Lab 2: Remote Control of Intersection

ME 348 Advanced Mechatronics

James DeBacker and Nicholas Fuselier

Emailed to: Benito Fernandez – benito@mail.utexas.edu

Attachments: Lab2.zip

1. Introduction

The goal of this lab was to program an intersection using LabVIEW FPGA on an embedded controller with a remote user control capability. Our solution was to use a NI myRIO with WiFi capability to generate the digital outputs and give control to a remote user. For instructor evaluation, a host version of our FPGA logic is included so it can be run without a target.

1.1 Overall Code Structure

The resulting program consists of three VIs executing on three different portions of the hardware. At the lowest level, the *main_FPGA.vi* program is compiled into a bitfile and loaded onto the myRIO's FPGA. The *main_RT.vi* runs on the processor of the myRIO and is responsible for loading the bitfile and interfacing with *main_FPGA.vi.* Finally, we have a host program *RemotePanelMethods.vi* which opens a connection to the remote panel running on the myRIO and allows a user to interface with the program over a WiFi connection.

1.2 Deliverables

Delivered to the Professor consists of: 1) An in-person demonstration in the AML on March 23, 2017, 2) A zip file containing all of the source code and project files used for the lab and 3) this report which explains each portion of the code.

2. Code Explanations

2.1 **main_FPGA.vi** (Basic Light)

This VI runs on the FPGA and is responsible for controlling stoplight operation and interfacing physical and digital I/O necessary to alter stoplight functionality. All of the main intersection logic takes place in this VI.

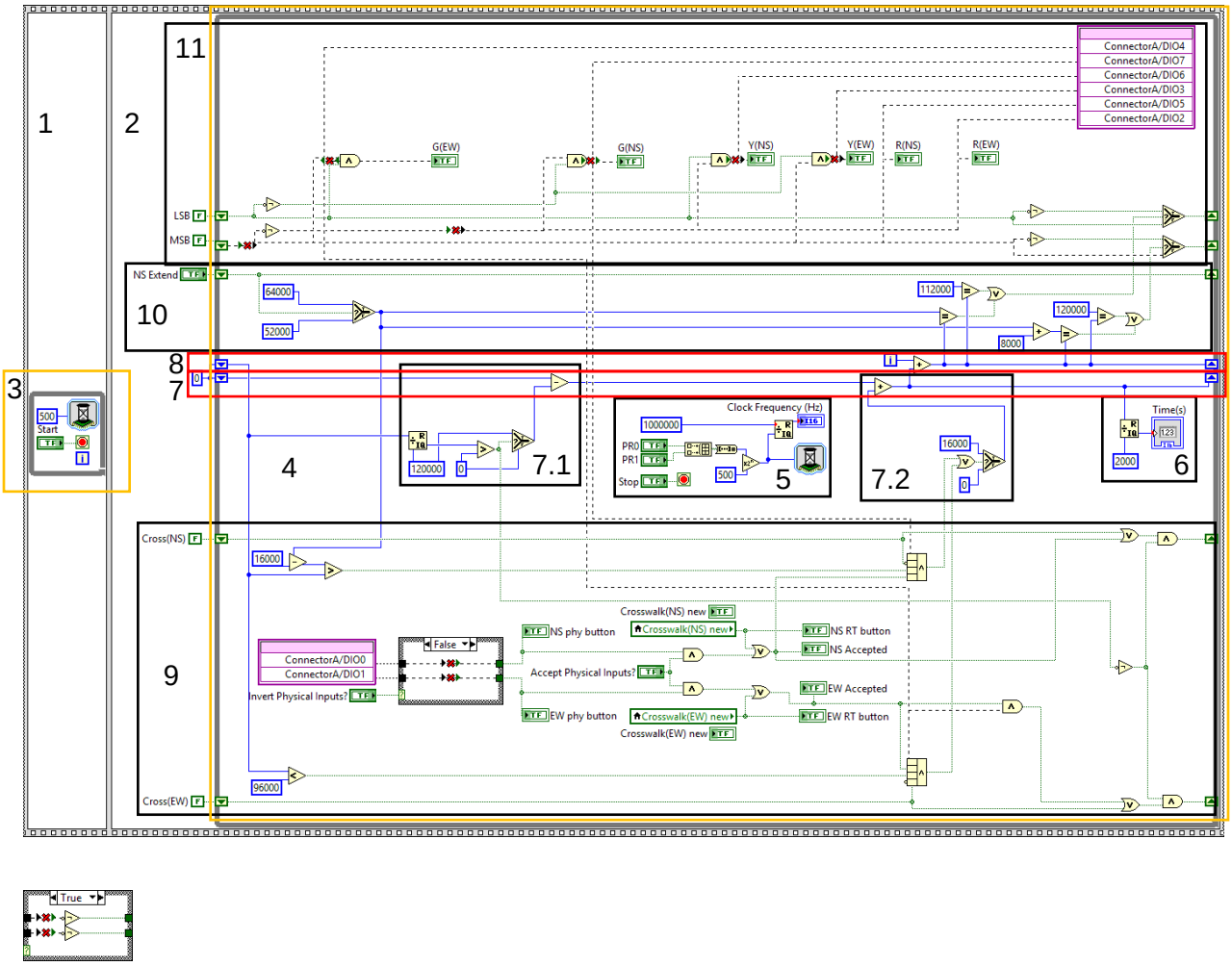

Fig. 1 – Block Diagram of the main_FPGA.vi

2.1.1 Functional Subunits

1: Frame 1 – Prevents execution of frame 2 until stoplight operation is desired
2: Frame 2 – Contains the stoplight loop

