

ME 461 LabVIEW #3

Due October 31st 2:00 PM

This task will be performed on a lab bench robot. Remember to power your robot with the programmable power supply with 12V, 3A.

Most of the work for this final task is to create the LabVIEW program pictured below. This LabVIEW program is an extension to the programs you created in LabVIEW assignment #2. If you have not completed that assignment, I recommend you finish that assignment first.

In LabVIEW assignment #2, you were shown how to draw lines/squares in a Picture Box. In the second exercise you were shown how to communicate using the TCPIP protocol over Ethernet. The TCPIP communication talked to an Echo Server that simply sent back whatever text was sent to it.

For this task you are going to take the experience you gained from LabVIEW assignment #2 and create (copy pictures below) a LabVIEW program that receives eight floating point numbers from your robot's red (F28379D) board at the periodic rate of every 250ms. These eight numbers are displayed in text boxes in your LabVIEW application. In addition, the first two numbers sent are assumed to be the X and Y coordinates of your robot car. The below given C code varies these X and Y points as if the robot was traveling in a circle. In Lab 6 and Lab 7, you will use this program to display in the LabVIEW Picture Box the actual X and Y location of your robot.

In order to have the robot send TCPIP Ethernet data wirelessly to LabVIEW, there is an addition microcontroller on the robot called the ESP32 that has Wi-Fi built in. The TMS320F28379D processor does not have Wi-Fi capability so the ESP32 needs to be added for this communication. You will see in the steps below that you will need to "telnet" into the ESP32 processor which is running an operating system called Nuttx. Easiest way to explain Nuttx is that it is a miniature Linux that can run on slower and small memory capacity microcontrollers. Once you telnet into Nuttx, you will run a C program that I wrote called "tcpLVCOM". This program waits for data either coming wirelessly from LabVIEW or coming serially through a UART serial port connected to the F28379D's UARTD (SCID). Summarizing the communication:

- If you click "Send" in your LabVIEW program, it sends eight floating point numbers over Ethernet/Wi-Fi to the tcpLVCOM application running in Nuttx. When tcpLVCOM receives the eight floats, it then sends the numbers to the red board over UART.
- Also, the given test code below sends eight floating point numbers through UARTD to the tcpLVCOM application every 250ms. When tcpLVCOM receives this new data over UART, it sends this data to LabVIEW over the same TCPIP/Wi-Fi port. Then these eight numbers are displayed in text boxes and the

first two numbers are also used to change the X and Y location of the robot that is drawn as a square in the LabVIEW picture box.

Perform the following steps:

1. Using LabVIEW exercise #2 as a guide, copy the LabVIEW program given in the below two pictures.

Notes for completing this exercise are as follows:

- a. Your robot's IP number is written on your robot car. The numbers written on our 12 robots range from 51 to 62. This number is the last number in your robot's IP 192.168.1.<yournumber>. If your robot has 55 written on it, your IP is 192.168.1.55. Also note that this new program has two while loops. One while loop is for sending eight floats to your robot and the other while loop is for receiving eight floats from your robot.
 - b. The Send code / Event structure from LabVIEW exercise #2 is a good starting point for the Send of this exercise reminding you about the Timeout Event and Stop Event. In this case however, the Send event structure only sends data to the robot. The receiving of data happens in the second while loop. Eight floats, converted to text, are sent and I add these \FD and \FF characters as start and stop characters. Make sure to right click on these pink string constants and select "\ Display Code." Also make sure there is only one "\ back slash in the string constant. This tells LabVIEW to send the hexadecimal number 0xFD and 0xFF instead of the ASCII characters 'F' and 'D'.
 - c. Make sure the Mechanical Action of both the Send and Stop buttons is set to "Switched When Pressed" by right clicking on the buttons.
 - d. After creating the "Size of Rectangle" control, set it to 20 and then right click on it and select "Data Operations->Make Current Value Default." This will reload 20 in this variable the next time you open this program.
 - e. To test your LabVIEW program you need to finish the remaining steps to setup both the red board and the ESP32.
2. In CCS create a new "LABstarter" project and rename it. Much of the code for communicating to and from LabVIEW, Nuttx, and the red board is given to you already in LABstarter. *If you are interested, the bulk of the communication code for the red board is in the UARTD receive interrupt function "RXDINT_recv_ready" in F28379dSerial.c.* The remaining code you need is given here:
 - a. Cut and paste the following global variables towards the top of your project's C file. For now you can use these variables below. But in the future you may want to give the eight LV variables more descriptive names.

```
float printLV1 = 0;
```

```

float printLV2 = 0;
float printLV3 = 0;
float printLV4 = 0;
float printLV5 = 0;
float printLV6 = 0;
float printLV7 = 0;
float printLV8 = 0;
extern uint16_t NewLVData;
extern float fromLVvalues[LVNUM_TOFROM_FLOATS];
extern LVSendFloats_t DataToLabView;
extern char LVsenddata[LVNUM_TOFROM_FLOATS*4+2];
extern uint16_t newLinuxCommands;
extern float LinuxCommands[CMDNUM_FROM_FLOATS];

