Tutorial

Antonio Rueda

June 6, 2019

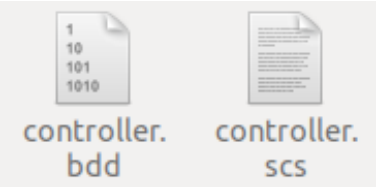
In this tutorial we explain the necessary steps to build, flash and simulate a closed loop of a controlled system that will run in 2 myRIOs.

1 Creating the controller with Scotsv0.2

Please refer to the documentation of Scotsv0.2 to see how to create a symbolic controller. In our tool, Scotsv0.2 has been included in the source files and we can build some of them by running the following bash comands:

```
$ cd Project/examples
$ ./0-build.sh
$ ./1-run.sh
```

These commands will create the controller files generated by Scots (`.scs + .bdd`).



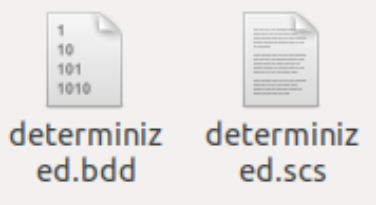
If we want to include another example, this will need to be created inside the `models` folder, and we will need to edit the `.sh` files to include the path to the new example.

2 Determinizing the controller with SCOTSv2.0 BDD controllers determinizer

Now we need to determinize the previous controller. Please refer to the documentation of SCOTSv2.0 BDD controllers determinizer to understand how this process works. Again, this tool has been included in the source files and we can determinize the controller by running:

```
$ ./2-determinize.sh
```

We will create the following files inside the `blgdet` folder:



3 Generating the blif file

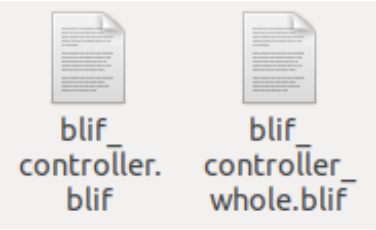
After that, we will need to convert the determinized controller to `.blif`. We have now two possibilities (Refer to the Thesis Project for details):

- Modify the BDD to set the control inputs as output of the controller BDD.

  ```
  $ ./3-generate_blif.sh
  ```

- Not modify the BDD but we will need to create a loop in LabVIEW to find a valid input for a particular state.

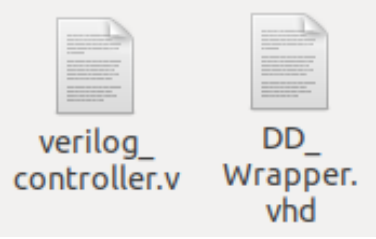
  ```
  $ ./5-generate_blif_whole.sh
  ```



4 Converting the blif file to verilog and generating the VHDL wrapper

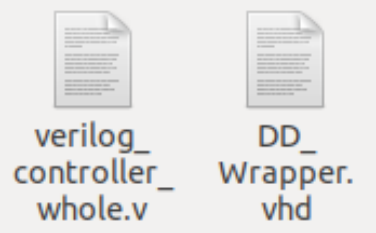
Using the abc tool, we can convert these files to verilog, and we have created a tool to generate a VHDL wrapper:

```
$ ./4-verilog_and_wrapper.sh
```



And for the second approach:

```
$ ./6-verilog_and_wrapper_whole.sh
```



5 Synthesizing the controller with Vivado

Now we need to create a new project in Vivado in order to generate the netlist using the previous files. For this, follow the steps 4 and 5 from the National Instrument tutorial `http://www.ni.com/tutorial/54793/en/` but adding both files (`.vhd` and `.v`) in the **Add Sources** page (step 4.5). This will create the `.dcp` file.