3: Hungry Loop – The mechanism by which frame 1 is maintained and frame 2 is held off

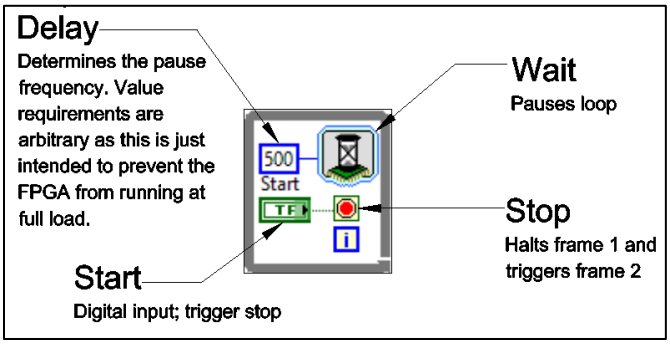

Fig. 2 – Hungry Loop in main_FPGA.vi

4: Stoplight Loop – Contains the logic for stoplight operation and handles I/O in a manner corresponding to that specified in the lab instructions.

5: Clock – Determines the frequency of the loop clock in response to PR0 and PR1, with a default frequency of 2kHz

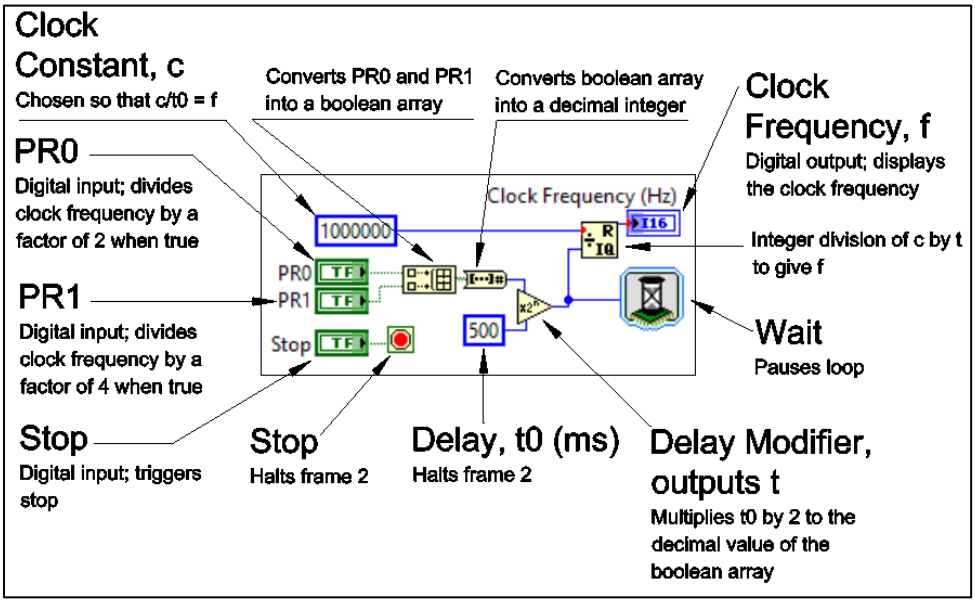

Fig. 3 – Loop Timer

6: Time – Converts the number of iterations (here being the equivalent to clock ticks) into seconds. 2000 ticks/second.

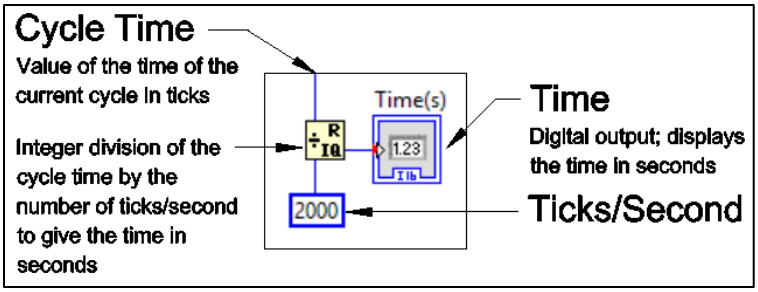

Fig. 4 – Ticks to Second converter

7: Time Adjustment Register (TAR) – Modifies the transmitted value of the clock register in response to the completion of a cycle of a crosswalk request.

