashes the previous five-minute mark and the second **for** loop flashes the next five-minute mark, according to where the time is between the two marks, for a total of five flashes.

As explained above, the Tiny Time Watch displays times later than half past the hour by lighting the next hour number; this is implemented with the correction +1859 in the calculation of **Hours**.

Compiling the program

I compiled the program using Spence Konde's excellent new ATTiny Core, which now supports all the ATtiny processors and supercedes the various earlier ATtiny cores [10]. Select the **ATtinyx5 series** option under the **ATtiny Universal** heading on the **Boards** menu. Then choose **Timer 1 Clock: CPU**, **B.O.D. Disabled**, **ATtiny85**, **8 MHz (internal)** from the subsequent menus. Choose **Burn Bootloader** to set the fuses appropriately.

I uploaded the program using a clip that fitted to the top of the SMD ATtiny85 [11], using Sparkfun's Tiny AVR Programmer Board; see ATtiny-Based Beginner's Kit. Because the timing is performed by the separate RTC chip there's no need to calibrate the ATtiny85's internal oscillator with this version of the watch.

Here's the whole Tiny Time 2 Watch program: Tiny Time 2 Watch Program.

Alternatively, get it on GitHub here together with the Eagle files for the PCB: Tiny Time 2 Watch on GitHub.

Or order a board from OSH Park here: Tiny Time 2 Watch Board.

---

1. ^ DS2417 1-Wire Time Chip with Interrupt on Maxim Integrated.
2. ^ MS1V-T1K 32.768kHz Micro Crystal on Farnell.
3. ^ 10x White 0805 Surface Mount (SMD/SMT) LED on Bright Components.
4. ^ Mini Push Button Switch - SMD on SparkFun.
5. ^ Mini Push Button Switch (SMD) on Proto-PIC.

6. ^ BAT-HLD-002-SMT Linx Technologies Battery Holder on Mouser.
7. ^ Coin Cell Battery Holder - 20mm (SMD) on SparkFun.
8. ^ Coin Cell Battery Holder - 20mm (SMD) on Proto-PIC.
9. ^ Ragworm PC prototyping service.
10. ^ ATTinyCore on GitHub.
11. ^ IC test Clip - SOIC 8-pin on SparkFun.

Bruno Guillemet • 5 months ago

Very good job !

2 • Reply • Share ›

Stephane Come • 4 months ago

David, this is one of the best project using the ATTiny series - and it actually works on a tiny 25 based on the compile size. I've just made one and my kids are going nuts over it. However, it keeps showing the same time and not incrementing. Any ideas? I've double-checked everything on a hardware level and the code compiles. Have you ran across this before? I'll need to take my scope out and check the 1 wire line if not. Thank you for sharing this. Rad!!

• Reply • Share ›

johnsondavies Mod ➚ Stephane Come • 4 months ago

Does it step through the LEDs when you first apply power? That should work even if the RTC isn't working. Have you programmed the fuses for an 8MHz internal clock? Finally, you could turn on the RTC's INT signal and check it gives a 1Hz signal.

• Reply • Share ›

Stephane Come ➚ johnsondavies • 4 months ago

David, thanks for the quick reply. Yes, on the steps when I power it up. Yes, on the 8MHz internal config. I will check the RTC chip and the signal you mentioned. This one is new to me. Sounds like some fun digging ahead. I'll keep you posted.

• Reply • Share ›

Georgi Gurari • 4 months ago

Very good job man! I love it, and made myself some watches with different leds. Keep the good work!

• Reply • Share ›

johnsondavies Mod ➚ Georgi Gurari • 4 months ago

Great! Thanks for the feedback.

• Reply • Share ›

Sergey • 4 months ago

Well!!!! And there is a video of work hours

• Reply • Share ›

johnsondavies Mod ➚ Sergey • 4 months ago

Sorry - I don't quite understand ...

• Reply • Share ›

David Connolly • 5 months ago

The evolution of this project amazes me. So well done.
Of course I will build this version as well.

Reply • Share ›

Hagen von Merak • 5 months ago

I want to see in video! seems nice.

Reply • Share ›

johnsondavies Mod ➚ Hagen von Merak • 5 months ago

OK, I'll try and make one!

Reply • Share ›

Jeremy Cook • 5 months ago

Very cool!

Reply • Share ›

johnsondavies Mod ➚ Jeremy Cook • 5 months ago

Thanks for the comment!

Reply • Share ›

Subscribe Add Disqus to your siteAdd DisqusAdd Privacy DISQUS