Fall 2012: Bicycle Speedometer SCADA using the Basic Stamp 2

Typically, bicycle speedometers are live reference devices. The rider will check speed or distance as they travel. At the completion of the ride, they may also check their maximum and average speeds along with how long the trip was. Usually, the only long term indication of the trip is available in the odometer.

The bicycle speedometer uses a reed sensor that is triggered by the passing of a magnet. The reed sensor is in a NO (normally open) state and will close when the magnet is close. In operation, the magnet will pass by the reed sensor several times, creating a detectable pulse. This signal is sensed by the speedometer which measures the time between pulses then calculates speed. In addition, the speedometer also averages the measurement, sets the maximum measured value, and calculates distance which is stored.

The primary goal of this project is to us the Basic STAMP 2 to measure pulses from the reed sensor, calculate RPM, and store data into external EEPROM. Aside from the logic functions of the BS2, the entire system must be able to handle rough road travel in cold and wet conditions. The secondary goal of the project is to indicate memory use by means of a LCD, with values displayed numerically and graphically. As an additional feature, the BS2 system will have a RGB LED to indicate Standby, Run, and Finished states. The objectives for this project are defined as follows:

- Detect reed sensor changes when wheel motion occurs
- Measure pulse width of sensor state changes and calculate RPM
- Store RPM values into external EEPROM
- Enclose electronics inside a weather and shock proof case
- Display memory usage on LCD in numeric value and graphically
- Indicate if system is in Standby, Running, or Finished with a RGB LED

Project History

It should be noted that this project was drawn from previous attempts to log data of a bicycle speedometer. The overall objectives were similar, but it never accomplished its major goals. However, it was able to do the following:

- Detect reed sensor changes when wheel motion occurs
- Measure pulse width of sensor state changes and calculate RPM

The end result was a system that was tethered to a workstation via the USB cable, while data was displayed using the **DEBUG** command. At most it could give indication of wheel spin decay over time. Since it offered a head start, the code was modified some and used into the current project.

Operational Flow Chart

Figure 1-1 is a flow chart of the operational logic and control. The items displayed in blue indicate reused process from the earlier project, while the items in black indicate developed process for this project.

When the system is powered on all declarations and initializations are done. The Green LED will blink 16 times to indicate progress from power on to the Main Routine. The system will check if EEPROM space is available and blink the Red LED if it has been processed. Next the system will check if the wheel time value has been set, which results from wheel motion. By default, this value is zero and causes the system to enter in to the Start Meter Routine.

Figure 1-1.

