Wireless Parking Meter—Final Project

Omer Peleg
Niraj Patel
Jay Patel
Dhairya Patel
Paul Solecki
5/8/04
Prof. Rose
ECE 426: Wireless Personal Communication System

Table of Contents

Introduction

In this project the main goal was to make the parking violation ticketing system more efficient. Conventionally, a parking attendant has to personally check the status of the parking meter and write up a violation ticket. Even then, not all the violators will ticketed because if the parking attendant does not get to an expired meter, the violator parked in the parking space will get away without paying the fine (free rider problem). The aim was to speed up the process of ticketing and ticket all the possible violators as well as keeping the overall cost of the system minimal. Hence, the idea of a wireless parking meter was born.

The first step was to come up with a feasible idea by which a wireless system could be implemented. After much brainstorming, it was decided to buy a digital parking meter. For the wireless aspect of the project the best course of action was to use MICA2 motes build by Crossbow Industries. The motes act as a transmitter and a receiver. Programming the mote was done using the TinyOS operating system developed by UC-Berkeley. The first task was to load TinyOS onto the PC and learning its commands. TinyOS is an operating system developed for the mote, which uses a Unix interface and uses a programming language based on C. Through a program loaded on to the motes with TinyOS, the motes would communicate with the parking meter and have the other mote (receiver) communicate with the laptop, which is to be used by the parking meter attendant to monitor the status of the meter.

To assist the parking attendant in locating the meter, we wrote a program, which embedded a map with the meter locations, displayed on the laptop. This was accomplished by the use of a GPS card. The status of the meter is always displayed on the laptop; if the meter is expired, the status of the meter changes to a red display. However, when the meter is not expired or is in use, then the display on the laptop is green.

Hardware Implementation

The first step in building the wireless parking meter is to get the Hardware logic off the meter. This is a very integral part, as it will be needed to identify the state of the meter. The meter is said to be “logic high” if there is time left in the meter, and said to be at “logic low” if there is no time left. This task was more difficult then expected because of the lack of help from Duncan (manufacturer of parking meter). We were successful in getting the logic off the meter but will first list-failed methods.

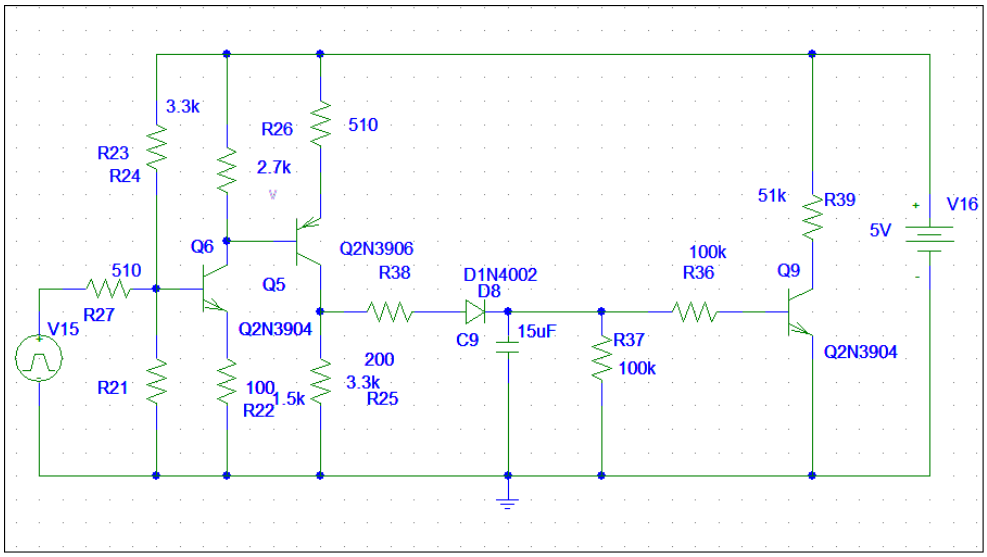
Failed methods of getting Hardware logic off meter:

- Probing the micro controller of the meter to check status, this method was not effective because voltages of all the pins were the same at both meter states.
- Use the integrated RS232 port on the meter. Unfortunately this method was also unsuccessful because of the meter has a proprietary port.
- Use the Infrared port on the meter. This method was also successful because the meter’s IR port is not IRDA compliant.

Successful methods of getting Hardware logic off the meter:

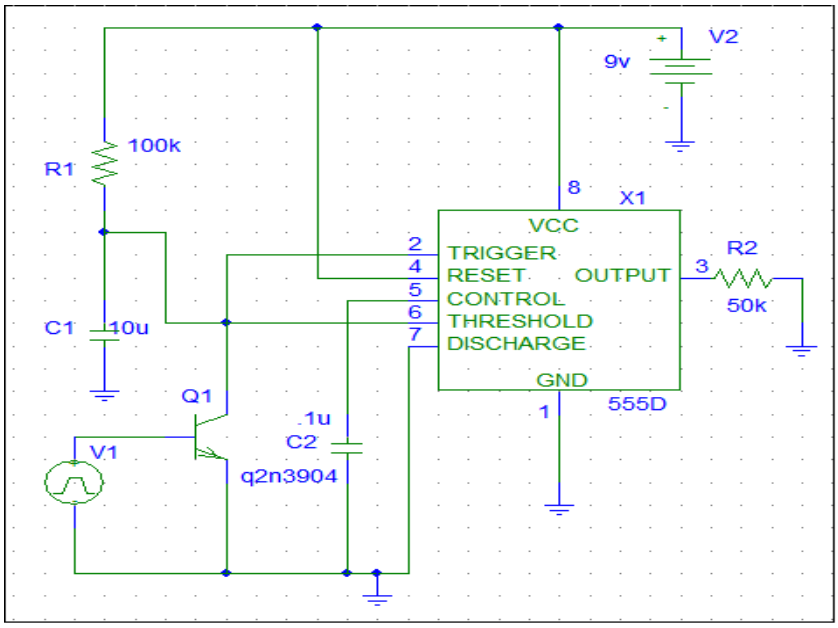
Finally with the failed attempts in getting the hardware logic off the meter, we decided we would exploit a flashing LED that starts only when the meter is expired. This LED presented us with an 800mV pulse with a duty cycle of just 2%.