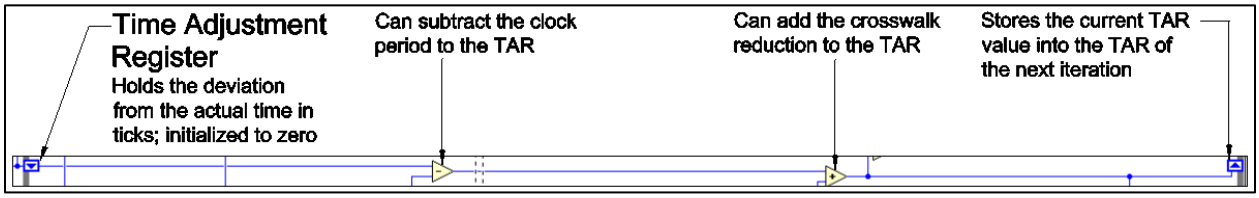

Fig. 5 – Time Adjustment Register

7.1: Cycle Adjust – Subtracts the cycle period (120s) from the TAR on the completion of a cycle

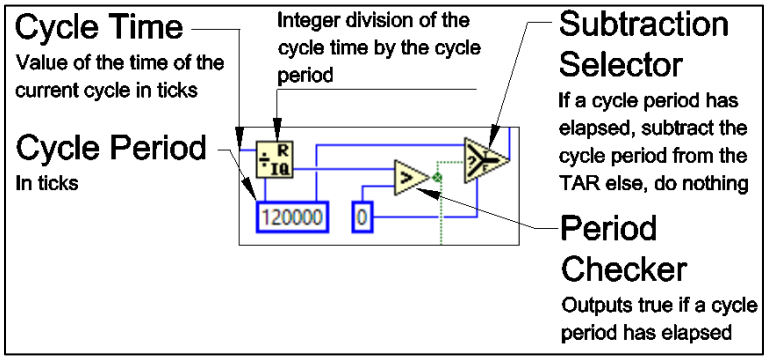

Fig. 6 – Resets the cycle timing

7.2: Crosswalk Adjust – Adds the crosswalk reduction (8s) to the TAR on a valid crosswalk request.

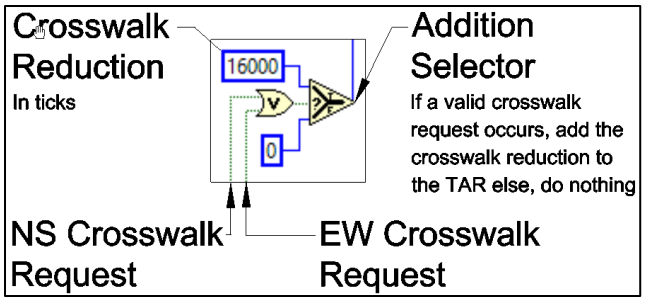

Fig. 7 – Crosswalk Adjust

8: Time Register – Adds the value in the TAR to the number of iterations (right) to get a cycle specific virtual time (left).

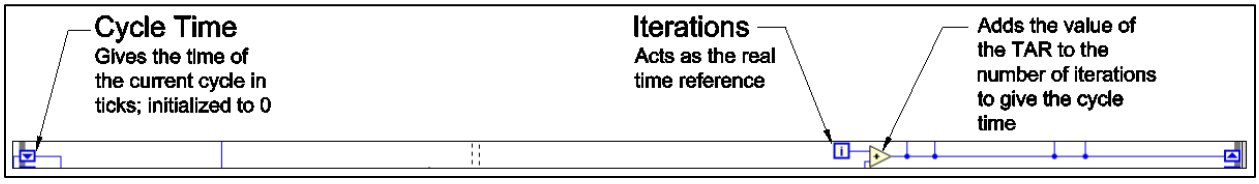

Fig. 8 – Time Register

9: Cross Request Validator – Approve the transmission of a crosswalk request in accord with the constraints specified in the lab instructions. Accepts physical and digital crossing requests. Most functionality is mirrored between the EW and NS validators.

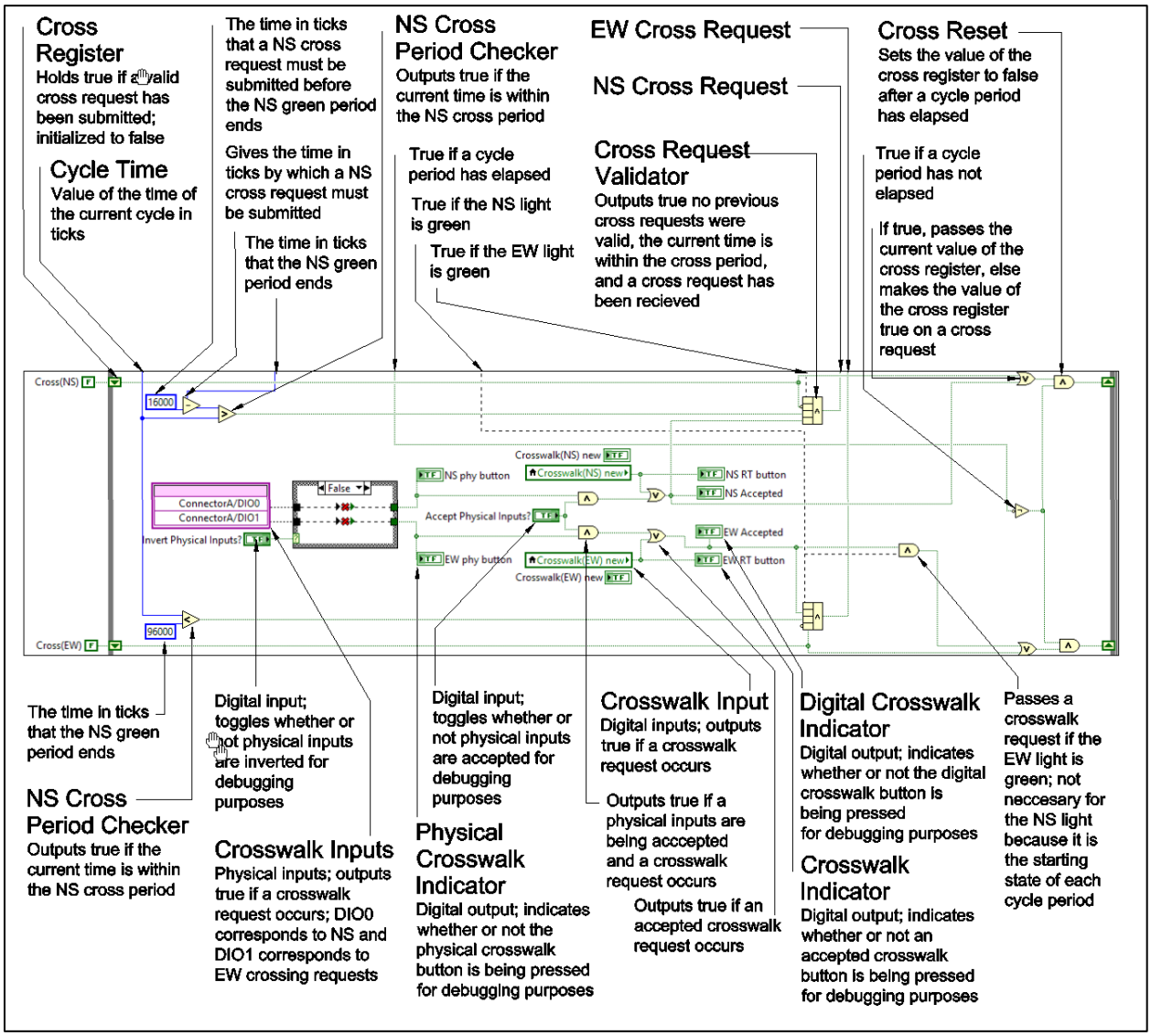

Fig. 9 – Cross Request Validator

10: Light Switcher – Determines the periods of light operation in response to the virtual time and the value of the NS Extend Register. Operated via the toggling of the LSB and MSB registers.