```

- b. Then setup Timer 0's interrupt function to be call every 1 ms.
- c. Copy the following code into Timer 0's interrupt function before the line of code that increments "numTimer0calls." Look as this code. "NewLVData" is a flag variable that is set for you when new data has been received from LabVIEW/Nuttx. When "NewLVData" is equal to one, copy the eight received floats into global variables you can use. Then also every 250 ms., this code sends eight floats up to Nuttx/LabVIEW. Look at the eight things we are sending. I use sin() and cos() and numTimer0calls to simulate X, Y coordinates. Then, so that we can see that the six other variables are being sent and received, I assigned them different fractions of "numTimer0calls." In the future you can change this code to upload the robot's actual X, Y coordinates and other data you want to send to LabVIEW.

```

if (NewLVData == 1) {
    NewLVData = 0;
    printLV1 = fromLVvalues[0];
    printLV2 = fromLVvalues[1];
    printLV3 = fromLVvalues[2];
    printLV4 = fromLVvalues[3];
    printLV5 = fromLVvalues[4];
    printLV6 = fromLVvalues[5];
    printLV7 = fromLVvalues[6];
    printLV8 = fromLVvalues[7];
}

if((numTimer0calls%250) == 0) {
    DataToLabView.floatData[0] = 3.0*sin(2*PI*.05*numTimer0calls*0.001);
    DataToLabView.floatData[1] = 3.0*cos(2*PI*.05*numTimer0calls*0.001);
    DataToLabView.floatData[2] = (float)numTimer0calls*.001;
    DataToLabView.floatData[3] = 2.0*((float)numTimer0calls)*.001;
    DataToLabView.floatData[4] = 3.0*((float)numTimer0calls)*.001;
    DataToLabView.floatData[5] = (float)numTimer0calls;
    DataToLabView.floatData[6] = (float)numTimer0calls*4.0;
    DataToLabView.floatData[7] = (float)numTimer0calls*5.0;
    LVsenddata[0] = '*'; // header for LVdata
    LVsenddata[1] = '$';
    for (int i=0;i<LVNUM_TOFROM_FLOATS*4;i++) {
        if (i%2==0) {
            LVsenddata[i+2] = DataToLabView.rawData[i/2] & 0xFF;
        } else {
            LVsenddata[i+2] = (DataToLabView.rawData[i/2]>>8) & 0xFF;
        }
    }
    serial_sendSCID(&SerialD, LVsenddata, 4*LVNUM_TOFROM_FLOATS + 2);
}

```

- d. In main()'s while(1) loop change your serial_printf statement so that it prints all eight values sent from LabVIEW.

```
serial_printf(&SerialA, "%.3f,%.3f,%.3f,%.3f,%.3f,%.3f,%.3f,%.3f\r\n",
printLV1,printLV2,printLV3,printLV4,printLV5,printLV6,printLV7,printLV8);
```

- e. Build and download this code to your robot. Power the robot with 12V, 3A from the programmable power supply in lab. In CCS run your red board's code and open a Tera Term session to see what is printing. You should see the eight variable values sent from LabVIEW printed to Tera Term. They should be all zero as we have not setup the communication yet.
- f. When the robot is powered on, the ESP32 board is also powered. It takes about 10 to 15 seconds for Nuttx to boot on the ESP32 board. After waiting 15 seconds, open a CMD prompt in Windows and test that Nuttx is ready by pinging your robot's IP:

```
ping 192.168.1.<your robot's IP number> (i.e., ping 192.168.1.55)
```