Using knowledge of TTL, an amplifier limiter circuit was built



This circuit uses the meter's 800mV input, which is fed into a common emitter amplifier. This common emitter inverts and amplifies the signal, which is then fed into a common collector that inverts the signal again producing a 4V P-P signal. After this signal is amplified it is sent through a limiter circuit that keeps the transistor on as long as there is a pulse coming in resulting in a VoL at the collector. When the pulses stop (meter is at logic high) Q9 will be off and VoH is 5V. Although this method did successfully get logic off the meter, we found there could be an easier way that would use less power and be more efficient.

2) 555 Timer missing Pulse detector



This circuit would also use the meter's 800mV signal but it is much simpler and more efficient then the amp-limiter circuit. The 555 timer's pin 2 and 6 act as a comparator in which pin 6 is set to 6V. If the voltage at pin 2 were less then 6V, the output voltage (pin 3) would be at 9V. If pin 2 voltage is greater then 6V the output voltage is 0V. When there is no pulse (meter has money in it) the voltage at pin 2 is 9V and Q1 is off which leaves a VoL at 0V. Once the pulse starts the transistor will switch on every 2ms, draining the 10u capacitor through the transistor causing the voltage at pin 2 to go to 0 every 2ms, which in turn will change pin 3 to VoH.

*Circuit Schematics and output files are located in Appendix A

Wireless Implementation

Once the data is taken off the meter, it is now important to transmit this data. Initially the idea was to use 802.11 to send this data off the meter, but then quickly realized that 802.11 would be excessive for the amount of data we are sending, and inefficient because of power consumption. We decided to use mica2 motes, a low power, low bandwidth programmable transmitter/receiver used for sensor networks.

Mica 2 Mote

- Processor Speed: 4MHz (ATmega 128L)
- Processor has 128kbytes of flash memory to store the program
- ATmega consumes only 8 milliamps when it is running, and only 15 micro amps in sleep mode
- Low power consumption allows a MICA mote to run for more than a year with two AA batteries
- Transmits at 433Mhz (Radio Link)
- Software implementation for the mote is done through OS called Tiny OS

To begin installing applications on the mote, Cygwin, a Unix emulation for windows, and TinyOS had to be installed on the PC. Once these programs were installed on PC, a program can be installed on a mote via a command called *make mica2 install.* An application and mote ID can be installed on the mote via this command, for example:

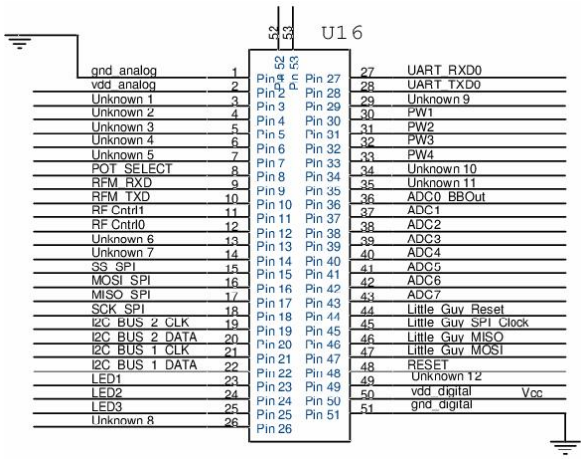
make mica2 install.4, installs an application and sets the mote ID to 4

make mica2 install.6 installs an application and sets the mote ID to 6

Once though Mote ID, now the focus was on reading meter status.

Because of time constraints we decided to use a program that came with TinyOS, called SenseToRfm. This program periodically samples the photo sensor and sends an ADC light reading in a packet over the radio. From online literature we found that SenseToRfm uses ADC channel 1 which corresponds to pin 37 on the

mote. With this information this pin and program could be exploited be used to our advantage when transmitting data. Once SenseToRfm was loaded on to the transmitter mote, we installed TOSbase to the receiver mote that sits on the programming board. TOSbase is a program that receives radio packets from all motes in



Mote Pin Configuration

the following hex data stream comes in through the serial port:

7E 42 FF FF 04 7D 5D 03 **03 01** 00 60 63 7E

7E 42 FF FF 04 7D 5D 03 **03 01** 00 60 63 7E

7E 42 FF FF 04 7D 5D 03 **03 01** 00 60 63 7E

7E 42 FF FF 04 7D 5D 03 **03 01** 00 60 63 7E

Repeating…

In this message basically only two words need to be interpreted. The two words in bold represent meter ID, and the reading from the ADC channel. From this data stream it can be interpreted that the meter ID is **1** and ADC reading is 3. Because the ADC pin reads for a voltage, we connected our 555-timer circuit to a relay that will be open (VoL), and leave the ADC pin with an open circuit, or close and ground the ADC pin (VoH). When the ADC pin is grounded the following hex data stream comes in through the serial port:

7E 42 FF FF 04 7D 5D 03 **00 01** 00 60 63 7E

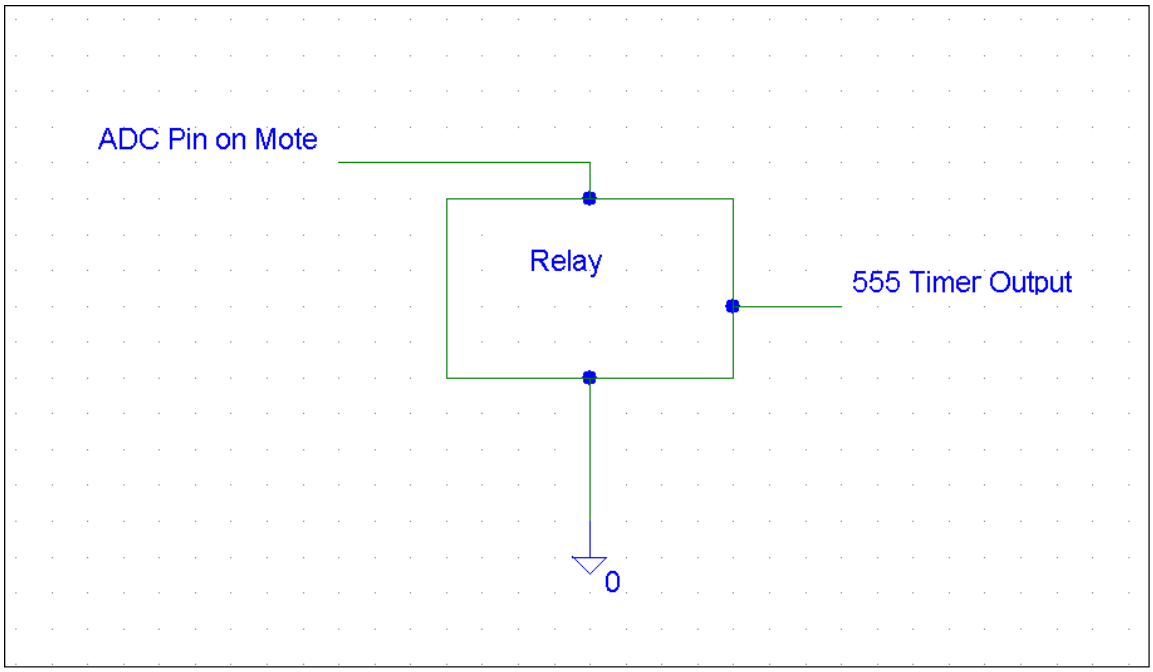
7E 42 FF FF 04 7D 5D 03 **00 01** 00 60 63 7E

7E 42 FF FF 04 7D 5D 03 **00 01** 00 60 63 7E

7E 42 FF FF 04 7D 5D 03 **00 01** 00 60 63 7E

Repeating…

The grounded ADC pin outputs a 0, corresponding to logic 0 on the meter. When meter logic is high, the ADC pin will output a 3. This relay circuit connects the meter to the mote.



*Appendix B includes code to applications installed on motes.

Software Implementation

The user end of the Wireless Parking Meter involves a software application running on a laptop, tablet PC, or PDA, optionally mounted in the parking attendant's vehicle. The application is a GPS Navigation system that includes parking meters and their status. Options such as a table of all parking meters in the system and parking meter history will also be available. The interface is user friendly and visually simple, so the parking meter can focus on driving as well.

The software application interfaces with two modules. The first is the base receiver connected to the serial communications port. The base receiver polls any incoming signals from nearby parking meters, and sends packets to the laptop. The application reads the serial port and updates the status for that meter. The communication is at 57600 baud, more than enough for the speed the system requires.

The second module interfaced by the application is the GPS. A PCMCIA or Compact Flash GPS card is connected to the computer. GPS uses serial communication as well (Com5 on this system). The global coordinates are used to display the parking meter attendant's position on the city map, as well as surrounding parking meters. The code for the application is in VB.NET, and is available at the end of this report.

Manufacturability & Cost Analysis

Wireless Parking Meter Cost Analysis		
Parts	Quantity	Cost
Duncan Eagle 2000 Parking meter	1	$180
Crossbow Mote Programming Board	1	$300
Crossbow Mica2 Motes	1	$200 each
Laptop Computer/ PDA	1	$500
GPS PCMCIA Card	1	$100
	Total Cost	$1280
	Total Cost of one Meter	$380

To implement this system in practice, a manufacturability analysis is in order. As evident from the table above, building this system will be fairly expensive on an individual level. However, buying parts in bulk can lower cost. It is also important to state that the potential buyers for this system are most likely large municipal governments; hence, the budget is large and the system can be marketed. Switching to this new system might be expensive for some cities but the revenue will also increase. Why? The city will have to hire less parking attendants because each expired meter can be attended to faster. The free rider problem is resolved also because as soon as the meter has expired the attendant will know instantaneously and will reach the meter in time to ticket the violator.

Difficulties

After we started working on this project, we made many strides but at same time we ran into problems. One major problem was finding out the logic from the parking meter. It was very important for our project because we needed to know the state in which the meter expires or is in service. We had to build another relay circuit in order to find out the logic of the meter. Another problem we faced was that we had a difficult time figuring out the mote ID and getting the motes talking to each other. Once a mote was connected to the laptop, the mote would send data in which the mote ID, group address, source address etc. were listed. The interpretation for that code was important because once both motes were talking to each other it was easier to recognize the data, which was being sent back and forth.

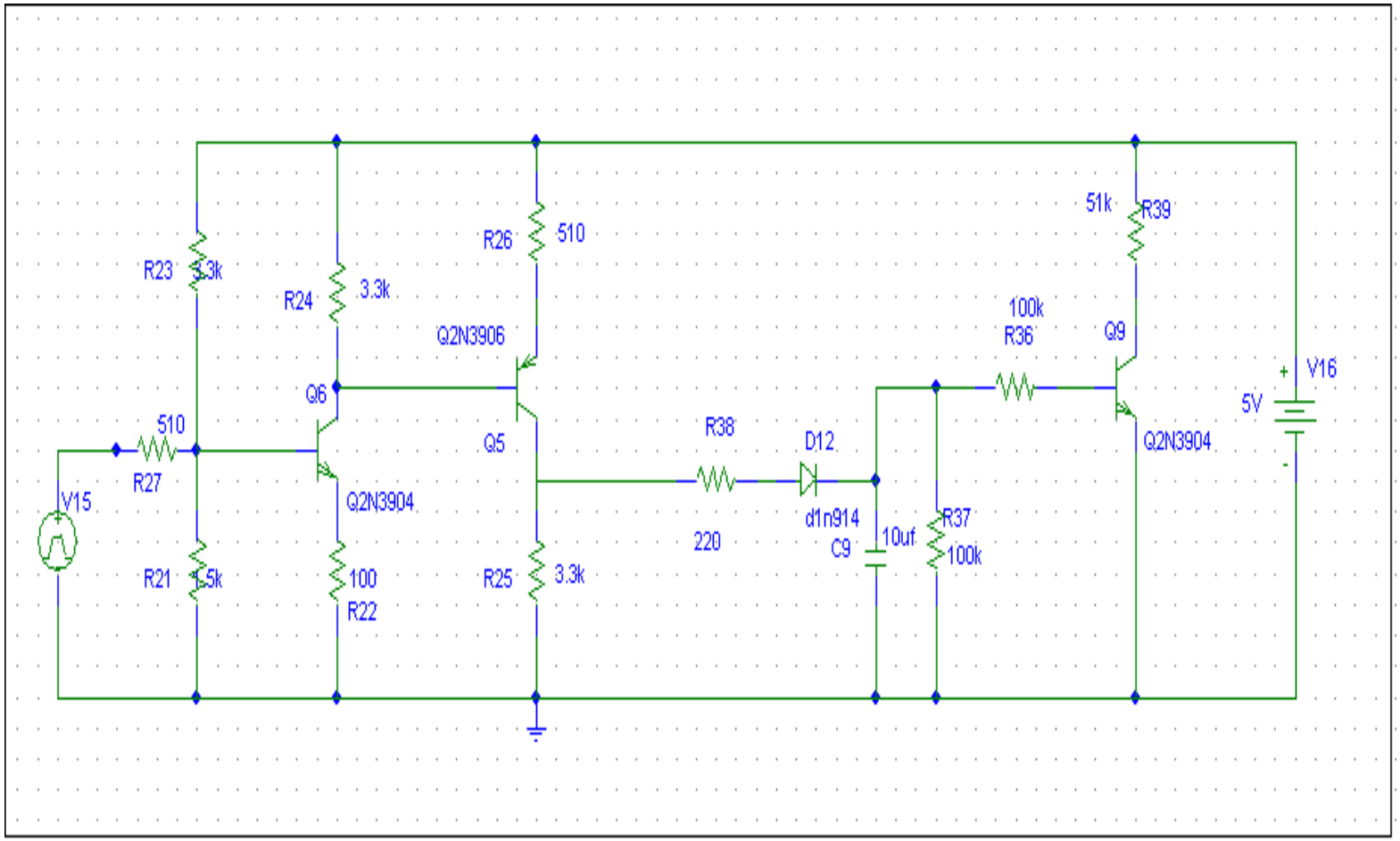
Conclusion

Working on this project was a great learning experience. The objective was to make ticketing more efficient so that violators would not get away without paying the fine. Majority of objective of the project was accomplished although, there were still things in the project that were not able to do due to time constraint and because the equipment needed for this project arrived late. The system was also supposed to locate the meter using the GPS system but GPS implementation was not done. The order of the GPS card came late therefore; work on this aspect of the project was not finished. With the help of the GPS card, the coordinates were obtained but interpreting the coordinates to the code was difficult. There is still a lot of work that can be done to make this system much efficient and much better.

We would like to Thank Prof. Rose for giving us an opportunity to work on a project of this magnitude. He provided us with all the resources we needed as well as the guidance when we had trouble moving forward. This project gave us real life experience because many of us will be working projects of this magnitude or much higher once we get into the professional environment.

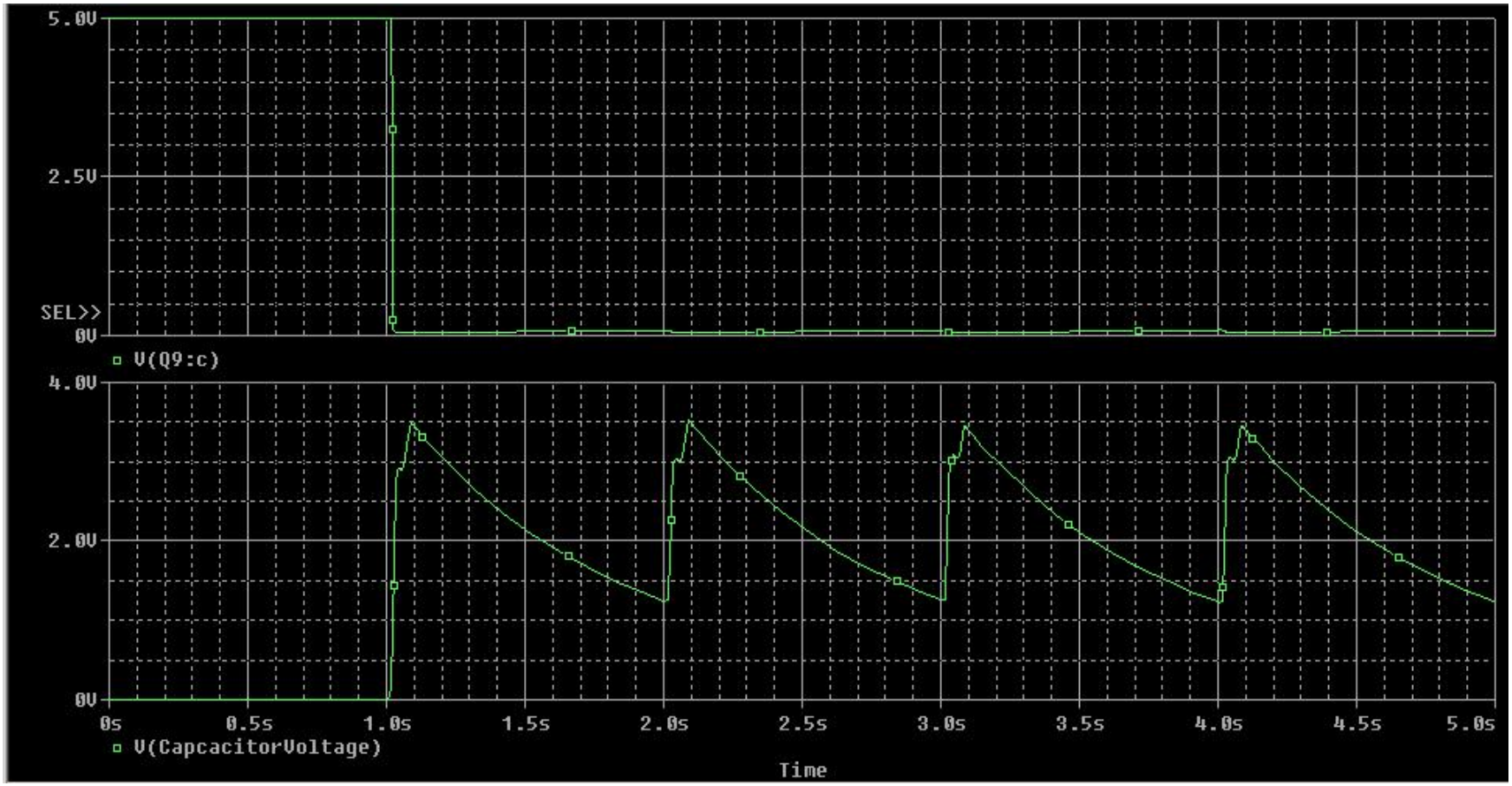
Appendix A

Amp-Limiter Circuit

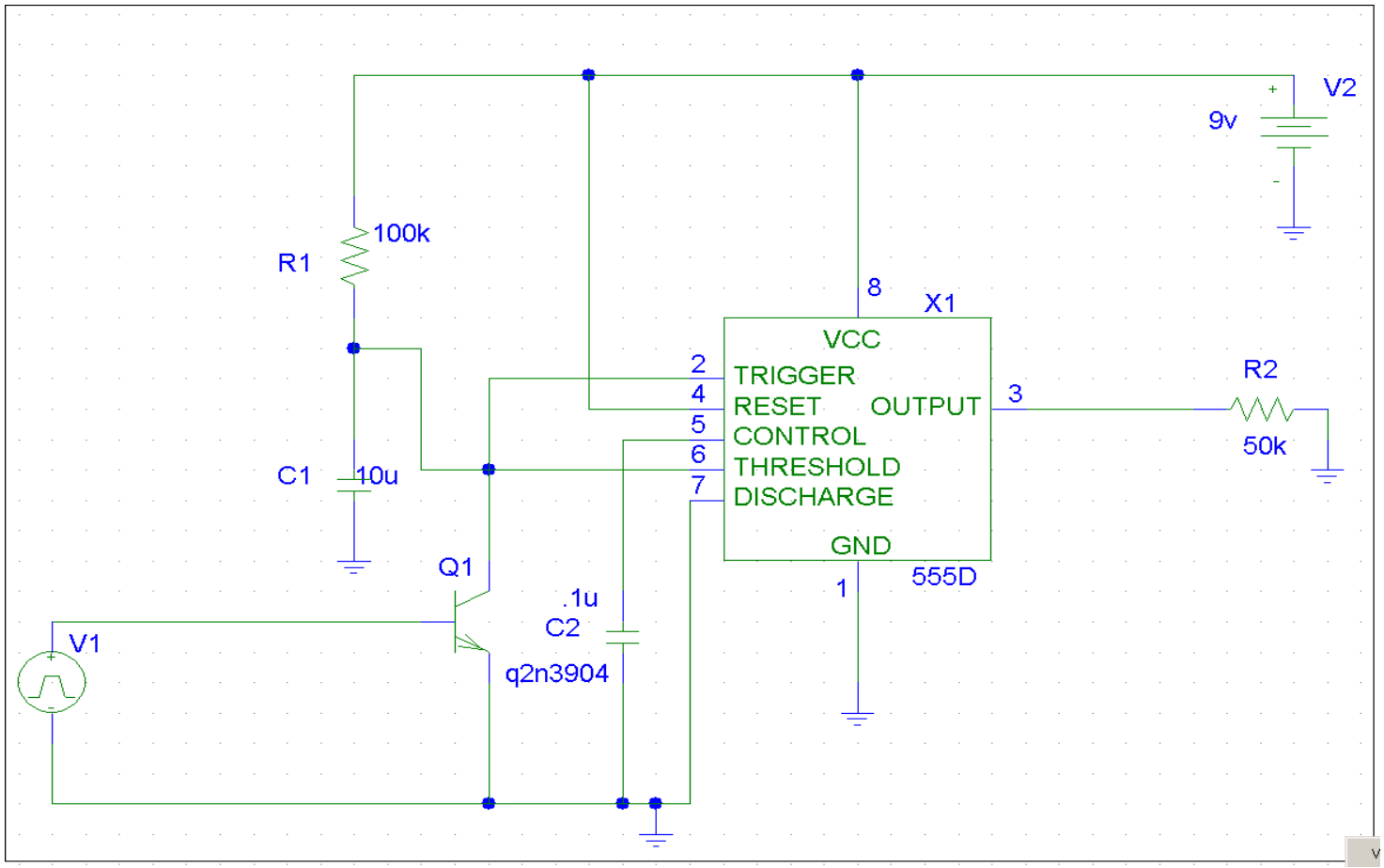


Amp-Limiter Circuit's Graphs

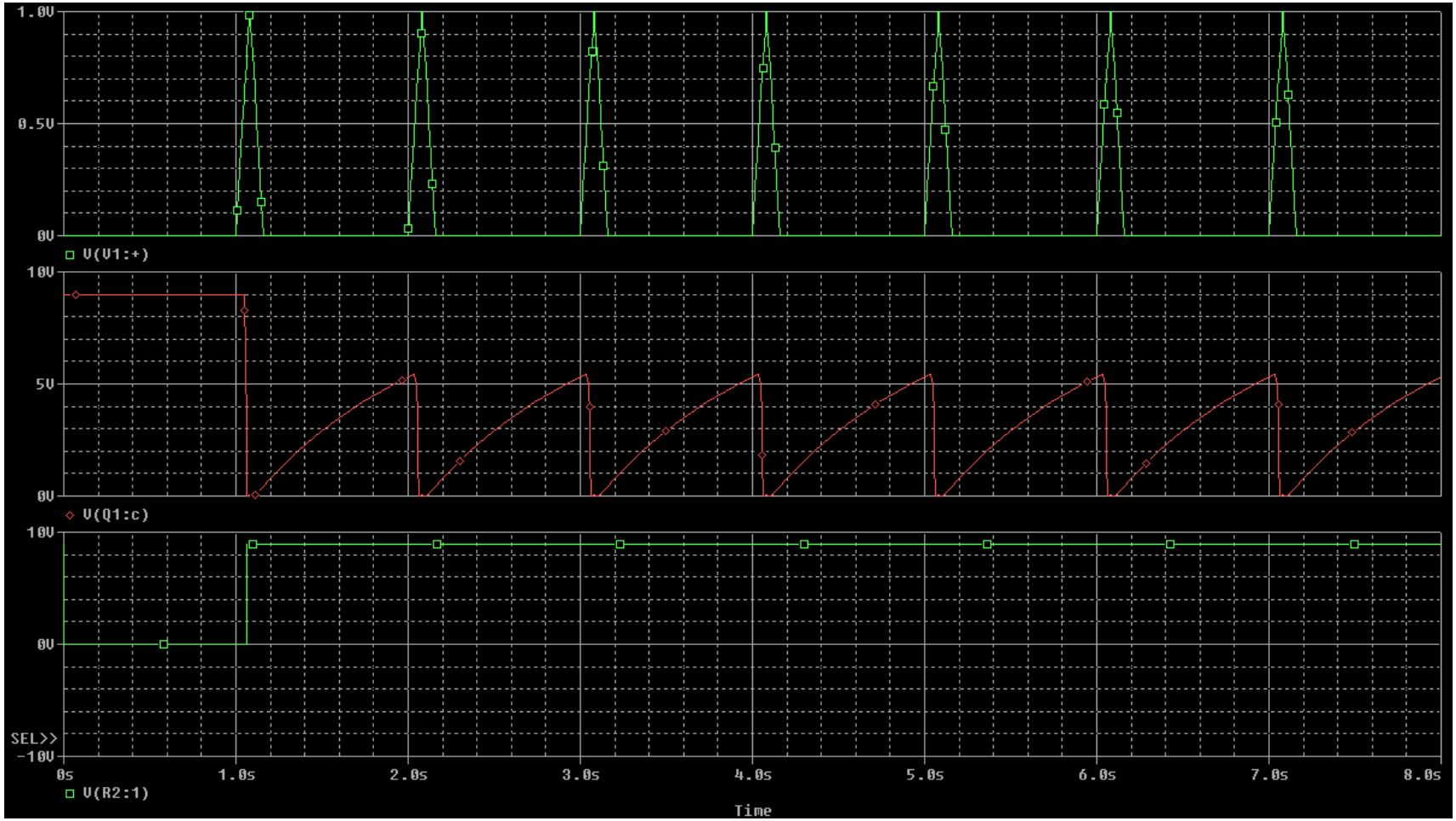
Amp-Limiter Circuit's Graphs



555 Timer Circuit



555 Timer Graphs



Appendix B (Mote Applications)

TOSBase

```
configuration TOSBase {
}
implementation {
  components Main, TOSBaseM, RadioCRCPacket as Comm, FramerM, UART, LedsC;

  Main.StdControl -> TOSBaseM;

  TOSBaseM.UARTControl -> FramerM;
  TOSBaseM.UARTSend -> FramerM;
  TOSBaseM.UARTReceive -> FramerM;
  TOSBaseM.UARTTokenReceive -> FramerM;
  TOSBaseM.RadioControl -> Comm;
  TOSBaseM.RadioSend -> Comm;
  TOSBaseM.RadioReceive -> Comm;

  TOSBaseM.Leds -> LedsC;

  FramerM.ByteControl -> UART;
  FramerM.ByteComm -> UART;
}
```

TOSBaseM

```
module TOSBaseM {
  provides interface StdControl;
  uses {
    interface StdControl as UARTControl;
    interface BareSendMsg as UARTSend;
    interface ReceiveMsg as UARTReceive;
    interface TokenReceiveMsg as UARTTokenReceive;

    interface StdControl as RadioControl;
    interface BareSendMsg as RadioSend;
    interface ReceiveMsg as RadioReceive;

    interface Leds;
  }
}
implementation
{
  enum {
    QUEUE_SIZE = 5
  };

  enum {
    TXFLAG_BUSY = 0x1,
    TXFLAG_TOKEN = 0x2
  };


  TOS_Msg gRxBufPool[QUEUE_SIZE];
  TOS_MsgPtr gRxBufPoolTbl[QUEUE_SIZE];
  uint8_t gRxHeadIndex,gRxTailIndex;

  TOS_Msg    gTxBuf;
  TOS_MsgPtr gpTxMsg;
  uint8_t    gTxPendingToken;
  uint8_t    gfTxFlags;

  task void RadioRcvdTask() {
    TOS_MsgPtr pMsg;
    result_t   Result;

    dbg (DBG_USR1, "TOSBase forwarding Radio packet to UART\n");
    atomic {
      pMsg = gRxBufPoolTbl[gRxTailIndex];
```

```
    gRxTailIndex++; gRxTailIndex %= QUEUE_SIZE;
  }
  Result = call UARTSend.send(pMsg);
  if (Result != SUCCESS) {
    pMsg->length = 0;
  }
  else {
    call Leds.greenToggle();
  }
}

task void UARTRcvdTask() {
  result_t Result;

  dbg (DBG_USR1, "TOSBase forwarding UART packet to Radio\n");
  gpTxMsg->group = TOS_AM_GROUP;
  Result = call RadioSend.send(gpTxMsg);

  if (Result != SUCCESS) {
    atomic gfTxFlags = 0;
  }
  else {
    call Leds.redToggle();
  }
}

task void SendAckTask() {
   call UARTTokenReceive.ReflectToken(gTxPendingToken);
   call Leds.yellowToggle();
   atomic {
     gpTxMsg->length = 0;
     gfTxFlags = 0;
   }
}

command result_t StdControl.init() {
  result_t ok1, ok2, ok3;
  uint8_t i;

  for (i = 0; i < QUEUE_SIZE; i++) {
    gRxBufPool[i].length = 0;
    gRxBufPoolTbl[i] = &gRxBufPool[i];
  }
  gRxHeadIndex = 0;
  gRxTailIndex = 0;
```

```
  gTxBuf.length = 0;
  gpTxMsg = &gTxBuf;
  gfTxFlags = 0;

  ok1 = call UARTControl.init();
  ok2 = call RadioControl.init();
  ok3 = call Leds.init();

  dbg(DBG_BOOT, "TOSBase initialized\n");

  return rcombine3(ok1, ok2, ok3);
}

command result_t StdControl.start() {
  result_t ok1, ok2;

  ok1 = call UARTControl.start();
  ok2 = call RadioControl.start();

  return rcombine(ok1, ok2);
}

command result_t StdControl.stop() {
  result_t ok1, ok2;

  ok1 = call UARTControl.stop();
  ok2 = call RadioControl.stop();

  return rcombine(ok1, ok2);
}

event TOS_MsgPtr RadioReceive.receive(TOS_MsgPtr Msg) {
  TOS_MsgPtr pBuf;

  dbg(DBG_USR1, "TOSBase received radio packet.\n");

  if (Msg->crc) {

    /* Filter out messages by group id */
    if (Msg->group != TOS_AM_GROUP)
      return Msg;

    atomic {
        pBuf = gRxBufPoolTbl[gRxHeadIndex];
        if (pBuf->length == 0) {
          gRxBufPoolTbl[gRxHeadIndex] = Msg;
```

```
          gRxHeadIndex++; gRxHeadIndex %= QUEUE_SIZE;
        }
        else {
          pBuf = NULL;
        }
    }

    if (pBuf) {
        post RadioRcvdTask();
    }
    else {
        pBuf = Msg;
    }
  }
  else {
    pBuf = Msg;
  }

  return pBuf;
}

event TOS_MsgPtr UARTReceive.receive(TOS_MsgPtr Msg) {
  TOS_MsgPtr  pBuf;

  dbg(DBG_USR1, "TOSBase received UART packet.\n");

  atomic {
    if (gfTxFlags & TXFLAG_BUSY) {
      pBuf = NULL;
    }
    else {
      pBuf = gpTxMsg;
      gfTxFlags |= (TXFLAG_BUSY);
      gpTxMsg = Msg;
    }
  }

  if (pBuf == NULL) {
    pBuf = Msg;
  }
  else {
    post UARTRcvdTask();
  }

  return pBuf;
```

```
}

event TOS_MsgPtr UARTTokenReceive.receive(TOS_MsgPtr Msg, uint8_t Token) {
  TOS_MsgPtr  pBuf;

  dbg(DBG_USR1, "TOSBase received UART token packet.\n");

  atomic {
    if (gfTxFlags & TXFLAG_BUSY) {
      pBuf = NULL;
    }
    else {
      pBuf = gpTxMsg;
      gfTxFlags |= (TXFLAG_BUSY | TXFLAG_TOKEN);
      gpTxMsg = Msg;
      gTxPendingToken = Token;
    }
  }

  if (pBuf == NULL) {
    pBuf = Msg;
  }
  else {

    post UARTRcvdTask();
  }

  return pBuf;
}

event result_t UARTSend.sendDone(TOS_MsgPtr Msg, result_t success) {
  Msg->length = 0;
  return SUCCESS;
}

event result_t RadioSend.sendDone(TOS_MsgPtr Msg, result_t success) {


  if ((gfTxFlags & TXFLAG_TOKEN)) {
    if (success == SUCCESS) {

      post SendAckTask();
    }
  }
  else {
    atomic {
```

```
        gpTxMsg->length = 0;
        gfTxFlags = 0;
      }
    }
    return SUCCESS;
  }
}
```

Appendix C (Laptop Software)

```
Public Class Form1
    Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

    Public Sub New()
        MyBase.New()

        'This call is required by the Windows Form Designer.
        InitializeComponent()

        'Add any initialization after the InitializeComponent() call

    End Sub

    'Form overrides dispose to clean up the component list.
    Protected Overloads Overrides Sub Dispose(ByVal disposing As
Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form
Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    Friend WithEvents box1 As System.Windows.Forms.Label
    Friend WithEvents box2 As System.Windows.Forms.Label
    Friend WithEvents Timer1 As System.Windows.Forms.Timer
    Friend WithEvents PictureBox2 As System.Windows.Forms.PictureBox
    Friend WithEvents Label1 As System.Windows.Forms.Label
    Friend WithEvents Label2 As System.Windows.Forms.Label
    Friend WithEvents Label3 As System.Windows.Forms.Label
    Friend WithEvents Label4 As System.Windows.Forms.Label
    Friend WithEvents Label5 As System.Windows.Forms.Label
    Friend WithEvents Label6 As System.Windows.Forms.Label
    Friend WithEvents Label7 As System.Windows.Forms.Label
    Friend WithEvents Label8 As System.Windows.Forms.Label

    <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
        Me.components = New System.ComponentModel.Container
        Dim resources As System.Resources.ResourceManager = New
System.Resources.ResourceManager(GetType(Form1))
        Me.box1 = New System.Windows.Forms.Label
        Me.box2 = New System.Windows.Forms.Label
        Me.Timer1 = New System.Windows.Forms.Timer(Me.components)
```

```
        Me.PictureBox2 = New System.Windows.Forms.PictureBox
        Me.Label1 = New System.Windows.Forms.Label
        Me.Label2 = New System.Windows.Forms.Label
        Me.Label3 = New System.Windows.Forms.Label
        Me.Label4 = New System.Windows.Forms.Label
        Me.Label5 = New System.Windows.Forms.Label
        Me.Label6 = New System.Windows.Forms.Label
        Me.Label7 = New System.Windows.Forms.Label
        Me.Label8 = New System.Windows.Forms.Label
        Me.SuspendLayout()
        '
        'box1
        '
        Me.box1.BackColor = System.Drawing.Color.Lime
        Me.box1.Location = New System.Drawing.Point(356, 160)
        Me.box1.Name = "box1"
        Me.box1.Size = New System.Drawing.Size(32, 32)
        Me.box1.TabIndex = 29
        '
        'box2
        '
        Me.box2.BackColor = System.Drawing.Color.Lime
        Me.box2.Location = New System.Drawing.Point(432, 160)
        Me.box2.Name = "box2"
        Me.box2.Size = New System.Drawing.Size(32, 32)
        Me.box2.TabIndex = 30
        '
        'Timer1
        '
        '
        'PictureBox2
        '
        Me.PictureBox2.Image =
CType(resources.GetObject("PictureBox2.Image"), System.Drawing.Image)
        Me.PictureBox2.Location = New System.Drawing.Point(16, 4)
        Me.PictureBox2.Name = "PictureBox2"
        Me.PictureBox2.Size = New System.Drawing.Size(676, 548)
        Me.PictureBox2.SizeMode =
System.Windows.Forms.PictureBoxSizeMode.StretchImage
        Me.PictureBox2.TabIndex = 32
        Me.PictureBox2.TabStop = False
        '
        'Label1
        '
        Me.Label1.BackColor = System.Drawing.Color.Lime
        Me.Label1.Location = New System.Drawing.Point(208, 380)
        Me.Label1.Name = "Label1"
        Me.Label1.Size = New System.Drawing.Size(16, 16)
        Me.Label1.TabIndex = 33
        '
        'Label2
        '
        Me.Label2.BackColor = System.Drawing.Color.Lime
        Me.Label2.Location = New System.Drawing.Point(124, 220)
        Me.Label2.Name = "Label2"
        Me.Label2.Size = New System.Drawing.Size(16, 16)
        Me.Label2.TabIndex = 34
```

```
'
'Label3
'
Me.Label3.BackColor = System.Drawing.Color.Lime
Me.Label3.Location = New System.Drawing.Point(208, 408)
Me.Label3.Name = "Label3"
Me.Label3.Size = New System.Drawing.Size(16, 16)
Me.Label3.TabIndex = 35
'
'Label4
'
Me.Label4.BackColor = System.Drawing.Color.Lime
Me.Label4.Location = New System.Drawing.Point(364, 372)
Me.Label4.Name = "Label4"
Me.Label4.Size = New System.Drawing.Size(16, 16)
Me.Label4.TabIndex = 36
'
'Label5
'
Me.Label5.BackColor = System.Drawing.Color.Lime
Me.Label5.Location = New System.Drawing.Point(148, 220)
Me.Label5.Name = "Label5"
Me.Label5.Size = New System.Drawing.Size(16, 16)
Me.Label5.TabIndex = 37
'
'Label6
'
Me.Label6.BackColor = System.Drawing.Color.Lime
Me.Label6.Location = New System.Drawing.Point(364, 396)
Me.Label6.Name = "Label6"
Me.Label6.Size = New System.Drawing.Size(16, 16)
Me.Label6.TabIndex = 38
'
'Label7
'
Me.Label7.BackColor = System.Drawing.Color.Lime
Me.Label7.Location = New System.Drawing.Point(208, 360)
Me.Label7.Name = "Label7"
Me.Label7.Size = New System.Drawing.Size(16, 16)
Me.Label7.TabIndex = 39
'
'Label8
'
Me.Label8.BackColor = System.Drawing.Color.Lime
Me.Label8.Location = New System.Drawing.Point(172, 220)
Me.Label8.Name = "Label8"
Me.Label8.Size = New System.Drawing.Size(16, 16)
Me.Label8.TabIndex = 40
'
'Form1
'
Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
Me.ClientSize = New System.Drawing.Size(708, 569)
Me.Controls.Add(Me.Label8)
Me.Controls.Add(Me.Label7)
Me.Controls.Add(Me.Label6)
Me.Controls.Add(Me.Label5)
```

```
        Me.Controls.Add(Me.Label4)
        Me.Controls.Add(Me.Label3)
        Me.Controls.Add(Me.Label2)
        Me.Controls.Add(Me.Label1)
        Me.Controls.Add(Me.box1)
        Me.Controls.Add(Me.box2)
        Me.Controls.Add(Me.PictureBox2)
        Me.Name = "Form1"
        Me.Text = "h"
        Me.ResumeLayout(False)

    End Sub

#End Region

    Dim WithEvents objPort As SerialNET.Port
    Dim str As String
    Dim message As Byte()
    Dim i As Integer = 0

    Structure Meter
        Dim status As String
        Dim node As Integer
        Dim x As Integer
        Dim y As Integer
        Dim Timeout As Integer
    End Structure
    Dim Meters() As Meter


    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        ' You can get a valid developer key at
        ' franson.biz/serialtools/
        ' That key will be valid for 14 days. Just cut and paste that
key into the statement below.
        ' To get a key that do not expire you need to purchase a
license
        Dim license As New SerialNET.License
        license.LicenseKey = "o8h1h84qtTkkTIKiv2Cnnxjdn6yefj6YoCi6"

        objPort = New SerialNET.Port
        Timer1.Interval = 500
        Timer1.Enabled = True

        ReDim Meters(4)

        For i = 0 To Meters.GetUpperBound(0)
            Meters(i).node = i

        Next

        box1.BackColor = System.Drawing.Color.Green
        box1.Text = ""

        Try
            objPort.ComPort = 1
```

```
            objPort.BaudRate = 57600

            ' objPort.Timeout = 5000
            objPort.StartTrigger = "~"
            objPort.EndTrigger = "~"
            objPort.Enabled = True
            'objPort.StopBits = 126

        Catch ex As Exception
            MessageBox.Show(ex.Message)
        End Try
    End Sub

    Private Sub Form1_Closing(ByVal sender As Object, ByVal e As
System.ComponentModel.CancelEventArgs) Handles MyBase.Closing
        ' You must dispose the object before the form unloads
        ' else Port might throw events to the unloaded form with a
crash as result.
        objPort.Dispose()
    End Sub

    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick

        Dim Node As Integer
        Dim status As Integer

        str = objPort.Read(0, 5000)
        message = SerialNET.Port.StringToByteArray(str)
        Node = message(9)
        status = message(8)


        If (Node = 1) Then
            If (status = 0) Then
                box1.BackColor = System.Drawing.Color.Red
            Else
                box1.BackColor = System.Drawing.Color.Green
            End If
        End If

        If (Node = 4) Then
            If (status = 0) Then
                box2.BackColor = System.Drawing.Color.Red
            Else
                box2.BackColor = System.Drawing.Color.Green
            End If
        End If

        'Meters(1).status = message(8)
        'Meters(1).node = message(9)

        'If Meters(i).Timeout = 5 Then
        '    box1.BackColor = System.Drawing.Color.Yellow
        '    Meters(i).status = "No Communication"

        'End If
```

```
        'If str Is Nothing Then
        '    Meters(i).Timeout = Meters(i).Timeout + 1
        'End If


        '    Else
        'message = SerialNET.Port.StringToByteArray(str)
        'Meters(1).status = message(8)
        'Meters(1).node = message(9)

        'If Meters(1).node = 1 Then
        '    Text1.Text = Meters(1).status

        '    If (Meters(1).status = 0) Then
        '        box1.BackColor = System.Drawing.Color.Red
        '    Else : box1.BackColor = System.Drawing.Color.Green
        '    End If


        'End If
        '    End If

        'For i = 0 To Meters.GetUpperBound(0)
        '    If Meters(i).node <> node Then
        '        If Meters(i).Timeout = 10 Then

        '        End If
        '    End If
        'Next
    End Sub


    Private Sub PictureBox2_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles PictureBox2.Click

    End Sub
End Class
```

Appendix D (Dummy Help)

Step 1.
After all the products have been obtained, the first objective should be to build the relay circuit in order to read the logic from the parking meter (Refer to Appendix A for relay circuit).

Step 2.
Connect a mote to the parking meter.

Step 3
Once the mote at the parking meter is transmitting the meter id and status, connect the base mote to the laptop via serial connection.

Step 4
After the base mote communicates with the laptop properly now the mote-to-mote communication should be implemented.

Step 5
Write a program to implement the meter location on the laptop.

Step 6
Put the money into the meter and see if the status of the meter on the laptop changes. If yes, then project is successful.

Appendix E (Images)

Project Flow Chart

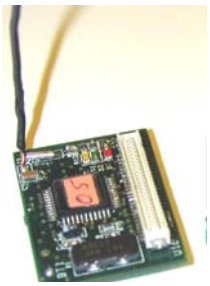
No Money in Meter

LED Pulses

LED pulses cause a VoH, turning the relay on which grounds ADC1 Pin on the mote.

555 Timer Circuit

When ADC pin is grounded the mote begins transmitting msg corresponding to expired meter.



Mica2 Mote

Base Mote