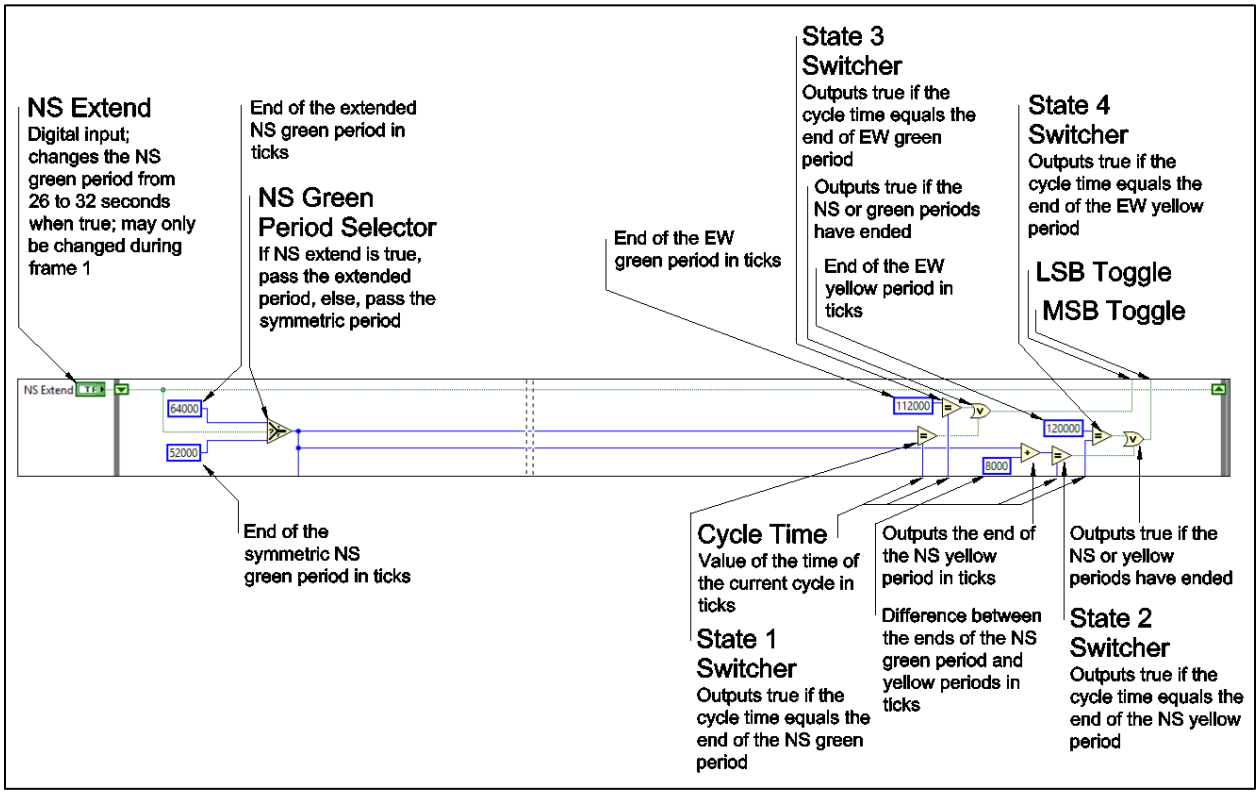

Fig. 10 – Light Switcher

11: Light Logic – Determines the output configuration of the lights in response to the values of the LSB and MSB register in accord with the truth table specified in the lab instructions. Outputs both to the front panel and to physical digital outputs.

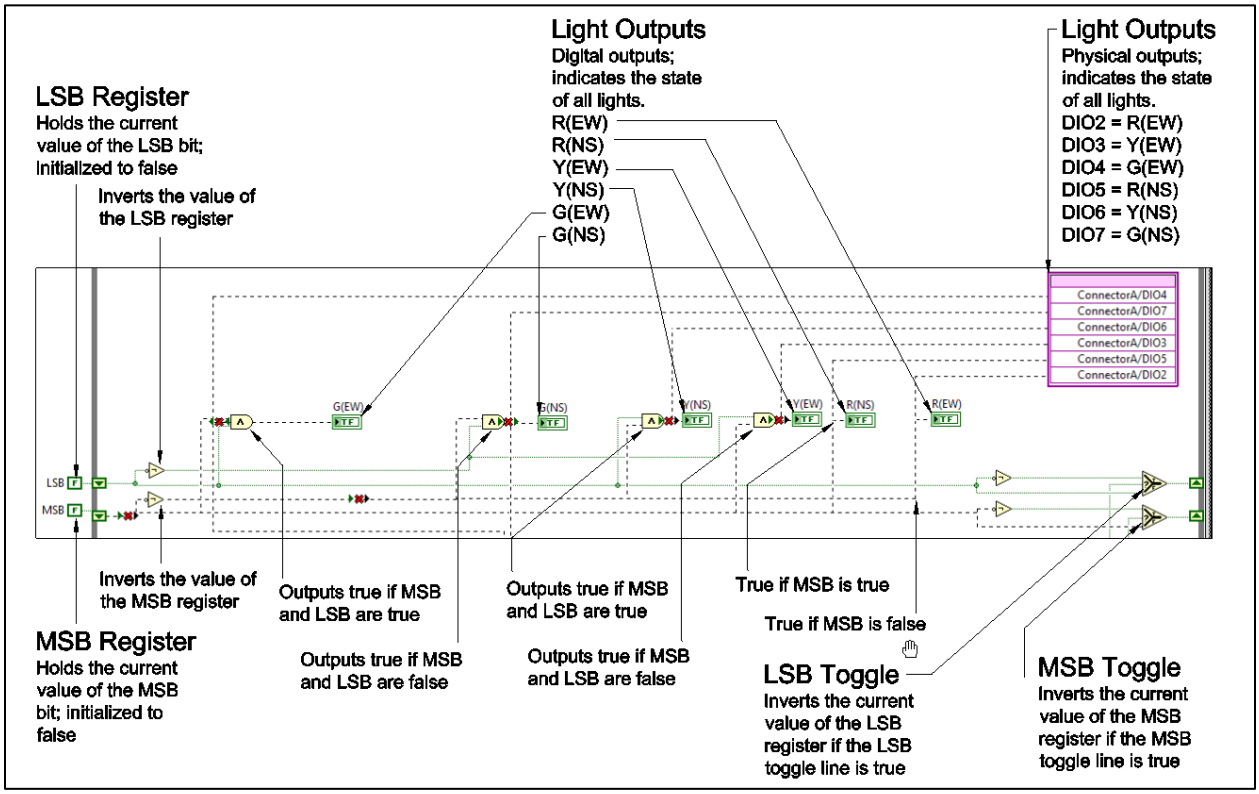

Fig. 11 – Light Logic

2.2 main_RT.vi

This VI runs on the RT module and is responsible for routing physical and digital I/O to the targeted FPGA bitfile.

