Bicycle Speedometer with LED Design

By
Jake Conway and Adam Helmerich

Introduction

For our project we are designing a speedometer for a bicycle. This project will then take the speed of the bike using calculations done in the program and then generate a specific LED pattern that corresponds with the speed the bicycle is traveling.

User Manual

The included two sketches show how our bicycle speedometer with LED design would fit on a bicycle. The magnets in Fig.1 are used to tell the microcontroller which is seen in Fig. 2 how fast the bicycle is going.

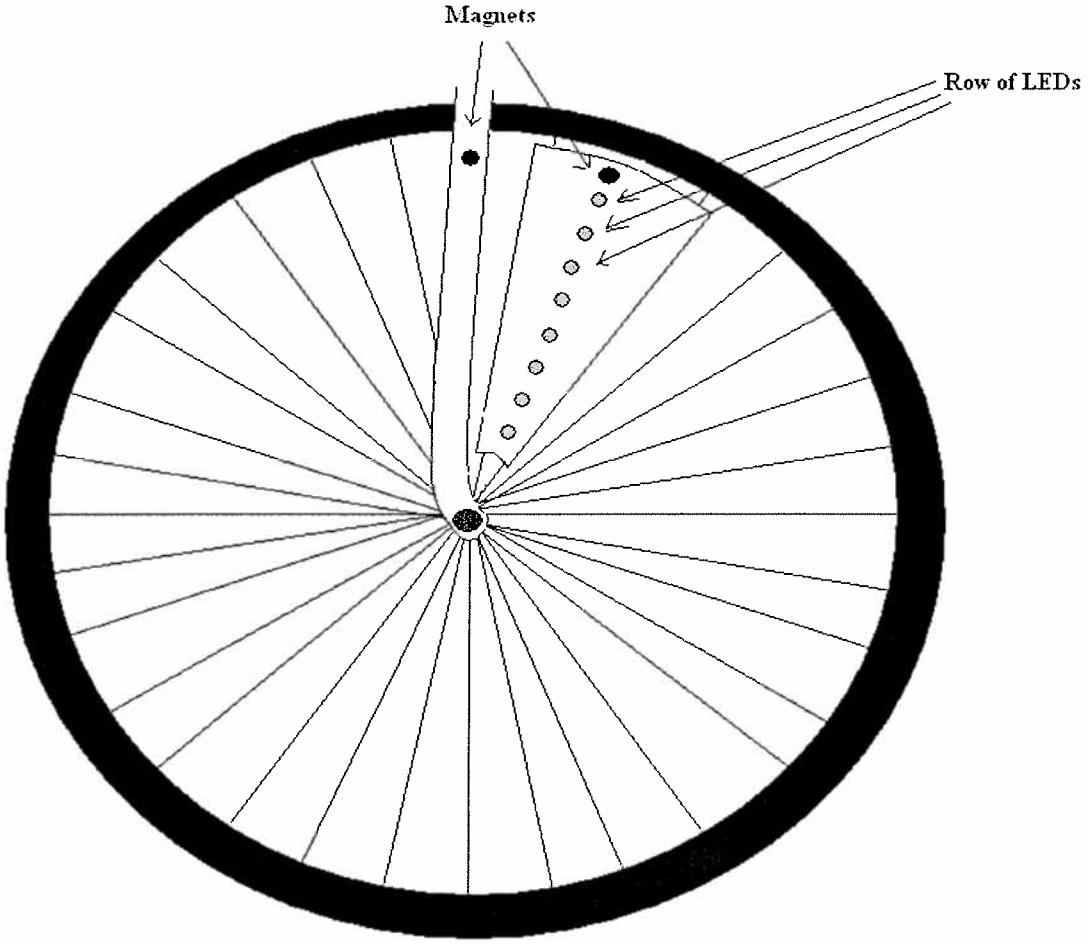


Fig. 1 – Front Tire of Bicycle (Left Side)

This will then correlate with the microcontroller on how many lights to light up on the Row of LEDs seen in Fig. 1. The LEDs light up based on the speed of the bicycle. As seen in Fig. 2 the back of the LED display holds the microcontroller and also the battery compartment to make the LED display work.

