To-mate-Oh

1.1

Generated by Doxygen 1.8.6

Fri Jan 9 2015 15:24:54

Contents

1 To-mate-Oh - A small Pomodoro timer with a capacitive button

Figure 1: Home made To-mate-Oh (not an actual tomato)

The following files comprise To-mate-Oh: ./docu/docu.pdf, ./docu/doxygen.cfg, ./docu/doxygen.sh, ./firmware/-Makefile, ./firmware/To-mate-Oh.c, ./firmware/To-mate-Oh.h, ./hardware/To-mate-Oh.brd, ./hardware/To-mate-Oh.sch, ./hardware/To-mate-Oh-board.png, ./hardware/To-mate-Oh-etch.pdf, ./hardware/To-mate-Oh-schematic.pdf, ./hardware/To-mate-Oh-prototype.jpg, ./hardware/To-mate-Oh-final.jpg, ./hardware/To-mate-Oh-kit.jpg, LICENSE.txt, README,txt

Their use is permitted under the terms of GNU General Public License as well. If you want to use any part under different terms, please feel free to ask the author for his permission.

1.1 Introduction

I saw the POMODORO project by Robin Schreibler and decided to build it. Some may know Robin by the name FakuFaky. His' is a very nice layout with beautifully clean code.

That can't stand, I thought, so first thing I did was soil the code and complicate the layout. And here is why: I was measuring the power consumption of the original POMODORO. With a fresh coin cell at 3.3 V, the POMODORO drained 1.6mA, even when in power down. Your average coin cell has about 200 mAh to give, ignoring the voltage decay. That means after 5 days at most, the battery is gone.

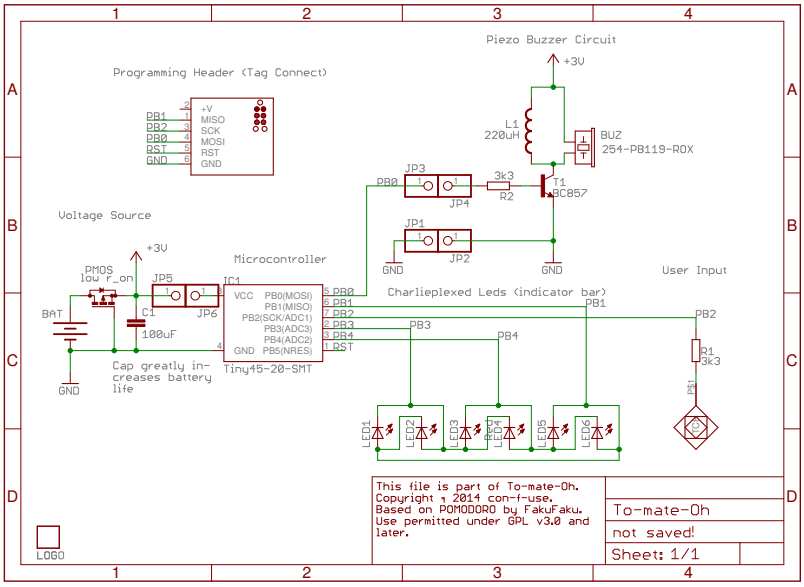

Figure 2: New To-mate-Oh schematic. It has reverse polarity protection, four less resistors and replaced the button by a capacitive touch pad.

I thought I can do better, and that kind of exploded in my face. Now the board is a quarter of Robin's footprint, has proper reverse polarity protection, drains only 7 uA when inactive and uses less components. Most notably I could do away with four resistors and the switch. The design uses capacitive sensing now. The code still works with the old hardware though. On the down side, the code is much uglier (but uses only a third of the space on the micro controller). I think it's an improvement, but judge for yourself.

Anyways, many thanks to Robin for making his project open source and being awesome!

1.2 Making the Board

You can order the circuit board at a board house of your choice. e.g. Dirty PCBs. The board is still one sided, but you have to solder the components, break it in two, and sandwich-connect the two halves by soldering them together with solid leads of metal (pin headers work fine).

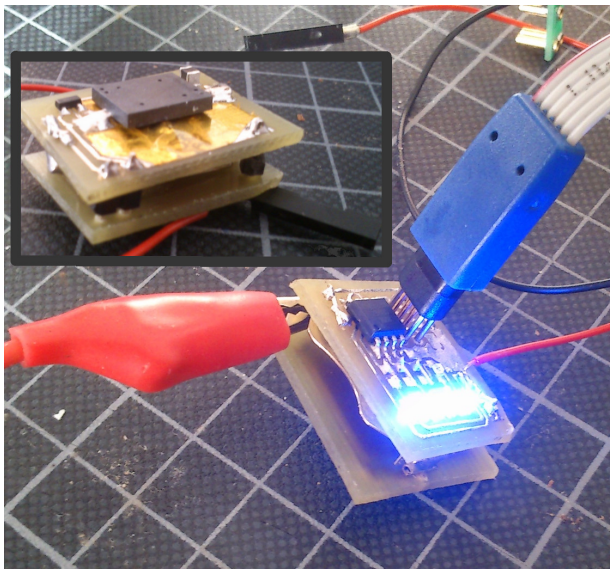

Figure 3: First prototype being programmed

If you choose to make even the board yourself, there are tons of good introductions on how to etch circuit boards at home and how to solder components to them. With that in mind, the eagle files, pictures and schematics in the ./hardware directory should be all you need, except for these components (available e.g. on Mouser):

