

# **POWER-SAVING PEDOMETER MEASURES SPEED**

*-by Varun Aggarwal*

## **CONTENTS**

- I. INTRODUCTION**
- II. DESIGN FEATURES**
- III. STEP COUNTING MECHANISM**
- IV. POWER SAVING MECHANISM**
- V. DESIGN OF HARDWARE**
  - A. DECIDING THE MICROCONTROLLER
  - B. PHYSICAL ASSEMBLY OF HARDWARE
  - C. POWER FOR THE DEVICE
  - D. CLOCK FREQUENCY
  - E. INPUT CAPABILITY
  - F. OUTPUT CAPABILITY
- IV. PROGRAMMING AT90S2313 FOR THE PRESENT APPLICATION**
  - A. ISSUES CONCERNING PRACTICAL USE
  - B. PROGRAMMING ISSUES
  - C. WORKING OF DIFFERENT IMPORTANT MODULES
- V. PROBLEMS FACED AND SOLUTIONS**
- VI. CONCLUSION**
- VII. FURTHER IMPROVEMENTS**
- REFERENCES**
- ACKNOWLEDGEMENT**
- APPENDIX I**
- APPENDIX II**
- APPENDIX III**

## I. INTRODUCTION

Pedometer is a device, which is worn by a person on his belt and it measures the number of steps and the distance walked by the person. It is generally used by people who do measured exercise every day. The device counts the number of steps taken by the person and multiplies it by the average step length fed to give the distance walked by the person.

In the present project, we measure the number of steps, distance and instantaneous speed of the person using it.

## II. DESIGN FEATURES

The design solution for pedometer presented measures the number of steps taken by the user and the distance covered by him. Apart from these traditional features, it includes the following new features:

1. The present pedometer has a power-saving design. This save in power is highly beneficial since pedometer is a portable device and has power on-board. (*Discussed in Section III*)
2. This design includes measurement of instantaneous speed. Measurement of instantaneous speed becomes important for individuals, who have been prescribed by the doctor to walk at a minimum speed. (*Discussed in Section V,B (3)*)

## II. STEP COUNTING MECHANISM

The basic principle behind the counting mechanism is that when a person moves, he bends a little, so his center of gravity goes down. For every step first the center of gravity goes down and then up, this to and fro motion continues throughout walking.

Commercially available pedometers use many different mechanisms to measure the number of steps taken by the person while walking. For example, the ball mechanism, in which a ball is put in a vertical tube, which oscillates up and down on each step and hence counting is done. Another mechanism is the spring and magnet mechanism, where the spring dips on each step, registering the step through the reed switch and then spring pulls it back to the original position.

In the present project can use any mechanical contraption to count the number of steps, given that (after interfacing with supply and AVR port) it toggles from logic 0 to logic 1 and back to logic 0 on each step. We used a mercury switch to test our design. Mercury switch has a small tube containing mercury (half-filled), and two wires soldered to it, which dip into the mercury [1]. In the horizontal position, both the leads are dipped in mercury and hence the wires are short. But, on tilting the tube, the mercury slides sideway and the connection between the two wires break.

We made use of the tilting motion while walking to note logic zero and one. One end of the mercury switch is attached to an interrupt port (details later) and other to the ground. An instance of wires going open tells us that a step has been taken. Hence, we used a positive-edge triggered interrupt.

### **III. POWER SAVING MECHANISM**

For a pedometer, once the user puts it on his belt and starts walking, he doesn't use its input buttons any more till the time he wants to either change the display mode (as to display number steps, step length, velocity, etc.) or reset the count, etc. again. So there is no requirement to have the input buttons operational that this time, though counting of number of steps and display of steps/distance/speed is required.

In our design, we exploited this fact to enhance performance. We had the following software implementation structure. We polled the input push buttons used for different operations (function of different input buttons shall be elaborated later) and put the mercury switch on the interrupt line (INT0). The steps were counted and the display was refreshed using an ISR (interrupt service routines). When none of the input buttons are pressed for ten seconds (an indication that the pedometer has been put on the belt), the chip went into power down sleep mode. Thereafter none of the input buttons could be used, though the pedometer keeps on registering steps, refreshing display and refreshing time on timer0 overflow (Refer Section (IV.B)), which are interrupt driven. The RESET button may be pressed to make all the input buttons operational once again.