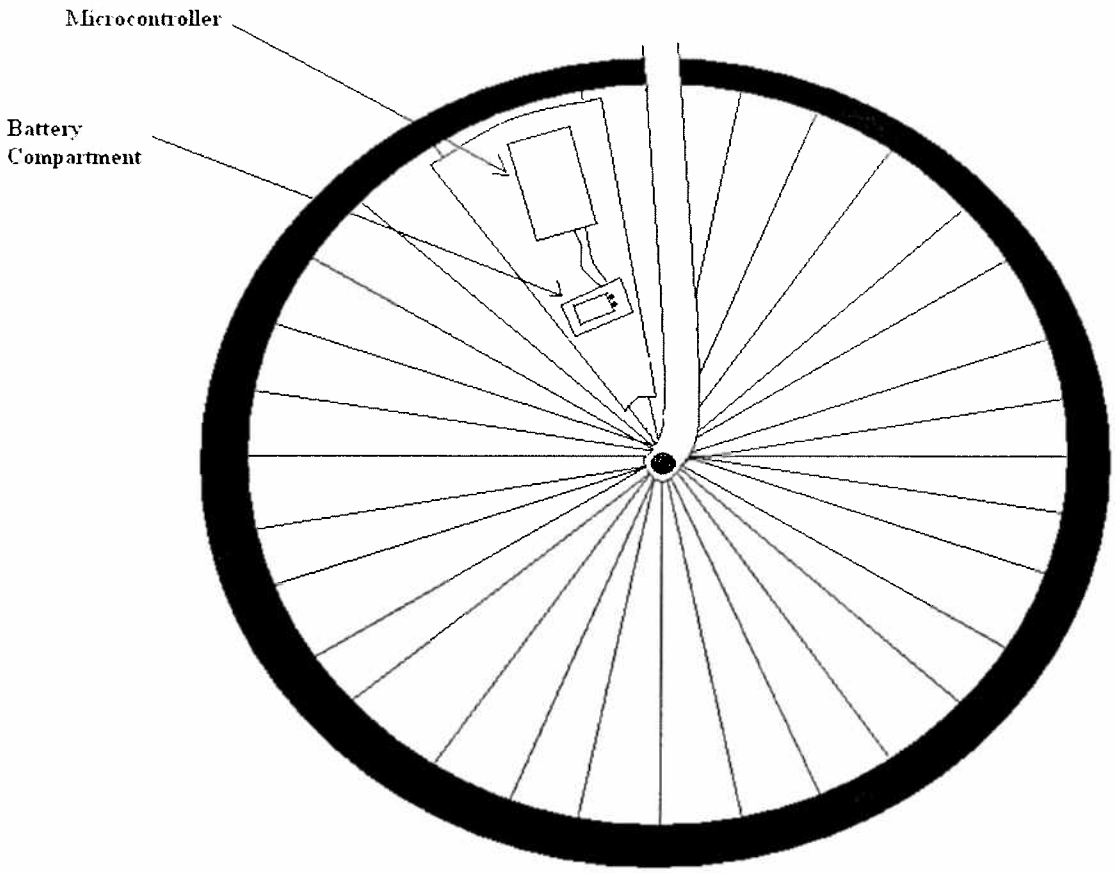


Fig. 2 – Front Tire of Bicycle (Right Side)

If an LED ever needs replaced a person would need to buy a LED and plug it into the spot of the bad LED as seen in Fig 3. The flat side of the LED would need to correspond to the flat side of the socket also seen in Fig. 3.

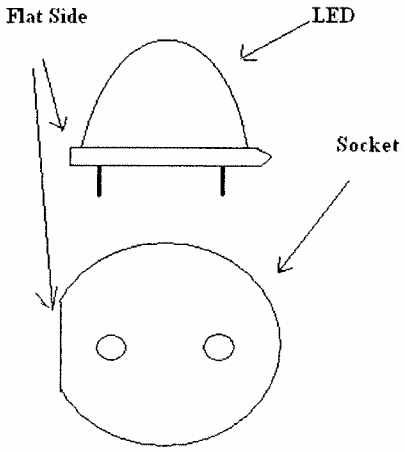


Fig. 3 – LED Replacement

If the batteries run out on your bicycle speedometer and LED display you will need to replace them with a 9V battery. In Fig. 4 the proper placement of the battery to the

connector is shown. After this connection has been made the battery and connector need to be replaced into the battery compartment also shown in Fig. 4 and in Fig. 2.