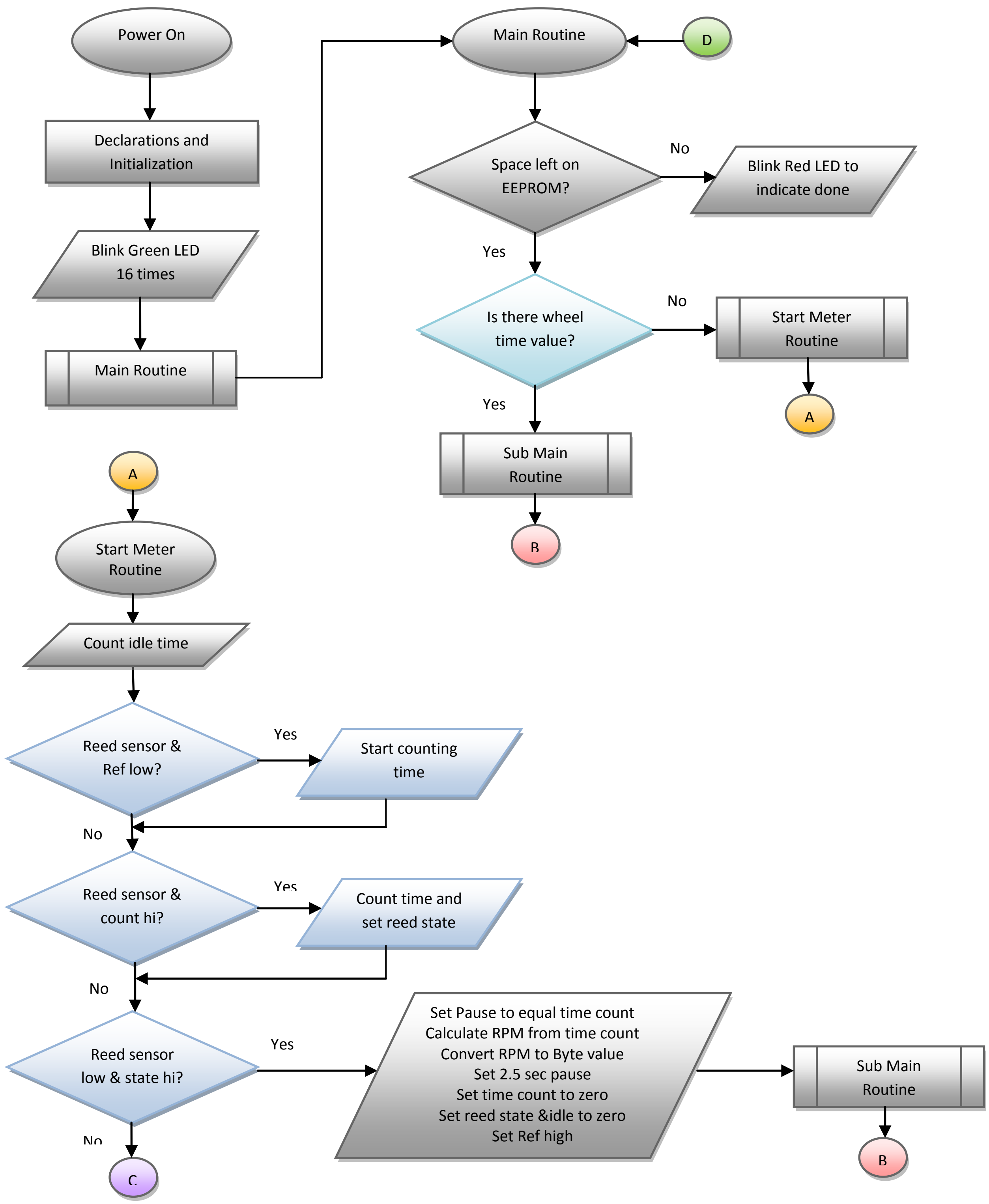
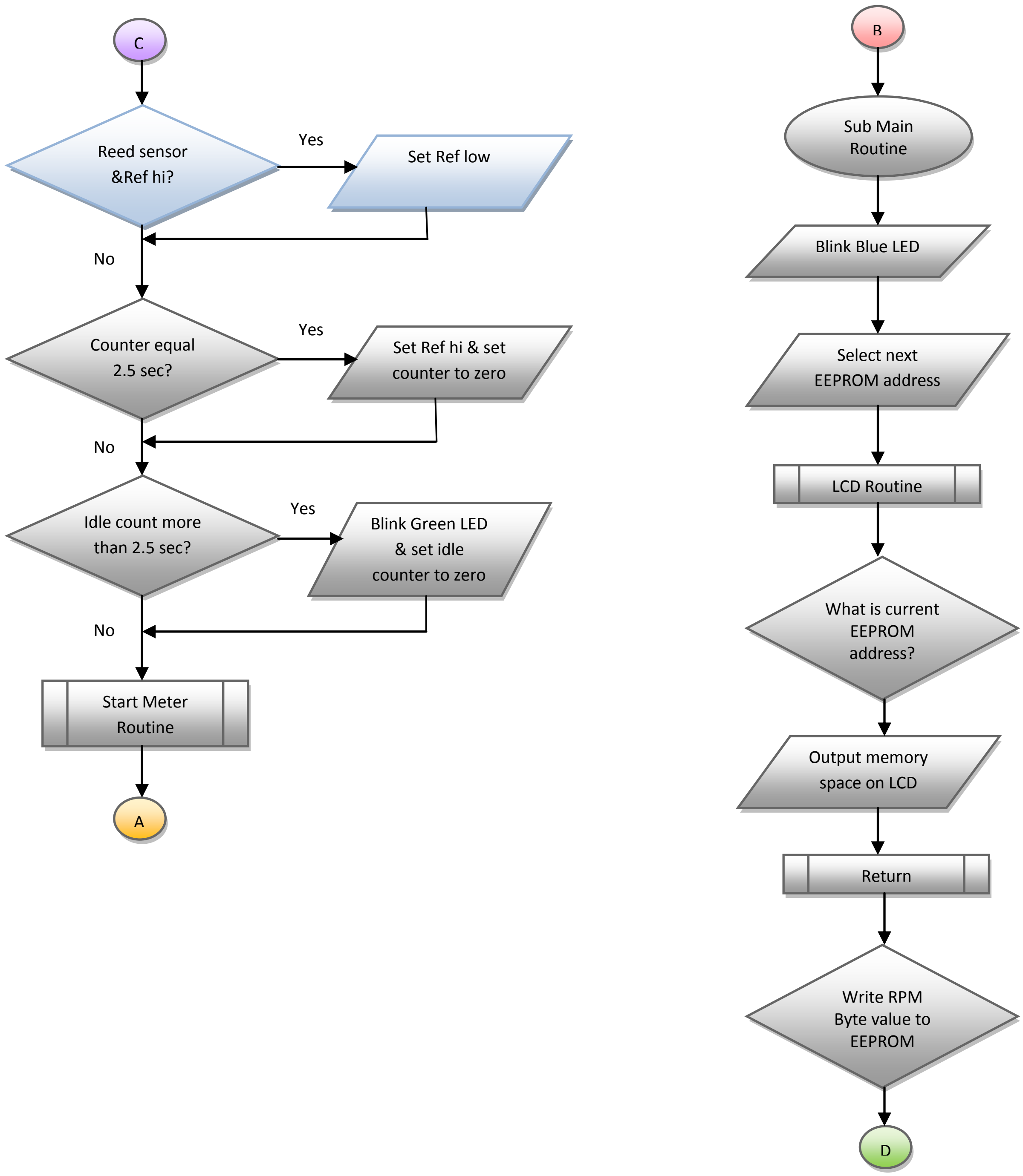

