iRobotics sbRIO Control System

User and Developer Manual

**Scope and Purpose**

The purpose of this document will cover the basic setup, connection, and operation of the sbRIO and associated hardware, specific to the iRobotics Club. The control algorithm in the repository is not covered. This document only covers moving data from a USB controller through a Host/Debugging PC over TCP/IP to the sbRIO. Some parts are purposfully left vague as to give general guidance to new developers, but ensure that they make engineering decisions which will be specific to their team's requirements.

**Revision List**

* 1.0 - Chris Gerth - Initial Release

**Getting Started**

For this system, you will need the following:

* sbRIO-9642 with appropriate PWM, Encoder, and Analog adapters installed on IO ports
  + Runs on 19-24VDC
* Wifi Router
* Power Supply for Router and sbRIO (Blue FRC PowerDistro Board has all the required output voltages)
* Ethernet Cables
* Xbox Controller (2x)
* Xbox Controller PC wireless dongle (1x)
* Labview 2012 SP1 & associated add-on's

**Labview**

Labview is the only development environment allowed for creating code for all National Instruments products, including the sbRIO. Its all-graphical coding style emphasizes prebuilt libraries like Simulink and Matlab, and is built around a focus on parallelizing execution of code. It is helpful to think of code in parallel, similar to how FPGAs get programmed in VHDL.

Labview can be downloaded for free from Labview's website.Go to <https://lumen.ni.com/nicif/us/evaltlktembdes/content.xhtml>, and install all the packages under step 1. The modules may all be downloaded in parallel, as well as paused and restarted. When each one completes, unzip the files to a temporary location, and install each one. When you are done, open the NI MAX (Measurement and Automation Explorer) program. NI MAX displays all the connected systems, and all the installed software on each one. Ensure that you have Labview, Labview Real-Time module, Labvew FPGA Module, FPGA Compilation Tools, and the NI-RIO Drivers installed on your local machine.

Licenses for Labview are available for free from UIUC's Webstore. "Purchase" the software to get info on how to install the license.

Next, connect the sbRIO to the same subnet as your computer. Turn on DIP switch 3 (factory reset), power on the sbRIO, wait twenty seconds, and turn off the DIP switch. The sbRIO should show up under NI MAX with ip address 0.0.0.0 . Set a static IP address within the valid range allowed by your router/subnet, as it will be easier to find the sbRIO when it is networked to a vi later on. Save all settings. Also, go ahead and give it a unique name, saving settings again.

**Git**

Git is the versioning software used to keep track of changes to code. It is needed to access the code source files which are hosted on github.com, download them to your local machine, and uploading/merging any edits you make to the code.

As with any versioning software, it is strongly recommended that you pull from the remote repository (or "update" in SVN terminology) each time you start working on the project, and push to the repository ( "commit") after every change you make is stable and functional. Be sure to use the -m flag to add an appropriate comment to each commit, saying what was changed.

Git needs to install software locally on your computer. Download it from <https://code.google.com/p/msysgit/downloads/detail?name=Git-1.8.1.2-preview20130201.exe&can=2&q=full+installer+official+git>, and install with all the default settings. Additionally, since Labview's files are all graphical and proprietary in nature, you need to use Labview's tools to difference or merge different versions of files during a push or pull operation. To do this, follow the instructions at <https://docs.google.com/document/d/1Lmx9WI1g_ObB03Vrlfb7QU3ochz1q_YRGM8dPi0R8g8/edit> to help set Git up to use Labview's tools and not its own.

The current repository for the source code is at <https://github.com/gerth2/iRoboticsSbRio>. This is the repo for the Midterminator Team's code. If you are on a different team or developing a new robot, it is suggested to either start a new repository, or create your own branch from this code and develop solely off of that branch.

**Hardware**

Setting up the router is highly specific to what sort of hardware you are using.

When setting up your router, it is suggested that you limit the number of incoming wireless connections, encrypt the connect, use some form of security (WAP, etc), and do not broadcast the SSID. This will prevent malicious teams from being able to access the data link between your host PC and your robot during competition. Additionally, if the router will not be connected to the internet, you can disable the firewall to make the connection run faster.

Labview can use any USB-based controller as input for your robot. The main branch of the repo is set up to use two Xbox controllers as input devices. One should note that there is a bug in the current Xbox driver installer, but many workarounds exist. Google is your friend on finding fixes to any problems you have. However, as long as windows recognizes your device as a USB Input device, Labview can utilize it to send data to the sbRIO.

The sbRIO itself requires little maintenance. Ensure the backup battery is fresh (it should last a good couple years). Also, be very careful with the power connector - it is not designed to withstand the stresses of robotics (it almost broke off during JSDC2013). Finally, note that to connect pwm or spike cables to the board, you will need to fabricate custom IO headers to mate with the IO ports on the sbRIO, and connect to the three-wire cables. Refer to the sbRIO manuals for pinout info, as well as the Victor and Spike manuals to see what sorts of signals they will be expecting on each wire.

**Running the system**

This section references the Midterminator's operating procedure. It will have to be altered if you change the communication process significantly.

To start the system, follow the following steps:

First, power on the robot. All parts should begin to initialize. The router will probably take the longest (about 30 seconds). By the time you can connect the host laptop to the wireless network, the sbRIO should have had plenty of time to boot up and initialize the IO ports.

While this process occurs, open the project in Labview if it is not already open. Open both Computer.vi and sbRIO.vi, but do not deploy them yet.

Once the router has been connected, you should be able to view any IP camera data in the your internet browser.