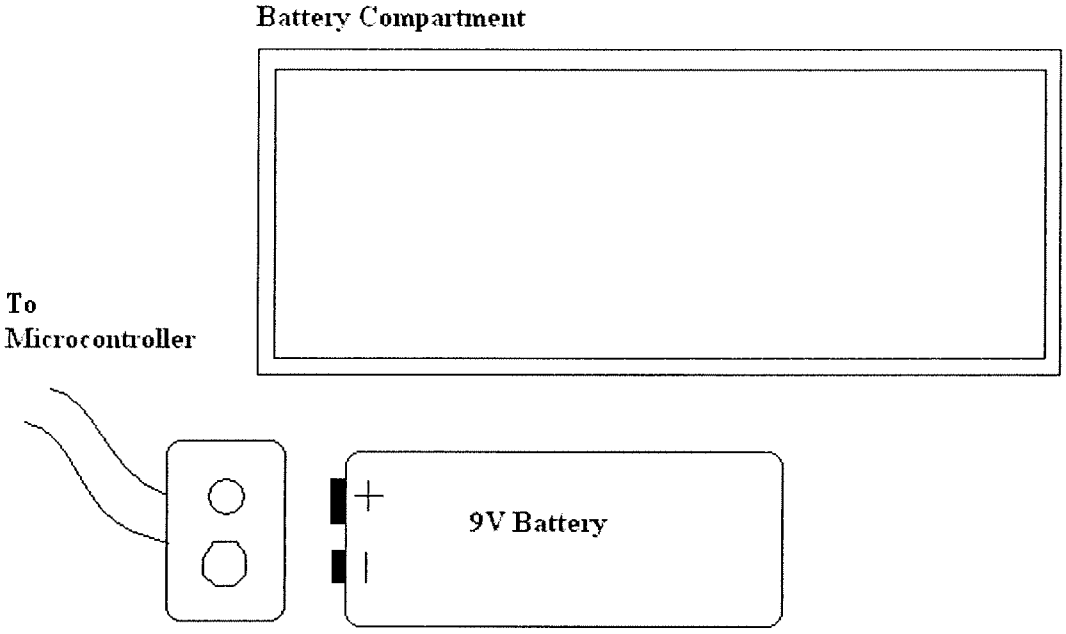


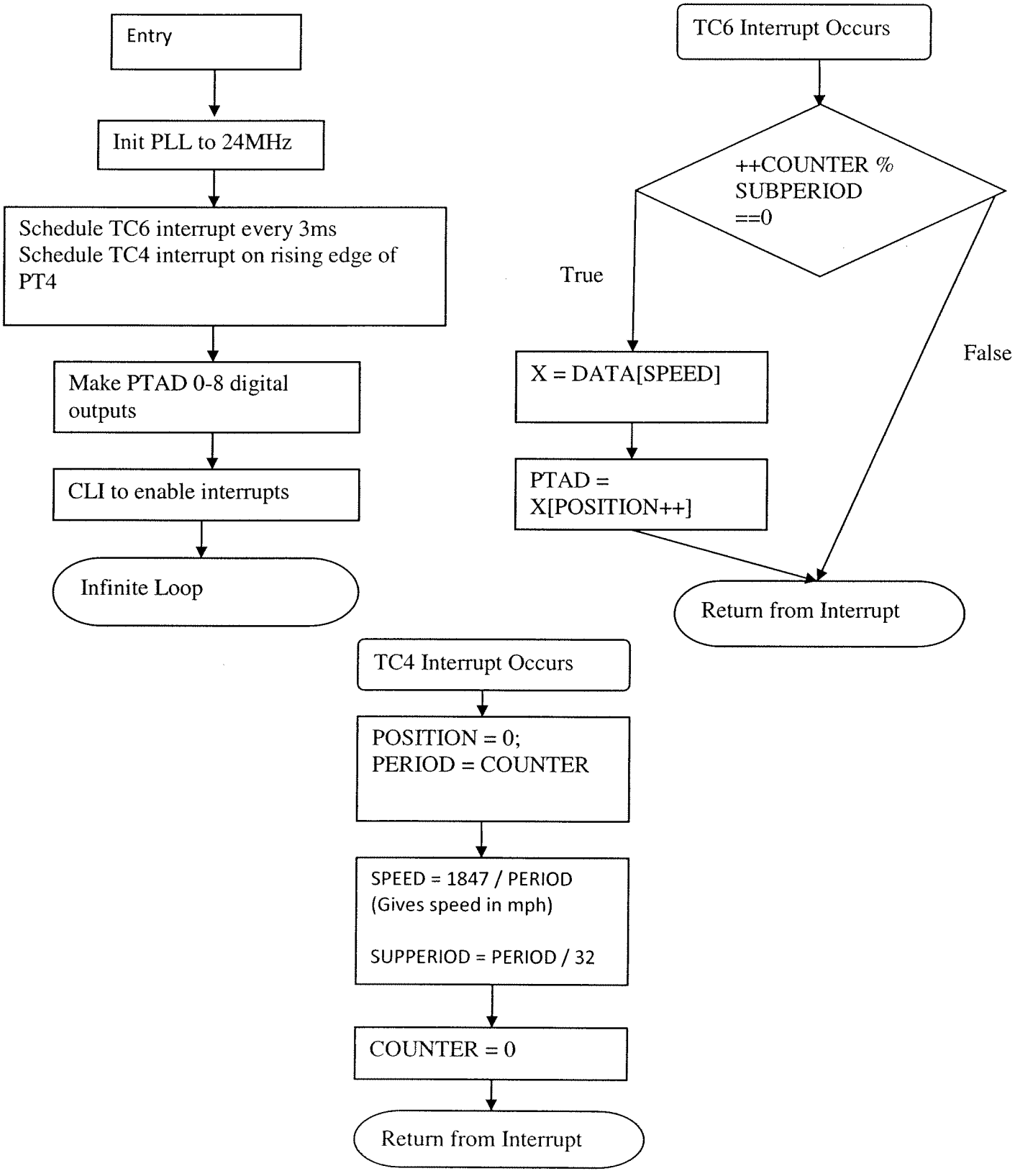
Fig. 4 – Battery Diagram

If the microcontroller would go bad please consult the hardware block diagram to see how to wire up the individual pieces of the board. If you are not familiar with these kind of diagrams are think you are incapable of wiring them up yourself please return the part to the builders and they will wire up your microcontroller to the beginning specifications.

Internal Operation

Attached on the next page is our software flowchart of how our code calculates the speed of the bicycle and then turns this into and LED display.

Flowchart of program operation



The following is a hardware block diagram of how our project works. From the battery compartment comes a 9V charge from the battery. This then passes through our voltage