Power On
Declarations and Initialization
Blink Green LED 16 times
Main Routine
Main Routine
D
Space left on EEPROM?
No
Blink Red LED to indicate done
Yes
Is there wheel time value?
No
Start Meter Routine
A
Yes
Sub Main Routine
B
A
Start Meter Routine
Count idle time
Reed sensor & Ref low?
Yes
Start counting time
No
Reed sensor & count hi?
Yes
Count time and set reed state
No
Reed sensor low & state hi?
Yes
Set Pause to equal time count
Calculate RPM from time count
Convert RPM to Byte value
Set 2.5 sec pause
Set time count to zero
Set reed state &idle to zero
Set Ref high
Sub Main Routine
B
No
C

Figure 1-1 continued.



The Start Meter Routine will count values for idle time which is used to enter the system in standby. By default the Reed Reference is high and Time Counter value is zero. This causes the logic system to wait for the Reed Sensor to go high before setting the Reed Reference low. If the Reed Sensor does not go high in a 2.5 second period, the Green LED blinks and the Idle Counter is set back to zero. This process continues until the Reed Sensor goes high.

When the wheel moves, the Reed Sensor detects the passing magnet. This causes the Reed Sensor to go high, and the Reed Reference is set low. After a short period, the Reed Sensor goes low and this condition along with the Reed Reference being low causes the Time Counter to start.

If the wheel magnet passes by the Reed Sensor within 2.5 seconds then the Reed Reference is set high, otherwise the Time Counter is reset to zero and the Reed Reference is set high.

With a Reed Reference set high, a Time Counter value not zero, and the Reed Sensor back at low, the system takes the Time Counter value and subtracts it from the 2.5 second cycle to create a Pause Counter. The counters for Time and Pause are added to create a final pause cycle that is consistently 2.5 seconds. This is used to set the interval for EEPROM writes to 2.5 seconds.

RPM is calculated from the Time Counter and the Byte value is derived from the RPM value. The Reed State, Idle Counter, and Time Counter are set to zero and the Reed Reference is set high, in preparation for the next routine cycle. The Start Meter Routine now passes the RPM Byte value on to the Sub Main Routine.

The Sub Main Routine blinks the Blue LED and addresses the next EEPROM address block to write the RPM Byte value to. Before writing the value, the Sub Main Routine calls the LCD Routine to display memory usage.

The LCD Routine checks the value of the current EEPROM address and displays the numeric percentage of space left on EEPROM and a graphical bar graph based on the same value. It then returns to the Sub Main Routine.

The Sub Main Routine continues with the write operation to the EEPROM address block. Once the RPM Byte value has been entered, the Sub Main Routine returns to the Main Routine. The next available EEPROM address is selected and the process loops through until all EEPROM address space is used.