When the pedometer is locked (sleep mode), the measured input current drawn by the system is 4.70mA compared to 12.65mA in normal mode. Therefore the power requirement (where  $V_{in}=6V$ ) falls down to 37.87% (25.2mW) of the earlier value (75.9mW). Since the pedometer remains on the belt of the user for most of the time, the reduction in power translates to similar saving of energy. This is a considerable improvement and will be highly beneficial since pedometer is a portable device and the power for device is on-board.

*Once may note that this save in power is at cost of no additional hardware or financial cost.*

### **IV. DESIGN OF HARDWARE**

#### **A. DECIDING THE MICROCONTROLLER**

We planned to use AVR microcontroller for building the pedometer. We had to decide which AVR chip to use. We were using a LCD display because we wanted low power consumption (which wouldn't have been the case with an LED display). At least 6 pins had to be connected to the LCD display. Another 6 pins were required for input purpose. Hence we finalized using AT90S2313, which had two ports i.e. 15 I/O lines including two interrupt lines. AT90S1200 also had the same number of ports/lines available, but it

doesn't have a software stack. Since my application was interrupt driven (and used stack extensively), I planned to use AT90S2313, which had a software stack implemented in it.

Finally and luckily, all the 15 available I/O lines were used for the project.

## **B. PHYSICAL ASSEMBLY OF HARDWARE**

Initially the hardware design was developed using Eagle Layout Editor and later the boards were soldered referring to it. I used a General Purpose Printed Circuit board for assembling the circuit. Rather than using a single board, two mating boards were used. The purpose for using such a design was flexibility.

On the main board, all the ports of the AVR microcontroller were connected to male headers. Two 10-pin box headers were used to connect PORTB and PORTD together with the GND and VCC signal. A third box header was used to connect lines required for programming the AVR microcontroller. Such a board can be used for any application using the AT90S2313 by providing the required I/O capabilities on the second board.

A second board was used to assemble all the input and output peripherals and connecting the mercury switch. Initially for experimental purpose, I placed a push button in parallel of the mercury switch. Inputs were registered using push buttons, while output was displayed on the LCD display.

The details are present in the schematic.

*A list of the components used is included in Appendix I.*

*A description of the boards accompanied by the picture is included in Appendix II.*

*The schematic is included in Appendix III.*

## **C. POWER FOR THE DEVICE**

Since pedometer is essentially a portable device, we used batteries to power the board. We used four batteries, pencil cells of 1.5V each connected in series to provide a voltage of 6.0V to the AVR. This is the maximum voltage AT90S2313 works upon. Such a high voltage was used, because LCD display was being used and it needs 5V to become operational.

## **D. CLOCK FREQUENCY**

A quartz crystal of frequency 4.15MHz was used to clock the microcontroller.

## **E. INPUT CAPABILITY**

There are 7 push buttons used for inputting data to the pedometer. Their details are as follows:

1. NO. OF STEPS button (PD6): Pressing this button puts the pedometer in the STEP mode. In this mode, the number of steps taken by the person is displayed.
2. DISTANCE button (PD5): Pressing this button enables the DISTANCE mode. In this mode the distance walked by the person is displayed.
3. VELOCITY button (PD4): Pressing this button enables the velocity mode. In this mode the instantaneous velocity is displayed.
4. START/HALT Button (PD1): After this button is pressed the pedometer either starts counting steps or stops counting steps. It toggles from the previous state. This can be used when the person is not walking or is changing some settings.
5. RESET NUMBER button (PD3): On pressing this button, the pedometer sets the number of steps taken and distance covered to zero. It also halts the counting of steps (which can be restarted using the START/HALT button).
6. STEP SET button (PD0): This button helps the user set his step length. Pressing it automatically halts the counting of steps. Once he has set the step length, pressing this button re-enables the normal mode. This step length has to be inputted in centimeters ranging from 0 to 255cms. Once this button is pressed, the operation of two of the above five buttons change. They are listed underneath.
  - a) NO. OF STEPS to INCREASE: NO. OF STEPS button is now used to increase the step length. Pressing the button increases the step length by one centimeter.
  - b) DISTANCE to DECREASE: DISTANCE button is now used to decrease the step length. Pressing this button decreases the step length by one centimeter.
7. RESET button (RESET PIN): This button is connected to the RESET pin of the AVR. To save the power of the battery, an internal mechanism has been implemented in the pedometer. When none of the six above buttons are pressed for ten seconds, the chip goes to power down sleep mode. Thereafter none of the above 6 buttons can be used any more, though the pedometer keeps on registering steps, which is interrupt driven. To make these buttons functional again, the RESET button has to be pressed.
8. The mercury switch: The mercury switch is attached to the INT0 line of the controller. A push button is presently connected in parallel to it for diagnostic purposes.

## **F. OUTPUT CAPABILITY**

All the outputs are displayed using a 16X1 (gdm1601A) LCD display. To understand the functionality of the LCD display, an Internet resource was used [2]. Details of interfacing the LCD display with the microcontroller are as follows:

1. VSS connected to ground
2. The VDD line was given a voltage of roughly 5.3V. Basically the 6V available from the battery were given a drop of 0.7V using a diode (forward biased) and a small capacitor was put in parallel to avoid spikes. Details are present in the schematic.
3. The V0 line was given a voltage of roughly 0.8V, whose details are present in the schematic.
4. RS was connected to PB2
5. RW was connected to PB1
6. EN was connected to PB3
7. DBO-DB3 pins were grounded. An LCD display can be interfaced with the microcontroller in two ways, one being an 8-bit interface and the second being a 4-bit interface [2]. When the 4-bit interface is used, we can save on ports of the microcontroller, but the clock cycles used to display or send a command to the LCD display increases. Since, our application is not highly time sensitive and there are limited ports on the AT90S2313 chip, the 4-bit interface was used, where only DB4-DB7 pins were used and DB0-DB3 pins were grounded.
8. PB4 to PB7 connected to DB4 to DB7 of the LCD display
9. Pins 15 and 16 of the LCD display remained unused.

Secondly, a red coloured LED is connected to the PBO pin. The light of the LED goes off once the pedometer goes in the SLEEP mode (power saving mode). The LED is simply connected to the PBO line through a resistor with its other end on VCC (6V) supply. Details present in schematic.

## **V. PROGRAMMING AT90S2313 FOR THE PRESENT APPLICATION**

### **A. ISSUES CONCERNING PRACTICAL USE**

Before coding the pedometer, some issues concerning the practical usability of the device were addressed. They are as follows.

1. Unit of measurement of length: I decided to use cm as the unit for measuring the step length and distance. The basic reason to do so was to save myself from working in

fractional numbers with a RISC architecture controller. I observed that the average step length is 60cm and hence using meters would have involved fractional numbers and lot of number crunching.

2. Range of step length: The range in which the step length can vary is 0 to 255cm. The average step length is 60cm, hence a maximum of 255 cm is sufficient. An 8-bit binary number (stored in a single register) was used to contain this value.
3. Maximum number of steps walked, distance covered: The pedometer can measure upto 65535 steps before resetting the value to 0. This implies that the distance that a person can measure using this pedometer is around 39 km, which is sufficient (calculated using step length equal to 60cm). To store the step value, two concatenated registers were used and for keeping record of distance three concatenated registers were used.
4. Velocity measurement issues: The unit chosen for velocity measurement is cm/sec. The maximum velocity that can be measured by the pedometer is 65535cm/sec. The microcontroller only understands a maximum time gap of 9 seconds between two steps, which is logical enough. Hence the processor can calculate a minimum velocity of 6cm/seconds.

## **B. PROGRAMMING ISSUES**

The following are the basic constructs of the software for the pedometer:

1. Maintaining a timer: Since, the velocity had to be calculated and the sleep mechanism had to be implemented, a constant record of the time had to be kept. For the same purpose, `tcnt0` register was used. But this register could record time for a maximum of 0.063 seconds when it increments on every 1024 clock cycles. To solve this problem, three registers were joined with the `tcnt0` register, which were refreshed on occurrence of overflow interrupt for `timer0`. This way time could be kept for a maximum of 294 hours (though using two joined registers would have been able to only count upto 1.5hr)!!!
2. Sleep Mechanism: Another group of 4 registers (the `tims` registers) were used to capture the time, whenever any button was pressed. Every time, at the end of polling the switches once, it was seen whether the last captured time plus 10 seconds is less than the present time. If it so, the controller got locked, else continued in the loop.
3. Capturing time: When we capture time in the 4 `tims` registers, it had to be seen that no increment (overflow) takes place, during this procedure. Therefore, the overflow interrupt was disabled and then re-enabled after capturing the time.
4. The SLEEP-RESET structure: To incorporate the power saving mechanism, it was decided to put the controller to SLEEP. For the purpose of waking up the pedometer to work with full functionality, the RESET key was used. Since, on RESET all

register, port values are reset, a copy of important register values (for step value, distance, etc.) were made in the SRAM.

### C. WORKING OF DIFFERENT IMPORTANT MODULES

A short description of functionality of important modules is included underneath.

1. Module to manage the timer0 overflow (timovfl): Whenever a timer 0 interrupt occurs, the above mentioned module increments the tim1 register and 'adds with carry' zero to tim2 and tim3 registers. Therefore these set of three registers together with the tcnt0 register carry the instantaneous value of time measured from starting of the pedometer. Also the value of tcnt0, tim0, tim1, tim2 and tim3 are written to their allotted memory locations, so that they can be used even after waking from SLEEP. A maximum error of 0.062 seconds can take place in this way.

It might be argued that the timer0 overflow interrupts occur repeatedly reducing the advantage of power down mode. But, timer0 overflow interrupt occurs only every 0.063 seconds, and is serviced in  $3.614 \times 10^{-6}$  seconds!!! Therefore the processor is in sleep mode for 0.063 seconds and power up mode for some microseconds, thus this doesn't form an appreciable part of the power consumed. Similar explanation exists for interrupt on int0 line.

2. Mercury switch Interrupt (int): On getting a positive edge on the line where mercury switch is interfaced, the int routine is called. Then the number of steps is incremented, the step value is added to distance covered. Thereafter, distance or step is displayed according to the present status. Also these values are written in memory. If velocity is chosen, then another routine calc\_vel is called which calculates and displays velocity.
3. Module to calculate velocity (calc\_vel): To calculate velocity, the step length is divided by the time difference between two consecutive interrupts (one step). Since the time difference will be recorded from tcnt0 and tim registers, we get it in clock cycles/1024. So the formula for velocity in cm/sec is given by:

$$\begin{aligned} \text{Velocity} &= \text{step value} / (\text{tim\_diff} * 1024 / (4.15 \times 10^6)) \\ &= \text{Step value} * 4053 / \text{tim\_diff} \end{aligned}$$

Therefore, first step value was multiplied by 4053 to get a value in three joined registers. The time difference between two interrupts is noted in two joined registers, which makes it a maximum of 16 seconds difference. Then the value calculated above is divided by the time difference registers to get the velocity. Finally this velocity is displayed.

*The multiplication precedes the division to reduce the quantization error.*

4. Module to display values on LCD display: Different display modules are built to display different kind of values, i.e. a different function is called to display step value

(one register long) and a different one to display number of steps (two registers long), etc. In these subroutines, the value is first moved to temporary registers. Then the number is converted from binary to decimal. Digits are extracted one by one by division by 10000, 1000, 100, 10 (for maximum value of 65535) and displayed simultaneously on the LCD display. The remainder is finally displayed to complete the operation.

5. Module to set Step (step\_set): First of all, obviously, the debounce routine is called to check that the button is released and has stopped oscillations between logic 1 and 0. Pressing the button also stops recording of further interrupts. As mentioned in the input capabilities section, the step and distance button are used to increment and decrement the value of step value, while pressing the step set button again indicates that the step has been set and pedometer returns to normal mode. These three buttons were polled and the step value is incremented, decremented or frozen according to the button pressed. The step value is written in the memory.
6. Module to set distance, number of steps or velocity status: These modules set status according to the button chosen and display the chosen value. The status keeps record to be used by the interrupt routine, so that it knows what to display when the interrupt occurs.
7. Start/Halt routine (st\_hl): It simply disables/enables interrupt on int0 line by writing 0b00000000 / 0b01000000 in the gimsk register. It toggles from previous state.