6 Flashing the controller onto the FPGA using LabVIEW

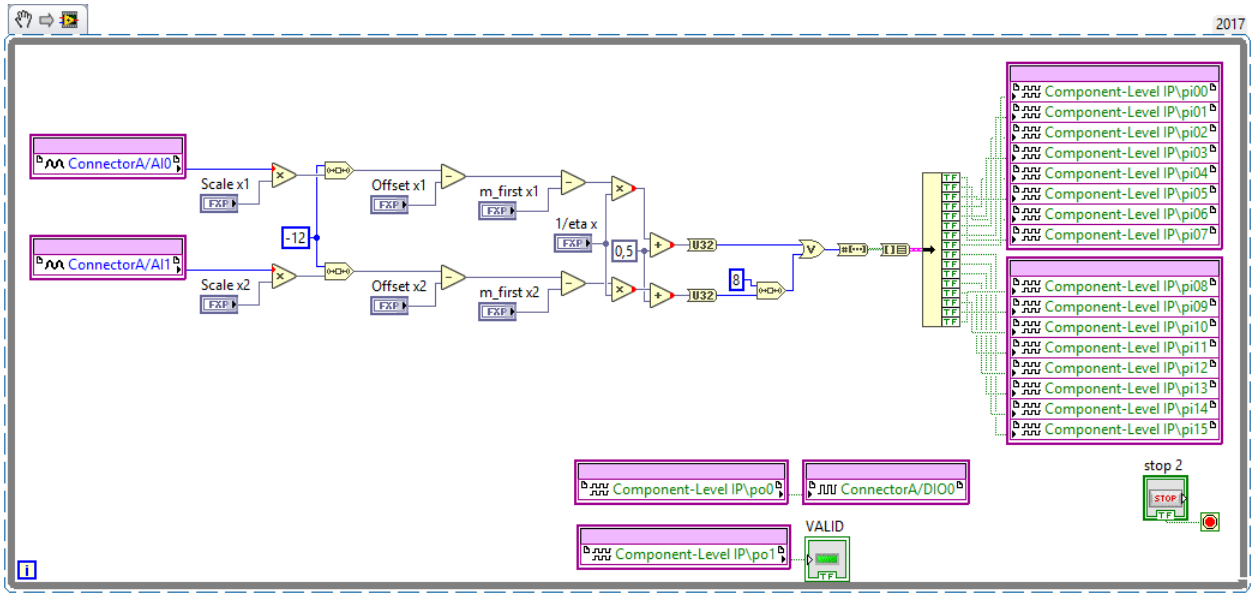
Once we have the `.dcp` and `.vhd` files we can follow step 2 from another National Instrument tutorial: `http://www.ni.com/tutorial/7444/en/` adding both files in step 2.2. This LabVIEW `.vi` will run on the first myRIO FPGA.

6.1 Example 1: DC-DC converter

In this example, we will have 2 states (current and voltage) as analog input of the controller and 1 digital output (control input). Please refer to the Scots documentation for more information about this system.

Now we have 3 possibilities in order to flash the controller onto an FPGA:

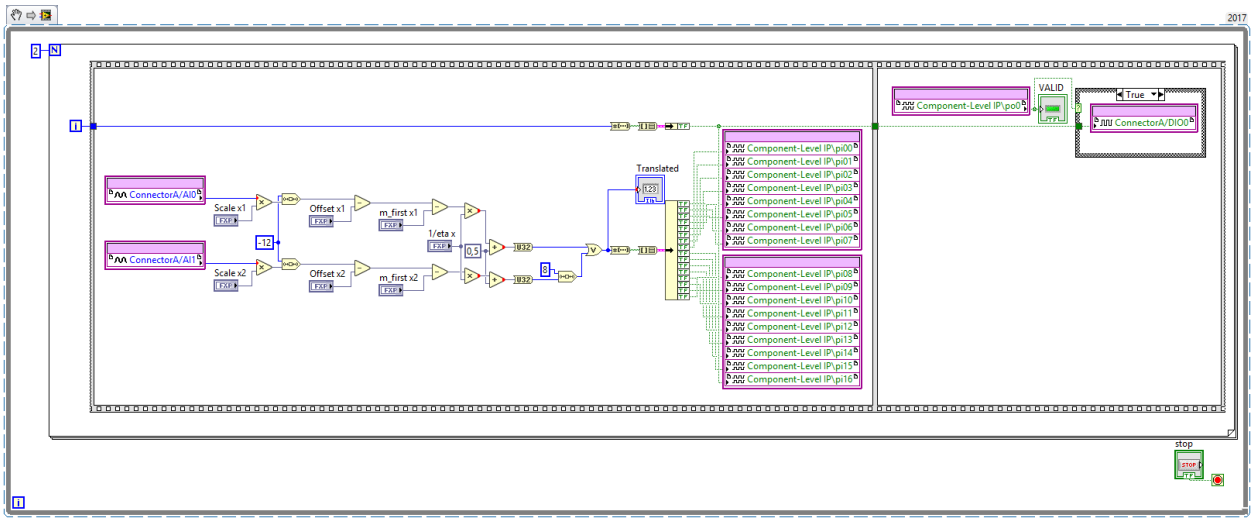
1. If we decided to modify the BDD to set the control inputs as output of the controller BDD, we will have the following in the controller side:



Where:

- The analog inputs AI0 and AI1 will read the values of the states.
- The user will need to adjust the parameters `Scale` and `Offset` according to the physical magnitude of the states that need to be read and according to the specifications of the inputs of the selected myRIO.
- The user will need to introduce the parameters `m_first` and `1/eta`. These parameters are defined when the controller is created using Scots.
- The BDD of the controller is shown as the Component-Level IP, where the inputs are the states translated to boolean. The number of bits dedicated to each state is obtained from the quantization parameter (`eta`) , which is coming from Scots (8 bits for each state in this example).
- The outputs of the BDD will be $po0$ (control input) and $po1$ (checks if the states are defined in the domain of the controller).