- Atmel AtTiny25/45/85 microcontroller
- 5x LED all one color, SMD 0805 (VLMB1300-GS08)∗
- 1x LED any other color, SMD 0805 (VLMS1300-GS08)∗
- 2x 3.3 k Ohm Resistor, SMD 0805
- 1x NPN BJT transistor, SOT-23 (BC857)
- 1x 220uH Inductor, SMD 1007
- 1x Buzzer, SMD (Kobitone 254-PB119-ROX)
- 1x CR2032 coin cell and holder
- Optional:

  1x MOSFET P-Channel, SOT-23 (SI2371)∗,

  1x 10 uF Capacitor, SMD 1210∗

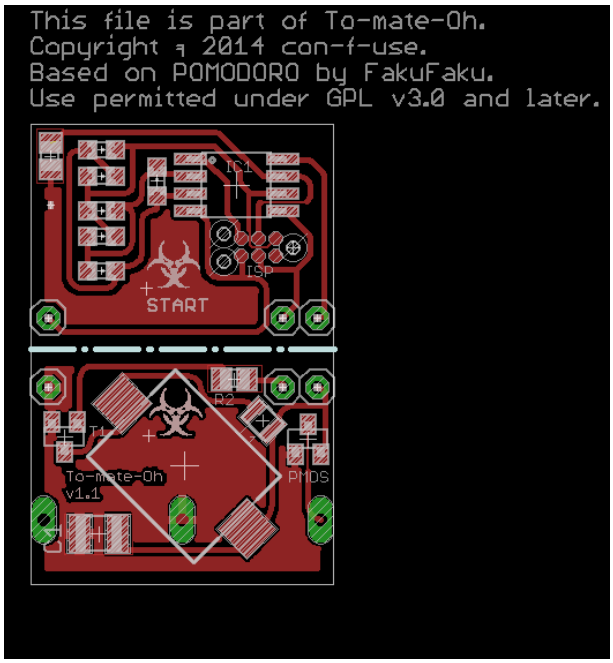

Figure 4: To-mate-Oh Board

And a few tools for soldering:

- Good soldering ion with small tip for SMD soldering
- Third hand tool
- Clothing ion
- Programmer for the micro controller
- Tag connect adapter∗

And making the board yourself (if you don't want to order it):

- Laser printer
- Dremel or similar tool for drilling small holes
- Tool to cut your copper clad board (for thin material -> scissors)
- Latex Gloves
- Acid resistant container to hold your etching solution

Note

∗The P-MOS should have a threshold voltage well below the battery voltage to work as reverse polarity protection. A low on resistance is nice, too. The 10uF cap is a good thing to enhance battery life by smoothing out peak drains/spikes. The 3k3 resistor on base of the NPN shouldn't be lower or it might interfere with programming.
The tag connect adapter is only needed if you don't want to solder temporary wires for programming or if you don't program the chip off board. It has a much smaller footprint and some other advantage. Some people have build clones.

Warning

If you use LEDs other than the Vishay 1300 series, you might get differences in brightness and even damage your micro controller. The LEDs are PWM-dimmed with no current limiting resistor. Not all LEDs have the same efficiency and forward voltage, meaning current through them may vary. And even at same current, the brightness may vary. The ATTiny must only supply up to 40mA max per output pin and only sink 10mA per input. Additionally all inputs together must not sink more than 60mA (150 mA source per output). At the low duty cycles in the program and 3.3V supply voltage from the battery there should be no problem no matter the LEDs. But if you use non-standard LEDs or supply the circuit with more than 3.3V, e.g. when programming, give it a thought or two and consult the ATTiny's datasheet .

1.3 Compiling and flashing the firmware

The firmware for this project requires avr-gcc and avr-libc (a C-library for the AVR controller). Please read the instructions at http://www.nongnu.org/avr-libc/user-manual/install_tools.html for how to install the GNU toolchain (avr-gcc, assembler, linker etc.) and avr-libc.

Lady Ada's avr tutorial covers the basics of flashing firmware to a microcontroller nicely.

However, running "make program" in the *firmware* direcotry should be sufficient. Maybe you will have to edit the *Makefile* especially if you are using a programmer other than USBtinyISP (btw.: you can build USBtinyISP at home).