Many other modules are used, but they are moreover simple and straightforward and hence not discussed.

## **VI. PROBLEMS FACED AND SOLUTIONS**

The following are some of the problems I faced. I had a hard time working on 57 variables with 32 available registers. I extensively used the Stack capability present in AT90S2313. In many routines, the registers being used in that routine were push at beginning and popped at the end of routine. An alternate solution was to keep a copy of the extra variables in the SRAM. Another problem I faced was with interrupts. When I disabled interrupts on the int0 and int1 line (by writing 0 on GIMSK register), and then created interrupts from the mercury switch (during the disable mode), on enabling interrupts back, the controller serviced 1 interrupt. I tried many solutions but nothing worked. Finally, I polled the interrupt line at the beginning of the ISR. And only if the interrupt line is at logic 1, then only service it. This solved the problem.

The following is a conceptual problem, which surfaced due to the use of mercury switch. I realised that the mercury switch has a problem. Because mercury is a viscous liquid, whether the leads of the two dipped wires are short not only depends on the position of the mercury switch but also on its past position. For example: If the mercury switch achieves tilt of 20° from 30°, and on the other hand if it achieves a tilt of 20° from 10°, the state of the wire at 20° might be different in both the cases. This may create errors in the

step counting mechanism. I can think of no solution at this time. All innovative ideas are invited.

## **VII. CONCLUSION**

I used all the ideas presented herein to build and design a pedometer, which worked satisfactorily. This design of pedometer not only combines all design features of a traditional pedometer, but is power-saving and measures instantaneous speed.

## **VIII. FURTHER IMPROVEMENTS**

Some more improvements can be made to this system. A copy of the number of steps, step length can be made in the EEPROM (AT90S2313 has a 128 bytes of EEPROM), so that the user doesn't have to re-set the pedometer every time he uses it after switching it off. Secondly to reduce the weight of the system, the four pencil cells can be replaced by a single 1.5V battery and a voltage amplifier can be used to get 6V. A third improvement can be to put a diode in series of the voltage supply, so that if the user mistakenly puts battery in the reverse polarity, the system doesn't get destroyed. These improvements can be done very easily using standard techniques.

# References

1. <http://space.tin.it/scienza/fladelle/Page9.htm>: Internet resource containing a digit pedometer circuit.
2. <http://www.doc.ic.ac.uk/~ih/doc/lcd/index.html>: Internet resource containing LCD display datasheet and instruction set.
3. “**Programming the AVR microcontroller**”, book (ISBN007134666x) by Mr. D. V. Gadre.
4. <http://www.avr-asm-tutorial.net>: Internet resource containing tutorial on AVR assembly.

# Acknowledgement

I acknowledge the support and help provided by Mr. Dhananjay Gadre, AP, NSIT. Without his support, guidance and encouragement, this project couldn't have been put together. I also thank him for providing the basic hardware and dongle for programming my board.

I also thank Mr. Gerhard Schmidt for answering my queries through email regarding AVR assembly.

**Varun Aggarwal**

# Appendix I

**General requirements:** Universal PCB, Screws and nuts to mount the board, hard wires, solder wire, soldering iron, wire stripper and wire cutter