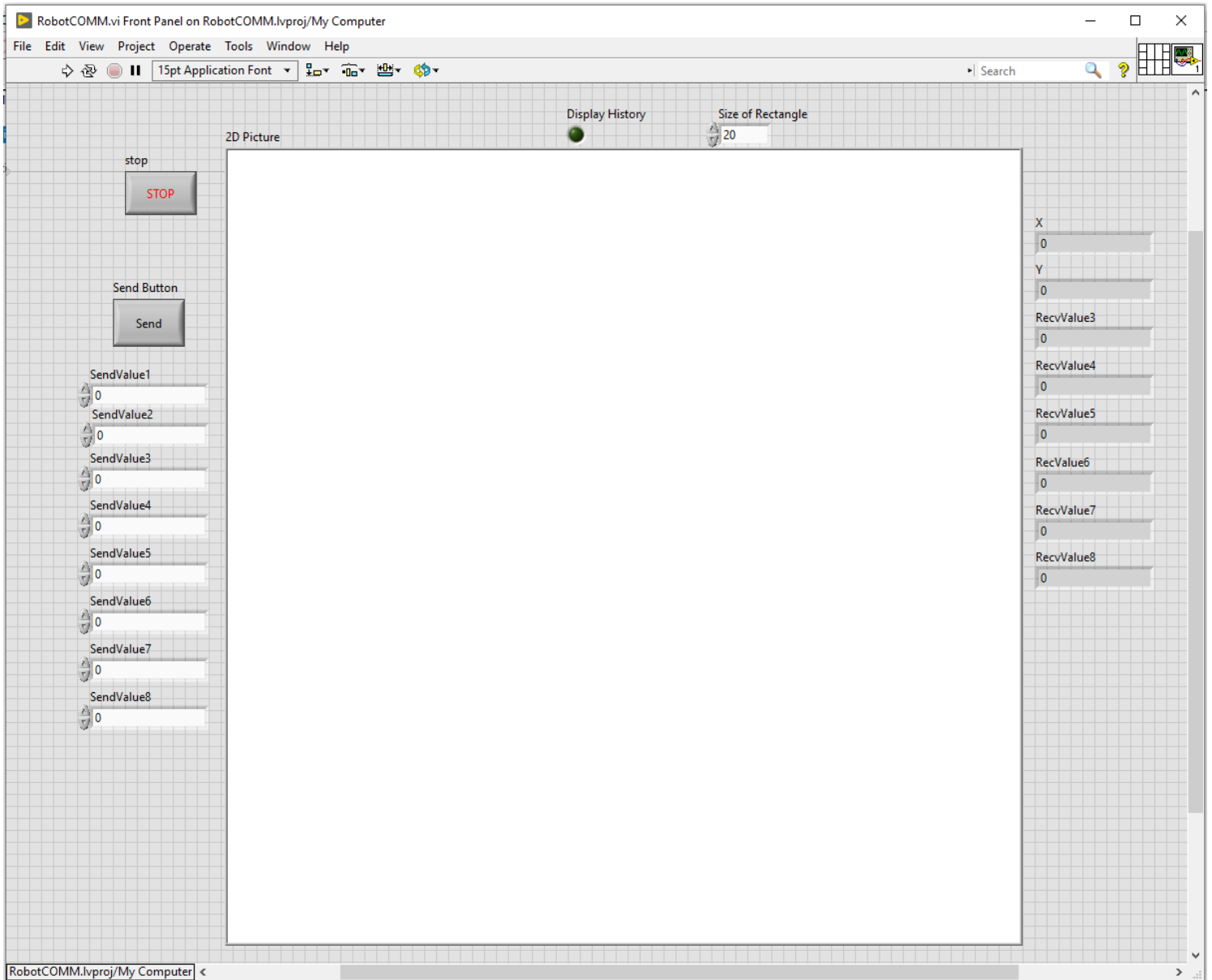
You should receive a response. Press "<CTRL> + C" to end the ping. If you do not receive a response with ping, check that you typed the correct IP. Also, you can try pressing the small "EN" button on the robot's ESP32 board, waiting 15 seconds, and trying again.
- g. Then from the Windows CMD prompt connect to Nuttx over telnet by typing:

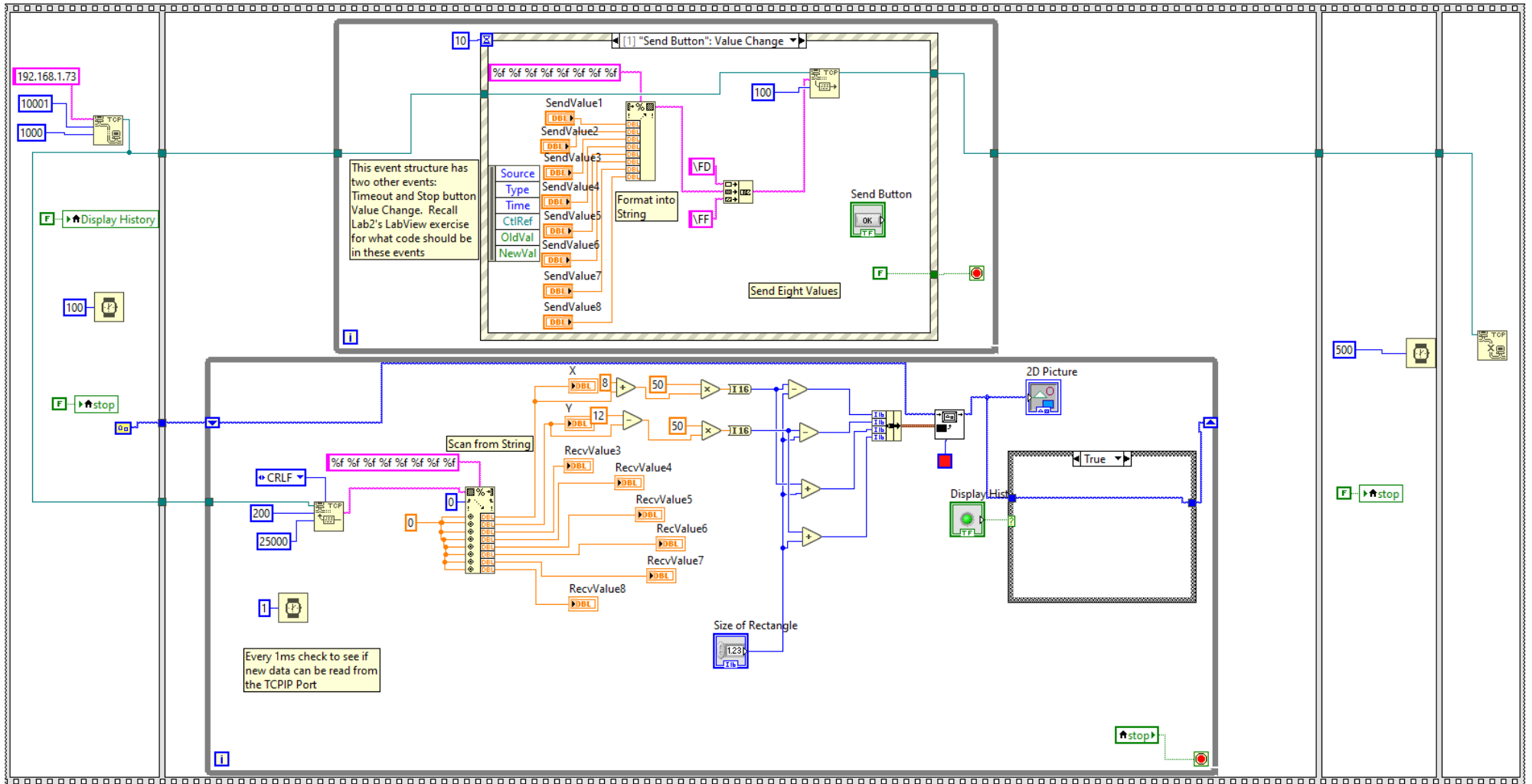
```
putty -telnet 192.168.1.<yourRobotsIPnumber>
```

This should give you a Nuttx Shell, nsh, prompt. At this prompt type:

```
tcpLVCOM
```

The tcpLVCOM application is now waiting for your LabVIEW program to connect.
- h. Make sure your red board's code is running in CCS and printing to Tera Term.
- i. Run your LabVIEW program.
- j. If everything is working, you should see your robot square moving in the Picture Box of LabVIEW. In the Nuttx shell, the values being received from the red board should be printing.
- k. To test, download from LabVIEW to the robot. Type different floating point values in the eight send variables and press the "Send" button. You should see the printed values in Tera Term change to the values you typed in LabVIEW.
- l. **Demo this working to your TA or Instructor** and/or continue to the last steps listed after the two below pictures.