2.2.1 Front Panel

Figure 12 is a screenshot of the realtime front panel. The left half is the main stoplight control panel, and the controls and LEDs are the virtual representation of the intersection. The right half are vestigial debug controls and indicators used during development.

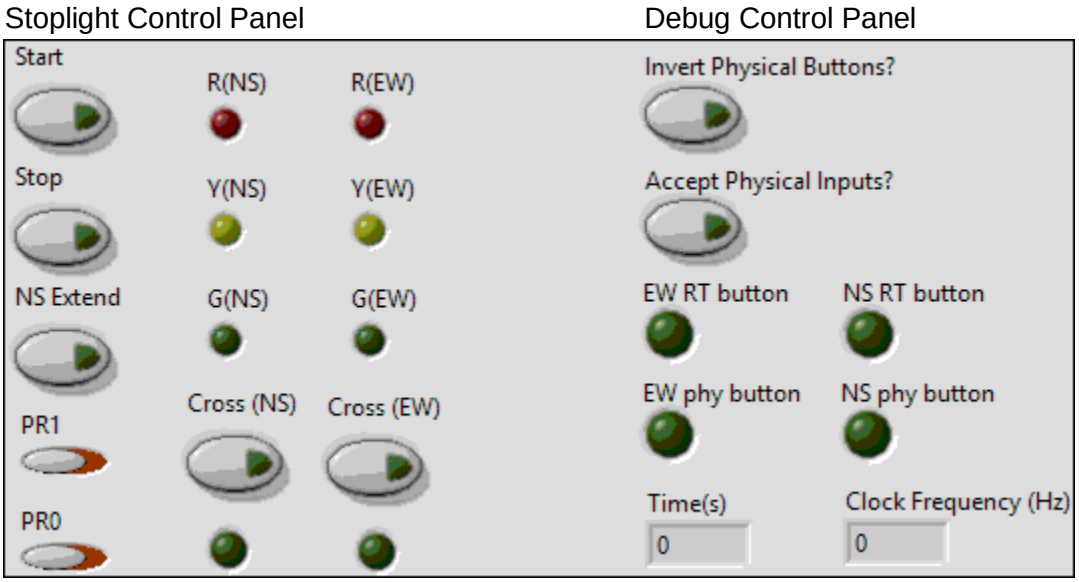

Fig. 12 – Front Panel of main_RT.vi

2.2.2 Block Diagram

The block diagram begins with a *Load Fpga Reference* block, which loads the compiled bitfile onto the FPGA. The main loop of the program is simply a *read/write control* which pushes inputs to and reads outputs from the FPGA. A *wait until ms multiple* block times the loop at 10 hz to reduce the CPU load, while maintaining a responsive user interface. This program was loaded onto the myRIO by building a startup.exe and setting it to “run as startup” from the project explorer. Thus, any time the myRIO starts up, it automatically runs this program.