## **List of components:**

### **Main board**

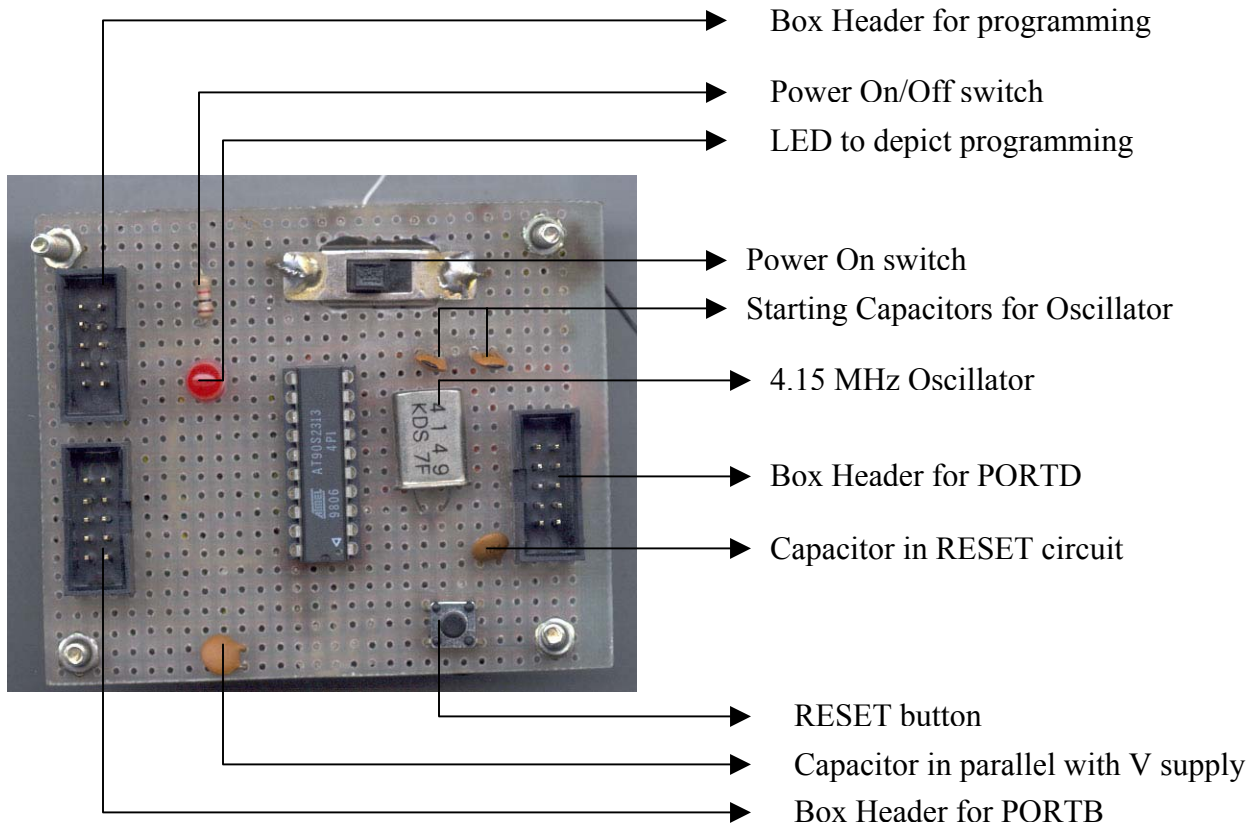
1. 20 pin IC holder
2. AT90S2313 chip
3. On/Off Switch (SPST)
4. 4-battery holder
5. 3 10-pin box headers (male and female both)
6. Cable to connect 2 females of box header.
7. 4.15MHz Crystal
8. 2 Starting capacitors 33pF (ceramic)
9. 2 capacitors 0.1uF (ceramic)
10. 1 LED
11. 1 push button
12. 1 1K resistance