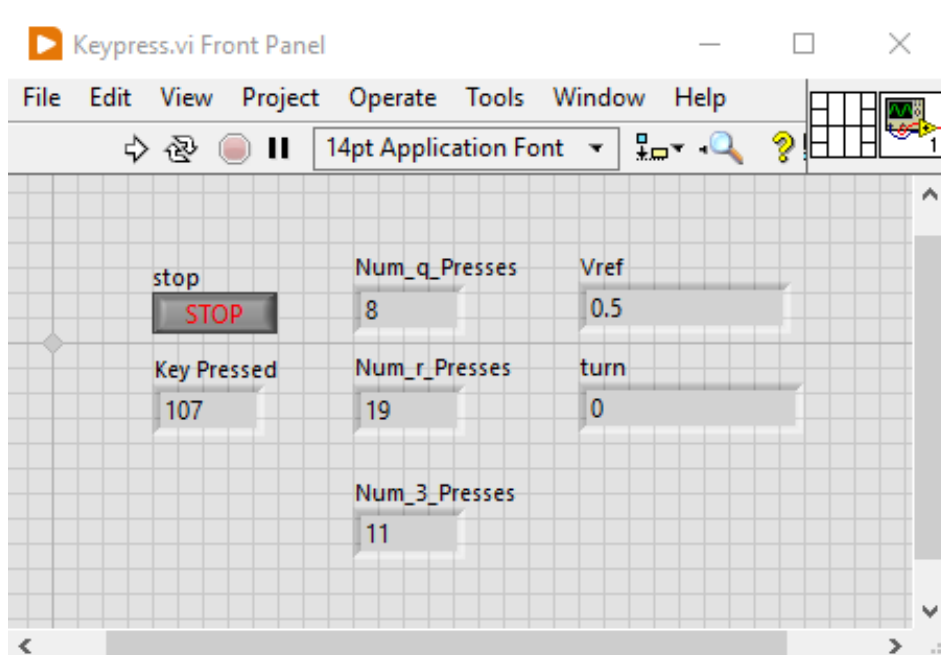
- m. Add one more feature to this LabVIEW program that will make steering around your robot wirelessly a bit easier in Lab 6 and Lab 7.

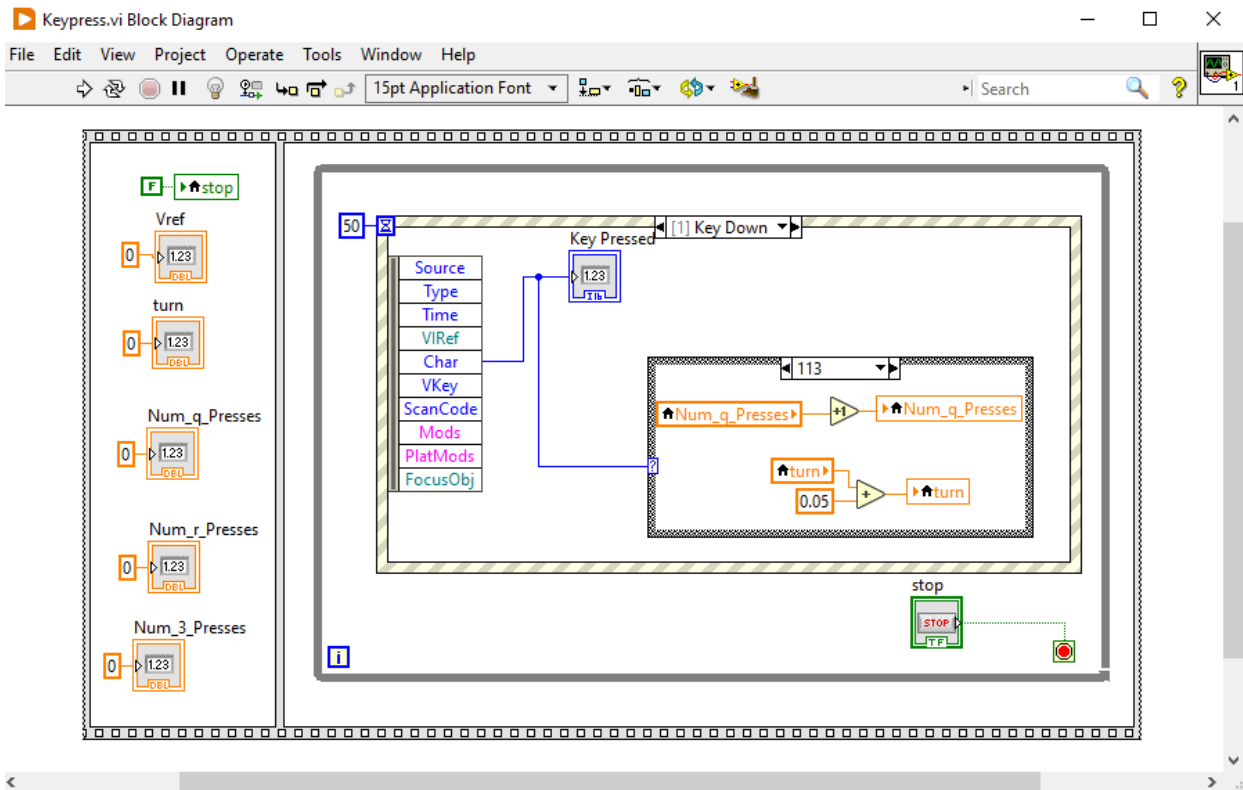
The Event structure has an event <This VI> → Key → Key Down. This event is run when your LabVIEW program has focus (mouse click the Front Panel View) and you press any key. Which key was pressed is given to you by the “Char” variable created by the event structure. So, using the below sample LabVIEW application, add to the above Event structure a Key Down event.