To flash the firmware to the microcontroller run "make flashfuse" after you connected the ISP header on the board to your programmer. The board does not connect the Tag Connect's plus pin to the micro controller. The power pin of your programmer will have to be connected manually to the battery holder's positive contact. This is to avoid exposing an inserted battery to the USB voltage, so **before programming, take the battery out.**

If you change the firmware or fuse bits, be aware that the cap sensing and led timing is very sensitive. In the ./firmware/To-mate-Oh.h are a few settings related to LED PWM and timeouts (scroll way down in the .h file until after this documentation).

Note

There is much room for optimization. For instance, one could get rid of the delays by using a timer for the short led pulses and putting the controller to sleep for the long ones (the watchdog timer can go down to 15 ms). The watchdog timer is not that accurate and temperature stable though. Speaking of accuracy, the delay lengths could be adjusted for more accurate timing.

1.4 Using the Board

From Robins original documentation:

1. One click starts a 25 minutes *work* counter. Five yellow LED indicates how many slices of five minutes are left. There is a discrete beep when the timer expires. The LED then flashes to remind you to go into *rest* mode.

2. A second click starts a 5 minutes *rest* counter. A **red** LED is turned on. There is a discrete beep when it expires. Then the timer turns off.

3. If you wish to turn the timer off during *rest* mode, click the button a third time.

When you insert a fresh battery, a calibration for the cap sense button is performed. After a successful calibration the To-mate-Oh will circle through its LEDs and beep. If you count the beeps or how often a different LED is lit, it will give you the calibration constant. If the calibration fails, the To-mate-Oh will blink LED2 for five seconds. Most likely the sense pad is connected to a large capacity or ground then.

2 File Index

2.1 File List

Here is a list of all files with brief descriptions:

/home/confus/misc/prog/eagle/To-mate-Oh/firmware/To-mate-Oh.c
Contains the firmware to flash the microcontroler on the To-mate-Oh **??**

/home/confus/misc/prog/eagle/To-mate-Oh/firmware/To-mate-Oh.h
Contains the prototypes, constants and global definitions as well as the documentation of To-mate-Oh, a little timer device for use with the `pomodoro technique` **??**

3 File Documentation

3.1 /home/confus/misc/prog/eagle/To-mate-Oh/firmware/To-mate-Oh.c File Reference

Contains the firmware to flash the microcontroler on the To-mate-Oh.