You can use NI MAX to verify that the computer and sbRIO are connected together, as well as check the sbRIO's operating state, and perform a software reset on the sbRIO by itself, without having to shut down the robot.

Set the IP Address box in Computer.vi to be the same as the sbRIO, and set the port on both Computer.vi and sbRIO.vi to the same number.

After establishing a connection with the robot's wireless network, deploy sbRIO.vi by hitting the play button. The FPGA bitstream will be automatically loaded if it is not already running on the FPGA. At this point the blue status light on the top of the sbRIO case will turn on solid, with no blinking. This indicates the sbRIO to be in the "Ready" state, waiting for a TCP connection from the host machine.

When the blue light is on, and everyone is a safe distance away from the machine, deploy Computer.vi. As soon as the vi starts running, it should make a connection to the sbRIO.

Control data will be acquired from the USB joysticks, packaged and transmitted to the sbRIO. From there, it is processed first by the ordinary processor, then by the FPGA, which passes it on to the circuitry which drives the IO pins.

To stop the system, press the "StopVi" button in Computer.vi. The sbRIO will return to a known safe state, and wait for the computer to reconnect. To fully stop the sbRIO, you must also press "Stop" in sbRIO.vi. If the network connection is interrupted at any time, the sbRIO will attempt to reconnect to the computer, and the computer will throw errors at the user, indicating what the problem is. Depending on the problem, Computer.VI might have to be redeployed. Regardless, the sbRIO should automatically reconnect and resume safe operation once the computer is back working again.

Due to the robustness of the TCP communication protocol, previous safeguards such as multiple watchdog timers and adjustable variable read timeouts have been done away with for simplicity. The current safeguard simply puts the outputs in a safe state if the TCP link is ever broken, and attempts to restore the link.

**High Level Explanation**

At a high level, the system is highly linear. User inputs occur at a USB device. Labview communicates with this device, picks out which signals are relevant, packages them into a Cluster (Labview's equivalent of a C struct), and transmits that cluster over the network connection to the robot. The robot receives this data, gives it to a control algorithm which generates motor values. These are then passed on to the FPGA, where the numbers are turned into appropriate control signals. Encoders and sensors are also interfaced through the FPGA back to the control algorithm in the processor.

**Computer.vi**

The purpose of this vi (virtual instrument) is to acquire user data, package it in a format useful to the sbRIO, and transmit it via a TCP protocol over Wifi and Ethernet. To this end, it acts like a TCP server, initiating the connection with the robot and transmitting data. Data currently only flows one direction (from computer.vi to sbRIO.vi), however the front panels of the .vi's can be used for debugging information. All errors are handled appropriately by either attempting to reconnect over TCP or halting the .vi.

To communicate, the .vi has to open two different instances of USB devices, with identifiers 0 and 1 (the first two joysticks plugged into your computer). Then, it attempts to make a TCP connection. If the sbRIO is not yet listening, an error is generated, and the user can choose to try again or quit. Once the connection is made, the computer runs on 10 millisecond loops. Each loop, it reads the current state of both Xbox controllers, takes the pertinent button and axis data (D-pad is unused), and puts it into a custom-defined cluster. Then, this cluster is flattened into a sequence of bits to be transmitted over the existing TCP connection.

First, the length of the cluster in bits is transmitted as a 32 bit (4 byte) integer. Then the cluster itself is transmitted. If any errors occur during this transmission, a user dialog will be generated, and the connection possibly reset and reinitialized.

**sbRIO.vi**

The purpose of this vi is to take information from the computer, and use it to control the robot. It is assumed that there is a separate .vi which contains the actual control algorithm. The vi starts by initializing the FPGA, and then listening for a TCP connection from the host computer. The "Listen for Connection" block will not allow anything else to execute until it gets a good connection from the Host Laptop. Once a connection has been made, it reads the first four bytes off of the stream and interprets them as the size of the rest of the data to be read in. Next, it reads in the actual data, and conforms it back to the known cluster type. From there, the individual axes and buttons are split off from the cluster, available for any control algorithm to utilize.

Once the control algorithm has decided what values to set the motors and digital outputs at, these numbers and Booleans are passed on to the FPGA via a set of internal registers. In Labview, the large purple block shows all the possible inputs and outputs from the FPGA which you can connect signals to.

The sbRIO code runs on 5ms loops. This ensures it will consume data faster than the computer produces it. As long as this holds, there will be little or no data in the network buffers, minimizing latency between a user input and the change in control signal output.

**FPGA.vi**

The purpose of the FPGA, in this application, is to perform time critical, processor-intensive tasks that would have otherwise bogged down the main processor with interrupts time-dependent calculations. This means generating stable PWM signals at 50Hz with a 5-10% duty cycle to control the motors, and counting encoder ticks as they occur.

Note that ALL input and output has to be routed through the FPGA in order to get to the processor. Even simple digital input or output lines, which require no processing, must still be passed through the FPGA's vi.

Compile times on the FPGA code are quite significant. They can range from 10 minutes on a $1200 desktop, to five hours on a laptop. It is best to prototype and debug circuits and logic on a different, computer-based vi and project, and then copy known-good algorithms to the actual FPGA vi. This is the beauty of labview - the same vi can be compiled for different platforms, and changing which platform it runs on is simply a matter of dragging the vi from one spot to another.

Note that the FPGA boots into an unknown state by default. It is ideal to write the FPGA code in such a way that on starting the .vi, it is in a known safe state. Then, the vi can be set to be run at boot up, minimizing the potential time for an unknown and unsafe state to be present. If an unsafe state persists, actuators and motors could start running without your control.