### **LCD board**

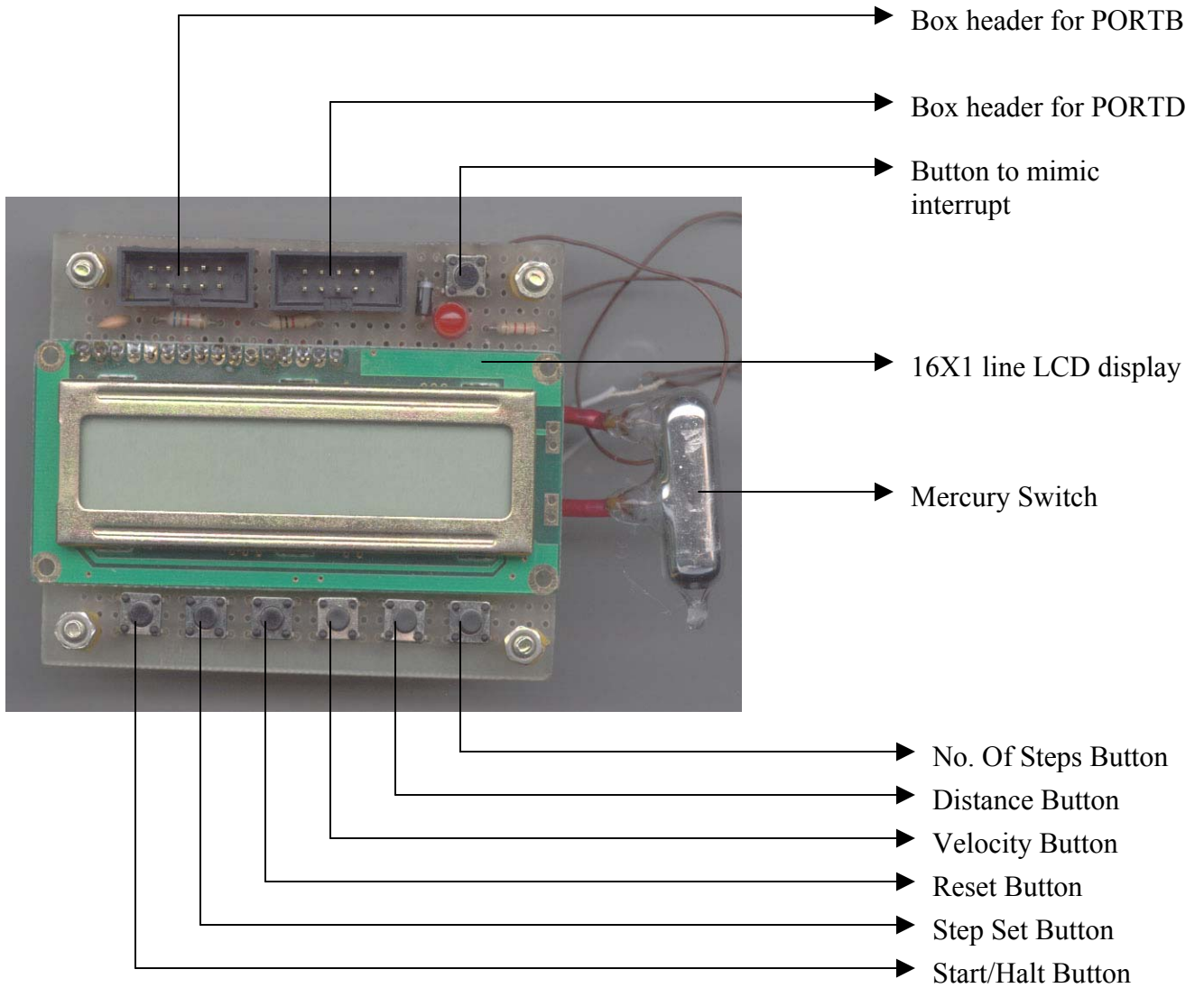
1. 16X1 line LCD display
2. 1 Mercury Switch
3. 7 push buttons
4. 2 box headers (male and female)
5. 1 LED
6. Array of 16 straight pins
7. 1 diode (0.7V)
8. 1 6.8K resistor
9. 1 1K resistor
10. 1 2.2K resistor