regulator, which then converts the voltage to 5 volts which then powers our microcontroller. Our microcontroller gets one input into it from the magnetic switch which is attached the bicycle frame to calculate the speed of the bicycle. This speed is then calculated using the microcontroller and then sent out to the row of LEDs. The LEDs will then display a pattern showing how fast the bicycle is going.

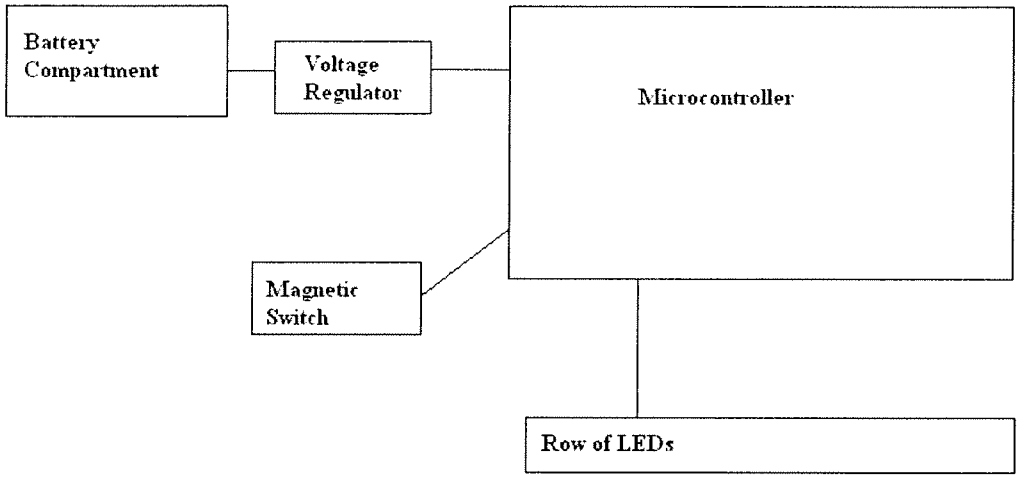


Fig. 5 – Hardware Block Diagram

In Appendix A: Schematic Design of Bicycle Speedometer you will find a detailed version of the hardware block diagram. This will show you the different connections the microcontroller has with the magnetic switch and the LEDs. Appendix B: Code of Bicycle Speedometer will contain the code that was used to program the microcontroller as seen above in Fig. 5.

Testing Procedures and Results

We were going to test our project on a bike to make sure that our switch was working properly with the program and the LEDs. However after trying to use a magnetic reed switch we found that this type of switch is unreliable. So we either used a pushbutton switch to test whether our project was working correctly or we hooked up the function generator using a square wave to test our project. Either one of these methods worked and gave us the results we were looking for. The different speeds were evident when the LEDs were lit up by the changing of the frequency of function generator or by pushing the pushbutton switch faster. Our project worked to how we wanted it to and without any problems besides not getting it mounted on a bicycle.

Bill Of Materials

I. Microcontroller and Voltage Regulator - $43.00
II. 11 Resistors - $0.40(1), $0.25(100+)[1]
III. Magnetic Switch - $1.15(1), $0.75(100+)[1]

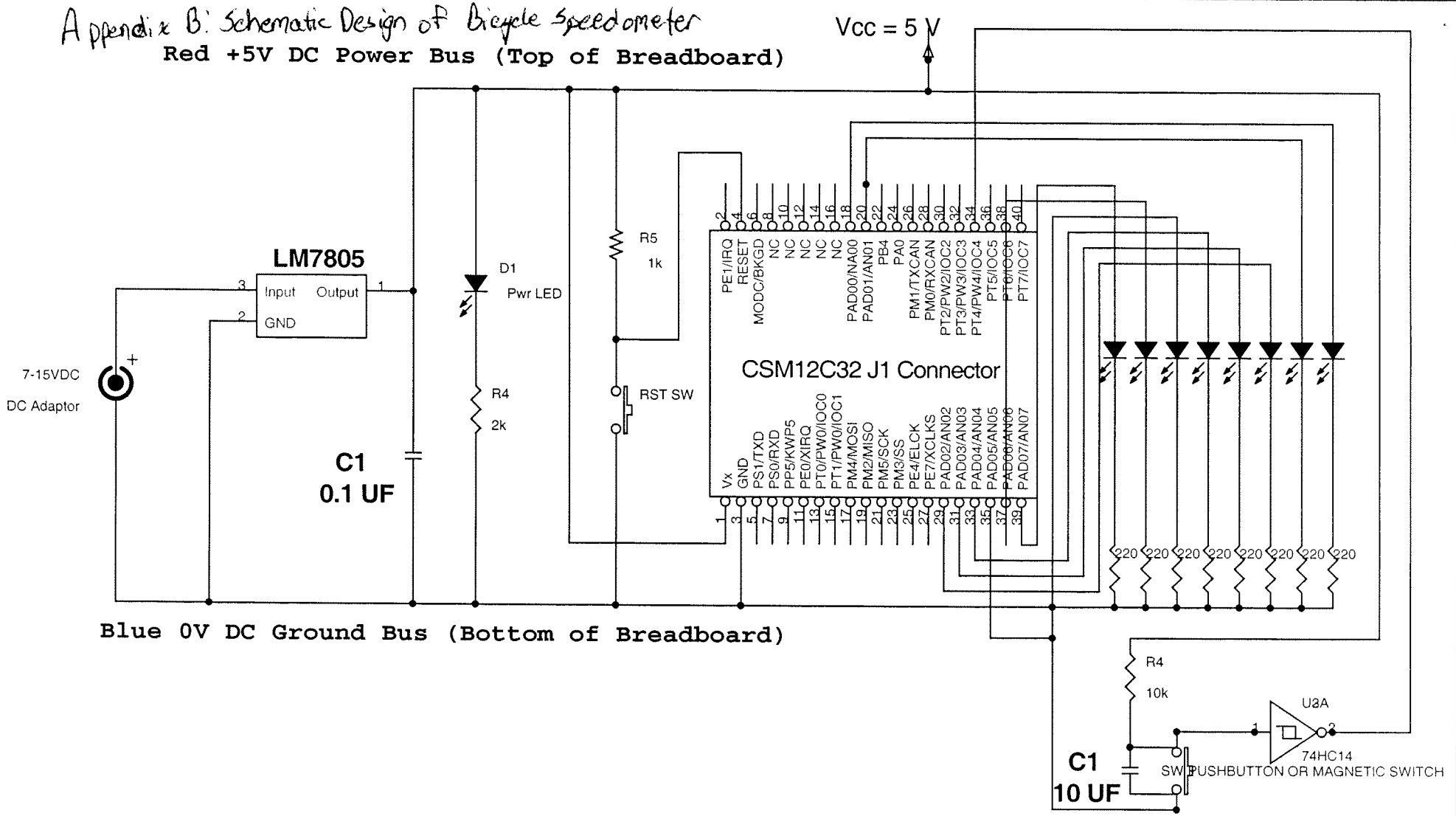
[1] Electronix Express http://www.elexp.com/

IV. 9 LEDs - $0.90(1), $0.75(10+)$_1$
V. Schmitt Trigger - $0.60(1), $0.50(10+)$_1$
VI. Pushbutton Switch - $2.35(1), $2.00(10+)$_1$
VII. 2 Capacitors - $0.15(1), $0.08[2](100+)$_1$
VIII. 9V Connector - $0.25(1), $0.18(100+)$_1$
IX. 9V Batter - $2.50(1), $1.70(100+)$_1$

For us to make one of our designs our estimated cost would be $62.65. For us to make a "1000-unit production quantity" the cost would come up to be $57790.00. The bulk pricing would save us a cost of $4860.00 if we would be each part individually to make all 1000 units.

[2] Electronix Express http://www.elexp.com/

Appendix B: Schematic Design of Bicycle Speedometer



Title: Orcad Design of Bicycle Speedometer		
Size: A	Document Number: <Doc>	Rev: <RevCode>
Date: Wednesday, November 07, 2007	Sheet 1 of 1	

Appendix B: Code of Bicycle Speedometer

```
;*****************************************************************
;* This stationery serves as the framework for a                 *
;* user application (single file, absolute assembly application) *
;* For a more comprehensive program that                         *
;* demonstrates the more advanced functionality of this          *
;* processor, please see the demonstration applications          *
;* located in the examples subdirectory of the                   *
;* Freescale CodeWarrior for the HC12 Program directory
;* PTAD(0-7) LED Outputs
;* PT(4) Reed Switch input (must be hardware debugged)
;*****************************************************************

; export symbols
            XDEF Entry            ; export 'Entry' symbol
            ABSENTRY Entry        ; for absolute assembly: mark this as application entry point

; include derivative specific macros
            INCLUDE 'mc9s12c32.inc'

  ORG   RAMStart
COUNTER: rmb 2  ;Space for storing counter data between interrupts
PERIOD:  rmb 2  ;Space for storing period
SUBPERIOD: rmb 2;Space for storing subperiod
SPEED:   rmb 1  ;Space to store the current speed
POSITION:rmb 1  ;Space to store position
  ORG   ROMStart
DAT4:   fdb 1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0
DAT5:   fdb 3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1
DAT6:   fdb 3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1
DAT7:   fdb 3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1
DAT8:   fdb 7,3,7,3,7,3,7,3,7,3,7,3,7,3,7,3,7,3,7,3,7,3,7,3,7,3,7,3,7,3,7,3
DAT9:   fdb 7,3,7,3,7,3,7,3,7,3,7,3,7,3,7,3,7,3,7,3,7,3,7,3,7,3,7,3,7,3,7,3
DAT10:  fdb 7,3,7,3,7,3,7,3,7,3,7,3,7,3,7,3,7,3,7,3,7,3,7,3,7,3,7,3,7,3,7,3
DAT11:  fdb 15,7,15,7,15,7,15,7,15,7,15,7,15,7,15,7,15,7,15,7,15,7,15,7,15,7,15,7,15,7,15,7
DAT12:  fdb 15,7,15,7,15,7,15,7,15,7,15,7,15,7,15,7,15,7,15,7,15,7,15,7,15,7,15,7,15,7,15,7
DAT13:  fdb 15,7,15,7,15,7,15,7,15,7,15,7,15,7,15,7,15,7,15,7,15,7,15,7,15,7,15,7,15,7,15,7
DAT14:  fdb 31,15,31,15,31,15,31,15,31,15,31,15,31,15,31,15,31,15,31,15,31,15,31,15,31,15,31,15,
31,15,31,15
DAT15:  fdb 31,15,31,15,31,15,31,15,31,15,31,15,31,15,31,15,31,15,31,15,31,15,31,15,31,15,31,15,
31,15,31,15
DAT16:  fdb 31,15,31,15,31,15,31,15,31,15,31,15,31,15,31,15,31,15,31,15,31,15,31,15,31,15,31,15,
31,15,31,15
DAT17:  fdb 63,31,63,31,63,31,63,31,63,31,63,31,63,31,63,31,63,31,63,31,63,31,63,31,63,31,63,31,
63,31,63,31
DAT18:  fdb 63,31,63,31,63,31,63,31,63,31,63,31,63,31,63,31,63,31,63,31,63,31,63,31,63,31,63,31,
63,31,63,31
DAT19:  fdb 63,31,63,31,63,31,63,31,63,31,63,31,63,31,63,31,63,31,63,31,63,31,63,31,63,31,63,31,
63,31,63,31
DAT20:  fdb 127,63,127,63,127,63,127,63,127,63,127,63,127,63,127,63,127,63,127,63,127,63,127,63,
127,63,127,63,127,63,127,63
DAT21:  fdb 127,63,127,63,127,63,127,63,127,63,127,63,127,63,127,63,127,63,127,63,127,63,127,63,
127,63,127,63,127,63,127,63
DAT22:  fdb 127,63,127,63,127,63,127,63,127,63,127,63,127,63,127,63,127,63,127,63,127,63,127,63,
127,63,127,63,127,63,127,63
DATHI:  fdb 127,255,127,255,127,255,127,255,127,255,127,255,127,255,127,255,127,255,127,255,127,
255,127,255,127,255,127,255,127,255,127,255



Entry:
  lds #$3f00

  ldaa #2
  psha
  ldaa #1
  psha
  jsr init_pll
  leas 2,SP         ;initialize PLL to 24 MHz

  movb  #0,TSCR2    ;Set prescaler bits to 5 so TCNT increments every
                    ;1/24MHz = 41 nanoseconds.
  movb  #$80,TSCR1  ;Enable Timer TCNT to begin counting
  movb  #$40,TIOS   ;Make TC6 an Output Compare register
  bset  TIE, #%01000000 ;Enable TC6
  ldd   #0          ;Loads 0 into register D
  std   TC6Hi       ;Schedule next output compare interrupt to occur when counter overflows (Every
ms)
```

```
  movb  #$40,TFLG1  ;Make sure TC6 interrupt flag is cleared


  bset  TIE,#$10    ;Locally Enable TC4 interrupts
  bclr  TIOS,#$10   ;Make TC4 an input capture pin
  bset  TCTL3,#$02  ;Trigger on falling edge for PT4
  bclr  TCTL3,#$01  ;Don't trigger on rising edge for PT4
  movb  #$10,TFLG1  ;Make sure TC4 interrupt flag is cleared

  movb  #$FF,ATDDIEN;Make PTAD 0-8 digital
  movb  #$FF,DDRAD  ;Make PTAD 0-8 outputs

  cli

LOOPER:
  bra LOOPER




TOC6ISR:
  ldx   COUNTER     ;Load Counter
  inx               ;Increment it by 1
  stx   COUNTER     ;Store Counter back
  tfr   x,d         ;move x to d
  ldy   #0          ;clear y
  ldx   SUBPERIOD   ;Load subperiod into x
  ediv              ;Y:D/X => Y r D
  cmpb  #0          ;compare a to 0
  bne   NOUPDATE    ;if COUNTER%SUBPERIOD==0 update the LEDS
  jsr   UPDATELED   ;jump to subroutine update LED
NOUPDATE:
  movb  #$40,TFLG1  ;Clear interrupt bit
  rti               ;Return


UPDATELED:
  ldaa  POSITION    ;A = position
  ldab  SPEED       ;b = speed
  cmpa  #32
  bne   NOSKIPALL     ;if position >= 32 don't update data
  jmp   SKIPALL
NOSKIPALL:

  cmpb  #4
  blt   LT4         ;if speed < 4
  cmpb  #5
  blt   LT5         ;if speed < 5
  cmpb  #6
  blt   LT6         ;if speed < 6
  cmpb  #7
  blt   LT7         ;if speed < 7
  cmpb  #8
  blt   LT8         ;if speed < 8
  cmpb  #9
  blt   LT9         ;if speed < 9
  cmpb  #10
  blt   LT10        ;if speed < 10
  cmpb  #11
  blt   LT11        ;if speed < 11
  cmpb  #12
  blt   LT12        ;if speed < 12
  cmpb  #13
  blt   LT13        ;if speed < 13
  cmpb  #14
  blt   LT14        ;if speed < 14
  cmpb  #15
  blt   LT15        ;if speed < 15
  cmpb  #16
  blt   LT16        ;if speed < 16
  cmpb  #17
  blt   LT17        ;if speed < 17
  cmpb  #18
  blt   LT18        ;if speed < 18
  cmpb  #19
  blt   LT19        ;if speed < 19
  cmpb  #20
```

```
   blt   LT20        ;if speed < 20
   cmpb  #21
   blt   LT21        ;if speed < 21
   cmpb  #22
   blt   LT22        ;if speed < 22
   bra   LTHI        ;else


LT4:
   ldx #DAT4
   bra UPDATEDONE
LT5:
   ldx #DAT5
   bra UPDATEDONE
LT6:
   ldx #DAT6
   bra UPDATEDONE
LT7:
   ldx #DAT7
   bra UPDATEDONE
LT8:
   ldx #DAT8
   bra UPDATEDONE
LT9:
   ldx #DAT9
   bra UPDATEDONE
LT10:
   ldx #DAT10
   bra UPDATEDONE
LT11:
   ldx #DAT11
   bra UPDATEDONE
LT12:
   ldx #DAT12
   bra UPDATEDONE
LT13:
   ldx #DAT13
   bra UPDATEDONE
LT14:
   ldx #DAT14
   bra UPDATEDONE
LT15:
   ldx #DAT15
   bra UPDATEDONE
LT16:
   ldx #DAT16
   bra UPDATEDONE
LT17:
   ldx #DAT17
   bra UPDATEDONE
LT18:
   ldx #DAT18
   bra UPDATEDONE
LT19:
   ldx #DAT19
   bra UPDATEDONE
LT20:
   ldx #DAT20
   bra UPDATEDONE
LT21:
   ldx #DAT21
   bra UPDATEDONE
LT22:
   ldx #DAT22
   bra UPDATEDONE
LTHI:
   ldx #DATHI
   bra UPDATEDONE

UPDATEDONE:
   ldab a,x
   stab PTAD
   inca              ;A++
   staa  POSITION    ;Store back incremented position
SKIPALL:
   rts
```

```
TOC4ISR:
  ldaa  #0          ;Load 0 into A
  staa  POSITION    ;Reset Position
  ldx   COUNTER     ;Load Counter
  stx   PERIOD      ;Store D as period



  ;Calculate speed
  ldd    #1847      ;Load X with 1847, since 1847/x_ticks = y_mph
  ldy    #0         ;Make sure y is clear
  ediv              ;Y:D/X => Y r D
  tfr    y,d
  stab   SPEED

  ;Divide to find interval
  ldd   COUNTER     ;Reload Counter
  ldx   #32         ;Load number of segments into X
  ldy   #0          ;Make sure Y is cleared
  ediv              ;Y:D/X => Y r D
  sty   SUBPERIOD   ;Store Y as SUBPERIOD


  ldd   #0          ;Load 0 into X
  std   COUNTER     ;Reset Counter
  movb  #$10,TFLG1  ;Relax the TC4 interrupt flag
  rti               ;Return




;*******************************************************************************
;* init_pll                                                                    *
;* author: J. Conway                                                           *
;* last modified: 9/26/07                                                      *
;* notes: Initializes the PLL using the top two arguments on the stack as the  *
;*        multiplier and divider                                               *
;* example: psh multiplier                                                     *
;*          psh divider                                                        *
;*          jsr init_pll                                                       *
;*          ins                                                                *
;*          ins                                                                *
;* WARNING: Temporarily suspends the PLL, will throw off the debugger          *
;*******************************************************************************
init_pll:
  psha      ; Push the A and B registers onto the stack, as we will use them
  pshb
  leas 4,SP       ; Jump over the data we just put on the stack
  pulb      ; B = divider
  pula      ; A = multiplier

  bclr CLKSEL,$80   ; Disconnect PLL from system
  bset PLLCTL,$40   ; Turn on PLL

  staa SYNR       ; set PLL multiplier
  stab REFDV      ; set PLL divider
  ;PLLCLK = OSSCLK*(SYNR+1)/(REFDV+1) = 16MHz * (2+1)/(1+1) = 24 Mhz
  nop
  nop                ; Allows time for CRGFLG to become valid
wt_PLL_LOCK:
  brclr CRGFLG,8,wt_PLL_LOCK  ; Wait for PLL to lock
  bset CLKSEL,$80             ; Connect PLL into system

  leas -6,SP      ; Go back to where we put a and b on the stack
  pulb
  pula     ; Restore registers A and B
  rts      ; Return


*************************************************************
*                 Interrupt Vectors                         *
```

```
;*************************************************************
  ORG   $FFFE
  DC.W  Entry           ; Reset Vector
  ORG $FFE2
  fdb     TOC6ISR     ;Make TC6 interrupt vector point to TC6 interrupt rtn
  ORG $FFE6
  fdb     TOC4ISR     ;Make TC4 interrupt vector point to TC4 interrupt rtn
  ORG $FFEE
```