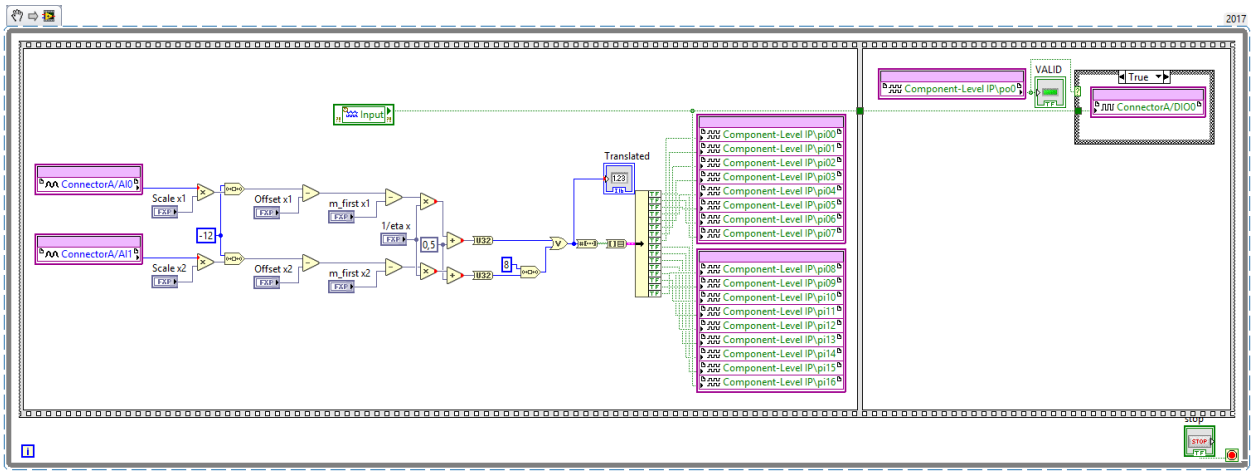
2. If we decided to create a loop to look for the correct input on the FPGA we will have:



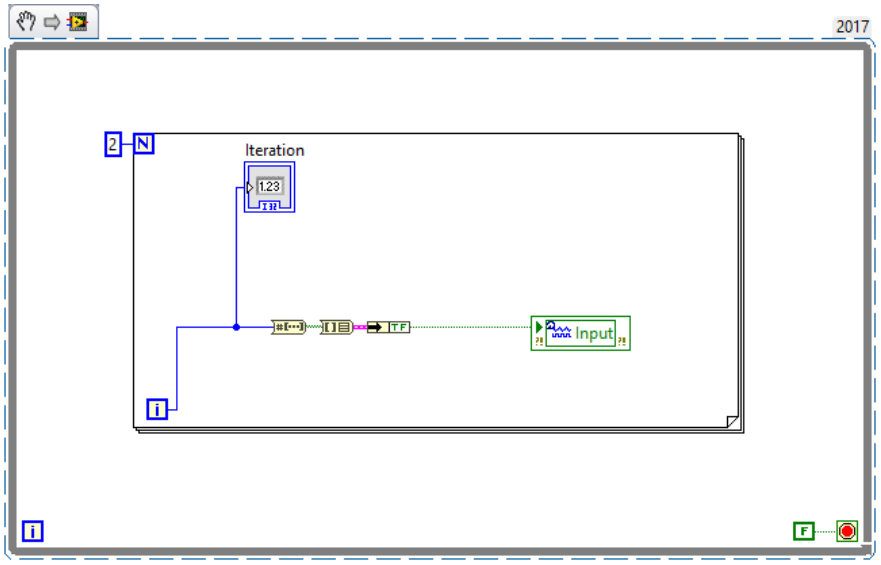
Where:

- We will have a new input to the BDD ($pi16$) in which we will try all the possible inputs (0 or 1 in this example).
- When the output of the controller returns "valid", we will pass that control input value to the digital output.
- If there is no valid control input, we know that we are out of the domain of the controller.

3. On the other hand, if we decided to create a loop on the CPU we will have on the FPGA side:



And on the CPU side of the same myRIO:

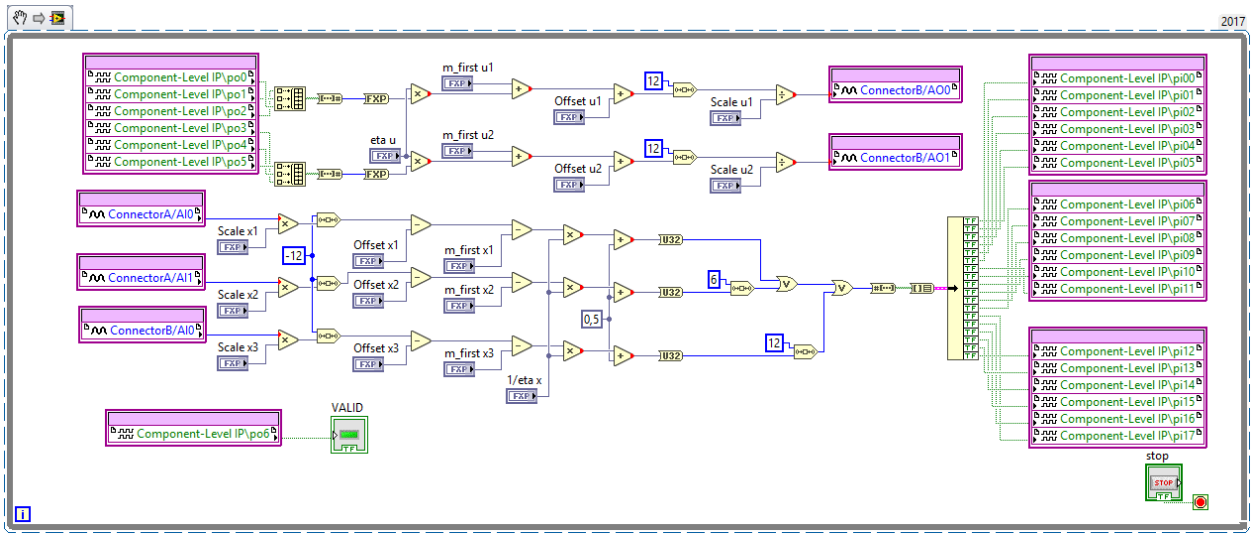


Where the input that we need to test is passed from the CPU to the FPGA using real-time variables.

6.2 Example 2: Vehicle

Please refer to the documentation of Scots to see the details about this example. Now, the controller will have 3 states and 2 analog outputs (control inputs). Again, we have 3 different possibilities to flash the controller:

1. With the modified BDD:

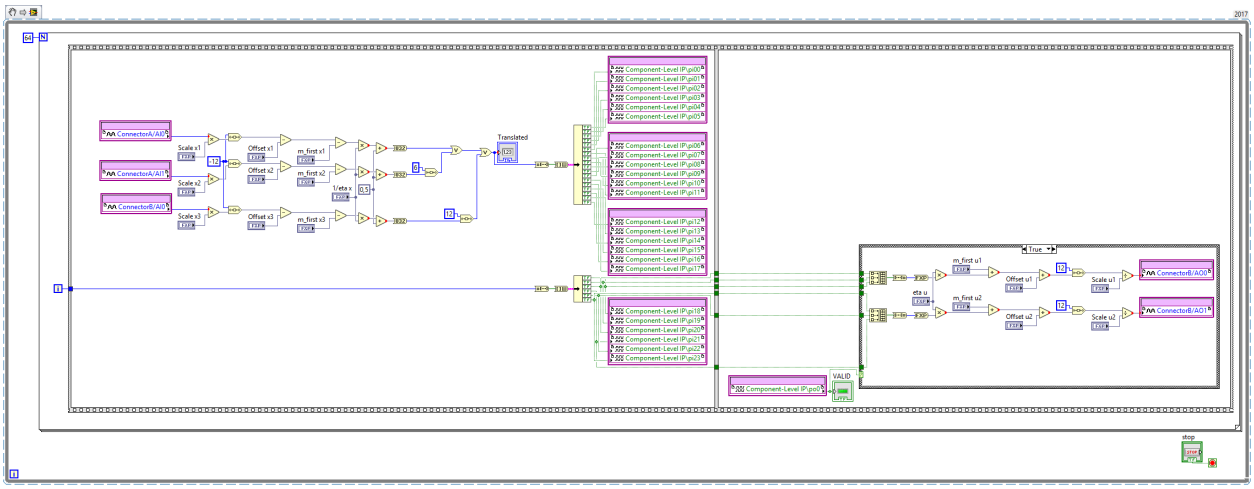


Where:

- The analog inputs AI0_A, AI1_A and AI0_B will read the values of the states.

- The user will need to adjust again the parameters of `Scale` and `Offset` for the states and also for the control inputs, since we are working now with analog values.
- The user will need to introduce the corresponding parameters `m_first` and `1/eta`. These parameters are defined when the controller is created using Scots.
- In this new example, we see that every state needs 6 bits. The conversion from analog to boolean is similar to the dc-dc example.
- The output of the BDD (from $po0$ to $po5$) is an array of 6 bits now (3 bits for each control input). These outputs will be converted to analog following the algorithm shown in the image.
- Again, the last output ($po6$) will tell us if the read states are included in the domain of the controller.

2. If we run a loop on the FPGA to find the correct control input:



Where:

- We will have 6 new inputs to the BDD ($pi18$ to $pi23$) in which we will try all the possible inputs (from 0 to 63 converted to boolean in this example).
- When the output of the controller returns "valid", we will convert all the control input values to analog, and send them to the plant.
- If there is no valid control inputs we know that we are out of the domain of the controller.

