

Super Awesome Multitasking Microcontroller Interface for Electromechanical Systems (S.A.M.M.I.E.S.) Pinball Table

Group 13
April 29th, 2008

Faculty Advisor:
Professor Haibo He

Group Members:
William McGuire
Michelle Attilio
Robert Iannacone

I pledge my honor to abide by the Stevens Honor System.

Table of Contents

I.	ABSTRACT.....	1
I.1.	ACKNOWLEDGEMENTS	2
II.	IMPLEMENTED PROTOTYPE.....	3
II.1.	INTRODUCTION	3
II.2.	PROTOTYPE SPECIFICATION	6
II.3.	PROTOTYPE PERFORMANCE AND EVALUATION.....	11
II.4.	FINANCIAL BUDGET	14
II.5.	PROJECT SCHEDULE	15
III.	CONCLUSION	16
IV.	REFERENCES	17
V.	APPENDICES	18
V-1	APPENDIX A: SENSOR POLLER (USART - NO 9 BIT).C.....	A1-A2
V-2.	APPENDIX B: POLLER SUBROUTINES.H.....	B1
V-3.	APPENDIX C: POLLER SUBROUTINES.C.....	C1-C2
V-4.	APPENDIX D: DRIVER (USART - NO 9 BIT).C.....	D1-D3
V-5.	APPENDIX E: USART STUFF.H.....	E1
V-6.	APPENDIX F: USART STUFF.C.....	F1-F2
V-7.	APPENDIX G: SOLENOIDSUBROUTINES.H.....	G1
V-8.	APPENDIX H: SOLENOIDSUBROUTINES.C	H1
V-9.	APPENDIX I: LIGHTSUBROUTINES.H	I1
V-10.	APPENDIX J: LIGHTSUBROUTINES.C	J1-J2
V-11.	APPENDIX K: SERIALTEST.C.....	K1-K3
V-12.	APPENDIX L: HOST.VI	L1-L2
V-13.	APPENDIX M: SAMMIES.VI (CLIENT)	M1-M8

I. Abstract

This project is called the Super Awesome Multitasking Microcontroller Interface for Electromechanical Systems Pinball Table, or the S.A.M.M.I.E.S. Pinball Table. Its purpose was to provide a way for a normal person to be able to once again enjoy the pinball experience. The group's goal was to find a way to affordably create a working pinball table complete with some unique features.

The table used a laptop running LabVIEW to simulate several microcontrollers for its central processing unit, graphics processing unit, and all subsystems. The group faced problems that included obtaining all of the parts needed at an affordable price, properly programming the interface to work together with each of the parts, as well as implementing the more unique features of the table. These new features included a sub-system to recover your ball, as well as a score that can be used to activate additional features within the game.

The group fully expected to be able to create a complete and working pinball table by the end of this senior design course. In spite of many last-minute problems, the group had completed all the software and testing for the project. However, given the age of the electro-mechanical parts, the table itself did not function. If a newer interface were used, the table would have been a resounding success.

I.1. Acknowledgements

We would like to acknowledge and thank Professor He for his ideas and support as well as his dedication to the success of this project.

Professor McNair for his ideas, advice and help with putting us in contact with all of the right people as well as lending us a power supply last-minute.

Microchip.com for the free microchip samples that they sent us for our project.

We would also like to thank all of our family and friends for their support and help through the completion of this project, especially Michael Pomponio for supplying us with primer for the table, Michael and Loretta Pomponio and the Attilio family for supplying us with paint and various other materials for the table, and the McGuire family for housing the table over the winter break as well as supplying the tubing, glue, pinballs, tools, and various metal components for entities including the plunger.

II. Implemented Prototype

II.1. Introduction

The game of Pinball has existed for many years. It has become more sophisticated and complex during each year of development. The modern day pinball table is perhaps more enjoyable than it ever has been. It utilizes features like flashing lights, entertaining music, fun sound effects, an animated video display and thrilling speed to make it the game that it is today. However, it is also extremely expensive to construct. Many companies have left the pinball market through corporate merges or simply halting development of pinball tables. Presently, there is only one pinball manufacturer left in the world: Stern Pinball.¹

The Super Awesome Multitasking Microcontroller Interface for Electromechanical Systems Pinball Table, or S.A.M.M.I.E.S. Pinball Table, is a way for a typical person to be able to enjoy the pinball experience again. People can usually only play pinball at arcades, attractions that have become rarer every year due to home video game consoles becoming better. The S.A.M.M.I.E.S. Pinball Table is relatively inexpensive when compared to a new pinball table. This is because it contains simple microcontrollers that can be purchased cheaply as its central processing unit, graphics processing unit, and all subsystems.

These microcontrollers use parallel data ports to communicate rather than high speed serial. This allows the microcontroller to run at a much lower speed than it otherwise would require, greatly reducing power

¹ <http://www.sternpinball.com>

consumption. Not only does this reduce the cost of electricity to power the table for the end user, but the cost of the power supply is reduced as well. The cost of the extra wiring is far outweighed by the lesser cost of the power supply. This is all around better for the end user.

Something new was added to the game itself to make it a different experience for the consumer so the product will be desirable and these features integrate with the theme of the pinball table. The S.A.M.M.I.E.S. Pinball Table is themed after what its designers have in common, in addition to what they know best: college life. There were four key features that are in the design plans. Unfortunately due to time and budget constraints, the third and fourth features were not able to be implemented into the table.

The first feature is that of a global high score. Due to the microcontrollers being used having a wireless function already built into them, wireless can be utilized in the table. No other pinball tables have implemented a global high score and this will be a unique selling point for users. It enables them to play pinball and easily compete against their friends.

The second feature also utilizes the wireless feature. The table will have the function of remote diagnostics. This will allow for easier maintenance of the table. Rather than having the user try to figure out what is wrong with the table or sending support to the table, the manufacturer can remotely check on the table using the wireless

connection to troubleshoot the problem. Then when the support is sent to the table, they have the advantage of already knowing where the problem with the table is, which will greatly reduce the amount of time needed for repairs.

The third feature was that of a score that is more than just an arbitrary summation of points. The score would represent campus cash and these points would be used when the ball goes to a specified hole to activate the “school store.” Features such as multiball or a textbook (see below) could have been purchased using points.

The fourth feature was an “exam.” Each stage was to be known as a “semester” and at the end of each stage, there would be a final exam. Random multiple-choice questions were going to be displayed on the display screen. The user would then shoot a different set of targets depending on what his or her chosen answer is. A different target is then hit to submit the answer. Bonus points are awarded for correct answers. A jackpot is scored if all answers submitted are correct. If a textbook is purchased from the school store, one incorrect answer will be removed from the selection list during the “final exam” stage of the game.

II.2. Prototype Specification

The SAMMIES pinball table emulated a PIC16F97J60 through the use of three PIC16F887s, a MAX232, and a Microsoft Windows laptop running LabVIEW. The PIC16F887 microcontrollers acted as the sensor detector, solenoid driver, and light driver for the table. The MAX232 converted the serial data from the microcontrollers from TTL voltages (0 and +5V for low and high, respectively) to RS-232 voltages (+12V and -12V for low and high, respectively²) and vice-versa. The USB-Serial converter required because the laptop did not have a serial port forced a baud rate of 9600 bps. LabVIEW acted as the central processing unit of the system. It also functioned as the display and the interface for internet connectivity.

More light drivers would be required to drive every light on the table, and a two character long protocol was established. Originally, 9th-bit addressing was to be used. However, LabVIEW seemingly did not support this as the COM port of the PC must be opened with a specific parity setting and cannot be changed mid-process. For demonstration purposes, the protocol was shortened to one byte with the most significant bit determining which microcontroller was to use the received data, the solenoid driver or the light driver.

One feature that was lost due to this change was the ability to set or clear and entire port on the microcontroller. However, since this feature

² The RS-232 protocol calls for an inverted signal.

was programmed but never actually required, its loss presented no problem.

Due to their similarity, the solenoid driver and light driver were the same program with slight modifications. Due to power draw considerations, the group decided that the solenoid driver should automatically disable the solenoid shortly after its activation. Also, the solenoid driver was active low rather than active high like the light driver. The light driver was created with the ability to blink a light without input from the PC. This was not done on the solenoid driver because it was unnecessary.

To make programming easier and cleaner, the drive routines of the drivers were put into separate source files. A preprocessor directive instructed the compiler whether the compiled hex file should be made for a solenoid driver or a light driver. The shut off time and blink time were hard coded as well. The compiler, SourceBoost BoostC, removed the unnecessary subroutines to conserve memory. The data byte sent to the microcontrollers was in the following format:

bit7 - chip address (0 - lights, 1 - solenoid)

bit6..5 - port (a - 0, b - 1, c - 2, d - 3)

bit4..2 - pin number (0 to 7)

bit1 - set/clear

bit0 - should this pin blink (1 - yes, 0 - no, don't care for solenoid)

The byte sent from the sensor poller was the ID number of the pin struck by the ball:

$$\text{ID number} = (\text{port multiplier}) * ((\text{bit number of port}) + 1)$$

Port multipliers:

$$\text{porta} = 1$$

$$\text{portb} = 2$$

$$\text{portc} = 3$$

$$\text{portd} = 4$$

$$\text{porte} = 5$$

Because all the ports of the sensor poller were to be monitored, active polling was required. The inputs were buffered before the input was tested because the ports are volatile. It is possible, although unlikely, that a pin set high when tested in the main loop would become low when evaluated in the subroutines. Also under consideration was the fact that the poller was to wait for the ball to leave the sensor before polling again. This was to prevent the score from increasing for every instruction cycle the ball was on the sensor.

The SAMMIES system also takes advantage of wireless network technology to maintain a global high score server. Upon the completion of each game, the player is asked to use the flippers to select their initials. The player's initials and score are then submitted via TCP to a server, either global or local, and uploaded into a central database.

The system makes use of two parts - a TCP client and host, both written in LABVIEW. The TCP client accepts a set of 3 single-byte characters and an 8-byte integer score from the server. It then formats both data into strings and concatenates them. Finally, it establishes a one-time TCP connection to the high-score server, sends the data as a single TCP packet, waits for acknowledgment, then closes the connection.

The server-side host loops a listener on an unused port number. If it detects a connect request from a remote client, it connects, reads the incoming packet, and saves the score. It then imports the data from the existing high score file as an array of clusters. Each cluster is composed of a three-character string and a score variable (which must be converted from a string). The host adds the new score to the array as a final entry, uses a LABVIEW VI to re-sort the array in descending order by score, and rewrites the data into a text file. It then returns to the primary loop to listen for another client to connect. The host loop can be switched off using a shutoff button located on the VI front panel.

The resulting text file can then be read by any other service which might make use of it, such as a web page, or a file download service. A more local instance of the host could be run on a consumer machine for smaller-scale use. This would allow local arcades and theaters to host small-scale pinball tournaments, automatically track scores, and produce no drain on the global machines.

Potential future applications for the wireless system include a remote troubleshooting service. By issuing a command to the table over the wireless, the game can be commanded to return its current logic state and run several diagnostic programs, compile a report, and send it back to the technician. This would greatly reduce labor costs and troubleshooting time, as the support agency is no longer required to make a house call to service the table, and has the further advantage that it encourages the consumer to use an in-house repair service rather than a third-party dealership.

Another potential improvement in this area is a web-camera system. This would allow at-home players to host competitive games against each other over the Internet. In future iterations of the system, the interconnectivity could even be used to cause the games on the two tables to interact with each other - a rollover target on one table could activate a series of drop targets on the other, or open a channel leading to the other player's drain.

II.3. Prototype Performance and Evaluation

While the prototype for the SAMMIES table experienced mechanical problems and did not function as expected, the core of the project was in working order by the demonstration. While the refurbished actuators on the flippers and bumpers were unable to properly interface with modern transistor-based circuitry, the SAMMIES project revolved around the microchip protocol and interface with the physical parts, and all tests and simulations showed these systems performing their functions admirably.

After testing and several hours before the project demonstration, as the group assembled the final structure of the table, it was discovered that several of the transistors controlling current flow through the solenoids were burned-out and unable to function. Removal and replacement of the faulty parts resulted in a second burn-out. Further investigation revealed that the solenoids, dating back to the 1970's, carried a thick layer of corrosion, and may have rotted on the inside. As a result, while they continued to function properly when triggered via a direct switch, these solenoids possessed resistances quite different than the given specifications. This caused their power draw to burn out each and every transistor that that was connected. These solenoids were absolutely critical to the function of many of the larger parts on the table, including the pop bumpers, the drop targets, and most importantly, the flippers. The pinball game could not be played at all.

Several ideas were exchanged to deal with the problem. However none of them proved practical, due to extreme time constraints. Several would have worked admirably given slightly more time and money. The first option was to attempt to replace the damaged solenoids with newer parts, or similar ones in better repair. This was deemed impractical, as insufficient time remained to have the parts shipped online.

Second, coupling the transistor circuit to the solenoids with a set of sufficiently high-amperage 5-volt relays would have solved the problem, isolating the solenoids from the transistor logic circuit and protecting the smaller parts from being destroyed. However, relays are largely obsolete parts, and local supplies of electronics proved too small to meet our need. No available source possessed any of these components, and the relays proved too difficult to order and ship within the sharp time constraints.

A third, more practical option was also considered. By using a voltage divider circuit made from sturdy resistors to cut the power moving through the transistors to a fraction, the group hoped to reduce the power load to the point where several transistors could have split the power into manageable loads, but the resistors too, proved impossible to secure before the demonstration.

The flaw in this portion of the project is not truly a devastating one, owing more to faulty components and poor timing than bad design. Estimates of the cost to repair the glitch run less than \$50.00, a week to ship the necessary components, and an hour or two of labor installing

them. It is also worth noting that more modern solenoids consume much less power than the inferior parts used on the SAMMIES prototype and are specifically designed to better interface with delicate transistor circuitry. A well-financed attempt to produce a SAMMIES table would not use such old parts, and would be extremely unlikely to suffer from a similar problem.

It should also be considered that the interface itself, which was the main thrust of the SAMMIES project, functioned correctly in every way by the time of the demonstration. When struck with the intended inputs from a power supply rather than the broken sensors on the table, the chips performed their functions exactly as designed. Likewise, the LABVIEW simulation of the microchip logic and the wireless systems had been completely debugged by the time of the presentation and responded correctly when fed inputs by the VI front panel rather than the hardware. Because this method of input was used, the simple formatting procedure to convert the RS-232 input to the point value to add to the player's score was left out. It was not needed for the demonstration.

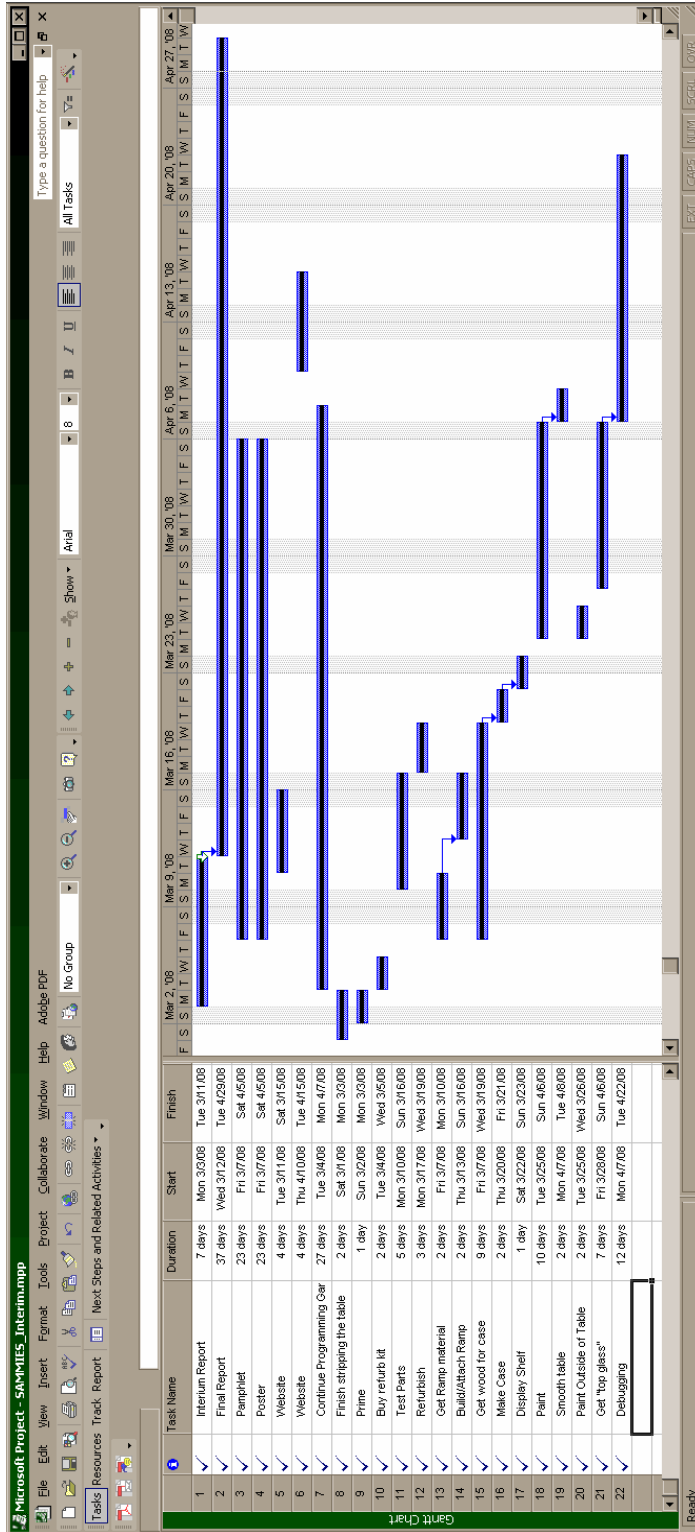
II.4. Financial Budget

Playfield	\$188
Electrical Components (MOSFETs, wire, connectors, etc.)	\$38
USB-Serial converter	\$40
Refurbish Kit	\$88
Microcontrollers	\$0
Microcontroller Programmer (rush shipped)	\$150
MAX232	\$1
Case materials (wood)	\$0
Toys and decorations (including paint)	\$30
Power supplies	\$0
Microcontroller programming license	\$80
TOTAL	\$615

The microcontroller programmer malfunctioned two weeks before the day of the design fair. This cost would have not occurred if not for this malfunction as the group already had a programmer before the project started.

II.5. Project Schedule

S.A.M.M.I.E.S. Gantt Chart:



III. Conclusion

Even though there were mechanical problems that prevented the table from physically working, all of the software needed to create the table was completed and the project was a success. A pinball table that is affordable enough for an average person to own in their home was created.

In order to improve this product for a future project it is highly recommended that new pinball parts are used rather than refurbishing an old pinball table. It is also recommended that a PIC16F97J60 microchip is used because it will cut down costs slightly. Because microcontroller is common to hobbyists, a future approach could aim toward an open-source pinball table which the SAMMIES hardware is required. Remote diagnostics can be implemented in a future prototype. The SAMMIES project was a resounding success and is the foundation for many more improvements, both in business and technology.

IV. References

- 1) <http://www.sternpinball.com>
- 2) <http://www.ipdb.org/machine.cgi?id=1133>
- 3) Discovery Science Channel's "How It's Made" Season 9, Episode 2; Airdate: September 14, 2007
- 4) <http://www.microchip.com/>
- 5) <http://www.sourceboost.com/>
- 6)
http://instruct1.cit.cornell.edu/courses/ee476/FinalProjects/s2007/afs26_hsb6/afs26_hsb6_pinball_machine/index.html
- 7) <http://www.woodweb.com/>
- 8) <http://www.thesolenoidcompany.com/>
- 9) Various offers on <http://www.ebay.com/>

V. Appendices

V-1 Appendix A: Sensor Poller (USART - no 9 bit).c

```

#include <system.h>
#include ".\Poller Subroutines.h"

//      Written for PIC16F887
#pragma CLOCK_FREQ 4000000

#pragma DATA _CONFIG1, _DEBUG_OFF & _LVP_OFF & _FCMEN_OFF &
_BOR_ON & _CPD_OFF & _CP_OFF & _MCLRE_OFF & _PWRTE_ON &
_WDT_ON & _INTOSCIO
#pragma DATA _CONFIG2, _WRT_OFF & _BOR40V

/*
Usage:

Now 9600 baud (Converter compatibility)
9 bit send
9th bit unused (enabled for compatibility with driver chip)

Data byte:
    Sends ID number of sensor struck
    ID number = (port multiplier) * ( bit number of port) + 1 )

Port multipliers:
    porta = 1
    portb = 2
    portc = 3
    portd = 4
    porte = 5
*/

// Define symbols
#define txif pir1.4
#define gie intcon.7
#define peie intcon.6
#define tmr2if pir1.1
#define tmr2ie pie1.1
#define rcif pir1.5
#define brgh txsta.2
#define brg16 baudctl.3
#define sync txsta.4
#define txen txsta.5
#define tx9 txsta.6
#define tx9d txsta.0
#define rx9 rcsta.6
#define rcie pie1.5

```

```

#define spen rcsta.7
#define adden rcsta.3
#define cren rcsta.4

unsigned char polldata[5];

/*
void interrupt (void) {
    if (tmr2if) {
        static unsigned char blinkcount = 0;
        tmr2if = 0;
        if ((++blinkcount) == 100) {
            porte ^= 0x01;
            blinkcount = 0;
        }
    }
}
*/
//t2con |= 0x04; to turn on timer

void main (void) {
    startup();
    initUSART();
    //setupblinktimer();

    gie = 1;

    unsigned char mycounter = 0;

    while (1) { // Active polling required since more than portb is to
be polled
        polldata[0] = porta;
        polldata[1] = portb;
        polldata[2] = (portc & 0x3F); // Don't read USART pins
        polldata[3] = portd;
        polldata[4] = (porte & 0x07); // Only three pins exist

        for (mycounter = 0; mycounter < 5; mycounter++) { // Check polled
data
            if (polldata[mycounter]) {
                transmitdata(polldata[mycounter], mycounter);
            }
            // Send detected data
            while (polldata[mycounter]);
            // Wait for ball to roll off sensor
        }
    }
}

```

V-2. Appendix B: Poller Subroutines.h

```
#ifndef _POLLER_SUBROUTINES_H_
#define _POLLER_SUBROUTINES_H_

void startup (void);

void initUSART (void);

unsigned char findpin (unsigned char collected, unsigned char byteme);

void transmitdata (unsigned char collected, unsigned char byteme);

void setupblinktimer (void);

#endif // _POLLER_SUBROUTINES_H_
```

V-3. Appendix C: Poller Subroutines.c

```

#include <system.h>

// Define symbols
#define txif pir1.4
#define gie intcon.7
#define peie intcon.6
#define tmr2if pir1.1
#define tmr2ie pie1.1
#define rcif pir1.5
#define brgh txsta.2
#define brg16 baudctl.3
#define sync txsta.4
#define txen txsta.5
#define tx9 txsta.6
#define tx9d txsta.0
#define rx9 rcsta.6
#define rcie pie1.5
#define spen rcsta.7
#define adden rcsta.3
#define cren rcsta.4

void startup (void) {
    trisa = 0xFF;
    trisb = 0xFF;
    trisc = 0b10111111;
    trisd = 0xFF;
    trise = 0xFF;

    porta = 0x00;
    portb = 0x00;
    portc = 0x00;
    portd = 0x00;
    porte = 0x00;

    ansel = 0x00;           // disable analog input on porta
    anselh = 0x00;         // disable analog input on portb
    wpub = 0x00;           // disable weak pull-ups on portb
}

void initUSART (void) {
    spbrgh = 0;             // set baud rate to 9600
    spbrg = 25;            // set baud rate to 9600
    brgh = 0;              // set baud rate to 9600
    brg16 = 1;             // set baud rate to 9600
    sync = 0;              // enable asynchronous mode
    spen = 1;              // enable serial port
    gie = 0;               // ensure all interrupts are disabled until setup
complete
    tx9 = 0;               // disable 9-bit mode (for address detection)
    txen = 1;              // enable data transmission circuitry

```

S.A.M.M.I.E.S. Pinball Table, C-2

```

        tx9d = 0;                // 9th bit will always read 0
        peie = 1;                // enable peripheral interrupts (in case it is needed
later)
    }

    unsigned char findpin (unsigned char collected, unsigned char byteme) {
        unsigned char testpin = 0;

        for (testpin = 0; testpin < 8; testpin++) {    // Find pin detected
            if (collected & (1 << testpin)) {
                return testpin;
            }
        }
    }

    void transmitdata (unsigned char collected, unsigned char byteme) {
        unsigned char bytetosend = 0;

        bytetosend = (byteme + 1) * ( findpin(collected, byteme) + 1 );

        while (! txif);        // Wait to transmit
        txreg = bytetosend;
        // Normally have to wait an instruction cycle to poll TXIF after writing to
TXREG
        // Since this is the end of the subroutine, this doesn't matter in this case
    }

    void setupblinktimer (void) {
        // 1/T = Fosc * Prescaler * 1/(Period Register) * Postscaler
        // For T = 0.01s, prescaler = 1/16, Period Register = 125, postscaler = 1/10 for
TMR2

        t2con = 0b01001010;    // Set up
        pr2 = 125;
        tmr2 = 0;
        tmr2ie = 1;            // Enable the timer as an interrupt
    }

```

V-4. Appendix D: Driver (USART - no 9 bit).c

```

#include <system.h>
#include ".\USART stuff.h"
#include ".\solenoidSubroutines.h"
#include ".\lightSubroutines.h"

#pragma CLOCK_FREQ 4000000

#pragma DATA _CONFIG1, _DEBUG_OFF & _LVP_OFF & _FCMEN_OFF &
_IESO_OFF & _BOR_ON & _CPD_OFF & _CP_OFF & _MCLRE_ON &
_PWRTE_ON & _WDT_OFF & _INTOSCIO
#pragma DATA _CONFIG2, _WRT_OFF & _BOR40V

// START COMPILE-TIME VARIABLES!

#define SOLENOID
// Uncomment above startment if this chip is to be a solenoid driver

const unsigned char SOLENOID_TIME = 5;// time to leave solenoid actuated in 10ms
increments
const unsigned char BLINK_TIME = 5;           // time of half-cycle of blinking
light in 10ms increments (50% duty cycle)

// END COMPILE-TIME VARIABLES!

/*
Usage:

All pins are active low because the solenoids must be done that way

Now 9600 baud (converter compatibility)
9 bit send disabled

Data byte:
    bit7 - chip address (0 - lights, 1 - solenoid)
    bit6..5 - port (a - 0, b - 1, c - 2, d - 3)
    bit4..2 - pin number (0 to 7)
    bit1 - set/clear
    bit0 - should this pin blink (1 - yes, 0 - no, don't care for solenoid)
*/

// Define symbols
#define txif pir1.4
#define gie intcon.7
#define peie intcon.6
#define tmr2if pir1.1
#define tmr2ie pie1.1

```

```

#define rcif pir1.5
#define brgh txsta.2
#define brg16 baudctl.3
#define sync txsta.4
#define txen txsta.5
#define tx9 txsta.6
#define tx9d txsta.0
#define rx9 rcsta.6
#define rcie pie1.5
#define spen rcsta.7
#define adden rcsta.3
#define cren rcsta.4
#define t2if pir1.1

void interrupt (void) {
    gie = 0;                // disable interrupts while interrupt is processing

    if (rcif) {             // check for if an address was received
        #ifdef SOLENOID
            receivedMessageSolenoid();
        #else
            receivedMessageLight();
        #endif
    }

    if (t2if) {             // Timer interrupt
        t2if = 0;
        #ifdef SOLENOID
            tripTimerSolenoid();
        #else
            tripTimerLight();
        #endif
    }

    gie = 1;                // Re-enable interrupts
}

void main (void) {
    #ifdef SOLENOID
        startup(1);
    #else
        startup(0);
    #endif

    initUSART();

    setupblinktimer();

    #ifndef SOLENOID
        t2con |= 0x04;      // Start timer
    #endif

    peie = 1;              // enable peripheral interrupts
    gie = 1;                // Turn on all interrupts
}

```

```
    while (1);           // Wait for interrupt  
}
```

V-5. Appendix E: USART stuff.h

```
#ifndef _USART_STUFF_H_
#define _USART_STUFF_H_

void startup (unsigned char isSolenoid);

void initUSART (void);
void setupblinktimer (void);

#endif // _USART_STUFF_H_
```

V-6. Appendix F: USART stuff.c

```

#include <system.h>

// Define symbols
#define txif pir1.4
#define gie intcon.7
#define peie intcon.6
#define tmr2if pir1.1
#define tmr2ie pie1.1
#define rcif pir1.5
#define brgh txsta.2
#define brg16 baudctl.3
#define sync txsta.4
#define txen txsta.5
#define tx9 txsta.6
#define tx9d txsta.0
#define rx9 rcsta.6
#define rcie pie1.5
#define spen rcsta.7
#define adden rcsta.3
#define cren rcsta.4
#define txie pie1.4

void startup (unsigned char isSolenoid) {
    trisa = 0x00;
    trisb = 0x00;
    trisd = 0x00;
    trise = 0xFF;

    switch (isSolenoid) {
        case 1:
            trisc = 0xC0;           // Set USART pins for input to not
interfere with parallel chips
            porta = 0xFF;
            portb = 0xFF;
            portc = 0xFF;
            portd = 0xFF;
            porte = 0xFF;
            break;
        case 0:
            trisc = 0x80;
            porta = 0x00;
            portb = 0x00;
            portc = 0x00;
            portd = 0x00;
            porte = 0x00;
            break;
    }

    ansel = 0x00;           // disable analog input on porta
    anselh = 0x00;         // disable analog input on portb
    wpub = 0x00;           // disable weak pull-ups on portb

```

```

        //oscon = 0b01101000;
    }

    void initUSART (void) {
        spbrgh = 0;           // set baud rate to 9600
        spbrg = 25;          // set baud rate to 9600
        brgh = 0;            // set baud rate to 9600
        brg16 = 1;          // set baud rate to 9600
        sync = 0;           // enable asynchronous mode
        spen = 1;           // enable serial port
        gie = 0;            // ensure all interrupts are disabled until setup
    complete
        rcie = 1;           // enable received data interrupt
        rx9 = 0;            // disable 9-bit mode (for address detection)
        //adden = 1;        // enable address detection
        cren = 1;           // enable data reception
        tx9 = 0;            // disable 9-bit mode (for address detection)
        txen = 0;           // disable data transmission circuitry
        txie = 0;           // disable transmission interrupt
        tx9d = 0;           // 9th bit will always read 0
        peie = 1;          // enable peripheral interrupts (in case it is needed
    later)
    }

    void setupblinktimer (void) {
        // 1/T = Fosc * Prescaler * 1/(Period Register) * Postscaler
        // For T = 0.01s, prescaler = 1/16, Period Register = 125, postscaler = 1/10 for
    TMR2

        t2con = 0b01001010; // Set up
        pr2 = 125;
        tmr2 = 0;
        tmr2ie = 1;         // Enable the timer as an interrupt
    }

```

V-7. Appendix G: solenoidSubroutines.h

```
#ifndef _SOLENOIDSUBROUTINES_H_
#define _SOLENOIDSUBROUTINES_H_

void receivedMessageSolenoid (void);
void tripTimerSolenoid (void);

#endif // _SOLENOIDSUBROUTINES_H_
```

V-8. Appendix H: solenoidSubroutines.c

```

#include <system.h>

extern const unsigned char SOLENOID_TIME;

void receivedMessageSolenoid (void) {
    unsigned char databuffer = rcreg;
    if (databuffer & 0x80) { // check address
        switch ( (databuffer >> 5) & 0x03 ) {
            case 0:
                porta &= (0 << ((databuffer >> 2) & 0x07)); // Active
                break;
            case 1:
                portb &= (0 << ((databuffer >> 2) & 0x07));
                break;
            case 2:
                portc &= (0 << ((databuffer >> 2) & 0x07));
                break;
            case 3:
                portd &= (0 << ((databuffer >> 2) & 0x07));
                break;
        }
        tmr2 = 0;
        t2con |= 0x04; // Start timer
    }
}

void tripTimerSolenoid (void) {
    static unsigned char timercycle = 0;
    if (++timercycle == SOLENOID_TIME) {
        t2con &= ~0x04; //Stop timer
        timercycle = 0;
        porta = 0xFF; // Shut off all solenoids
        portb = 0xFF;
        portc = 0xFF;
        portd = 0xFF;
    }
}

```

V-9. Appendix I: lightSubroutines.h

```
#ifndef _LIGHTSUBROUTINES_H_
#define _LIGHTSUBROUTINES_H_

void receivedMessageLight (void);
void tripTimerLight (void);

#endif // _LIGHTSUBROUTINES_H_
```


S.A.M.M.I.E.S. Pinball Table, J-2

```

portc &= ~(1 << ((databuffer >> 2) &
0x07)));
// Shut off blinking
blinklist[2] &= ~(1 << ((databuffer >> 2) &
0x07)));
}
break;
case 3:
if (databuffer & 0x02) { // Set bit
portd |= (1 << ((databuffer >> 2) & 0x07));
if (databuffer & 0x01){ // Determine
blinklist[3] |= (1 << ((databuffer >>
2) & 0x07));
}
} else { // Clear
portd &= ~(1 << ((databuffer >> 2) &
0x07)));
// Shut off blinking
blinklist[3] &= ~(1 << ((databuffer >> 2) &
0x07)));
}
break;
}
}
}

void tripTimerLight (void) {
static unsigned char timercycle = 0;
if (++timercycle == BLINK_TIME) {
timercycle = 0;
porta ^= blinklist[0];
portb ^= blinklist[1];
portc ^= blinklist[2];
portd ^= blinklist[3];
}
}

```

V-11. Appendix K: serialtest.c

```

#include <system.h>

// Made for pic16f887

#pragma CLOCK_FREQ 4000000

#pragma DATA _CONFIG1, _DEBUG_OFF & _LVP_OFF & _FCMEN_OFF &
_IESO_OFF & _BOR_ON & _CPD_OFF & _CP_OFF & _MCLRE_ON &
_PWRTE_ON & _WDT_OFF & _INTOSCIO
#pragma DATA _CONFIG2, _WRT_OFF & _BOR40V

// Blinking LED on RD0

// Compile time variables
#define MAXMESSAGE 64

// Define symbols
#define txif pir1.4
#define gie intcon.7
#define peie intcon.6
#define tmr2if pir1.1
#define tmr2ie pie1.1
#define rcif pir1.5
#define brgh txsta.2
#define brg16 baudctl.3
#define sync txsta.4
#define txen txsta.5
#define tx9 txsta.6
#define tx9d txsta.0
#define rx9 rcsta.6
#define rcie pie1.5
#define spen rcsta.7
#define adden rcsta.3
#define cren rcsta.4
#define txie pie1.4

// Define global variables

unsigned char testarray[32];
unsigned char index = 0;
unsigned char blinkcount = 0;

void transmit (void) {
    index = 0;
    while (testarray[index]) {
        while (! txif);
        txreg = testarray[index++];
        nop();
    }
}

```

```

    index = 0;
    tmr2ie = 1;
}

inline void startup (void) {
    trisa = 0x00;
    trisb = 0x00;
    trisc = 0x80;
    trisd = 0x00;
    trise = 0x00;

    porta = 0x00;
    portb = 0x00;
    portc = 0x00;
    portd = 0x00;
    porte = 0x00;

    ansel = 0x00;           // disable analog input on porta
    anselh = 0x00;         // disable analog input on portb
    wpub = 0x00;           // disable weak pull-ups on portb

    //oscon = 0b01101000;
}

inline void initUSART (void) {
    spbrgh = 0;             // set baud rate to 9600
    spbrg = 25;            // set baud rate to 9600
    brgh = 0;              // set baud rate to 9600
    brg16 = 1;             // set baud rate to 9600
    sync = 0;              // enable asynchronous mode
    spen = 1;              // enable serial port
    gie = 0;               // ensure all interrupts are disabled until setup
complete
    rcie = 1;              // enable received data interrupt
    rx9 = 0;               // disable 9-bit mode (for address detection)
    //adden = 1;           // enable address detection
    cren = 1;              // enable data reception
    tx9 = 0;               // disable 9-bit mode (for address detection)
    txen = 1;              // enable data transmission circuitry
    txie = 0;
    tx9d = 0;              // 9th bit will always read 0
    peie = 1;              // enable peripheral interrupts (in case it is needed
later)
}

inline void setupblinktimer (void) {
    // 1/T = Fosc * Prescaler * 1/(Period Register) * Postscaler
    // For T = 0.01s, prescaler = 1/16, Period Register = 125, postscaler = 1/10 for
TMR2

    t2con = 0b01001010;    // Set up
    pr2 = 125;
    tmr2 = 0;
    tmr2ie = 1;            // Enable the timer as an interrupt

```

```

    t2con |= 0x04;
    txreg = 65;
}

void interrupt (void) {
    if (rcif) {
        /*index = 0;
        rcie = 0;
        testarray[index] = rcreg;
        if (testarray[index++] == 0x0A) {
            testarray[index] = 0;
            tmr2ie = 0;
            transmit();
        }
        rcie = 1;*/
        while (! txif);
        txreg = rcreg;
    }

    if (tmr2if) {
        tmr2if = 0;
        if ((++blinkcount) == 100) {
            portd ^= 0x01;
            blinkcount = 0;
        }
    }

    gie = 1;
}

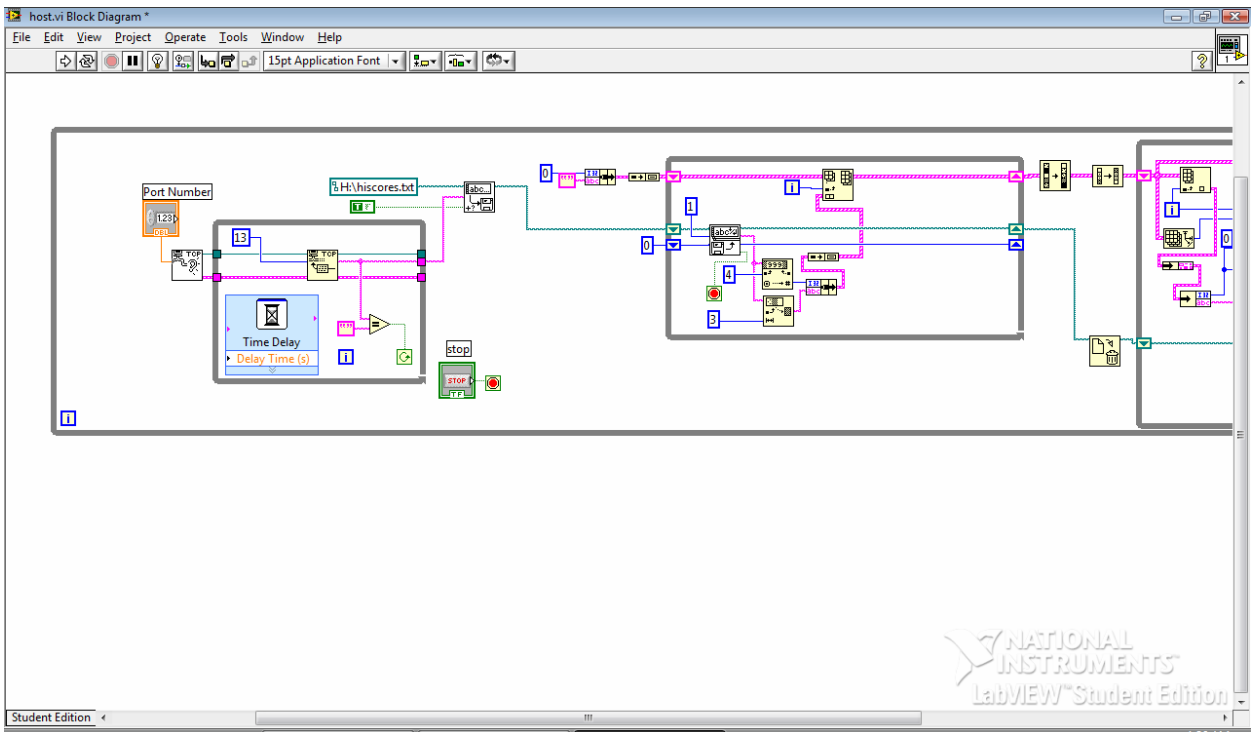
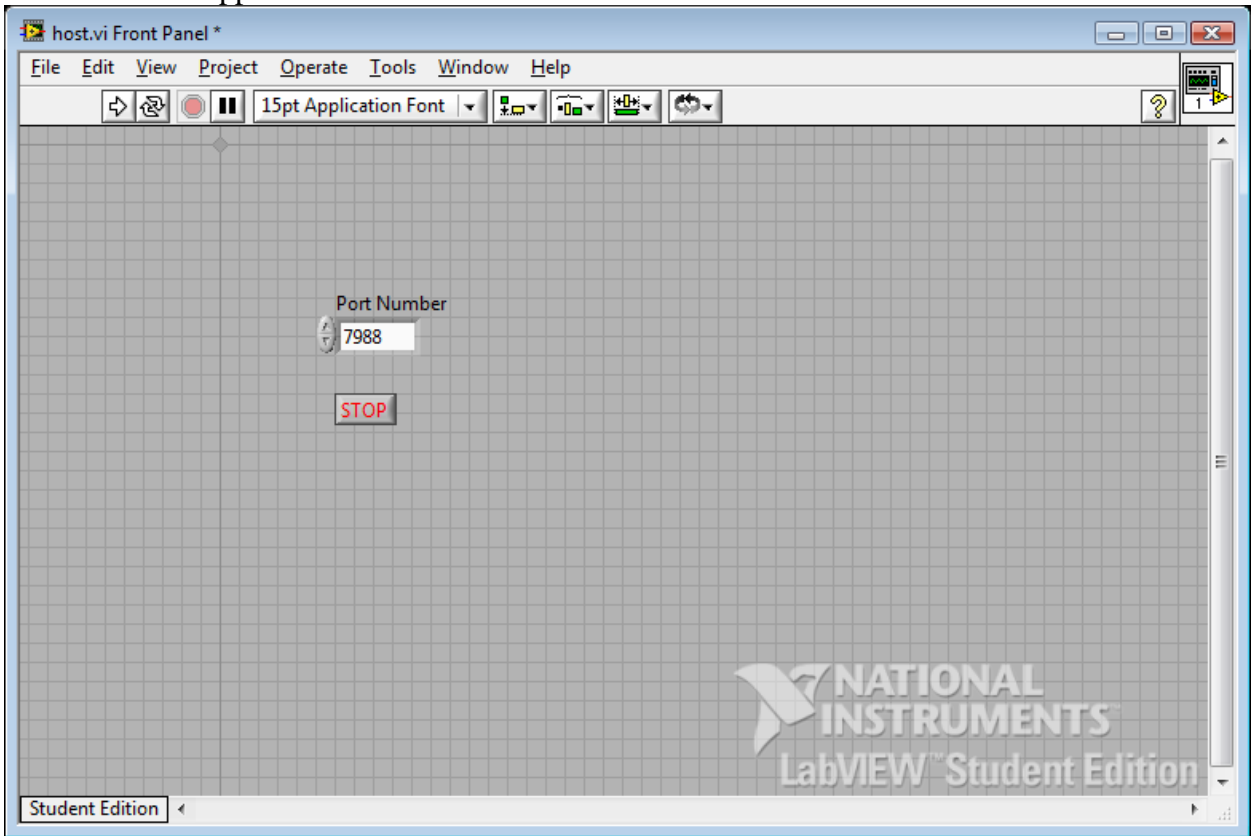
void main (void) {
    startup();
    initUSART();
    setupblinktimer();

    gie = 0;          // Turn on all interrupts

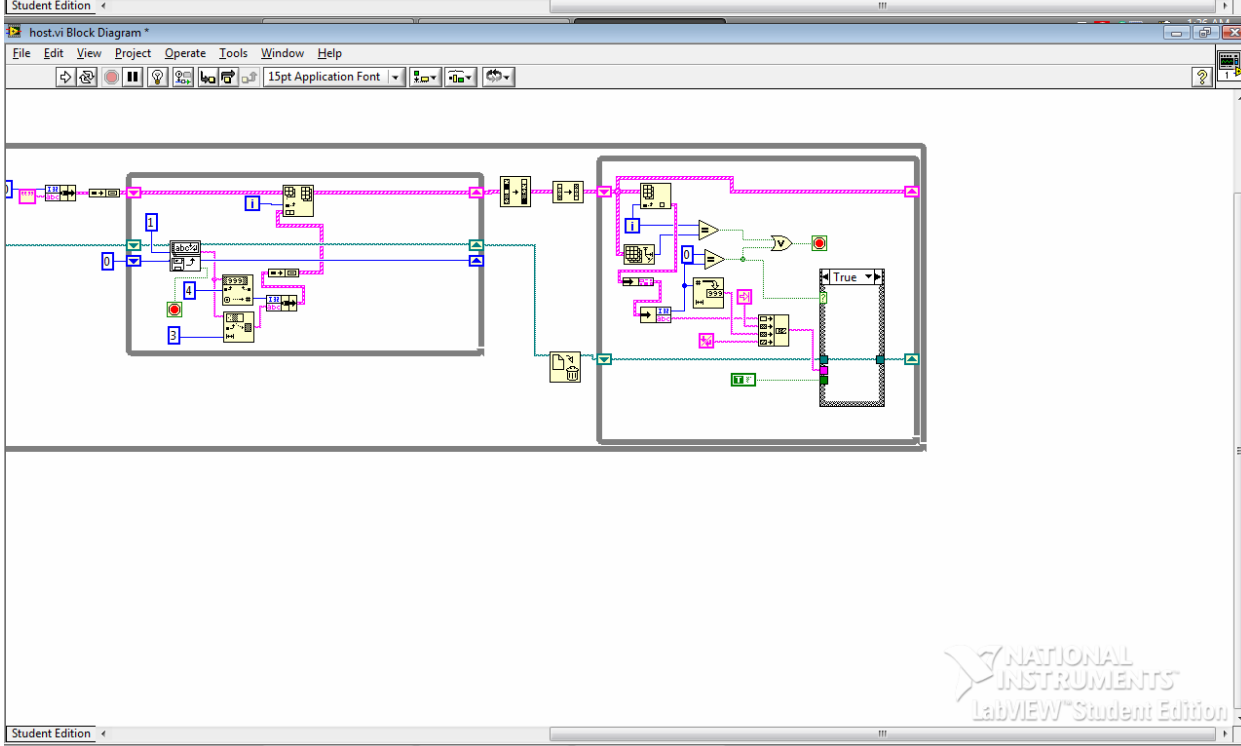
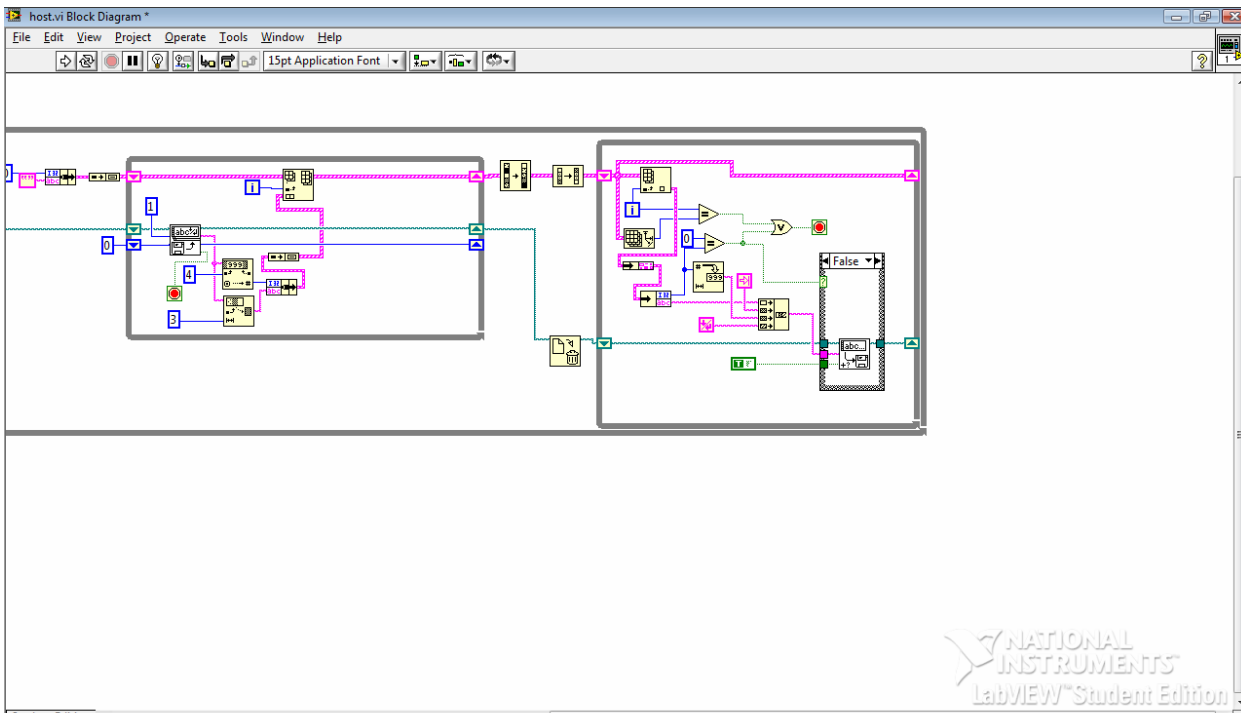
    while (1); /*
    {
        if (txif) {
            txreg = 0x65;
            delay_s (1);
        }
    }*/
}

```

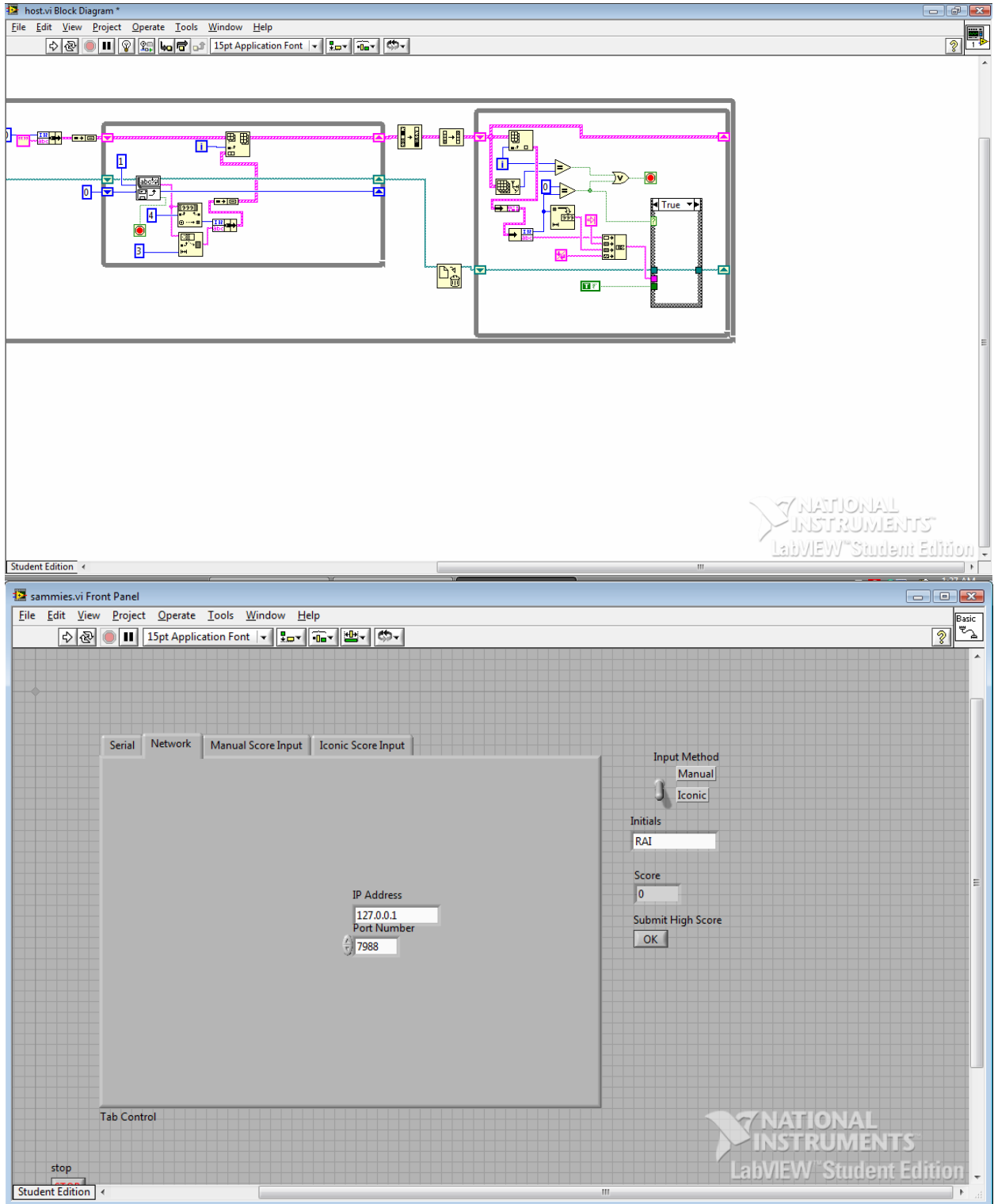
V-12. Appendix L: Host.vi



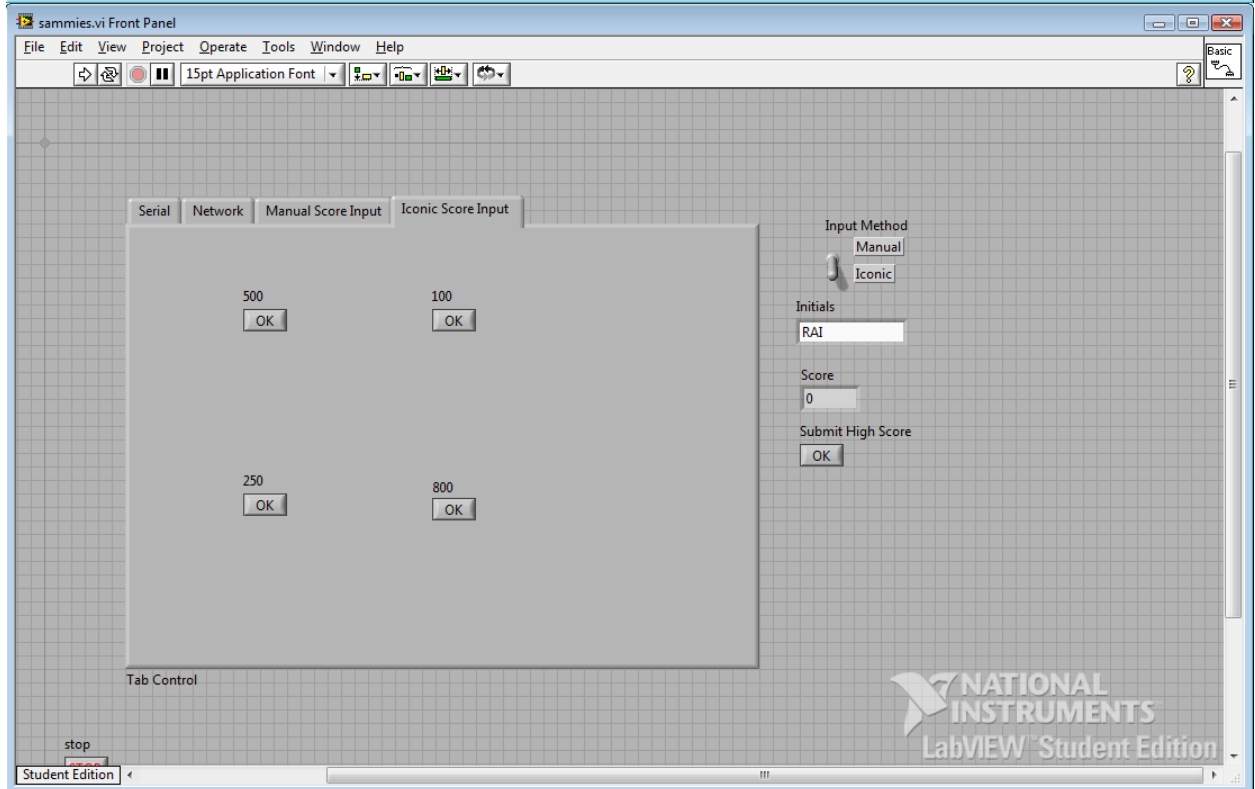
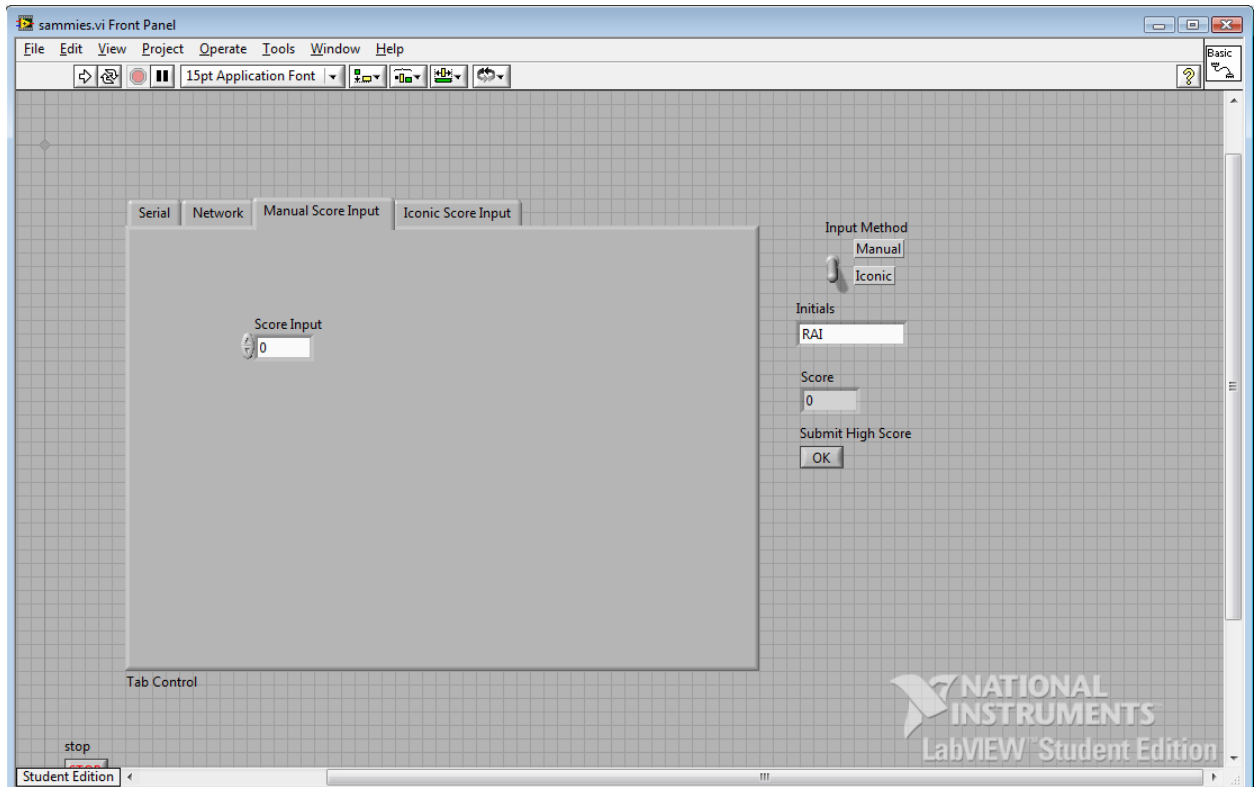
S.A.M.M.I.E.S. Pinball Table, L-2



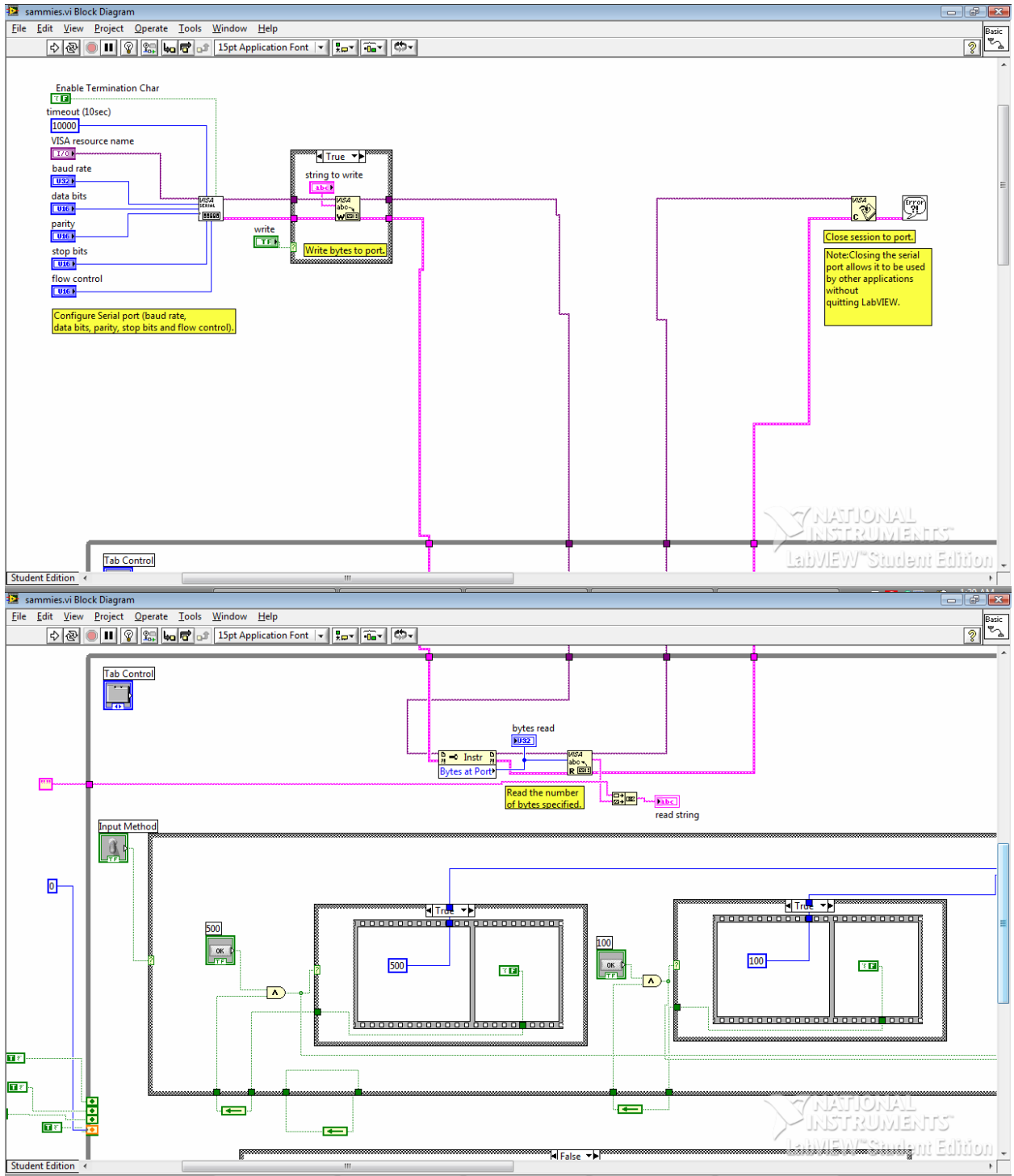
V-13. Appendix M: Sammies.vi (client)



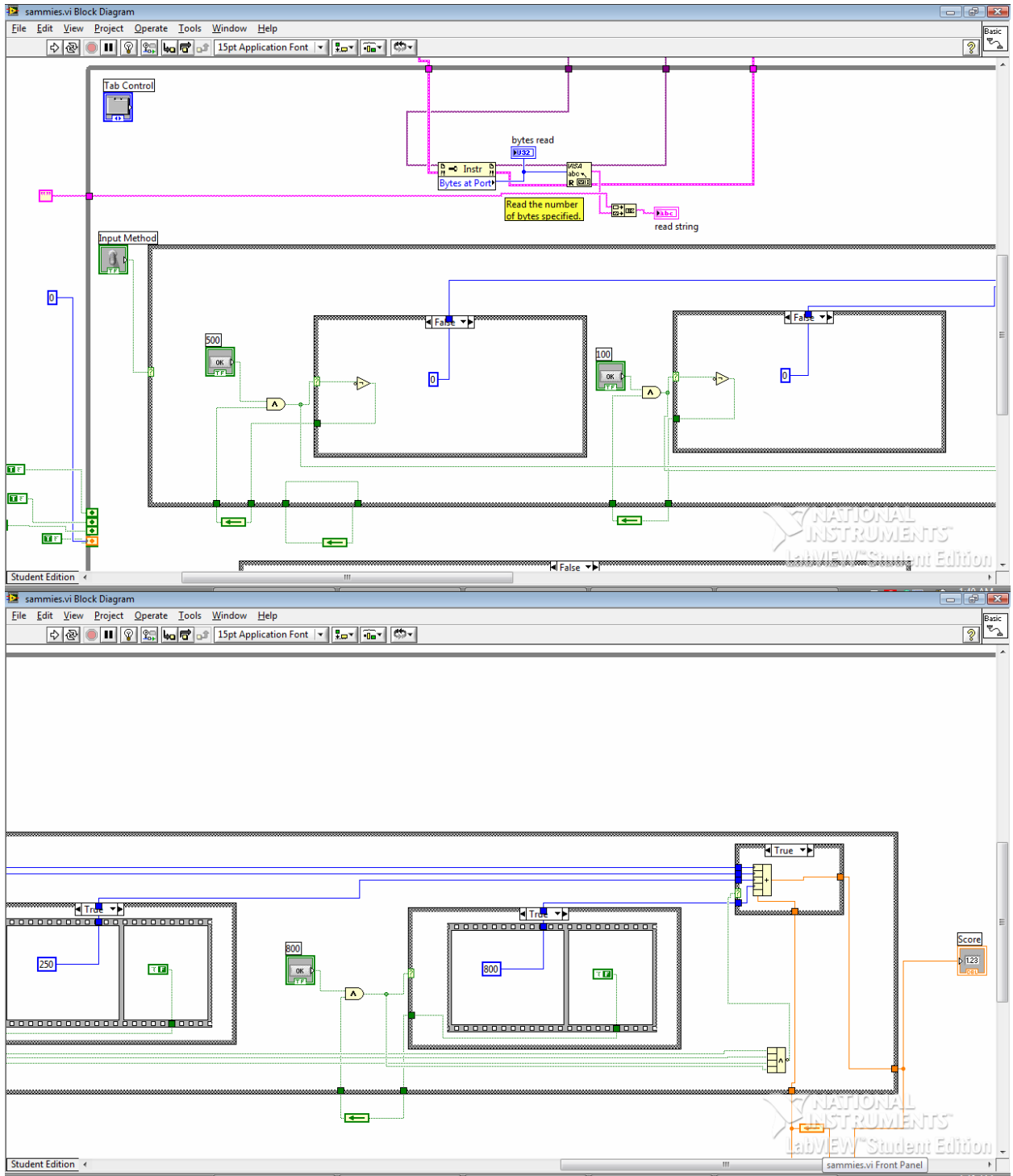
S.A.M.M.I.E.S. Pinball Table, M-2



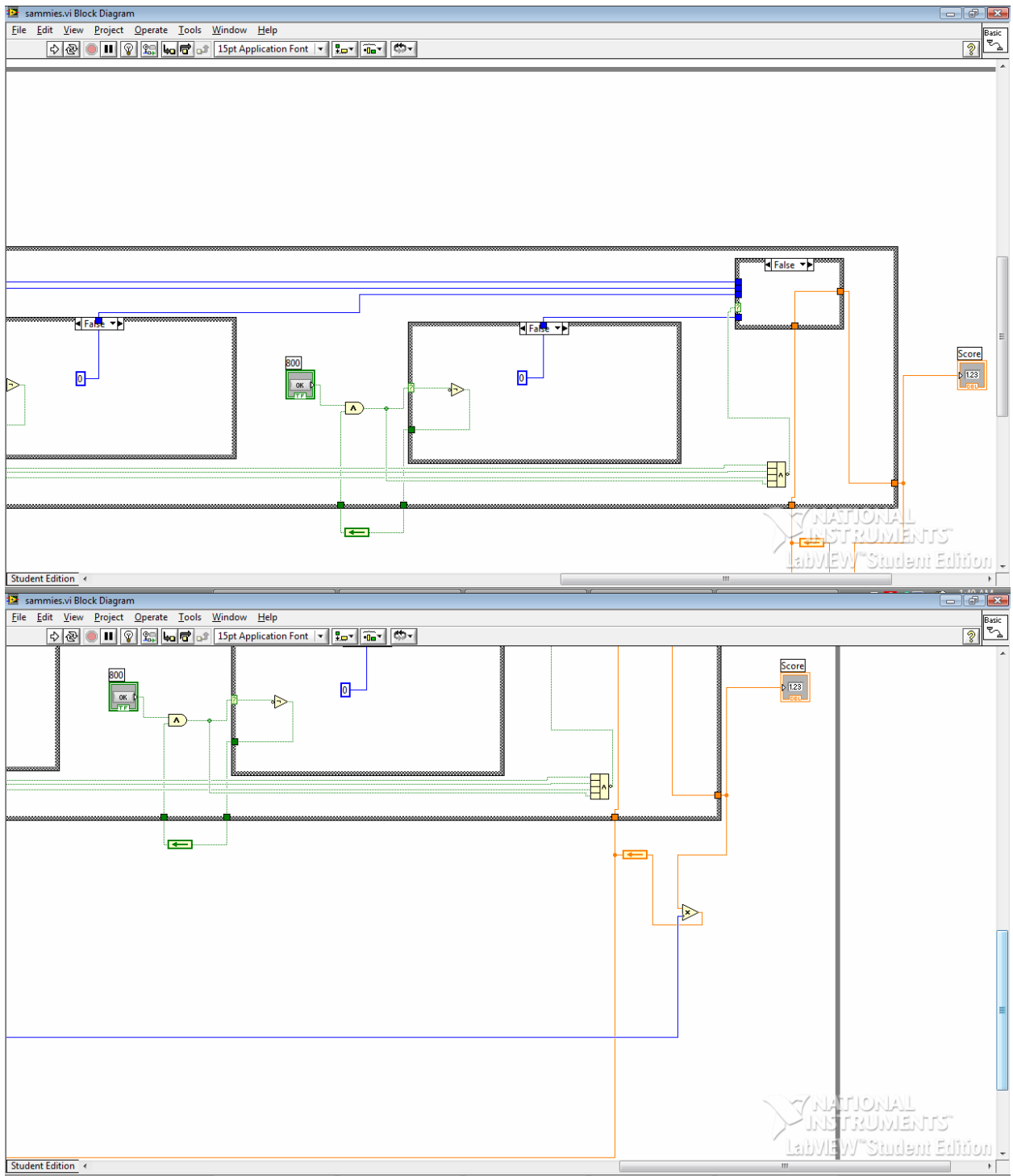
S.A.M.M.I.E.S. Pinball Table, M-3



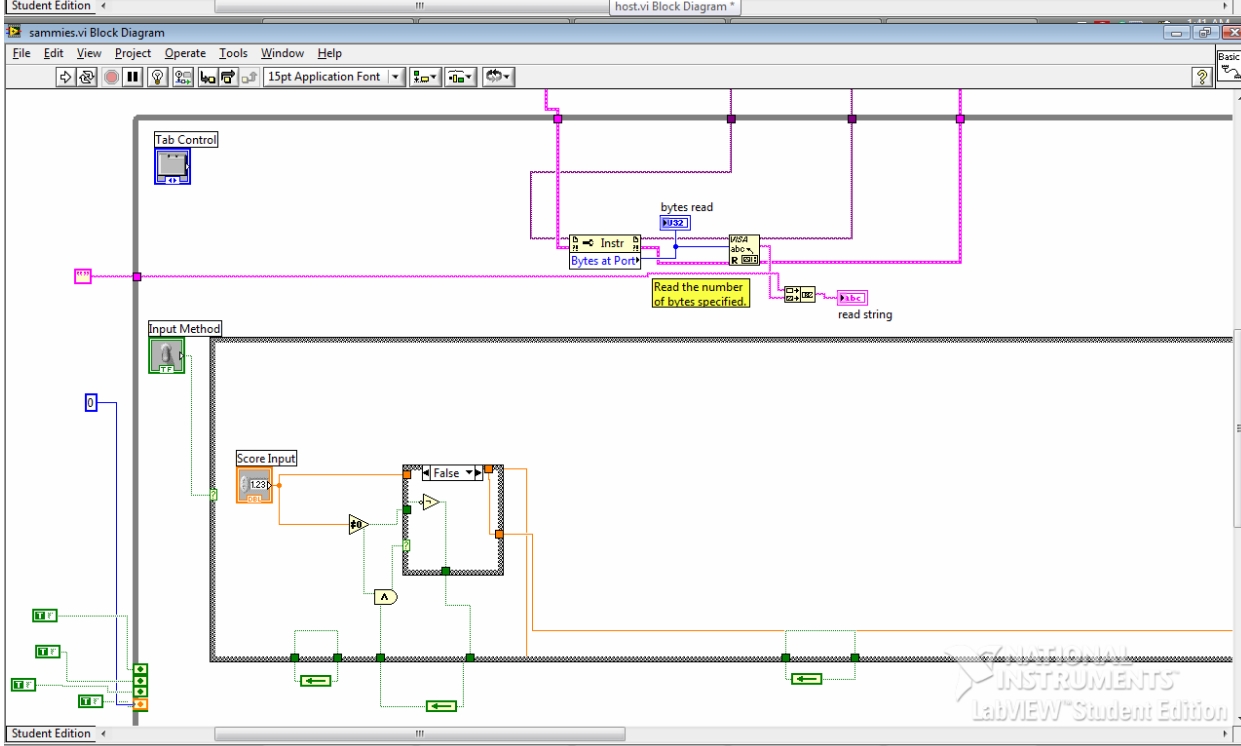
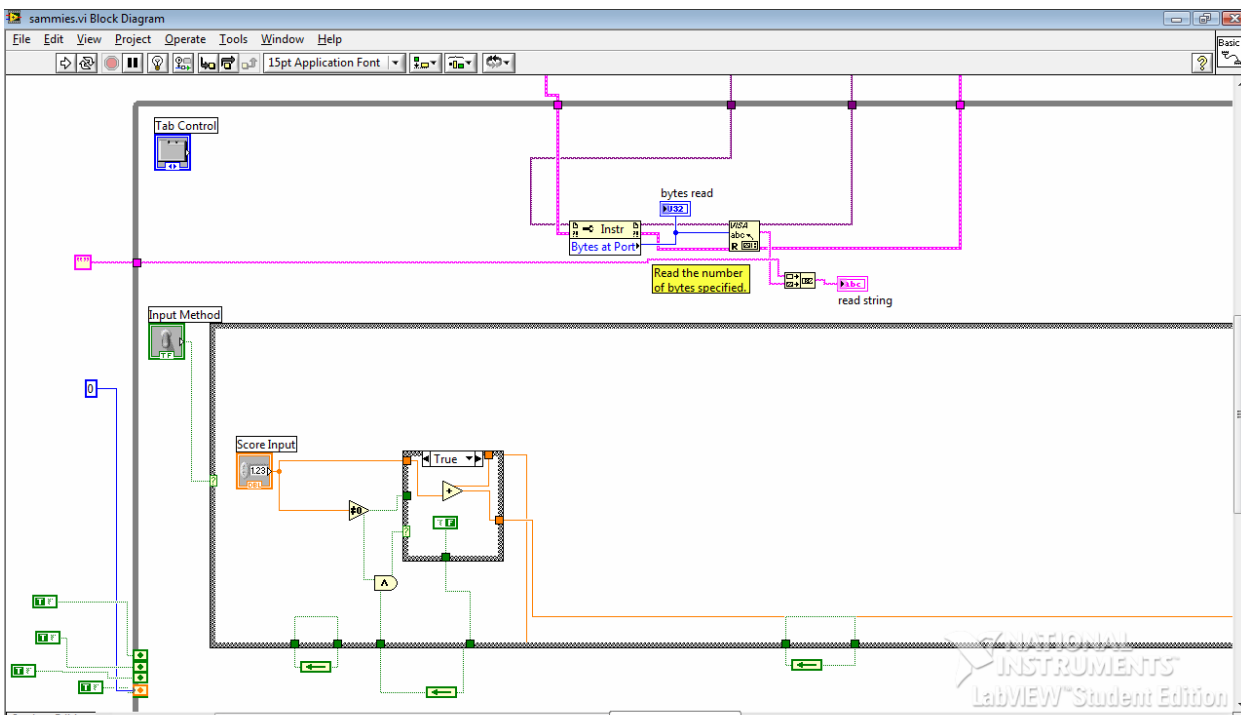
S.A.M.M.I.E.S. Pinball Table, M-4



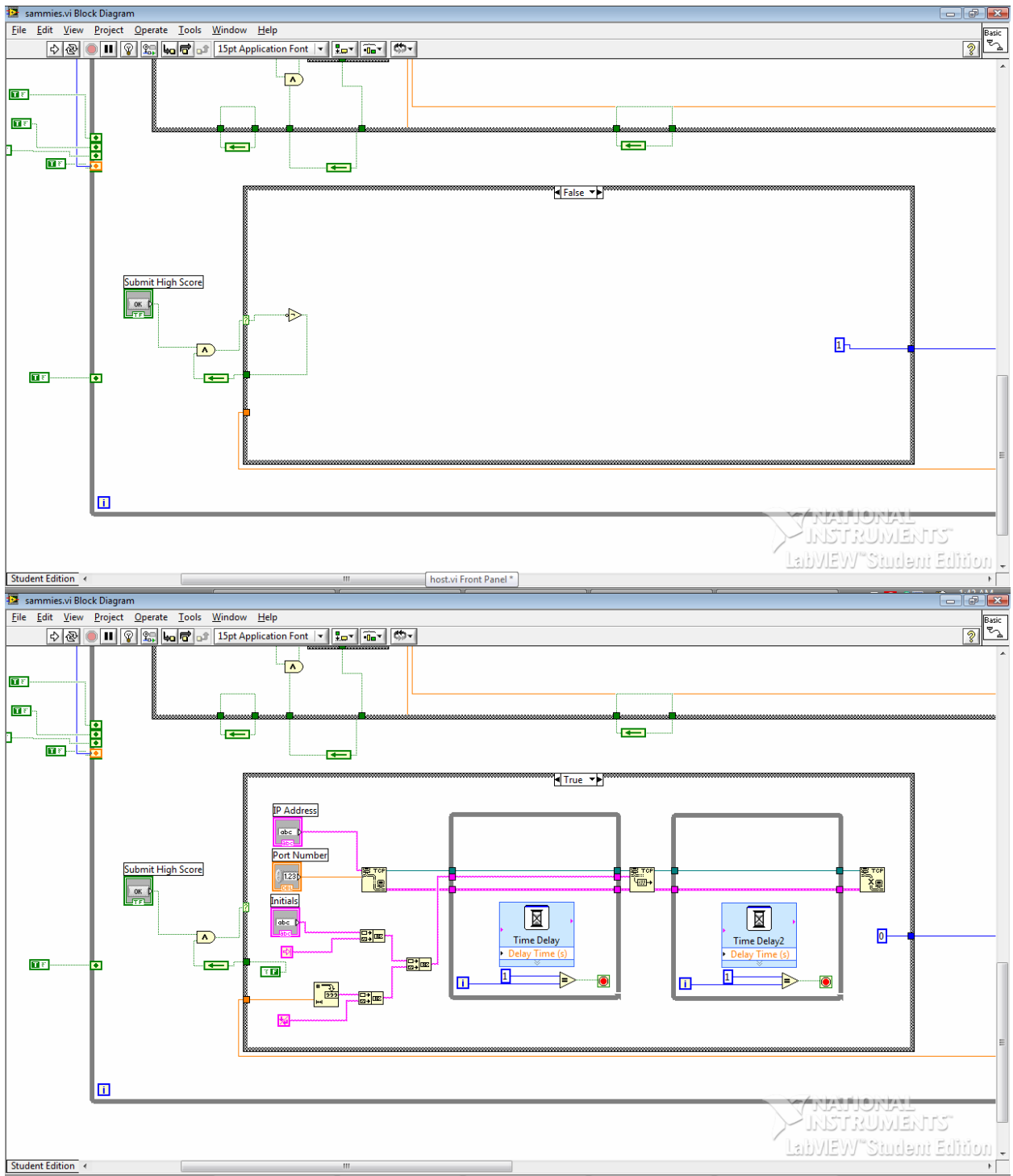
S.A.M.M.I.E.S. Pinball Table, M-5



S.A.M.M.I.E.S. Pinball Table, M-6



S.A.M.M.I.E.S. Pinball Table, M-7



S.A.M.M.I.E.S. Pinball Table, M-8