# Appendix II

**PHOTO 1: Photograph of the Main board (Containing the microcontroller)**

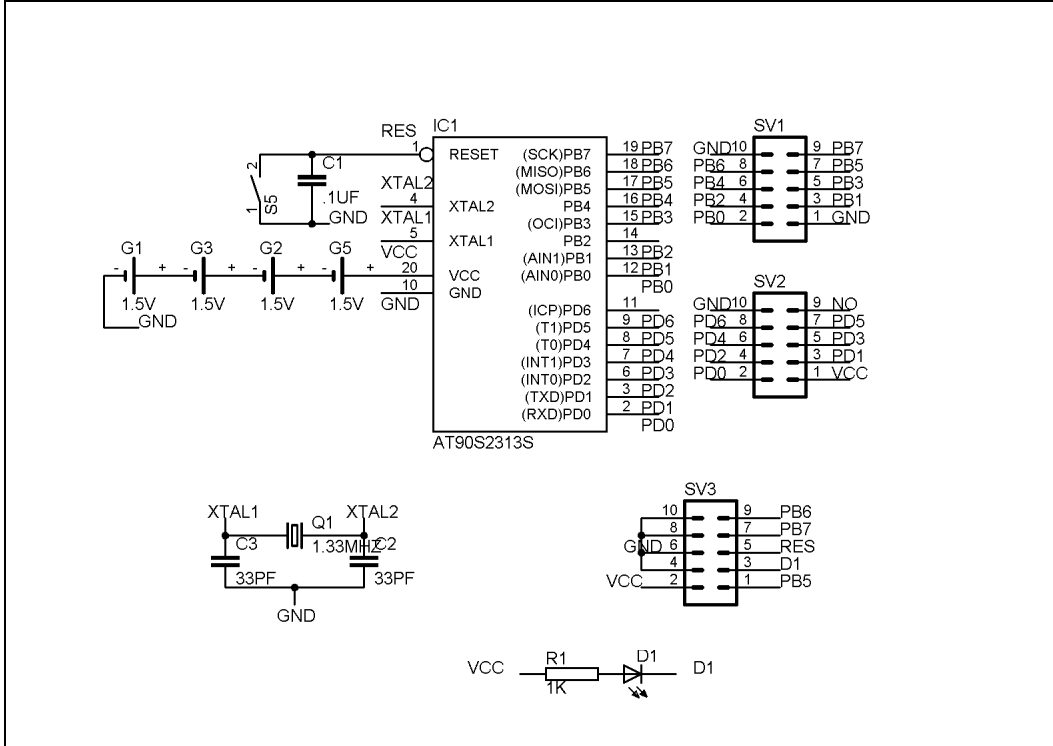


**PHOTO 2: Photograph of LCD board**

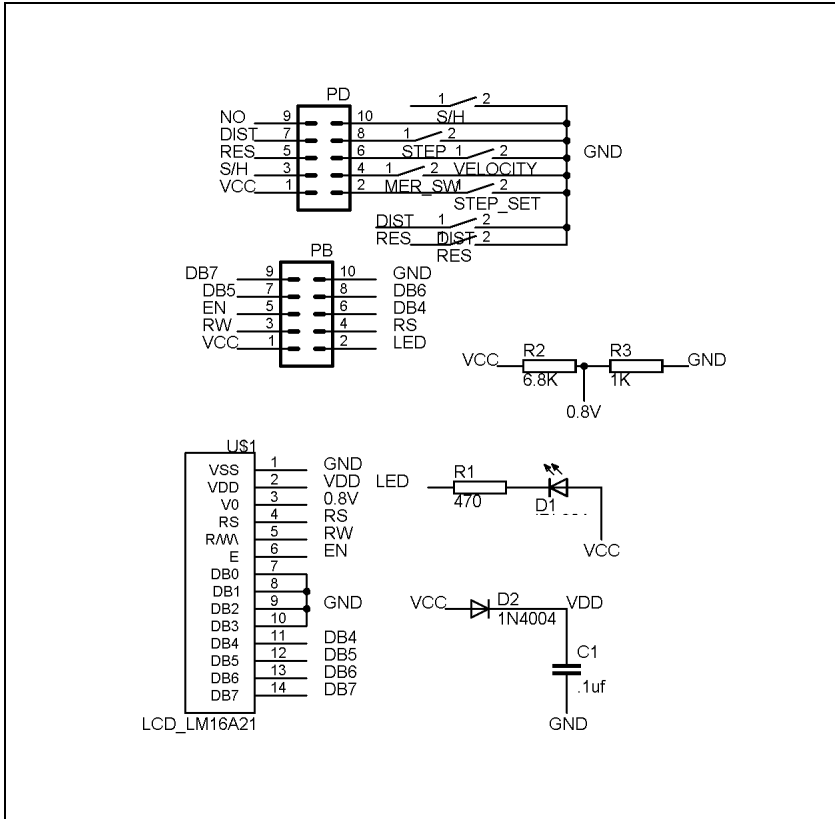


# Appendix III

FIGURE 1: Schematic of Main Board



**FIGURE 2: Schematic of LCD Board**



**FIGURE 3: Schematic of Complete Connections**

