ESE 441 – Spring 2007

Instructor: Prof. Gouzman

Student: Shrivathsa Bhargav

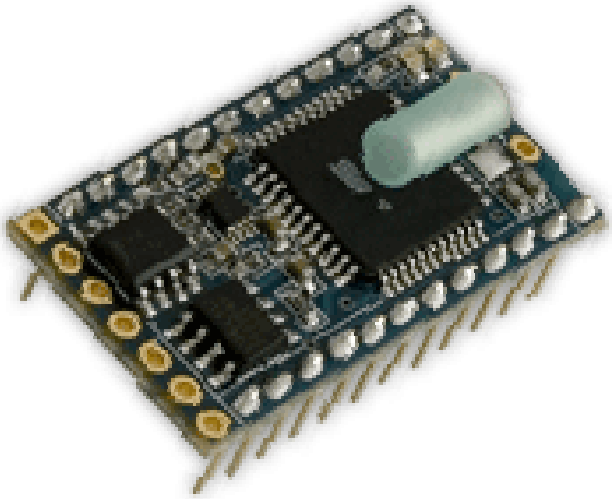


Figure 1 - The NetMedia BasicX24p

Abstract

Light detection is an important part of optoelectronics, as well as automation. While light detection systems are currently reliable, their accuracy leaves something to be desired. One of the most accurate light detection systems is a *single photon detector*. Using a single photon detector, light radiated from a body (luminescent or not) can be very finely determined. This project provides the brains of an inexpensive data collection/processing system that utilizes a single photon detector and an FPGA for processing.

Table of Contents

Abstract 2
Section 1: Goals 4
Section 2: Background 5
2.1 Survey 5
2.2 Project Planning 5
Section 3: System Design 6
3.1 Design Considerations 6
3.2 Designs 7
Programming the BasicX 7
3.3 Final Design 8
Section 3.4: Implementation and Testing 12
Implementation of the program 12
Section 3.5: Problems and Debugging 12
Section 3.6: Observations (Results) and limitations 13
Section 4: Experiences 14
Section 5: Conclusion 14
Acknowledgements 14
References and External Links 15
Attachments 15

Section 1: Goals

This project is based firmly in the realm of optoelectronics, which is a fascinating field involving, of course, optics and electronics. The task to be performed is to observe the decaying levels of photonic emission from a subject body. This is shown in a simplistic fashion and explained below.

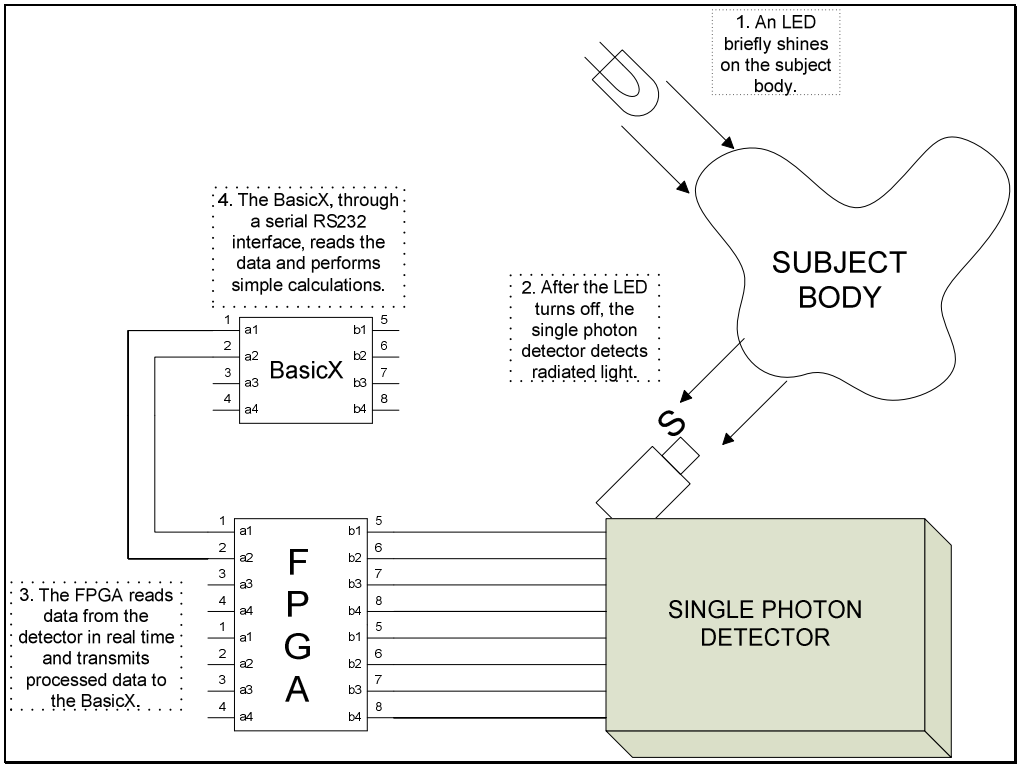


Figure 2 - Simplified model of the system

Without going into great detail, it suffices to say that the number of photons emitted is detected by a *single photon detector*, which, in real time, sends data to an FPGA (Field Programmable Grid Array). The FPGA, in turn, calculates the number of photons between two time intervals. This enables a good approximation of the emission of the subject body. The FPGA, by itself, has little to no memory for performing advanced calculations, and this is where an external microcontroller comes into play.

For this project, it is sufficient to perform a simple running average on two incoming numbers. There were dozens of microcontrollers suitable for the job, but the NetMedia BasicX was chosen after the successful implementation using the Parallax BASIC Stamp 2. The goal is to provide a quick, inexpensive method to read this photon count from the FPGA over many iterations and perform an averaging equation, after which the result is shown on an LCD display.

Section 2: Background

2.1 Survey

Professor Gouzman holds a patent on single photon detection-based photonic emission decay systems. This planned system has been realized in the past, using pure FPGA processing power. The FPGA, however, has no memory to speak of, and can therefore not perform any mathematical functions, such as the required averaging function. Such a modification on the project (by adding a microcontroller into the mix), therefore, brings its own challenges.

This project was attempted by in ESE 440 in the fall of 2006 with great success, and is being revamped as part of ESE 441 in the spring of 2007. Hence, there will be many similarities and assumptions throughout this report.

2.2 Project Planning

Since the NetMedia BasicX was chosen as the desired microcontroller, it is imperative that the programmer possesses solid programming skills, specifically in BASIC and C (NetMedia's version of BASIC is more akin to Visual BASIC). Interfacing with the FPGA involves serial data transfer through the RS-232 protocol. Additionally, some

rudimentary wiring skills will be required to interface the BasicX with the FPGA and some external control buttons, as well as the LCD.

Section 3: System Design

3.1 Design Considerations

Financially, the system, not including the single photon detector (which runs in the thousands of dollars) or the FPGA (which runs in the hundreds) must not be too expensive. The desired result would be a sub-$100 device. NetMedia offers just such a device with the BasicX, at $49.95 + s/h. In the previous design, the necessary external floating point processor cost $15 + s/h. The BasicX is internally capable of floating-point arithmetic; this eliminates the need for an external floating point processor, thereby saving space and expense.

Spatially, the system could not be much larger than the FPGA, since that was the largest component to be placed in the final casing. The BasicX meets this requirement, since it is only 1.2"x0.6"x0.4", and is the same footprint as the Parallax BASIC Stamp 2, so that previous circuits need not be modified to accommodate the BasicX.

Technically, the system had to be decently fast, with a wait time of less than 5 minutes for a result, since this was the second prototype. With an Atmel ATS8535 processor capable of executing ~8,000 instructions/sec, the BasicX supersedes the Parallax BASIC Stamp 2 in processing power and speed. Additionally, the power consumption must be low. Again, the BasicX meets this requirement with 20 mA run power consumption. The system must be able to display the results on an LCD screen, which was accomplished via a Parallax 2x16 LCD display (leftover from the last project).

Logically, a floating-point arithmetic was required, and, as mentioned before, this functionality was provided by the BasicX. The standards involved are CMOS voltage levels for powering the Stamp and the FPGA, and data transfer using the RS-232 single-wire protocol.

3.2 Designs

Programming the BasicX

The learning curve is relatively sharp, since the programming language used is not purely BASIC, but a proprietary variant of it, involving structures and object oriented programming. It incorporates several elements of Visual Basic, as well. Programming the foundational code was very easy, and the well-written documentation, as well as a little reference to the myriad of online resources dedicated to the BasicX definitely helped. Programming the serial communication was much trickier, given the variety of parameters to be set to achieve just a simple link. The interpreter is user-friendly, and transferring the program to the BasicX takes less than ten seconds.

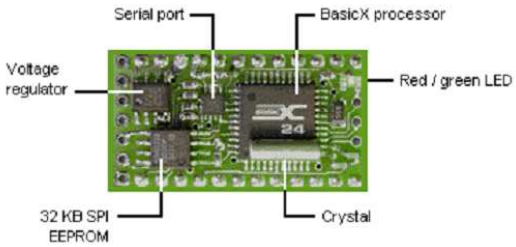


Figure 3 - BasicX

There is a rich library of mathematical functions available for dealing with floating-point mathematical operations.

3.3 Final Design

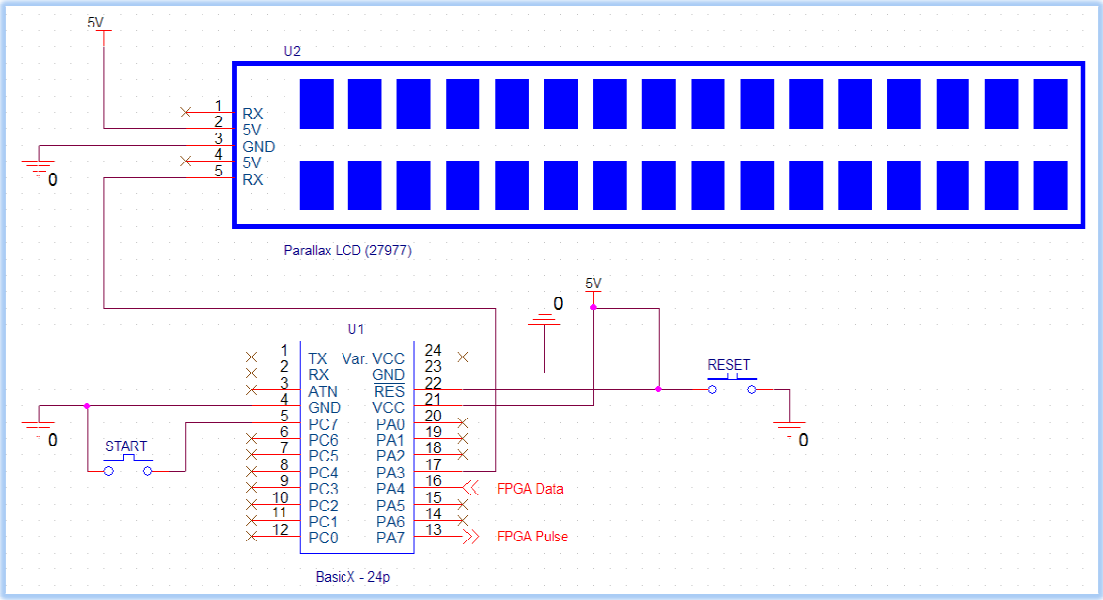


Figure 4 - Final schematic

Notice that the LCD has symmetric connection pins. This allows easier placement of the LCD on a final PCB and inside a casing. The LCD is backlit, but this feature is configurable from the BasicX during initialization. The current implementation has the backlight turned on, to enhance readability. However, future implementations may have this feature turned off to conserve power. It is even possible to have this toggle-able by the user on the fly using an interface button.

The RESET button allows an external reset of the BasicX's program in preparation for a new set of readings (signifying a new experiment). The START button allows the user to specify when the experiment begins, and when the BasicX should start pinging the FPGA for new values.

The software used was NetMedia's BasicX Editor v2.1.

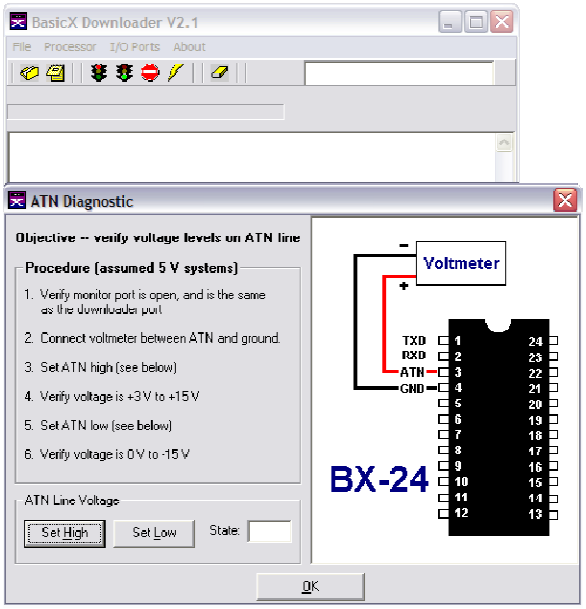


Figure 5 – Diagnostic routines make troubleshooting the BasicX easier

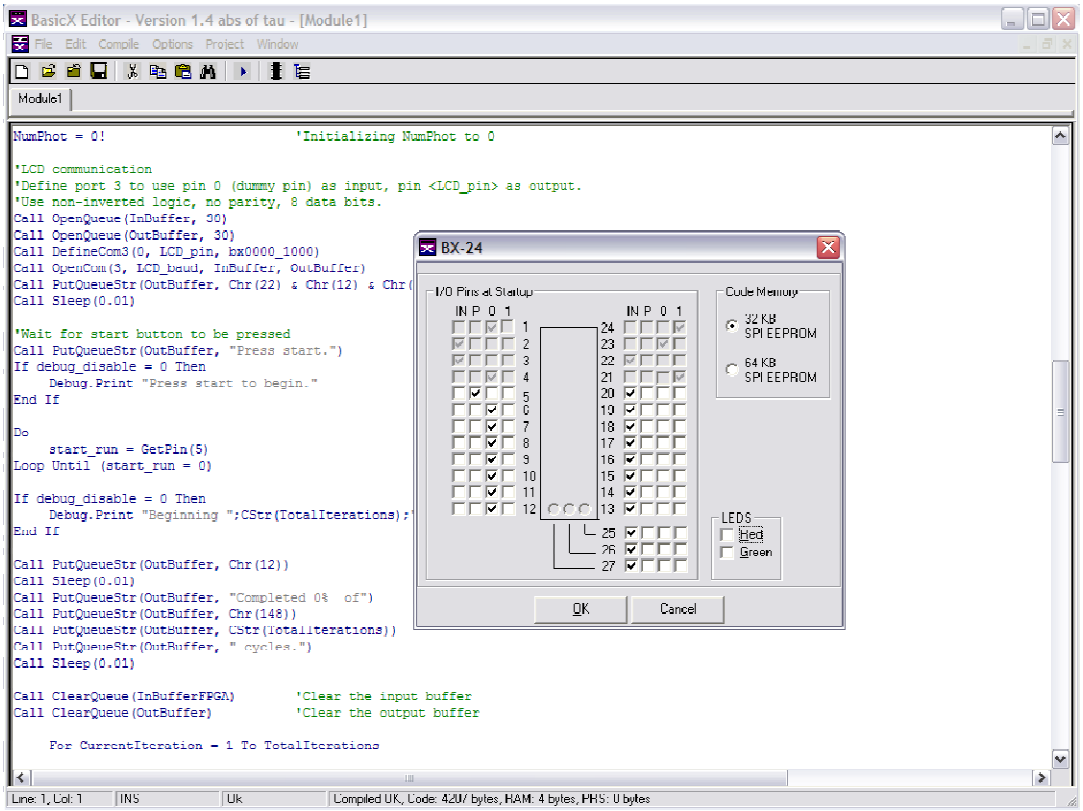


Figure 6 - Screenshot of the pin configuration in the IDE

The IDE has several nifty little tools, such as the pin configuration dialog shown in figure 6. This allows individual pins to be set as inputs, pull-ups or pull-downs.

In addition, the serial communication options are vast (see table 1). While it was tempting to use ports 1 or 2, which are dedicated hardware serial ports, it had to be kept in mind that transferring data from the FPGA is the least-processor intensive, and therefore need not be dedicated to hardware. Therefore, port 3, a software serial port, was used.

Here are some pictures of the final design:

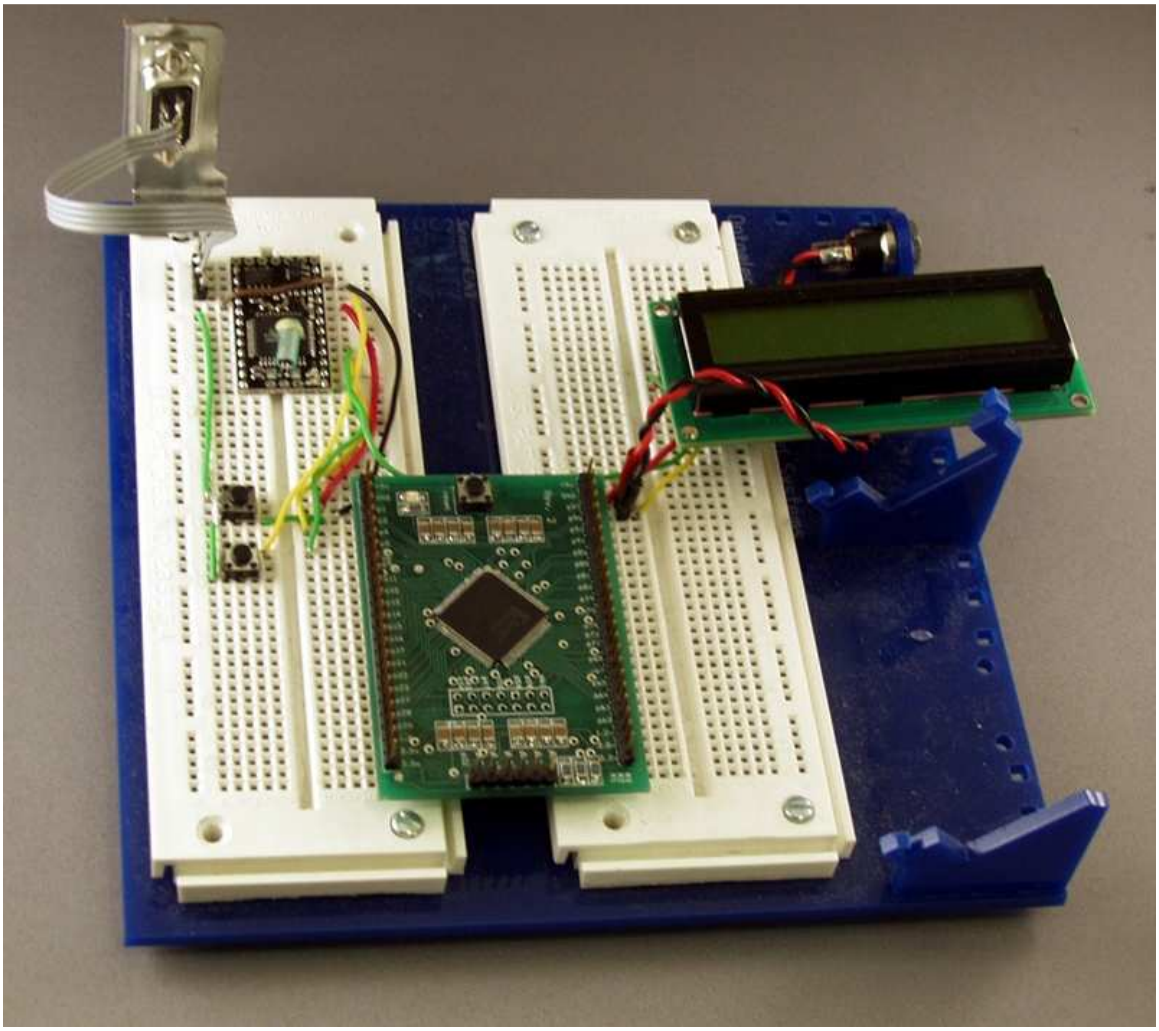


Figure 7 – Picture of final design

The final design, shown in figure 7, is only a prototyping setup. Note the BasicX in the top-left corner. It has a minimal footprint when compared to the FPGA (bottom-middle). The final, marketable design is expected to be much more compact. One possible implementation of the final design is shown in figure 9.

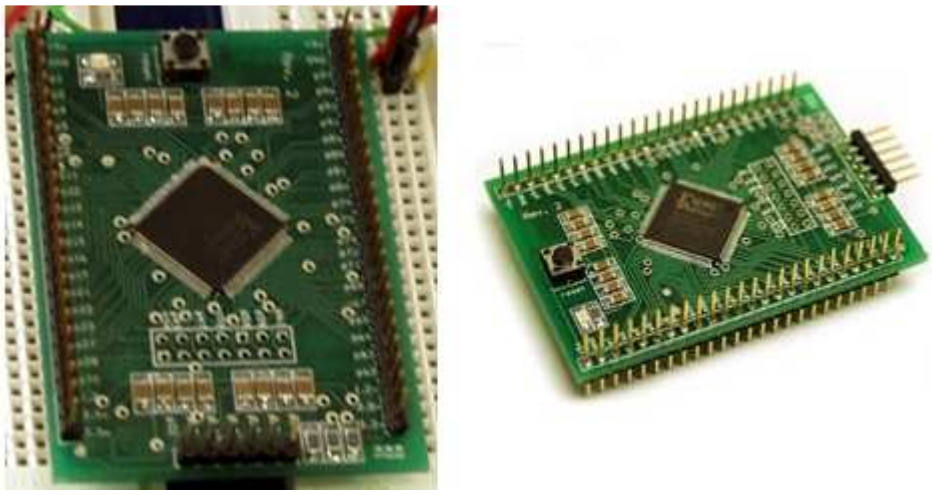


Figure 4 - Close-up of the FPGA and another angle

Figure 9 - One implementation of the final casing

Section 3.4: Implementation and Testing

Implementation of the program

The basic algorithm of the program is as follows:

- Initialization of the LCD, BasicX
- Wait for the (external) Start button to be pressed.
- Once it is pressed, go into a 16,000 iteration loop which consists of:
    - A short outgoing pulse to activate the FPGA
    - The serial-input of two eight-digit numbers, digit by digit
    - The recombination of the input into two eight-digit numbers
    - The application of an averaging formula on the ratio of the first number to the second number
- Finally, the results have to be displayed on the LCD.

This algorithm is represented by the attached flowchart. Greater insight may be gained by studying the attached program code.

Section 3.5: Problems and Debugging

The BasicX has no debug-functionality built-in, and, because of this, debug operations had to be performed through the LCD that was attached to the BasicX. This was painstaking, but enabled me to debug the problems quite easily. However, communicating with the FPGA was an exercise in trial-and-error, because the BasicX simply didn't seem to receive the data correctly from the FPGA. It later turned out that I was capturing a *wrong number* of digits from the FPGA, something that, in hindsight, could have been avoided. Also, the RS-232 has too many configurable parameters, so arriving at mutually acceptable parameters (with the FPGA) took time.

Table 1 – Configuring the COM port is straightforward, but has many options

Parameter	Value	Bit pattern (x = don't care)
Inverted logic	&H80	1 0 x x x x x x
Non-inverted logic	&H00	0 0 x x x x x x
Even parity	&H30	x x 1 1 x x x x (7 bit only)
Odd parity	&H20	x x 1 0 x x x x (7 bit only)
No parity	&H00	x x 0 0 x x x x
7 data bits	&H07	x x x x 0 1 1 1
8 data bits	&H08	x x x x 1 0 0 0

The final configuration used was 9600 baud / No parity / No stop bit / Inverted.

Section 3.6: Observations (Results) and limitations

The BasicX, while versatile, is moderately slow, due to the fact that data is transferred to and from the FPGA using a two-wire serial interface. Because the formula being used (shown in a previous section) is iterative, it is a slow process to compute, though not especially taxing. On average, for two seven-digit numbers taking the place of *firstnum* and *scondnum*, it takes about one-eighth of a second to perform three iterations. This works out to around 11 minutes for 16,000 iterations! This is extremely slow, but since the product is a prototype, it will suffice to run it through 1,000 iterations or so instead, till a faster device can be developed.

It is important to note the performance improvement that has been gained from switching the microcontroller platform from the Parallax BASIC Stamp 2 to the NetMedia BasicX. With an inherently faster processor, and no necessity of an external FPU unit, speeds improved generally by 3x. This is a marked improvement, and can only be topped by switching to a totally different (non-BASIC based) platform like the Rabbit.

Section 4: Experiences

Overall, it was a very enjoyable project, with a few setbacks. One of the main lessons learned was how to approach and assimilate documentation. When some items were unclear, I learned how to approach customer support, as well as community forums to learn from others who were much more knowledgeable than me. The community was moderately helpful, as the BasicX is usually considered a hobbyist's device (and a direct competitor to the more popular, but less capable, Parallax BASIC Stamp 2). I am quite prepared to utilize both NetMedia's, as well as Parallax's products in future projects, based on the happy experiences I had learning their technology.

Section 5: Conclusion

The BasicX was not as easy to use as the BASIC Stamp 2, but it was certainly more versatile. Unlike the Stamp, the BasicX had a built-in floating-point processor, which enabled faster computations with less error and transfers. The speed of the device, however, is still not as fast as desired. This is mostly due to the speed of the FPGA, which must be tweaked. The next logical step would be to try a faster/more capable microcontroller with a revamped FPGA. That, however, should be an entire project by itself. In the meanwhile, this was a most exciting research course.

Acknowledgements

Prof. Gouzman, as well as his PhD student Vladis Kuzminsky were both extremely helpful. Even when things looked bleak, both of them expressed their immense confidence in me and kept me going.

References and External Links

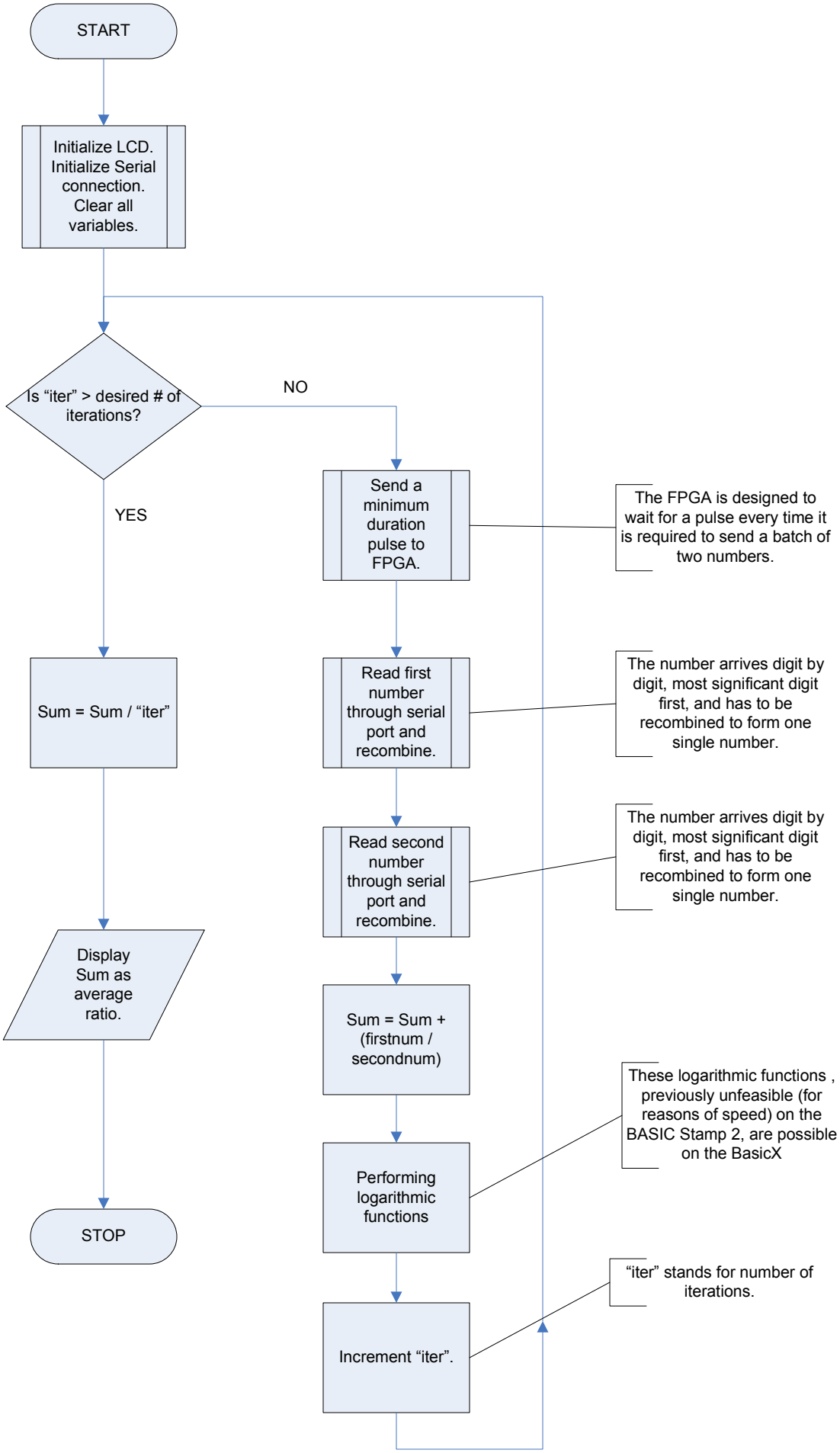
NetMedia's site for the BasicX: http://www.basicx.com/

Parallax's backlit LCD: http://www.parallax.com/detail.asp?product_id=27977

Attachments

- Flowchart showing program flow
- BasicX program code
- Final schematic
- BasicX internal schematic

Attachments - Flowchart



Attachments - Program code

```
'Author: Shrivathsa Bhargav
'Coded for ESE 441
'Version history (only shows significant versions):
'       Version 1.4  -      Implemented Tau calculation
'       Version 1.0  -      Final version, everything works
'                                   Start & Reset button added
'                                   Summation now displays correctly
'       Version 0.9 -       Speed optimization
'       Version 0.22-       Attempting to iron out bugs in FPGA communication
'       Version 0.1  -      LCD works


Public Sub Main()

'Allocate global data buffers for serial transfer
Dim InBuffer(1 To 30) As Byte          'LCD data buffer
Dim OutBuffer(1 To 30) As Byte         'LCD data buffer
Dim InBufferFPGA(1 To 30) As Byte      'FPGA data buffer
Dim OutBufferFPGA(1 To 30) As Byte     'FPGA data buffer


Dim C1(9) As Byte                      'We only need 9 * 1 spaces to catch 9 values of first number
Dim C2(9) As Byte                      'We only need 9 * 1 spaces to catch 9 values of second number
Dim CN(9) As Byte                      'This is to capture the 9 possible digits of N
Dim FirstNum As Single                 'The first number could be huge after recombination; Single is the largest data type
Dim SecondNum As Single                'The second number could be huge after recombination
Dim NumPhot As Single                  'The number of photons in the time interval
Dim Sum As Single                      'The sum needs to be as accurate as possible; Single is the largest floating point data type
Dim Tau As Single                      'The decay parameter that is calculated every iteration and must be averaged at the end
Dim CurrentIteration As Long           'How many iterations have been completed?
Dim Temp As Byte                       'Just a variable for temporary calculations
Dim Temp2 As Byte                      'Just another variable for temporary calculations
Const TotalIterations As Long = 160 'The number of iterations
Const ProgressUpdateRate As Long = 5      'Increase this to speed up process (by reducing update frequency)

Const FPGA_baud As Long = 9600         'The incoming baud rate to the FPGA
Const FPGA_pin As Byte = 13            'The pin that's connected to the FPGA (pulse)
Const FPGA_data_pin As Byte = 16       'The pin that's receiving data from FPGA (serial)

Const LCD_baud As Long = 19200         'The outgoing baud rate to the LCD
```

```
Const LCD_pin As Byte = 17          'Pin that's transferring data TO to LCD

Const debug_disable As Byte = 0     '1 = No status messages on console, 0 = otherwise

Const start_button As Byte = 5      'Start button is connected to pin 5
Dim start_run As Byte               'Variable to detect start button

'The following code initializes Catcher1
C1(0) = 0
C1(1) = 0
C1(2) = 0
C1(3) = 0
C1(4) = 0
C1(5) = 0
C1(6) = 0
C1(7) = 0
C1(8) = 0

'The following code initializes Catcher2
C2(0) = 0
C2(1) = 0
C2(2) = 0
C2(3) = 0
C2(4) = 0
C2(5) = 0
C2(6) = 0
C2(7) = 0
C2(8) = 0

'The following code initializes Catcher for NumPhot (CN)
CN(0) = 0
CN(1) = 0
CN(2) = 0
CN(3) = 0
CN(4) = 0
CN(5) = 0
CN(6) = 0
CN(7) = 0
CN(8) = 0
```

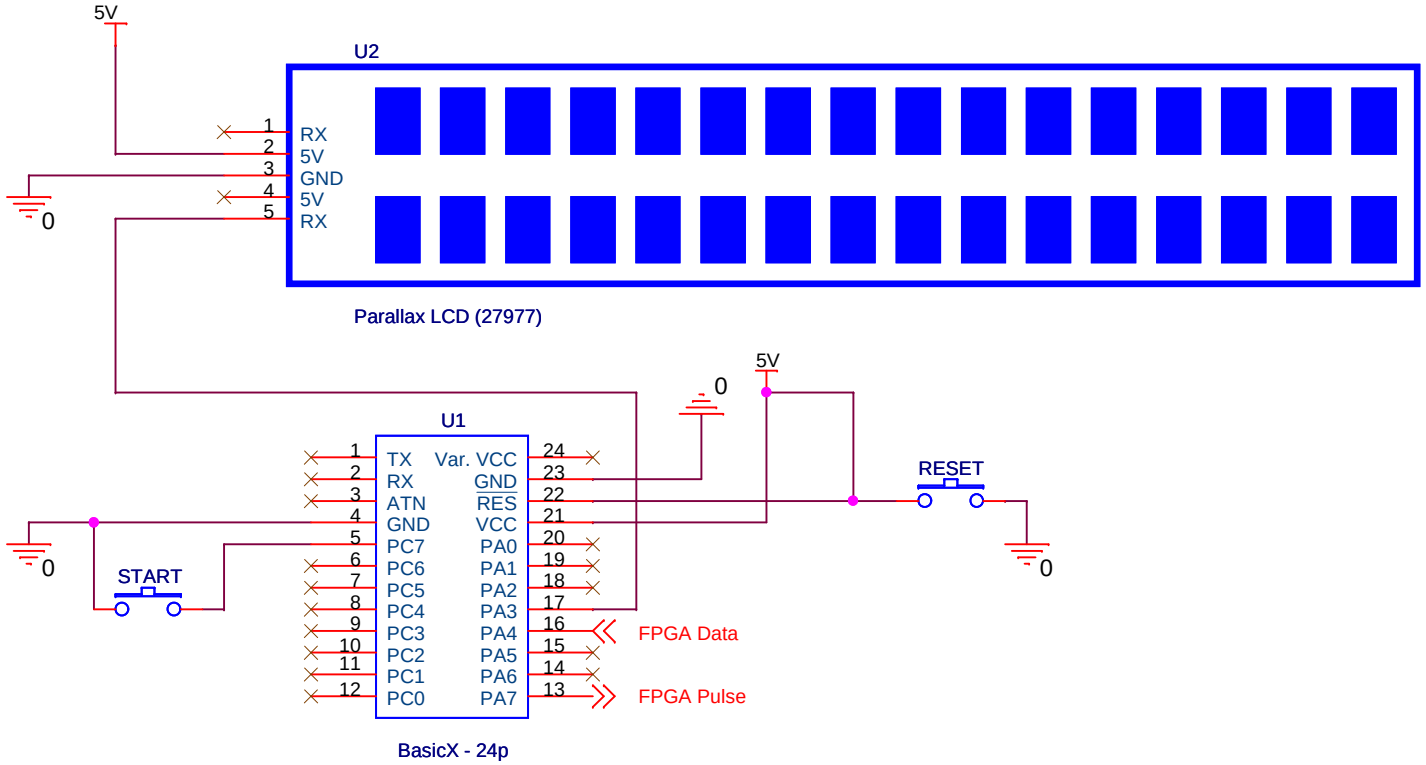
```
Sum = 0!                                    'Initializing Sum to 0
FirstNum = 0!                               'Initializing FirstNum to 0
SecondNum = 0!                              'Initializing SecondNum to 0
NumPhot = 0!                                'Initializing NumPhot to 0

'LCD communication
'Define port 3 to use pin 0 (dummy pin) as input, pin <LCD_pin> as output.
'Use non-inverted logic, no parity, 8 data bits.
Call OpenQueue(InBuffer, 30)
Call OpenQueue(OutBuffer, 30)
Call DefineCom3(0, LCD_pin, bx0000_1000)
Call OpenCom(3, LCD_baud, InBuffer, OutBuffer)              'Open the port to LCD
Call PutQueueStr(OutBuffer, Chr(22) & Chr(12) & Chr(17))    'Clear display, turn on backlight
Call Sleep(0.01)                                            'Recover from clear display

'Wait for start button to be pressed
Call PutQueueStr(OutBuffer, "Press start.")
If debug_disable = 0 Then
    Debug.Print "Press start to begin."
End If

Do
    start_run = GetPin(5)           'Read status of start button
Loop Until (start_run = 0)

If debug_disable = 0 Then
    Debug.Print "Beginning ";CStr(TotalIterations);" cycles."
End If

Call PutQueueStr(OutBuffer, Chr(12)) 'Clear LCD
Call Sleep(0.01)                     'Recover from LCD clearing
Call PutQueueStr(OutBuffer, "Completed 0%  of")
Call PutQueueStr(OutBuffer, Chr(148))
Call PutQueueStr(OutBuffer, CStr(TotalIterations))
Call PutQueueStr(OutBuffer, " cycles.")
Call Sleep(0.01)

Call ClearQueue(InBufferFPGA)       'Clear the input buffer
Call ClearQueue(OutBuffer)          'Clear the output buffer
```

```
For CurrentIteration = 1 To TotalIterations

    'Open port 3 to use pin <FPGA_data_pin> as input and pin 0 (dummy) as output
    'Use INVERTED logic, no parity, 8 data bits
    Call OpenQueue(InBufferFPGA, 20)
    Call OpenQueue(OutBufferFPGA, 20)
    Call DefineCom3(FPGA_data_pin, 0, bx10001000)
    Call OpenCom(3, FPGA_baud, InBufferFPGA, OutBufferFPGA)

    'The code below generates a pulse on FPGA_pin (which is needed to trigger the FPGA's output)
    Call PutPin (FPGA_pin, bxOutputHigh)     'Set FPGA pulse pin HIGH
    Call PutPin (FPGA_pin, bxOutputLow)      'Set FPGA pulse pin LOW

    'The code below captures 9 digits of each number
    For Temp = 0 To 8                        'Get first number (9 digits)
        Call GetQueue(InBufferFPGA, Temp2, 1)
        C1(Temp) = Cbyte(Temp2)              'Store digit in C1
    Next

    For Temp = 0 To 8                        'Get second number (9 digits)
        Call GetQueue(InBufferFPGA, Temp2, 1)
        C2(Temp) = CByte(Temp2)              'Store digit in C2
    Next

    For Temp = 0 To 8                        'Get number of photons (9 digits)
        Call GetQueue(InBufferFPGA, Temp2, 1)
        CN(Temp) = CByte(Temp2)              'Store digit in CN
    Next

    'The following debug code shows the digits gathered from the FPGA (in order), for debugging purposes
    If debug_disable = 0 Then
        'Debug.Print "FirstNum=";CStr(C1(0));" ";CStr(C1(1));" ";CStr(C1(2));" ";CStr(C1(3));" ";CStr(C1(4));" ";CStr(C1(5));" ";CStr(C1(6));" ";CStr(C1(7));" ";CStr(C1(8))
        'Debug.Print "SecondNm=";CStr(C2(0));" ";CStr(C2(1));" ";CStr(C2(2));" ";CStr(C2(3));" ";CStr(C2(4));" ";CStr(C2(5));" ";CStr(C2(6));" ";CStr(C2(7));" ";CStr(C2(8))
    End If

    'Recombination begins here
    FirstNum = 0!                            'Initialize values to 0 every iteration
    SecondNum = 0!                           'Initialize values to 0 every iteration
    NumPhot = 0!
        For Temp = 0 To 8
```

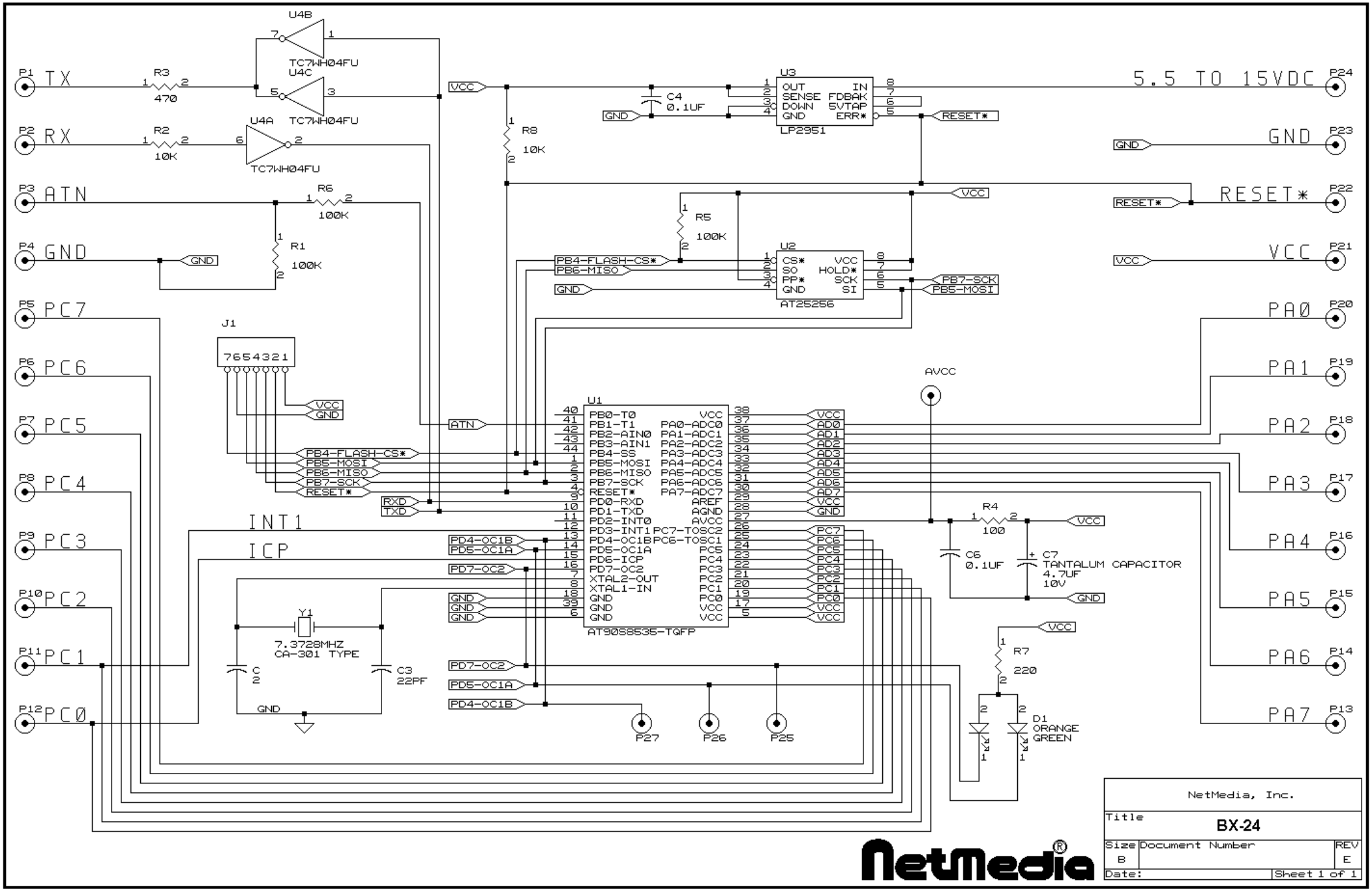
```
                FirstNum = FirstNum + (Csng(C1(8-Temp)) * (10.0 ^ Temp))
                SecondNum = SecondNum + (CSng(C2(8-Temp)) * (10.0 ^ Temp))
                NumPhot = NumPhot + (CSng(CN(8-Temp)) * (10.0 ^ Temp))
            Next

        'The following debug code shows the two numbers after recombination
        If debug_disable = 0 Then
            'Debug.Print "Firstnum=";CStr(CLng(FirstNum))
            'Debug.Print "Secondnu=";CStr(CLng(SecondNum))
        End If

        FirstNum = FirstNum / (10.0 ^ 9)          'Compensate for the fact that FirstNum is always in nS

        'Sum = (Sum + (FirstNum / SecondNum))                    'First part of Averaging equation
        Tau = Tau + (FirstNum/Log(1!-(1!/NumPhot)))             'First part of decay equation

        'The following code updates the LCD with current progress
        If (CurrentIteration Mod ProgressUpdateRate) = 0 Then
            'LCD communication
            'Define port 3 to use pin 0 (dummy pin) as input, pin 17 as output.
            'Use non-inverted logic, no parity, 8 data bits.
            Call OpenQueue(InBuffer, 30)
            Call OpenQueue(OutBuffer, 30)
            Call DefineCom3(0, LCD_pin, bx0000_1000)          'Define COM to LCD on previously chosen pin (LCD_pin)
            Call OpenCom(3, LCD_baud, InBuffer, OutBuffer)
            Call Sleep(0.01)

            If debug_disable = 0 Then
            '     Debug.Print "Progress: ";CStr(CByte((CSng(CurrentIteration)/Csng(TotalIterations))*100!));" %"
            End If

            Call PutQueueStr(OutBuffer, Chr(138))         'Position cursor at position 10
            Call PutQueueStr(OutBuffer, CStr(CInt((CSng(CurrentIteration)/Csng(TotalIterations))*100!)))     'Show progress %
            Call PutQueueStr(OutBuffer, "%")
            Call Sleep(0.005)
        End If

    Next

Tau = Abs(Tau)                                    'Tau can never be negative
```

```
'Sum = Sum/CSng(TotalIterations)                    'Second part of averaging equation
Tau = Tau/CSng(TotalIterations)                     'Second part of decay equation

Call PutQueueStr(OutBuffer, Chr(12) & Chr(17))      'Clear display, turn on backlight
Call Sleep(0.01)                                    'Recover from clear display

Call PutQueueStr(OutBuffer, "Job completed.")       'Show completion

If debug_disable = 0 Then
    Debug.Print "Job complete!";" Tau ="
    Debug.Print CStr(Tau)
End If

Call PutQueueStr(OutBuffer, Chr(148))               'Position cursor @ Line 2, Position 0
Call PutQueueStr(OutBuffer, "Tau=")
Call PutQueueStr(OutBuffer, CStr(CSng(Tau)))        'Show the final result

Call PutQueueStr(OutBuffer, Chr(128))
                                                    'Infinite loop, waiting for Reset button to be pushed
Do
Call Sleep(2.0)
Call PutQueueStr(OutBuffer, Chr(128))
Call PutQueueStr(OutBuffer, "Job complete! ")
Call Sleep(2.0)
Call PutQueueStr(OutBuffer, Chr(128))
Call PutQueueStr(OutBuffer, "Press Reset.  ")
Loop

End Sub
```

Attachments - Final schematic



Shrivathsa Bhargav		
Title: ESE 441 - Spring 2007 - Schematic		
Size: A	Document Number: ESE 441-001	Rev: 2
Date: Sunday, April 22, 2007	Sheet 1 of 1	


U4B
TC7WH04FU
U4C
TC7WH04FU
U4A
TC7WH04FU
P1 TX
P2 RX
P3 ATN
P4 GND
P5 PC7
P6 PC6
P7 PC5
P8 PC4
P9 PC3
P10 PC2
P11 PC1
P12 PC0
R3 470
R2 10K
R6 100K
R1 100K
R8 10K
R5 100K
C4 0.1UF
U3
OUT IN
SENSE FDBAK
DOWN SVTAP
GND ERR*
LP2951
5.5 TO 15VDC P24
GND P23
RESET* P22
VCC P21
U2
CS* VCC
SO HOLD*
PP* SCK
GND SI
AT25256
PB4-FLASH-CS*
PB6-MISO
PB7-SCK
PB5-MOSI
J1
7654321
U1
PB0-T0 VCC
PB1-T1 PA0-ADC0
PB2-AIN0 PA1-ADC1
PB3-AIN1 PA2-ADC2
PB4-SS PA3-ADC3
PB5-MOSI PA4-ADC4
PB6-MISO PA5-ADC5
PB7-SCK PA6-ADC6
RESET* PA7-ADC7
PD0-RXD AREF
PD1-TXD AGND
PD2-INT0 AVCC
PD3-INT1 PC7-TOSC2
PD4-OC1B PC6-TOSC1
PD5-OC1A PC5
PD6-ICP PC4
PD7-OC2 PC3
XTAL2-OUT PC2
XTAL1-IN PC1
GND PC0
GND VCC
GND VCC
AT90S8535-TGFP
PA0 P20
PA1 P19
PA2 P18
PA3 P17
PA4 P16
PA5 P15
PA6 P14
PA7 P13
AVCC
R4 100
C6 0.1UF
C7 TANTALUM CAPACITOR 4.7UF 10V
INT1
ICP
Y1
7.3728MHZ CA-301 TYPE
C2
C3 22PF
R7 220
D1 ORANGE GREEN
P27
P26
P25
NetMedia, Inc.
Title
BX-24
Size B
Document Number
REV E
Date:
Sheet 1 of 1
NetMedia®