ATTINY2313-BASED SIMPLE TIMER/COUNTER

KTimer is a simple ATtiny2313-based battery-powered timer/counter which beeps after a specified period much like microwave oven ones. It can control an external device and can also operate as a chronometer or hand counter.

Hardware Features

- Handheld size, 8.5 x 4.5 cm
- Powered from 3 AA alkaline batteries
- Battery life: ~4 years in standby mode, >80 hours in active mode
- 4-digit 7-segment display in multiplexed mode for maximum brightness
- 3 key keypad
- Buzzer with frequency synthetizer
- Connector with signals for external power stage allowing control of other devices
- Connector with 5V-level serial port for communications
- Crystal-less circuit
- Low cost, less than US$ 15 in materials

Firmware Features

- Countdown timer beeps for 30 seconds when timeout expires
- Extremely simple operation, just like microwave oven timers
    - Two buttons set up the time, incrementing or decrementing in one minute steps
    - Another button starts/stops the countdown
- 99 minute maximum countdown period
- Upcounting chronometer
- Hand counter, upcounts on keypress
- Auto-shutdown after 20 seconds of inactivity
- AVR910-compatible bootloader allows in-system firmware upgrade over serial port
- Instrumentation countes record number of reboots, wakeups, total time counted
- Serial console allows inspection of the instrumentation counters and escaping to the bootloader

Hardware Design

The design requisites were:

- The usual 'use as few components as possible'
- Make it as low power as possible, so as to make it battery powered

Notice there's no voltage regulator: the device is powered directly from the batteries. This eliminates quiescent currents, which can be quite substantial. We took advantage of the

wide voltage range (2.7-5.5V) supported by the ATtiny2313. Combined, these things increase battery life.

There's no crystal either, so it's not really meant to be high precision. About 2% acurracy may be achieved tuning the top value of Timer/Counter1, defined in the code. The software provides a calibration procedure that improves precision to about 0.5%.

The four numeric LED displays are multiplexed in the classic fashion with driver transistors in the common anode to overcome the AVR's limited current source capability. I considered using cleverer tricks such as alternating common-anode and common-cathode displays (would have saved two pins) or even Charlieplexing (would've saved three pins *and* four transistors), but rejected it because I supected I might have brightness problems.

Using Super Red displays (a LSD056BSR-10), I managed to get it bright enough to be read in daylight using just 30mA. Another, older orange-red display (Para-Light A-552E) took 72mA and didn't get anywhere as bright. The blue version uses Para-Light A-551UB displays, which are very bright -- you can almost use the timer as a flashlight. When using them, be adivised it's quite common for their brightness to vary significantly between units. To get uniform brightness, try to choose them from the same manufacturing lot. Finally, the smallish green version uses a LN543GA display module I found in the surplus market.

If you want less brightness and more battery life, use 330 or 470 Ohms for R1-R8 instead of the 220 shown in the circuit. On the other hand, if you want more brightness and less battery life, decrease them to 150 or 100 Ohms.

In the pictures you can see I used the classical trick of mounting the last two displays upside down so the decimal point of the third digit, along with the one of the second digit, emulates a colon.

To save a few pins, the keypad switches share the same lines as the display segments. A few resistors make the brightness drop imperceptible when you press some key while the display is lit.

The other side of the keypad is connected via a resistor and diode in parallel to the MCU's interrupt 0 line. This is used to wake it up from power down mode. This resistor is tricky -- in some instances, it needed to be a high value (the schematics say it's 100k), but it seems that with AVRs from newer lots it has to be quite low; I've been using 3k3 lately.

There are two versions of the board: one with a connector for 5V level serial and and the other with a two-transistor circuit to serve as level shifter, allowing direct connection to the serial port, although limited to half-duplex. None has an ISP (In-System Programming) connector: it is assumed you can do the initial firmware upload in a breadboard or you have a development board/starter kit such as Kanda's STK-200 or Atmel's STK-500.

The 'Countdown Timer' Firmware

Firmware Upload

If you use avrdude and usbasp like I do, check that the paths in the `Makefile` are OK for your system and type:

```
make fuses
```

```
make upload_usbasp_full
```

This will upload the `full.hex` image, which holds both the application code and the bootloader.

You can also upload only the serial bootloader with:

```
make fuses
```

```
make upload_boot
```

Then hook up the serial port and upload the application itself with

```
make upload_serial
```

If you use some other programmer, adapt the `Makefile` accordingly.

Building

Get yourself a recent avr-gcc (I used 3.4.4) with avr-libc 1.4.0 or higher. If you're under Windows, I guess recent WinAVR versions are fine, although GCC 4.x generates larger code under certain situations.

- **Note:** If you're using avr-gcc 4.0 or later, please add the clause "externally-visible" to the `__attribute__((...))` extension in the `SIGNAL` handler. Without this it will not work.

If you want to build the bootloader, you'll also need AvrAssembler2 from Atmel's AVRStudio. I've sucessfully compiled it with avra as well.

Change the paths in the `Makefile` so they make sense in your system, then type `make clean` to get rid of the precompiled versions or any other cruft, then `make`. You should get the files`ktimer.hex`, `eeprom.hex` and `avrboot2313.hex` as a result.

Operation

The first two buttons are called - and + and are used to ajust the initial value. The third is called ENTER and it starts/pauses the timer. When the timer is running, any key pauses the count. Pressing ENTER again resumes the count.

The last ten seconds of the countdown are marked by audible ticks (it does kinda look like a bomb is about to set off). When it reaches zero, it beeps and flashes for 30 seconds. PressingENTER stops the beeping.

Pressing + and - simultaneously stops the count and zeroes the display. This works in all modes, including hand counter mode as well.

If you start the timer on 00:00, it will enter 'chronograph mode', upcounting instead of downcounting. There is no alarm in this mode. If you pause then resume the count, it will continue to count up, even if you use the + or - buttons to change the counter value. To get back to countdown mode, use the counter reset command (press + and - at the same time) then adjust the initial value.

Auto-shutdown (sleep mode, actually) kicks in after 20 seconds of inactivity.

If you press all three keys simultaneously, the refresh rate will be halved. This is useful to demonstrate the 'multiplexing' principle to people. It is reset to the default 50Hz when the device wakes up.

The state of the countdown is reflected in the PD6 pin, available in the XTRNL_DEVICE connector. You can connect a power stage circuit on it to have the timer control another device.

The device operates in two modes: simple and advanced, described in further detail below. The distinction was introduced because some users got confused by inadvertently entering hand counter mode when all they actually wanted to do was to reset the counter to zero.

Simple Mode

This is the mode the device starts in when first turned on. In this mode there are a few restrictions:

- The - and + keys operate on the minutes counter only (the first two digits), so the time adjustment resolution is limited to one minute.
- When you turn on the device (wake it up, actually), the counter value is auto-reset to zero.
- Hand counter mode is disabled; the commands to enter and quit hand counter mode simply zero the timer.

Advanced Mode

To enter Advanced Mode, keep - and ENTER simultaneously pressed for about 15 seconds until the device starts to beep. To exit Advanced Mode and get back to Simple Mode, simultaneoulsy hold + and ENTER down for about 15 seconds until you hear the beeps.

This mode presents more functionality:

- The timer adjustment buttons have one-second resolution: they start subtracting/adding one second at a time to the counter value, then 10 seconds, then 1 minute, if you keep one of them pressed long enough.
- There is no auto-reset then the device is woken up; it shows the exact value it had before going to sleep. You may need to manually zero the counter by pressing - and +simultaneously if you don't want to continue the previous count.
- Briefly pressing - and ENTER simultaneously sends us to hand counter mode, described below. To go back to timer/chrono mode, briefly press + and ENTER simultaneously.

Hand Counter Mode

This mode is useful for counting things, like inventory items, so that we don't get lost in a possibly long count.

To make it visually obvious that we are in hand counter mode, the colon disappears and the leading zeroes are supressed.

To count up, press and release the ENTER key. You may keep the key pressed indefinitely; the count will actually happen when you release the ENTER key. There is no auto-repeat.

To count down (perhaps to correct a mistake), press the - key. As before, there is no auto-repeat and the count happens when you release the key.

In this mode, the + key doesn't do anything. It is useful to wake up the device without changing the count.

Source : http://www.postcogito.org/Kiko/SimpleTimerCounter2313.html