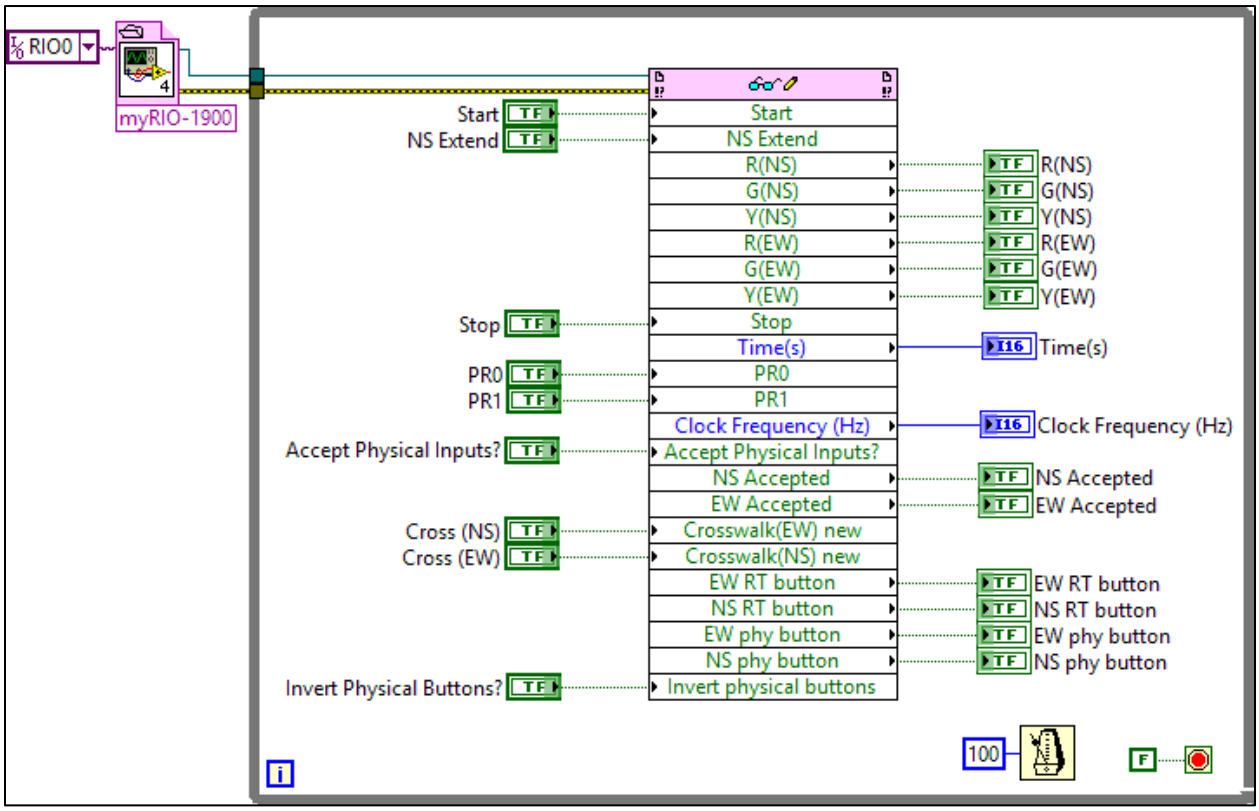

Fig. 13 – Block Diagram of main_RT.vi

2.3 Remote Panel Methods Client.vi

This VI illustrates how to programmatically connect to a remote front panel from a host computer. A connection is established to the specified VI and control can also be requested. Once control is requested, the VI can only be run once since the server will close the connection automatically after one run. Then, the RT program running on the cRIO is viewable over a WiFi connection from any PC that can run this executable while also connected to the WiFi.

2.3.1 Front Panel

This front panel allows the user to connect to any real time VI running remotely, as long as the correct parameters are specified. In this case, we made the settings in figure 14 as default so that the program always connects to the correct VI.

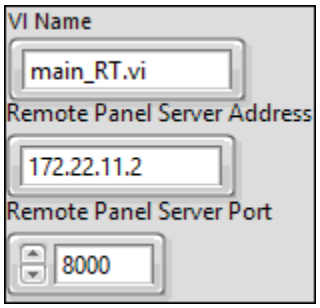

Fig. 14 – Front Panel of the Host Executable

2.3.2 Block Diagram

This program takes advantage of some native LabVIEW capabilities to establish a connection with the myRIO.

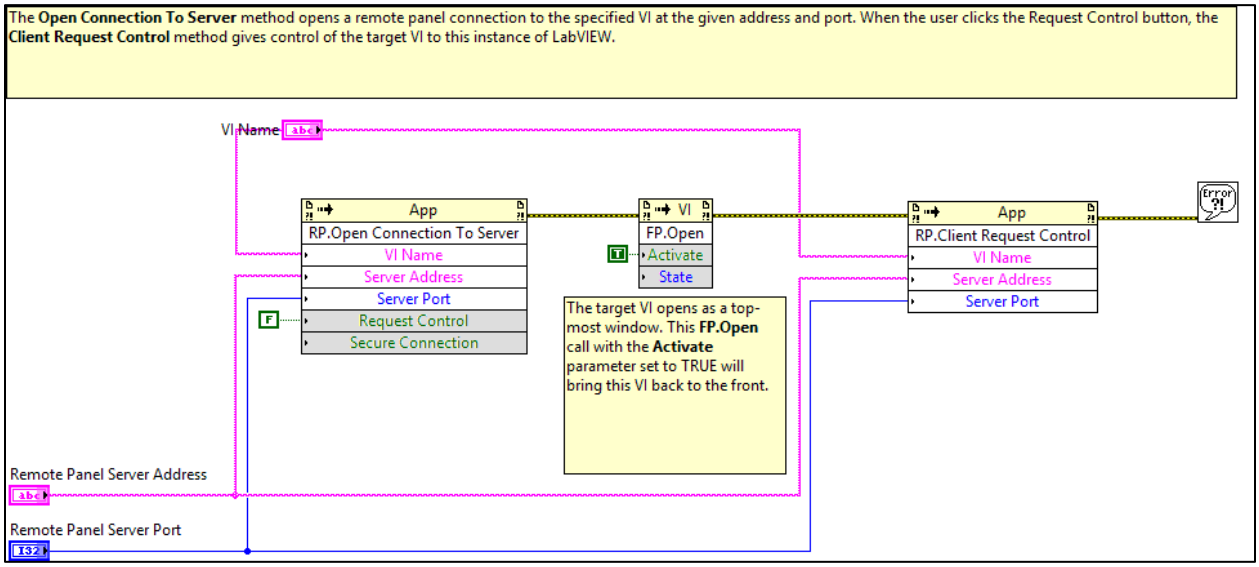

Fig. 15 – Block Diagram of the Host executable

3 Smart Intersection Proposal

main_FPGA.vi (With Sensors and Smart Timing)

The changes to this version over the above version are the addition of the NSEW sensors and the NS Priority and NS Smart modes. Their functional spread makes commenting impractical.

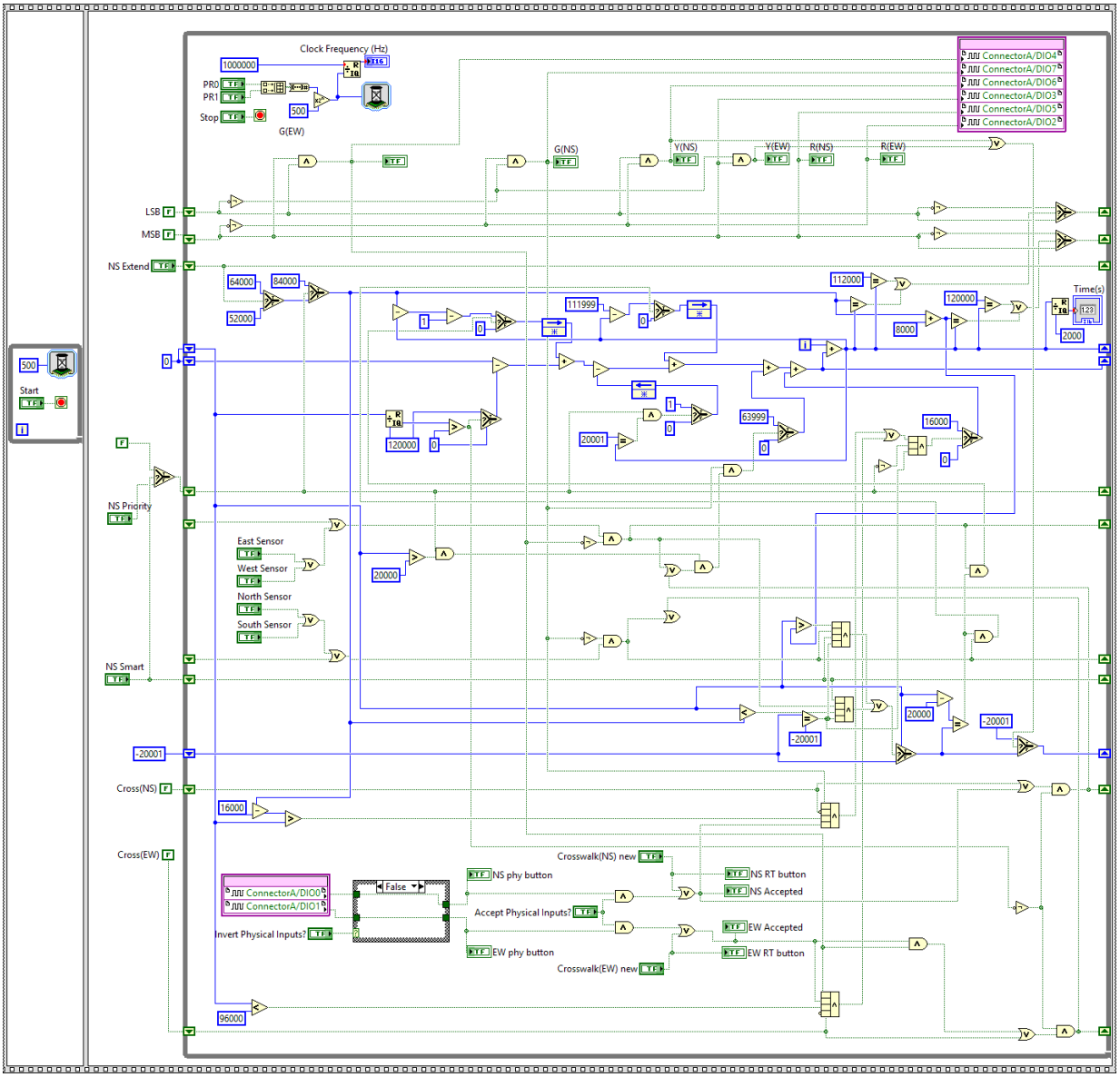

Fig. 15 – Block Diagram of the Host executable

1. In NS Priority mode, the light is by default NS green. If greater than 10 seconds have elapsed since the light has been NS green, then either an EW crosswalk request or the EW sensors can trigger the continuation of light changes. The NS green light will be on for 10 seconds in this partial cycle. The 8 second reduction triggered by the crosswalks is disabled in this mode.

2. In NS Smart mode the lights will operate normally or with NS Extend enabled; if a sensor cross to the current green light is triggered, then the light will cycle to the next green period within 10 seconds. Crosswalks function normally in this mode.