Copy most of what is in the “Send Button”: Value Change event into this new Key Down event. BUT NOTE: you will have to use local variables to get the values of SendValue3 through SendValue8. Instead of using local variables for SendValue1 and SendValue2, write the value of “Vref” (Use a Local Variable block) to the first “Format Into String” input and “turn” to the second “Format Into String” input. You will probably have to make your Event structure a bit bigger to fit all the below code into your Key Down event.

Looking at the below example code, notice that when you press “q” (ASCII 113) the value of 0.05 is added to “turn”. If “r” (ASCII 114) is pressed 0.05 is subtracted from “turn”. If “3” is pressed a value of 0.1 is added to “Vref”. If any other key is pressed, “0, Default”, “turn” is set to 0 and “Vref” is set to 0.5. Add this last feature to your application and then **demo to your TA or Instructor** that your LabVIEW application receives values from the Red Board (through ESP32) and that both your Send Button event and Key Down event work to send new values to the Red Board.

Example Application to show you how to use the “<This VI> → Key → Key Down” Event. It is up to you if you want to build this small application first before you add the Key Down event to your Red Board interface App above.



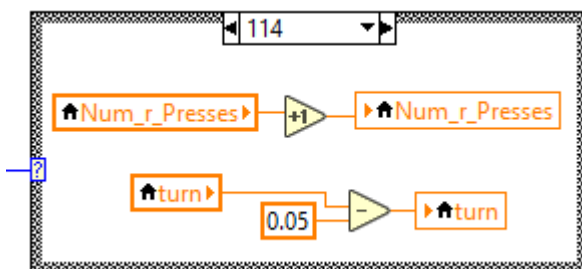


The three variables, “Num_q_Presses”, “Num_r_Presses”, and “Num_3_Presses” are not needed in your code. These variables are just incremented by one each time the Key Down event is called and that specific key was pressed.

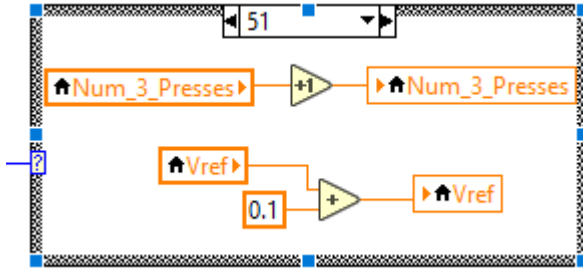
Notice how the Event Structure knows which key is pressed by looking at the event variable “Char”. “Char” is an I16 variable, so it is the ASCII value for the key pressed: “q” = 113, “r” = 114, “3” = 51. Using a Case Structure with four items, the below code changes “Vref” and “turn”. If “q” is pressed turn = turn + 0.05. If “r” is pressed turn = turn – 0.05. If “3” is pressed Vref = Vref + 0.1. If any other key is pressed, turn = 0, Vref = 0.5.

Key press case examples:

- Case for “r” being pressed:



- Case for “3” being pressed:



- Case for Any Other Key being pressed:

