# LABC2

**(Labels Counter Version 2)**

**Hardware and Firmware Information**

**Revision 2**

## Preface

This document describes the hardware and firmware specifications of the LABC2 project. Effort was made for this document to contain all the necessary information without errors in a form that can help the novice user as well as the experienced developer. The language used is as simple as possible in order for everyone to understand the concepts and functions of the project. All information presented here provided “as is”. No responsibility/liability is held true if any error occur by the usage of the data presented in this project. You may use all information for personal use – as always at your own risk!

## Contact info, bugs report and other info

If any bugs or errors found, please report them to tano@hol.gr (alternative email is nbovos@gmail.com) in order to be corrected as soon as possible.
For any questions feel free to send me an email, I will reply as soon as possible.
Please note, the construction and usage of data of this project is at your own risk.
Donations via paypal to tano@hol.gr would be very much appreciated :-)

This document was written using Apache Open Office™ 4.1.1 on my
Pentium 4 @ 3,2GHz P.C, running Microsoft Windows™ XP Professional©

Usually I'm told that i over-analyze things, I'm afraid I might do it also here, so I apologize in advance for that...

Open Office first build by Sun microsystems & the community (now acquired by Apache Software Foundation)
Apache Open Office is a trademark of Apache Software Foundation
Microsoft, Windows, Windows XP Professional are registered trademarks of Microsoft Corporation

## What is LABC2 ?

LABC2 is a adhesive labels counter device. It uses an 8-bit MCU (specifically the ATtiny2313*) to count labels printed by a barcode/label printer. This project has been used many years (as of revision 1) for my needs at work and now ported to fit a customer's needs. Obviously the same circuit can be used anywhere a similar count is needed.

## Main concept and operations

The project is to be used along with white or printed labels. The main goal is to count the labels passing through a photomicrosensor (transmissive sensor) and display the total number of labels to the user. As for this revision there is no demand the device to give any other form of signal or to signal when a pre-set value reached. In the first revision an audible signal produced when a pre-defined number of labels have been reached/counted, the logic and circuit is the same as used in a past project 'Count-down timer' which has been presented in AVR freaks also (a long time ago, search for it...).

The number of labels/counts will be displayed in five seven segment displays (for 0 to 99999 counting). As this implies, there must be 7 lines (a through g, the dot of the display is not used) to display the desired number to each display. As known, our eyes can't 'catch' quick changes, so this concept will be used and so there is no need to drive each segment alone, but all in parallel, in a time that can't be noticed. This saves a lot of I/O pins, so only 7 I/O pins will be used (for separate segment driving this could lead to 7 x 5 = 35 I/O pins of the MCU).

To be able to select which display will be driven, five I/O lines (one for each display) will be used. Since ATtiny2313 has a maximum of 18 I/O pins, so far 7+5 = 12 I/O lines are used...A multiplexer could be used to save some I/O pins, but since there are more available it wasn't necessary. One input must be assigned to the photomicrosensor input and this leads to 13 I/O.

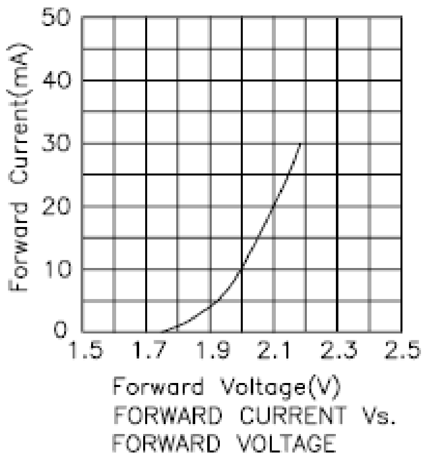
The number to be shown on the 7-Segment displays will be present to PORTB of the MCU, which is an 8-bit port. Since the dot is not used, pins PB0 to PB6 will be used leaving PB7 open (or tied to 'dot' but never to have active low signal).

The total number of I/O pins to be used so far are 14 (since PB7 will not be used). The microcontroller selected (ATtiny2313) fits quite nice for this project.

ATtiny2313 : 8-bit AVR micro controller
AVR: Atmel RISC micro-controllers

## The 7-Segment displays

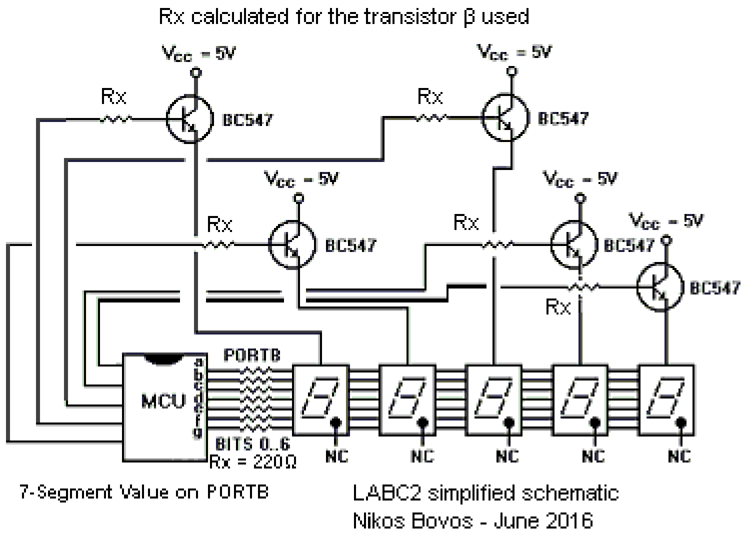
The components selected for this project are the Kingbright's SA52-11YWA seven segment displays. These are Common Anode displays in yellow color. By Kingbright's datasheet we note the typical dropout voltage of 2.1 V for current 20mA. This will help choosing the resistor value. From the previous values V/I gives about 100 Ω. A larger value will be used to keep the maximum current in lower levels (220Ω chosen).



The diagram on the left, from SA52 datasheet shows the relation between forward voltage and forward current.

The chosen current will be between 10mA and 15mA, thus each display will have 2 to 2.1 Voltage drop.

As mentioned earlier, each 7-segment display will have an individual selection circuit and common driving of segments. The following circuit will be used:



Each emitter of the NPN transistor is connected to Vcc pin of each 7-Segment display. The logic used for the digit selection is positive (+5V = Select/enable digit and 0V = Don't select digit) and the driving is done in negative logic (ON for 0V and OFF for 5V for each segment of the display, because the displays are common anode which leads to the negative logic necessity. Could be used common cathode displays and the driving would be in positive logic, just those were available from earlier projects so used instead of purchasing common cathode ones). Rx is 68kΩ (calculated for the transistors $\beta_{DC}$) and having a smaller value than calculated to produce the desired $I_C = I_E = 15mA$ approx)

## The port map of ATtiny2313

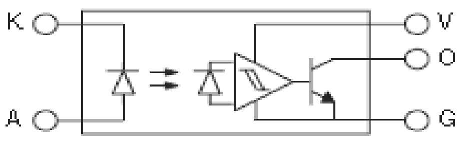
The following table is the port mapping of the microcontroller used.

| PORT A | Function | Direction | PORT B | Function | Direction |
|---|---|---|---|---|---|
| PA0 | Not used | OUT | PB0 | Segment a | OUT |
| PA1 | Not used | OUT | PB1 | Segment b | OUT |
| PA2/$\overline{\text{RESET}}$ | Configured as $\overline{\text{RESET}}$ | IN | PB2 | Segment c | OUT |
| | | | PB3 | Segment d | OUT |
| | | | PB4 | Segment e | OUT |
| | | | PB5 | Segment f | OUT |
| | | | PB6 | Segment g | OUT |
| | | | PB7 | *Not used / Dot* | OUT |
| | | | | | |
| **PORT D** | **Function** | **Direction** | | | |
| PD0 | Select 1's digit | OUT | PORT A is 2/3-bits port | | |
| PD1 | Select 10's digit | OUT | PORT D is 7-bits port | | |
| PD2 | Photomicrosensor | IN | PORT B is 8-bits port | | |
| PD3 | T.B.D later | IN / OUT | PA2/$\overline{\text{RESET}}$ is configured for Reset operation | | |
| PD4 | Select 100's digit | OUT | | | |
| PD5 | Select 1000's digit | OUT | | | |
| PD6 | Select 10000's digit | OUT | | | |

The PA2/$\overline{\text{RESET}}$ pin, performs a reset on the MCU, which re-starts the firmware for a new counting session. Obviously the counting is starting with '0' and count's up to the maximum number of 99999 before resets to 0 again (if the counting process continues).

PD2 was used as a second input in the first revision (for different type of input). By the time this document was written it wasn't sure if it will be used.

The photomicrosensor used in this project is the Omron's EE-SX398* model, which is a transmissive photosensor in 'Dark' edition (which means the output is ON for no-light / blocking of light). There is also the complementary EE-SX498 from Omron which is the 'Light' model (the output is ON for light and OFF for no-light, that could be used also (reversing the logic of the counting session). The internal circuit of the microsensor is presented below:

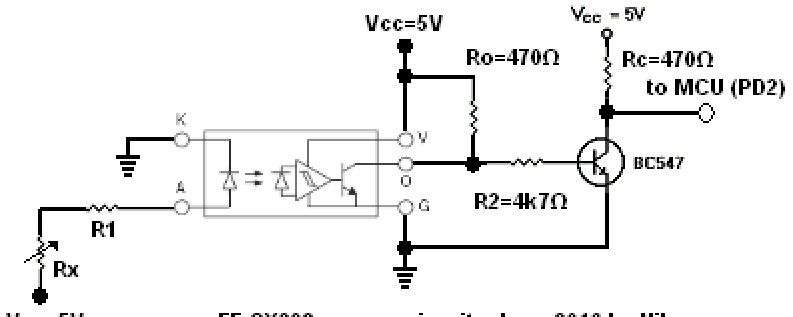


Maximum permitted current for the emitter (diode) is 50mA and for the detector circuit is 16mA. The dropout voltage for emitter is typical 1,2V. Of course to protect  the sensor, lower current values will be used for both the emitter and the detector. With a range of input voltage 4,5 to 16Volt, fits the available 5V supply we will use for the MCU.

Omron EE SX-398 pinout and diagram from Omron's datasheet

The following schematic shows the circuit used for the photomicrosensor EE-SX398. The potentiometer Rx is 1kΩ multi-turn one. The lowest current (with 1kΩ potentiometer at maximum position) is about 4mA, whereas the maximum current (with the 1kΩ potentiometer at minimum position) is about 38mA, inside the safe limits. This will be used as a sensitivity tuning. The value of $R_1$ is chosen to be in a value that the maximum current of emitter of SX-398 to be always less than the maximum (50mA) with the potentiometer to the zero value. For 50mA maximum current and 1,2V dropout voltage, the lowest value for $R_1$ is 76Ω. The resistor chosen is **$R_1$=100Ω**.

The output of the detector circuit can be connected to CMOS or TTL circuits. 5V supply is used for the detector as well (the detector has a maximum voltage of 16V). The output of the detector is driving a BJT inverter circuit. The inverter circuit is necessary first to not load the output of the detector and because the MCU will be configured for low level transition interrupt, thus a low level will appear if the detector has output a high signal (which in the dark model is true when no gap in the label is detected). The SX-398 was available so the inverter had to be inserted. You can use the SX-498 photomicrosensor and eliminate the need for the BJT inverter. It happened to have some of these sensors available - from previous projects, so I used them instead of purchasing a new one for this project.



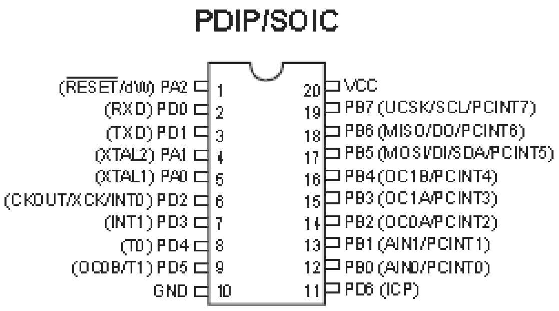
EE-SX398 sensor circuit - June 2016 by Nikos

The logic of usage is as follows:

Starting with anything (gap or no gap in a label), the MCU will wait for a high to low transition, to distinguish between the beginning and ending of each label counted and will increase the internal counters and show the number on displays. For better view, all displays are OFF before any counting is performed and only the necessary digits are displayed, while others remain in the off state (for instance if 514 is to be shown, the digits will be 514 instead of 00514 - which is of course a matter of taste for someone - I decided to do it that way). The counters keep counting for as long there is activity on the photomicrosensor circuit (and thus the INT0 line of the MCU). The maximum number displayed is as mentioned 99999 and after that the counting returns/overflows to 0.

The MCU is configured to use it's internal clock of 1MHz (nominal value at 25°C), which is the default setting of all ATtiny2313 produced (these MCUs have 8MHz internal clock and divide by 8 feature enabled, resulting in 1MHz clock). The timing is not critical, the counting process is way too slow compared to the clock cycle of the MCU and the material to count is not that small in height that the mcu will not 'catch' the transitions.

## The firmware's main operations



*ATtiny2313 pinout**

The firmware of ATtiny2313 is written in Assembly. Sometimes I find assembly easier to implement than any other method, so I use it instead of a high level language. Upon RESET, the mcu displays the message Lbc21 on the displays and after a small delay is switching off the displays and is ready to start counting.

The principal operations of the microcontroller are two:

a) Output the number of labels detected to the 7-Segment displays
b) Wait for a high to low transition on the PD2 (INT0) pin.

The mcu as mentioned is configured for a high to low transition (falling edge). This is done with the following commands (a fixed pitch font is used for better view):

```
;---------- INT0 Initialization ----------
; INT0 will be triggered on the falling edge
; (5V -> 0V or 1 to 0 transition)

ldi      genio, (0<<ISC00)|(1<<ISC01)
out      MCUCR, genio
ldi      genio, (1<<INT0)
out      GIMSK, genio
```

The **genio** register is defined with **.def genio = r16** (a high register (16-31) must be used because high registers only support the **ldi** instruction).

Negative logic is used for the display of the equivalent number :

| Number to show | 7 Segment display Equivalent | | | | | | | NEGATIVE logic equivalent | | | | | | | Negative HEX equivalent ($xx) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | g | f | e | d | c | b | a | g | f | e | d | c | b | a | |
| **0** | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | **$40** |
| **1** | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | **$79** |
| **2** | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | **$24** |
| **3** | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | **$30** |
| **4** | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | **$19** |
| **5** | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | **$12** |
| **6** | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | **$02** |
| **7** | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | **$78** |
| **8** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **$00** |
| **9** | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | **$18** |

The above values are stored to the EEPOM of the microcontroller, from 0 to 9 (which fit our needs since the number to show is also the address of EEPROM). The 'a' to 'g' signals are connected with MSB (most significant bit) of the mcu port to 'g' and the LSB (least significant bit) of the mcu port to 'a'. If reversed then the above values must be re-calculated.