Memory Write Code

Below is the code used to detect, display, and write RPM values to EEPROM. This code was based in large part from the "24LC16B demo" written by Jon Williams with Parallax dated from August 18th, 2004.

```
' =========================================================================
'
'   File....... 24LC16_MemoryWrite_ReedSwitch_LCDIndicator_Ver2.BS2
'   Purpose.... 24LC16 data logger for bicycle reed sensor
'   Edited..... Patrick Gilfeather, OrangeLine Solutions
'   Original... Jon Williams, Parallax (24LC16B demo with a BS2/BS2e/BS2sx)
'   E-mail..... patrick@orangelinesolutions.com
'   Updated.... 20 NOV 2012
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
' =========================================================================


' -----[ Program Description ]---------------------------------------------
'
' This is a BS2 datalogger storing bicycle rpm on external EEPROM using a reed switch.
'
' Reed sensor activates logging function.
' The sensor pulse width is counted and stored in to a variable.
' This variable is converted into RPM.
' The value is finally recorded into external EEPROM in successive increments of 2.5 seconds.
' If the system looses power, logging starts at the begining of the memory address.
' If pulse width exceeds 2.5 seconds, the process is in a holding loop and no memory functions
occur.


' -----[ I/O Definitions ]-------------------------------------------------
```

```
SDA             PIN     1                       ' I2C serial data line IC Pin 5  `
SCL             PIN     0                       ' I2C serial clock line IC Pin 6
RSW             PIN     3                       ' Reed switch input pin
RedLED          PIN     8                       ' Red LED on Pin 8
GreenLED        PIN     9                       ' Green LED on Pin 9
BlueLED         PIN     10                      ' Blue LED on Pin 10
LcdPin          PIN     14                      ' LCD I/O pin


' -----[ Constants ]-------------------------------------------------------

Ack             CON     0                       ' acknowledge bit
Nak             CON     1                       ' no ack bit
TCon            CON     60000                   ' 60 second time contant for 1 minute
TSam            CON     2500                    ' 2.5 time sample rate
EE24LC16        CON     %1010 << 4

T9600           CON     84                      ' True, 8-bits, no parity, 9600
LcdCls          CON     12                      ' Form feed -> clear screen
LcdOn           CON     22                      ' Turns display on
BckSpc          CON     8                       ' Backspace Cursor


' -----[ Variables ]-------------------------------------------------------

slvAddr         VAR     Byte                    ' slave address
devNum          VAR     Nib                     ' device number (0 - 7)
addrLen         VAR     Nib                     ' 0, 1 or 2
devAddr         VAR     Word                    ' address in device

i2cData         VAR     Byte                    ' data to/from device
i2cWork         VAR     Byte                    ' work byte for TX routine
i2cAck          VAR     Bit                     ' Ack bit from device

outVal          VAR     Byte

TimeCounter          VAR     Word               ' Main pulse width counter
TimePause            VAR     Word               ' Pause variable based on pulse width which
finally equals 2.5 seconds
ByteCounter          VAR     Byte               ' The Byte value of the RPM conversion sent to
EEPROM
ReedPoint            VAR     Bit                ' Reed switch starting point
RefPoint             VAR     Bit                ' Reed switch reference point

LEDBlinker           VAR     Nib                ' LED Blinker counter
HoldStart            VAR     Word                ' LED Blinker counter



' -----[ EEPROM Data ]-----------------------------------------------------


' -----[ Initialization ]--------------------------------------------------


Check_Module:
  #IF ($STAMP >= BS2P) #THEN
    #ERROR "Use I2COUT and I2CIN!"
  #ENDIF

SEROUT LcdPin, T9600, [LcdOn, LcdCls]          ' Initialize LCD
PAUSE 500

SEROUT LcdPin, T9600, [248, ' Define Custom Character 0
               %00000, '
               %00000, '
               %00000, '
               %00000, '
               %00000, '
               %00000, '
               %00000, '
               %00000] '
SEROUT LcdPin, T9600, [249, ' Define Custom Character 1
               %11111, ' *****
               %11111, ' *****
               %11111, ' *****
               %11111, ' *****
               %11111, ' *****
               %11111, ' *****
               %11111, ' *****
               %11111] ' *****
```