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/power.h>
#include <avr/wdt.h>
#include <avr/sleep.h>
#include <avr/interrupt.h>
#include "To-mate-Oh.h"
```

Functions

- int main (void)
- void handle_button_press ()

  Invokes the respective actions, when the button was pressed.
- EMPTY_INTERRUPT (WDT_vect)
- void Sleep_now (uint8_t periods, uint8_t timeout)

  Put microcontroller into sleep mode to conserve power.
- void Set_leds (int led)

  Access individual LEDs through charlieplexing.
- void Shine_led (int led, uint32_t duration)

  Poor PWM for dimming an led for a duration given in multiples of 10ms.
- void Start_buzz ()

  Start the piezo buzzer.
- void Stop_buzz ()

  Stop the piezo buzzer.
- void Play_sound ()

  Plays the indicator melody.
- unsigned Get_time ()

  Measure the time to charge the button.
- unsigned Get_cal ()

  Calibrate charge time of the button.

3.1.1 Detailed Description

Contains the firmware to flash the microcontroler on the To-mate-Oh. The program is for a little battery powered timer device that can be used with the pomodoro technique.

Date

25 Oct 2014

Author

con-f-use

Copyright

3.1.2 Function Documentation

3.1.2.1 EMPTY_INTERRUPT (WDT_vect)

3.1.2.2 unsigned Get_cal ( )

Calibrate charge time of the button.

This time is later used to detect a button press.

3.1.2.3 unsigned Get_time ( )

Measure the time to charge the button.

Takes the sum of several charge cycles to be more sensitive.

Returns

Time it took to charge the button

3.1.2.4 void handle_button_press ( )

Invokes the respective actions, when the button was pressed.

3.1.2.5 int main (void)

3.1.2.6 void Play_sound ( )

Plays the indicator melody.

3.1.2.7 void Set_leds (int *led*)

Access individual LEDs through charlieplexing.

3.1.2.8 void Shine_led (int *led*, uint32_t *duration*)

Poor PWM for dimming an led for a duration given in multiples of 10ms.

3.1.2.9 void Sleep_now (uint8_t , uint8_t)

Put microcontroller into sleep mode to conserve power.

Although the ATTiny25 supports it on paper, using timeouts over two seconds leads to strange behavior in test cases. So use "WDTO_2S" as absolute maximum timeout!

3.1.2.10 void Start_buzz ( )

Start the piezo buzzer.

3.1.2.11 void Stop_buzz ( )

Stop the piezo buzzer.

3.2 /home/confus/misc/prog/eagle/To-mate-Oh/firmware/To-mate-Oh.h File Reference

Contains the prototypes, constants and global definitions as well as the documentation of To-mate-Oh, a little timer device for use with the `pomodoro technique`.

Macros

- #define BTN_THRESHOLD 2

  Threshold for the button to detect user input.
- #define ON_TIME 5

  LED on time per PWN cycle in units of 100us.
- #define DUTY 5

  Duty cycle of the LED.
- #define WORK 1

  Timer is counting down a Pomodro (25min chunk)
- #define WAIT 3

  Waiting for user to start rest period.
- #define REST 4

  5 min rest
- #define T_ON ((uint32_t)(ON_TIME)∗100)
- #define T_CYC (T_ON∗100/(uint32_t)(DUTY))
- #define T_OFF (T_CYC-T_ON)

Functions

- void Config_wdt (uint8_t)

  Intializes the watchdog timer and sets the microcontroller's pins properly.
- void Sleep_now (uint8_t, uint8_t)

  Put microcontroller into sleep mode to conserve power.
- void Set_leds (int led)

  Access individual LEDs through charlieplexing.
- void Shine_led (int led, uint32_t duration)

  Poor PWM for dimming an led for a duration given in multiples of 10ms.
- void Start_buzz ()

  Start the piezo buzzer.
- void Stop_buzz ()

  Stop the piezo buzzer.
- void Play_sound ()

Plays the indicator melody.

- unsigned Get_time ()

  Measure the time to charge the button.

- unsigned Get_cal ()

  Calibrate charge time of the button.

- void handle_button_press ()

  Invokes the respective actions, when the button was pressed.

Variables

- unsigned cal

  Storage for the calibration values of the button.

- unsigned char state

  Current state the timer is in.

3.2.1 Detailed Description

Contains the prototypes, constants and global definitions as well as the documentation of To-mate-Oh, a little timer device for use with the pomodoro technique. User input is sensed via a time integration to sense capacity in the "button". An average charging time is computed before hand. This time in turn then serves as a calibration value to detect the button press.

The charging time changes with the capacity of the button, which in turn is changed when a human comes near the button. A daramtic increase in charging time means an increase in capacity and thus something touching the button.

Date

25 Oct 2014

Author

con-f-use

Copyright

 It is based on the Pomodoro timer by Robin Scheibler (FakuFaku). See: http://robinscheibler.org and https://github.com/fakufaku/Pomodoro for more details on the original project.

3.2.2 Macro Definition Documentation

3.2.2.1 #define BTN_THRESHOLD 2

Threshold for the button to detect user input.

3.2.2.2 #define DUTY 5

Duty cycle of the LED.

3.2.2.3 #define ON_TIME 5

LED on time per PWN cycle in units of 100us.

3.2.2.4 #define REST 4

5 min rest

3.2.2.5 #define T_CYC (T_ON∗100/(uint32_t)(DUTY))

3.2.2.6 #define T_OFF (T_CYC-T_ON)

3.2.2.7 #define T_ON ((uint32_t)(ON_TIME)∗100)

3.2.2.8 #define WAIT 3

Waiting for user to start rest period.

3.2.2.9 #define WORK 1

Timer is counting down a Pomodro (25min chunk)

3.2.3 Function Documentation

3.2.3.1 void Config_wdt (uint8_t)

Intializes the watchdog timer and sets the microcontroller's pins properly.

3.2.3.2 unsigned Get_cal ( )

Calibrate charge time of the button.

This time is later used to detect a button press.

3.2.3.3 unsigned Get_time ( )

Measure the time to charge the button.

Takes the sum of several charge cycles to be more sensitive.

Returns

Time it took to charge the button

3.2.3.4 void handle_button_press ( )

Invokes the respective actions, when the button was pressed.

3.2.3.5 void Play_sound ( )

Plays the indicator melody.

3.2.3.6 void Set_leds (int *led*)

Access individual LEDs through charlieplexing.

3.2.3.7 void Shine_led (int *led,* uint32_t *duration*)

Poor PWM for dimming an led for a duration given in multiples of 10ms.

3.2.3.8 void Sleep_now (uint8_t , uint8_t)

Put microcontroller into sleep mode to conserve power.

Although the ATTiny25 supports it on paper, using timeouts over two seconds leads to strange behavior in test cases. So use "WDTO_2S" as absolute maximum timeout!

3.2.3.9 void Start_buzz ( )

Start the piezo buzzer.

3.2.3.10 void Stop_buzz ( )

Stop the piezo buzzer.

3.2.4 Variable Documentation

3.2.4.1 unsigned cal

Storage for the calibration values of the button.

3.2.4.2 unsigned char state

Current state the timer is in.