ATtiny2313 pinout diagram from Atmel's datasheet

**Read and display**

Upon detecting a falling edge on INT0, the MCU will increase the counters and fetch the required number from the EEPROM and display it on the 7-Segment displays. As stated earlier, a different approach is used so to display only the necessary digits instead of all. This is done by defining a 'pattern' of digits in a register (the Mask register) and then checking it's value for each display to show. The Mask register is updated with a pattern according to the digits to be ON or OFF on each number to show. The following macro is used to fetch and output the equivalent value of a register to PORTB of the MCU:

```
;*****************************************************************
;* Macro Name   : ShowDigit
;* Parameters   : @0 = The Numeric Value To Show On 7-Segment Displays
;* Notes        : Read from EEPROM equivalent value of digit to show
;*              : All equivalent to 7-Segment display digits are
;*              : stored on positions 0 to 9 on EEPROM, otherwise
;*              : in the command after mov saveEEP,@0 we had to
;*              : add the offset (i.e +10 positions will be
;*              : saveEEP = saveEEP + $0A (+ 10dec) and then out ...)
;*              : Since we don't write to EEPROM there is no need
;*              : to wait for any checking before read.
;*****************************************************************
.MACRO ShowDigit                ;Show Digit Macro
        mov     saveEEP,@0      ;Copy to save_EEP the parameter to show
        out     EEAR,saveEEP    ;Define address to access
        sbi     EECR,EERE       ;Set READ EEPROM signal (EECR bit 0 =1)
        in      saveEEP,EEDR    ;Retrieve EEPROM contents to saveEEP
        out     PORTB,saveEEP   ;Out the result to PORTB
.ENDMACRO
```

As shown, the first parameter (@0) is copied to the saveEEP register. The EEPROM address is defined and the strobe to read the EERPOM is signaled. The value read from the EEPROM is saved also in the saveEEP register, which is then output to PORTB. Port bit 7 is not used, it's left unconnected. If you connect PB7 to the dot pin of the display and you want to light it up on counting, then you must use the numbers of the above table, otherwise you must add $80 to each value, to make it not lighting (negative logic, PB7 is the MSB).

Five registers are defined for the five digits to use. These are increased in every interrupt and checked for their values before displayed to the five digits. A little more complicated check is done to decide which digits to show and which not (according to the previous need for not displaying the leading zeros in the number). The macro/routine used is named ShowDigitAlt in the assembly listing and checks every digit before display (the alternative routine of the 'standard' ShowDigit that shows all the digits is also in the assembly file.

The assembled source is about 550 words (about 27% of chip's capacity). In these words there are also some routines that are not used (i.e larger delays and the standard implementation of the display), but size is not issue here so I left them inside.

## The 'Alternative Display' routine

This routine is used to display only the necessary digits on the displays. There is also in the assembly file the 'standard' routine (ShowCount) that can be used instead. This routine uses a 'clever' checking to decide which digits to show. The **Mask** register is loaded in the Interrupt Service Routine with the values checked below:

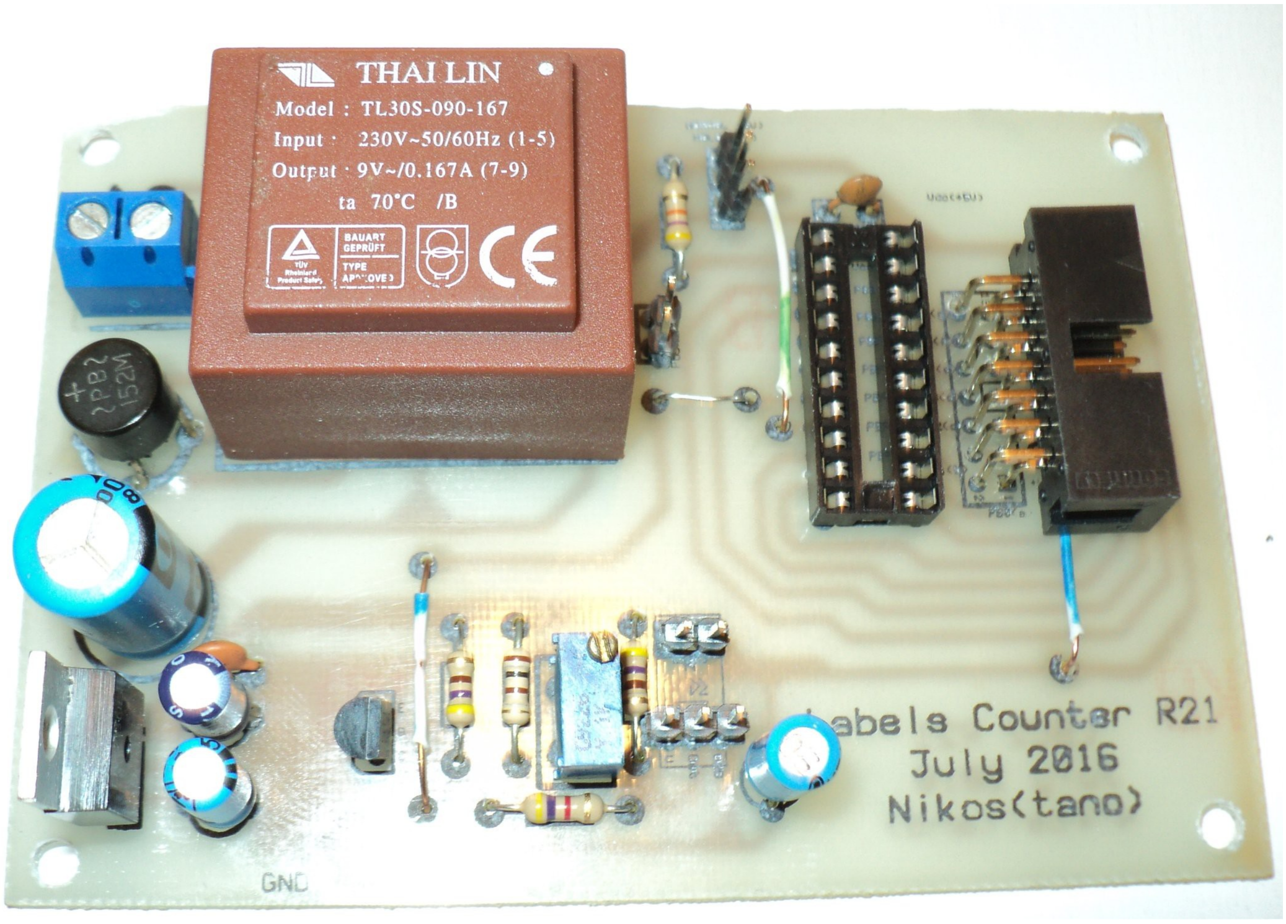
```
;*********************************************************************
;* Subroutine         : ShowCountAlt
;* Parameters         : -
;* Purpose            : Display necessary digits (Alternative display)
;* Created            : 10 Nov 2014
;* Updated            : 17 March 2015 for 2.00 , July 2016 for 2.10
;*********************************************************************
ShowCountAlt:
     digitsOFF
     rcall     delay1ms          ;Delay for 7segments
     sbrs      Mask, 4           ;Skip next if bit4 of Mask = 1 (ON)
     rjmp      ShowCnt3          ;Continue with next digit
     dispTensThou                ;Enable Tens Thousands Digit
     ShowDigit TensThou          ;Show Digit Tens Thousands
     rcall delay1ms              ;Delay for 7segments
ShowCnt3:
     sbrs      Mask, 3           ;Skip next if bit3 of Mask = 1 (ON)
     rjmp      ShowCnt2          ;Continue with next digit
     dispThou                    ;Enable Thou digit
     ShowDigit Thou              ;Show Digit Thousands
     rcall     delay1ms          ;Delay for 7segments
ShowCnt2:
     sbrs      Mask, 2           ;Skip next if bit2 of Mask = 1 (ON)
     rjmp      ShowCnt1          ;Continue with next digit
     dispHund                    ;Enable Thou digit
     ShowDigit Hund              ;Show Digit Thousands
     rcall     delay1ms          ;Delay for 7segments
ShowCnt1:
     sbrs      Mask, 1           ;Skip next if bit1 of Mask = 1 (ON)
     rjmp      ShowCnt0          ;Continue with next digit
     dispDecs                    ;Enable Thou digit
     ShowDigit Decs              ;Show Digit Thousands
     rcall     delay1ms          ;Delay for 7segments
ShowCnt0:
     sbrs      Mask, 0           ;Skip next if bit0 of Mask = 1 (ON)
     rjmp      ShowOut           ;Continue with next digit
     dispUnit                    ;Enable Thou digit
     ShowDigit Unit              ;Show Digit Thousands
     rcall     delay1ms          ;Delay for 7segments
ShowOut:
     ret                         ;Return (Came with rcall/icall)
```

All dispXXX shown above are macros for digit selection.

## Construction of the final device

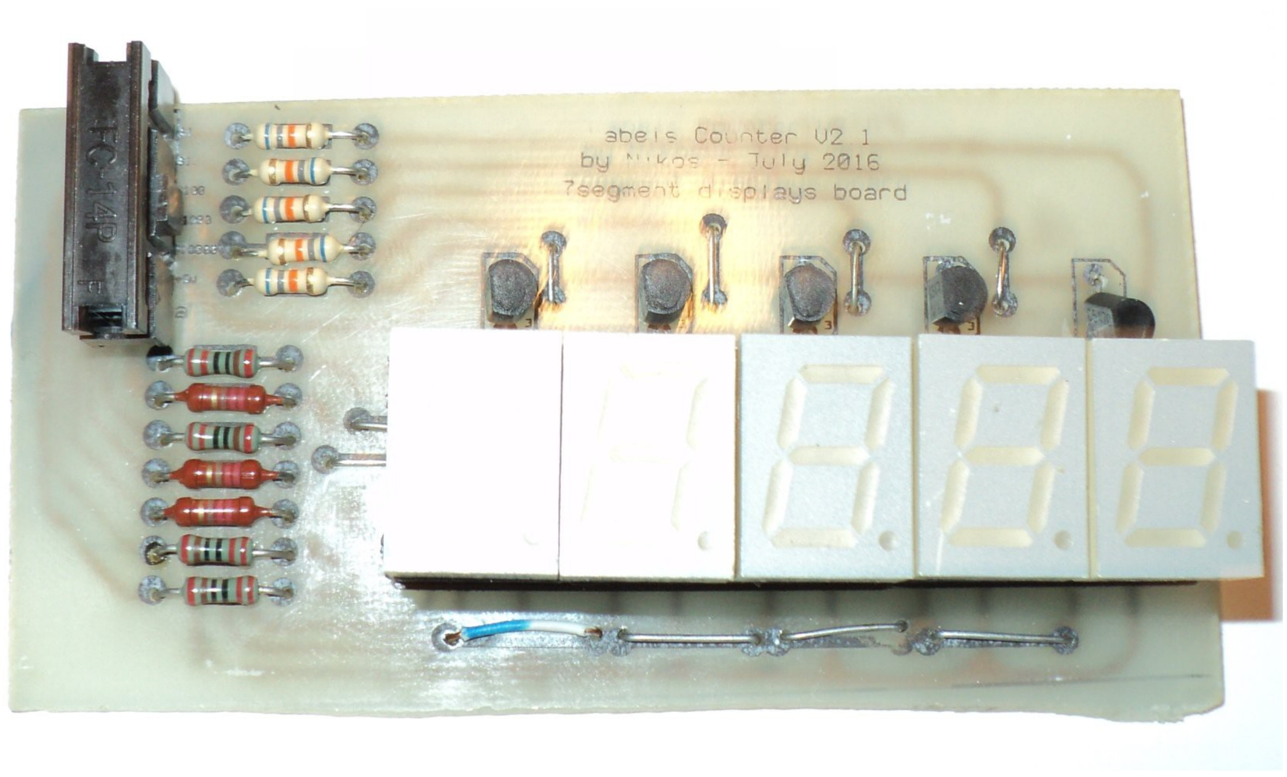
Because this will be a stand-alone device, a mains power supply included to the construction as well. It is a typical 5V power supply using the '7805' integrated circuit. A 9V PCB transformer was used along with a diode bridge and capacitor filter. The output of this supply is feeding a 7805 circuit that will power all other circuits of the device.

Two PCBs were created, the first one contain the power supply, the MCU circuit, the photomicrosensor circuit and the second PCB the displays and the driving circuits. The main PCB and the displays PCB are connected through a 14-pin ribbon cable, because the placement of the displays will be in a different location from where the main circuit will be. Also two plastic boxes were used to house the two PCBs. The photomicrosensor is guided to the path of the labels with a 5-pin ribbon cable. The following pictures show the main and displays PCB that were constructed.



**Main PCB (Mains power supply, 7805 , MCU , Photomicrosensor circuits)**

The photomicrosensor is attached to the 2 and 3 pins on the bottom of the PCB and it is placed in the labels path. A 14-pin ribbon cable is used to connect the main PCB above to the displays PCB. The upper 3-pin header is an optional for INT1 operation, but not implemented in this project. The 2-pin header in the right of the transformer will connect to $\overline{\text{RESET}}$, to reset the counting and restart the firmware from the beginning.



**7-Segment displays PCB**

The pin-out of the 14-pin header that is used is shown in the following table:

| Pin.No | Signal | Pin.No | Signal |
|---|---|---|---|
| 1 | Segment a | 2 | Segment b |
| 3 | + 5V | 4 | + 5V |
| 5 | Tens thousands digit signal | 6 | Segment c |
| 7 | Thousands digit signal | 8 | Segment d |
| 9 | Hundreds digit signal | 10 | Segment e |
| 11 | Decades digit signal | 12 | Segment f |
| 13 | Units digit signal | 14 | Segment g |

The +5V supply is used to power the 7-segment displays through the driving transistors. The segments a to g are following the values in the table presented earlier for negative logic driving (these are connected to PORTB according to PORT-map of the MCU).

Both PCBs were housed in plastic boxes. The main PCB has also some extra protective material added, because it has live contacts (to avoid shock).

**Bill of materials**

For future reference (and to have an idea of the cost) i usually create the BOM for my projects (at least most of the times):

| Component / Description | Items | Unit price(€) | Total (€) |
|---|---|---|---|
| PCB Terminal Block (2 pin) | 1 | 0,700 | 0,700 |
| PCB copper epoxy 1 side | 1 | 2,710 | 2,710 |
| PCB Transformer 9V | 1 | 3,600 | 3,600 |
| BJT Transistor BC547/BC550 | 5 | 0,024 | 0,120 |
| Resistors ¼ Watt various manufacturers & values | 17 | 0,010 | 0,170 |
| IDC Socket 14-pin (male) | 1 | 0,260 | 0,260 |
| Box header 14-pin (female) | 2 | 0,350 | 0,700 |
| Rectifier bridge | 1 | 0,500 | 0,500 |
| SA52-11YWA 7-segment displays common anode | 5 | 0,820 | 4,100 |
| Capacitors (various values and type) | 5 | 0,035 | 0,175 |
| L7805CV regulator | 1 | 0,330 | 0,330 |
| PCB pin headers | 20 | 0,005 | 0,100 |
| IC Base 20 pins (for the MCU) | 1 | 0,160 | 0,160 |
| Ribbon cable (for connecting main to display PCB) | 1 | 0,450 | 0,450 |
| MCU Atmel Attiny2313-20PU (DIP-20) | 1 | 4,950 | 4,950 |
| Plastic box/case 130x70x50mm | 2 | 5,500 | 11,00 |
| | | | |
| **Totals** | 65 | | 30,02 |

Thanks for reading up to here... See ya!