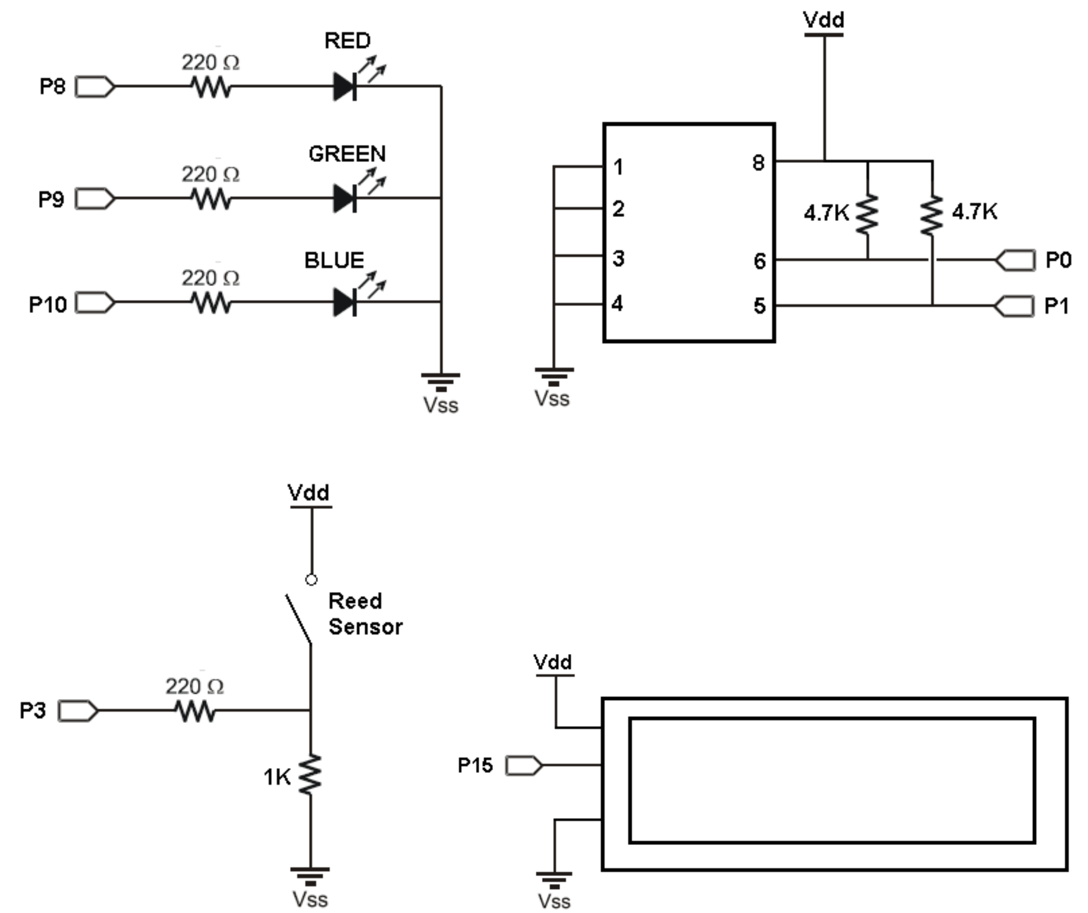
```
SEROUT LcdPin, T9600, ["Memory Used"]

Setup:
  addrLen = 1                                   ' one word address byte

TimeCounter = 0                                 ' Set variable values To default
ReedPoint = 0
RefPoint = 1

LEDBlinker = 0
HoldStart = 0

FOR HoldStart = 0 TO 3

  FOR LEDBlinker = 0 TO 3                                             ' Blink the Green LED
when system is waiting to write
    HIGH GreenLED
    PAUSE 1
    LOW GreenLED
    PAUSE 10
  NEXT

    PAUSE 2500
    LEDBlinker = 0

NEXT

HoldStart = 0

' -----[ Program Code ]----------------------------------------------------

Main:
  FOR devAddr = $000 TO $7FF                     ' Cycle through all EEPROM addresses


    IF (TimeCounter = 0) THEN
      GOSUB StartMeter
    ENDIF

SubMain:

  FOR LEDBlinker = 0 TO 3                        ' Blink the Blue LED when memory is written
    HIGH BlueLED
    PAUSE 1
    LOW BlueLED
    PAUSE 10
  NEXT
    LEDBlinker = 0

    slvAddr = EE24LC16 | (devAddr.BYTE1 << 1)
    GOSUB LCDDisplay
    outVal = ByteCounter
    i2cData = outVal
    GOSUB Write_Byte
    GOSUB I2C_Stop

  NEXT

  DO

  FOR LEDBlinker = 0 TO 3                        ' Blink the Red LED when all memory is full
    HIGH RedLED
    PAUSE 1
    LOW RedLED
    PAUSE 10
  NEXT
    PAUSE 1000
    LEDBlinker = 0

  LOOP


' -----[ Subroutines ]-----------------------------------------------------

StartMeter: ']----------------------------------------------------

HoldStart = HoldStart + 1
```

```
  IF (RSW = 0) AND (RefPoint = 0) THEN                                    ' Only count if Reference
Point has been cleared, isn't by defualt.
      TimeCounter = TimeCounter + 1
  ENDIF

  IF (RSW = 1) AND (RefPoint = 0) AND (TimeCounter <> 0) THEN             ' If count has occured
and Reference Point is clear then set ReedPoint.
      ReedPoint = 1
      TimeCounter = TimeCounter + 1
  ENDIF

  IF (RSW = 0) AND (ReedPoint = 1) AND (TimeCounter <> 0) THEN            ' Only if full pulse
width count has occured.
      TimePause = TimeCounter
      TimeCounter = TCon / TimeCounter                                    ' Calculate RPM
      ByteCounter = TimeCounter / 6                                       ' Calculate RPM to Byte
Value 0-255
      TimePause = TSam - TimePause                                        ' Set the pause time
based on pulse width so it equals 2.5 seconds
      PAUSE TimePause
      TimeCounter = 0
      ReedPoint = 0
      RefPoint = 1
      HoldStart = 0
      GOTO SubMain                                                        ' Set Reference Point so
next start occurs at Reed Switch drop.
  ENDIF

  IF (RSW = 1) AND (RefPoint = 1) THEN                                    ' Start on next Reed
Switch drop signal.
      RefPoint = 0
  ENDIF

  IF (TimeCounter > TSam) THEN                                            ' Prevent TimeCounter
Overflow by reseting time counter.
      RefPoint = 1
      TimeCounter = 0
  ENDIF

  IF (RefPoint = 1) AND (TimeCounter = 0) AND (HoldStart > TSam ) THEN
    FOR LEDBlinker = 0 TO 1                                               ' Blink the Green LED
when waiting for next sensor detection
      HIGH GreenLED
      PAUSE 1
      LOW GreenLED
      PAUSE 10
    NEXT
    LEDBlinker = 0
    HoldStart = 0
  ENDIF

  GOTO StartMeter

LCDDisplay: ']-----------------------------------------------------

  IF (devAddr > 0) AND (devAddr < 128) THEN
        SEROUT LcdPin, T9600, [140, "  0%"]
        SEROUT LcdPin, T9600, [148, 0]
  ELSEIF (devAddr > 127) AND (devAddr < 256) THEN
        SEROUT LcdPin, T9600, [140, "  6%"]
        SEROUT LcdPin, T9600, [148, 1]
  ELSEIF (devAddr > 255) AND (devAddr < 384) THEN
        SEROUT LcdPin, T9600, [140, " 13%"]
        SEROUT LcdPin, T9600, [148, 1, 1]
  ELSEIF (devAddr > 383) AND (devAddr < 512) THEN
        SEROUT LcdPin, T9600, [140, " 19%"]
        SEROUT LcdPin, T9600, [148, 1, 1, 1]
  ELSEIF (devAddr > 511) AND (devAddr < 640) THEN
        SEROUT LcdPin, T9600, [140, " 25%"]
        SEROUT LcdPin, T9600, [148, 1, 1, 1, 1]
  ELSEIF (devAddr > 639) AND (devAddr < 768) THEN
        SEROUT LcdPin, T9600, [140, " 32%"]
        SEROUT LcdPin, T9600, [148, 1, 1, 1, 1, 1]
  ELSEIF (devAddr > 767) AND (devAddr < 896) THEN
        SEROUT LcdPin, T9600, [140, " 38%"]
        SEROUT LcdPin, T9600, [148, 1, 1, 1, 1, 1, 1]
  ELSEIF (devAddr > 895) AND (devAddr < 1024) THEN
        SEROUT LcdPin, T9600, [140, " 44%"]
        SEROUT LcdPin, T9600, [148, 1, 1, 1, 1, 1, 1, 1]
  ELSEIF (devAddr > 1023) AND (devAddr < 1152) THEN
```

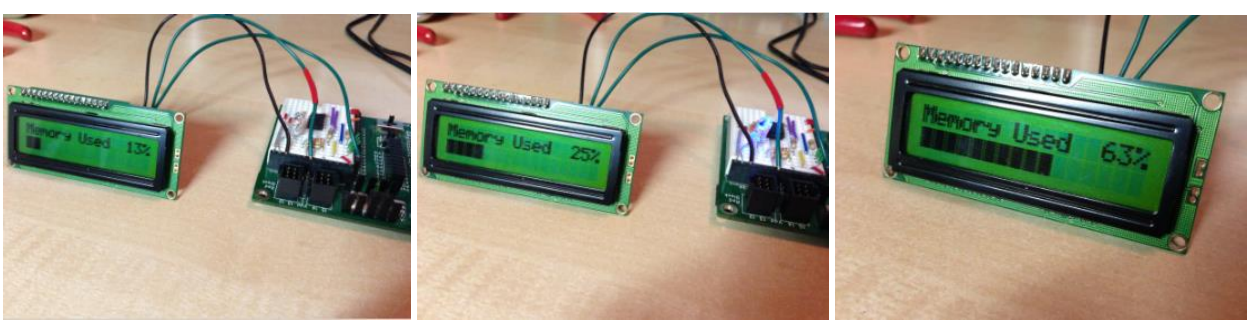
```
        SEROUT LcdPin, T9600, [140, " 50%"]
        SEROUT LcdPin, T9600, [148, 1, 1, 1, 1, 1, 1, 1, 1]
  ELSEIF (devAddr > 1151) AND (devAddr < 1280) THEN
        SEROUT LcdPin, T9600, [140, " 57%"]
        SEROUT LcdPin, T9600, [148, 1, 1, 1, 1, 1, 1, 1, 1, 1]
  ELSEIF (devAddr > 1279) AND (devAddr < 1408) THEN
        SEROUT LcdPin, T9600, [140, " 63%"]
        SEROUT LcdPin, T9600, [148, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
  ELSEIF (devAddr > 1407) AND (devAddr < 1536) THEN
        SEROUT LcdPin, T9600, [140, " 69%"]
        SEROUT LcdPin, T9600, [148, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
  ELSEIF (devAddr > 1535) AND (devAddr < 1664) THEN
        SEROUT LcdPin, T9600, [140, " 75%"]
        SEROUT LcdPin, T9600, [148, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
  ELSEIF (devAddr > 1663) AND (devAddr < 1792) THEN
        SEROUT LcdPin, T9600, [140, " 80%"]
        SEROUT LcdPin, T9600, [148, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
  ELSEIF (devAddr > 1791) AND (devAddr < 1920) THEN
        SEROUT LcdPin, T9600, [140, " 88%"]
        SEROUT LcdPin, T9600, [148, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
  ELSEIF (devAddr > 1919) AND (devAddr < 2047) THEN
        SEROUT LcdPin, T9600, [140, " 94%"]
        SEROUT LcdPin, T9600, [148, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
  ELSEIF (devAddr = 2047) THEN
        SEROUT LcdPin, T9600, [140, "100%"]
        SEROUT LcdPin, T9600, [148, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
  ENDIF


RETURN

' -----[ High Level I2C Subroutines]--------------------------------------

' Random location write
' -- pass device slave address in "slvAddr"
' -- pass address bytes (0, 1 or 2) in "addrLen"
' -- register address passed in "devAddr"
' -- data byte to be written is passed in "i2cData"

Write_Byte:
  GOSUB I2C_Start                               ' send Start
  i2cWork = slvAddr & %11111110                 ' send slave ID
  GOSUB I2C_TX_Byte
  IF (i2cAck = Nak) THEN Write_Byte             ' wait until not busy
  IF (addrLen > 0) THEN
    IF (addrLen = 2) THEN
      i2cWork = devAddr.BYTE1                   ' send word address (1)
      GOSUB I2C_TX_Byte
    ENDIF
    i2cWork = devAddr.BYTE0                     ' send word address (0)
    GOSUB I2C_TX_Byte
  ENDIF
  i2cWork = i2cData                             ' send data
  GOSUB I2C_TX_Byte
  GOSUB I2C_Stop
  RETURN


' Random location read
' -- pass device slave address in "slvAddr"
' -- pass address bytes (0, 1 or 2) in "addrLen"
' -- register address passed in "devAddr"
' -- data byte read is returned in "i2cData"

Read_Byte:
  GOSUB I2C_Start                               ' send Start
  IF (addrLen > 0) THEN
    i2cWork = slvAddr & %11111110               ' send slave ID (write)
    GOSUB I2C_TX_Byte
    IF (i2cAck = Nak) THEN Read_Byte            ' wait until not busy
    IF (addrLen = 2) THEN
      i2cWork = devAddr.BYTE1                   ' send word address (1)
      GOSUB I2C_TX_Byte
    ENDIF
    i2cWork = devAddr.BYTE0                     ' send word address (0)
    GOSUB I2C_TX_Byte
    GOSUB I2C_Start
  ENDIF
  i2cWork = slvAddr | %00000001                 ' send slave ID (read)
  GOSUB I2C_TX_Byte
  GOSUB I2C_RX_Byte_Nak
```

```
  GOSUB I2C_Stop
  i2cData = i2cWork
  RETURN


' -----[ Low Level I2C Subroutines]----------------------------------------

' *** Start Sequence ***

I2C_Start:                                      ' I2C start bit sequence
  INPUT SDA
  INPUT SCL
  LOW SDA

Clock_Hold:
  DO : LOOP UNTIL (SCL = 1)                     ' wait for clock release
  RETURN


' *** Transmit Byte ***

I2C_TX_Byte:
  SHIFTOUT SDA, SCL, MSBFIRST, [i2cWork\8]      ' send byte to device
  SHIFTIN SDA, SCL, MSBPRE, [i2cAck\1]          ' get acknowledge bit
  RETURN


' *** Receive Byte ***

I2C_RX_Byte_Nak:
  i2cAck = Nak                                  ' no Ack = high
  GOTO I2C_RX

I2C_RX_Byte:
  i2cAck = Ack                                  ' Ack = low

I2C_RX:
  SHIFTIN SDA, SCL, MSBPRE, [i2cWork\8]         ' get byte from device
  SHIFTOUT SDA, SCL, LSBFIRST, [i2cAck\1]       ' send ack or nak
  RETURN


' *** Stop Sequence ***

I2C_Stop:                                       ' I2C stop bit sequence
  LOW SDA
  INPUT SCL
  INPUT SDA
  RETURN
```

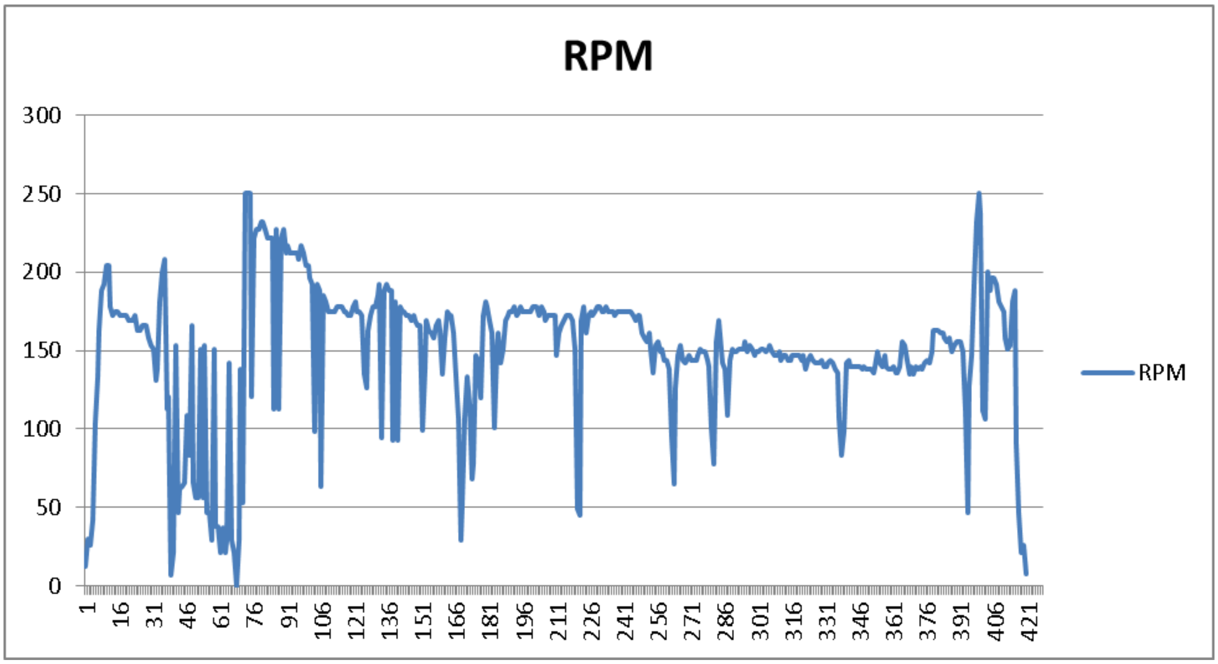
Hardware Schematics of Bicycle Speedometer SCADA



Bench Testing of Bicycle Speedometer SCADA



Final Installation and Data of Bicycle Speedometer SCADA



Summary and Conclusion

The project was successful in accomplishing the objectives defined. The data now can be process for further analysis outside of the system, which adds value to the project by offering a result far beyond its scope. Although the project achieved its objectives, there are some considerable discrepancies that should be pointed out.

The breadboard had a bad wiring connection for the reed sensor. The negative terminal of the reed sensor did not pass through the 1K ohm resistor as defined in the schematic, but was a short to ground. This caused the drain voltage to drop directly to ground, which could have damaged the board. This may explain some the RPM reading drops noticed in the output.

Also, the wiring to the LCD was not stable. Upon completion of the ride, the display was not readable. The ground pin had come loose from the connector which caused this to occur.

Aside from these pitfalls, the project was rewarding because it demonstrated the extended usefulness of the Basic Stamp. This was clearly evident from the expanded storage capacity, but also from the use of the I2C protocol.