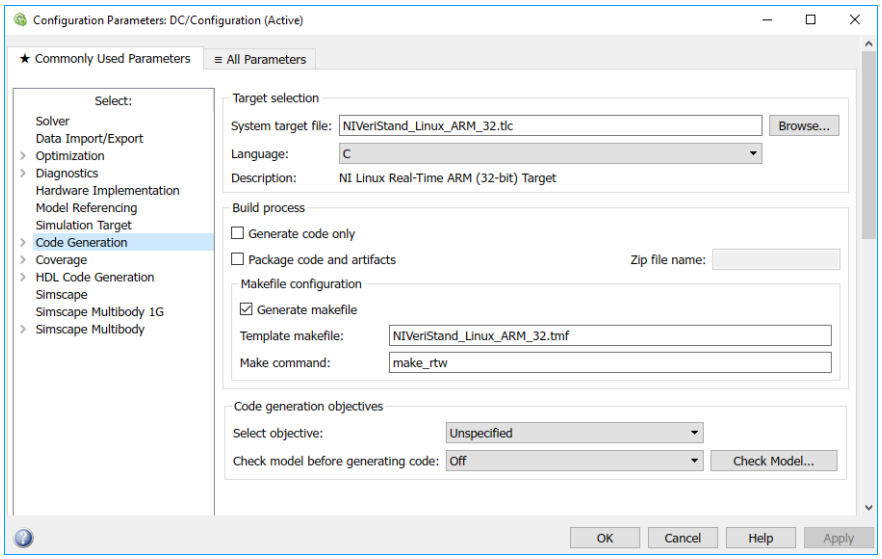
3. If we try now to run the loop on the CPU side of the same myRIO, the design won't fit because the use of real time I/O variables to pass the control input from the CPU to the FPGA will drastically increase the LUT utilization.

7 Creating the plant model using Simulink

The software needed to create and compile models in Simulink is:

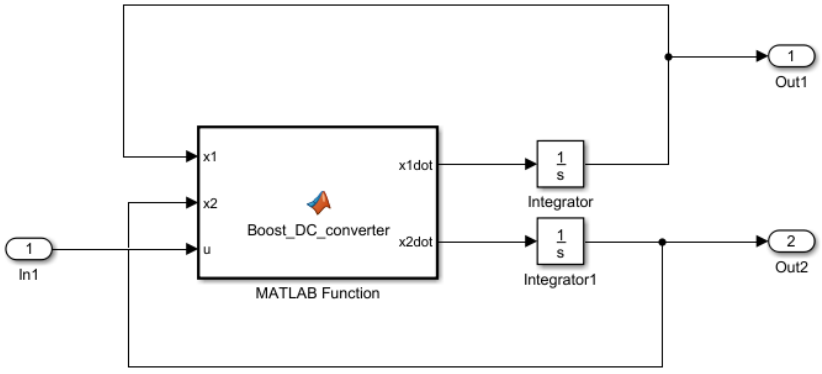
- Simulink Coder R2016b
- NI VeriStand Model Framework
- Microsoft Visual Studio 2010

In the Code Generation tab from the Configuration Parameters in Simulink we will have:

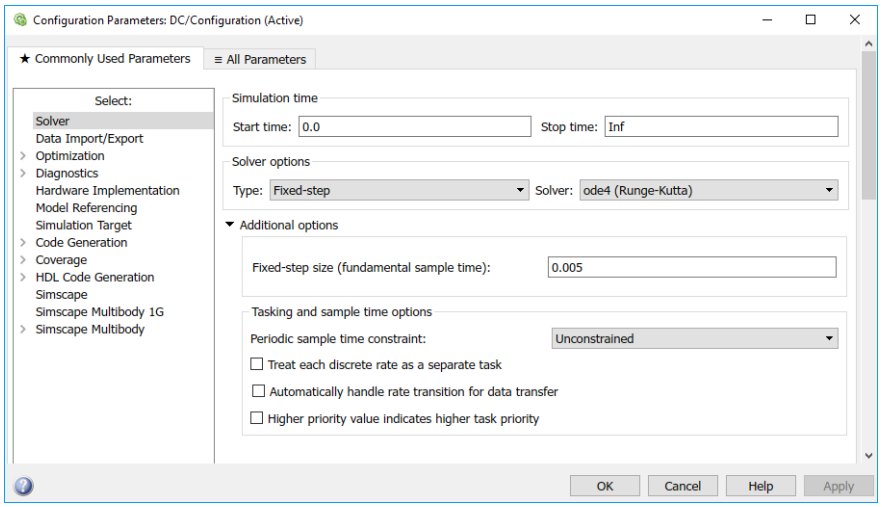


7.1 Example 1: DC-DC converter

We can use the `MATLAB Function` block containing the equations of the dynamics of the dc-dc converter:



Before building the model, we need to change the timestep size to 100 times smaller than the timestep used by Scots ($0.5s$ in this example):



The last step is to build the model. It will generate the `.so` file.

7.2 Example 2: Vehicle

The process here is similar to the one in the DC-DC converter example. But now we will have 3 states as output and 2 control inputs to the plant.

8 Importing the plant model from Simulink to LabVIEW and to myRIO

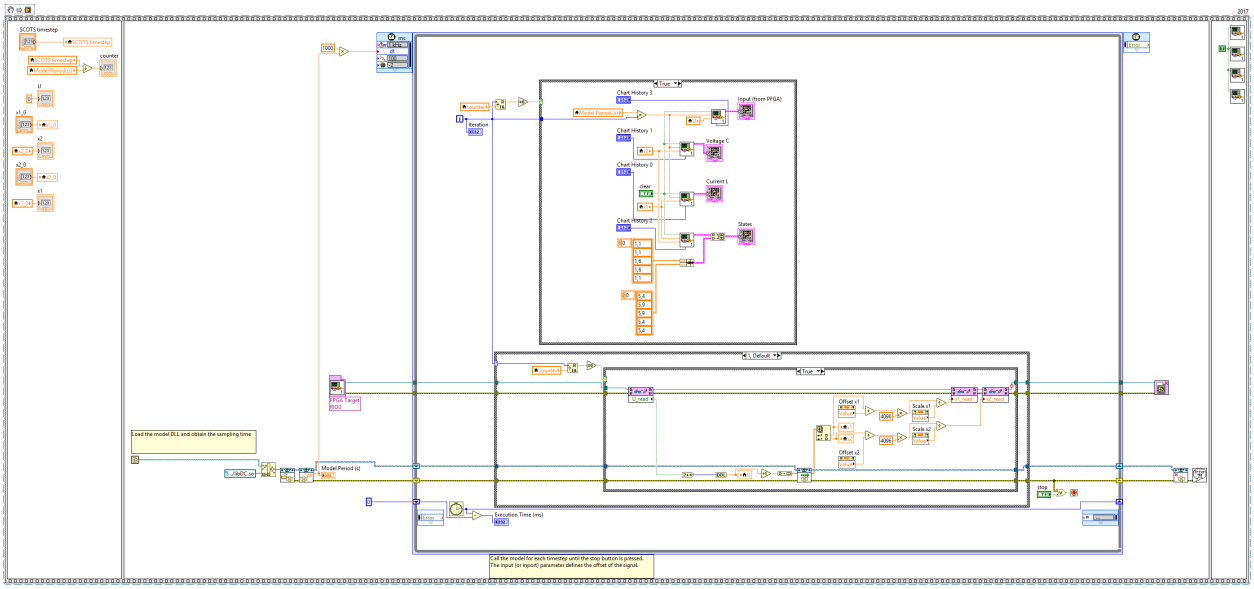
This LabVIEW `.vi` will run on the other myRIO.

8.1 Example 1: DC-DC converter

Now that we have the model, we need to save the `.so` file in the next folder of another myRIO:

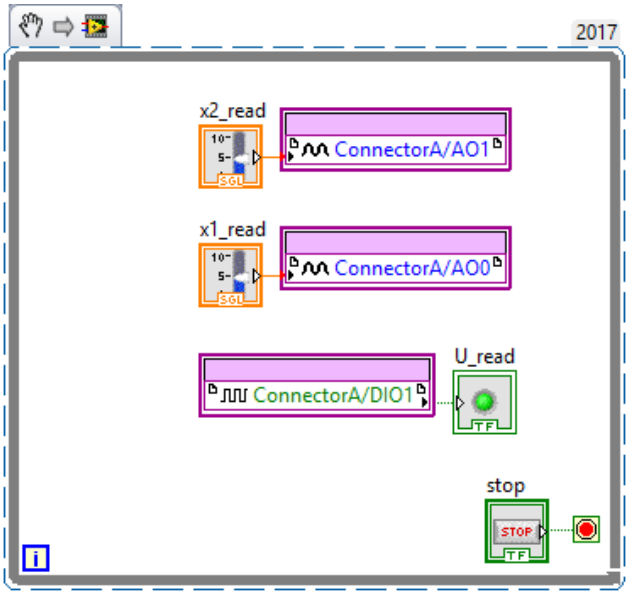
`http://172.22.11.10/files/c/ni-rt/startup`

The IP address correspond to the second myRIO. Create a LabVIEW file in the CPU side and call the Simulink model using the `Model Interface Toolkit`:



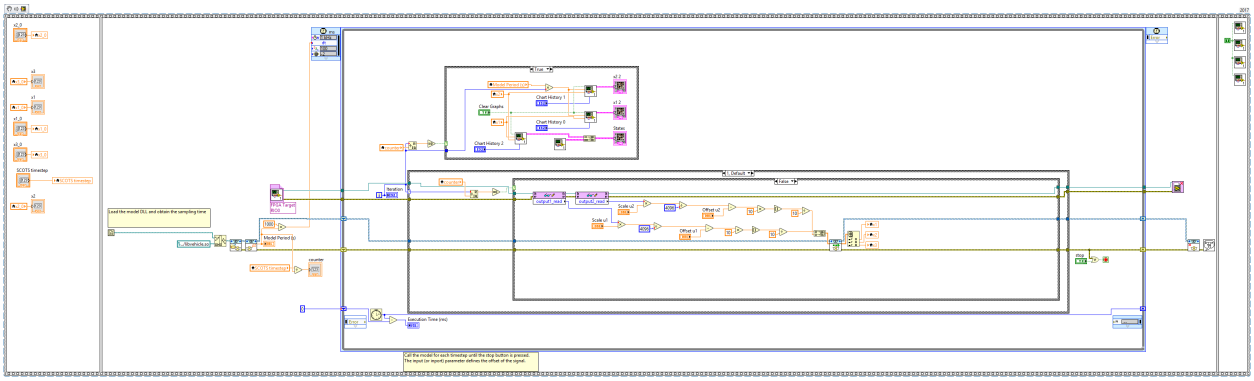
Where:

- We need to use the same parameters as in the other myRIO FPGA (`Scale`, `Offset`, `m_first` and `eta`).
- We will need to write the analog outputs and read the digital input. For this, since performing readings and writings is too slow on the CPU, we will use the FPGA to help us doing this faster:

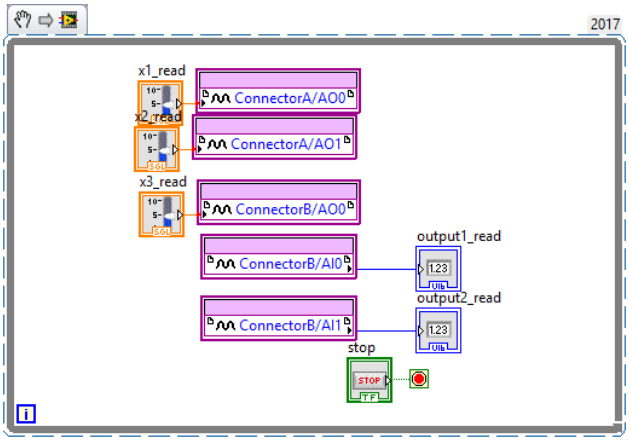


8.2 Example 2: Vehicle

This will be the `.vi` that will simulate the plant:



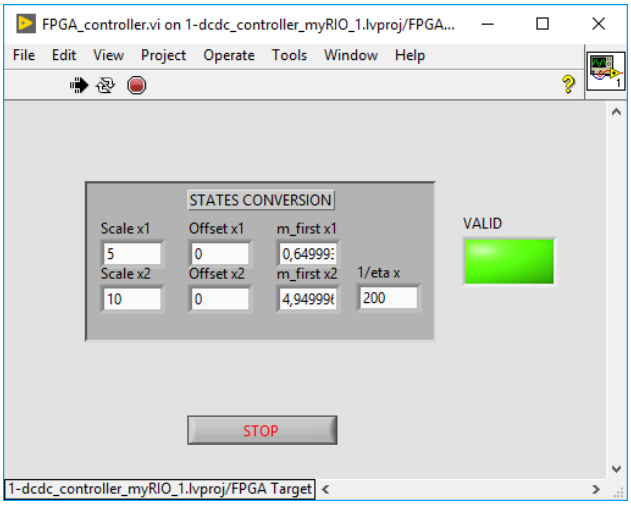
And to write and read the I/O we will use:



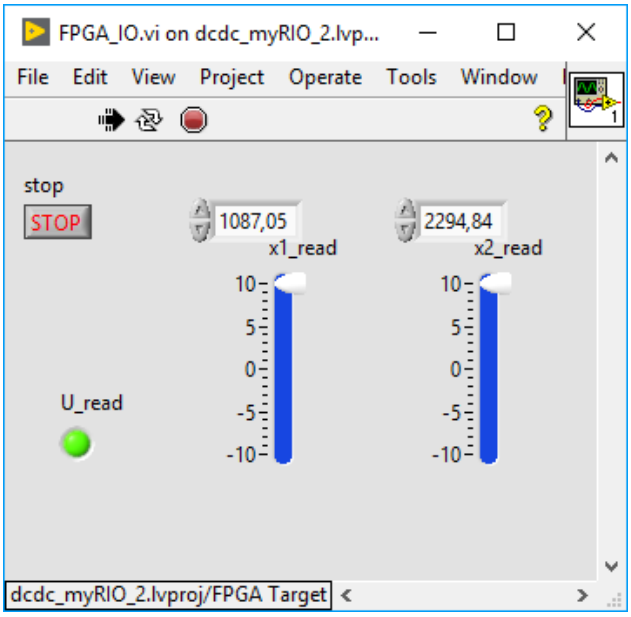
9 Simulating the closed loop

9.1 Example 1: DC-DC converter

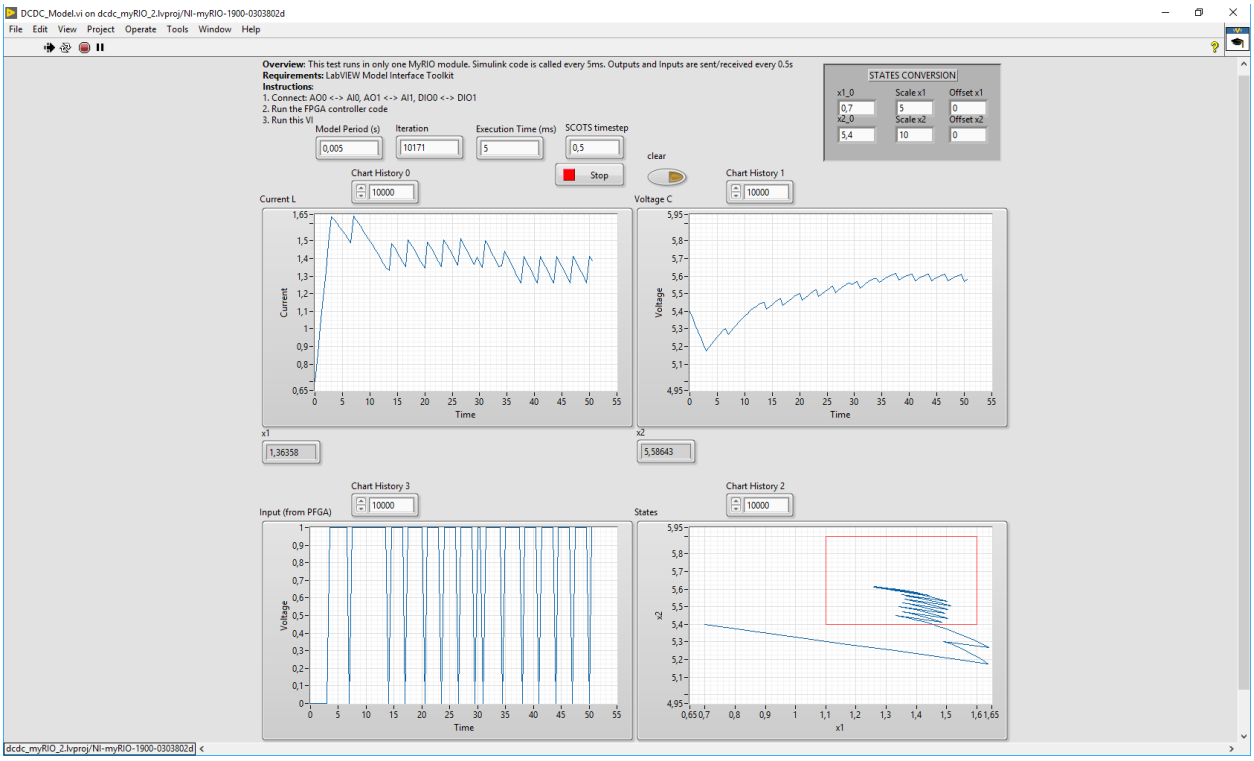
This will be the controller running on the first myRIO FPGA and following the first approach of splitting the controller:



On the other myRIO, we will have on the FPGA side to read the inputs and write the outputs of the plant:



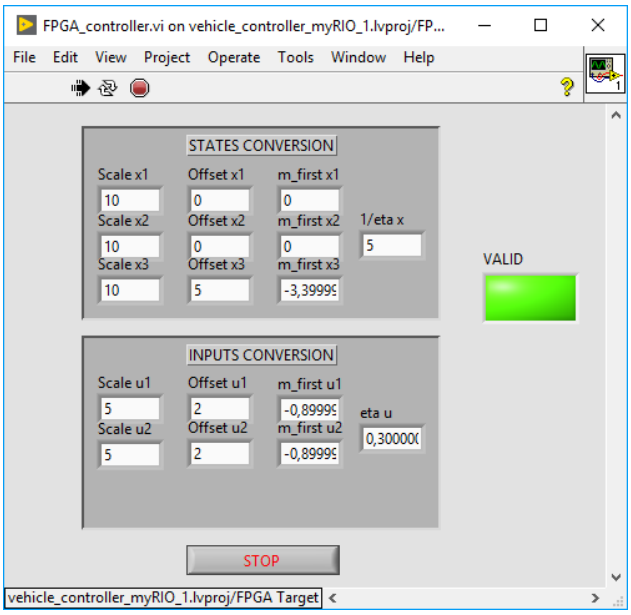
And on the CPU side, the results of the closed loop simulation:



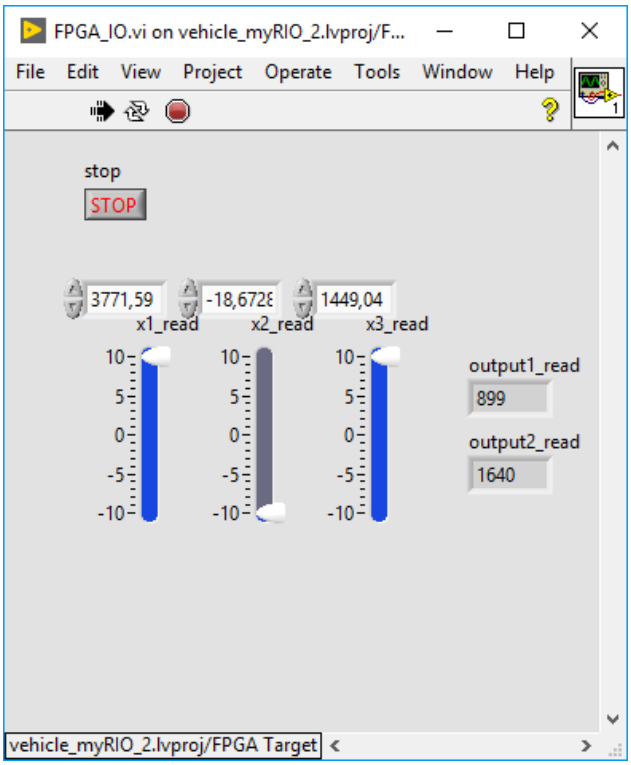
The other approach of running a loop to find the correct control will yield the same simulation results.

9.2 Example 2: Vehicle

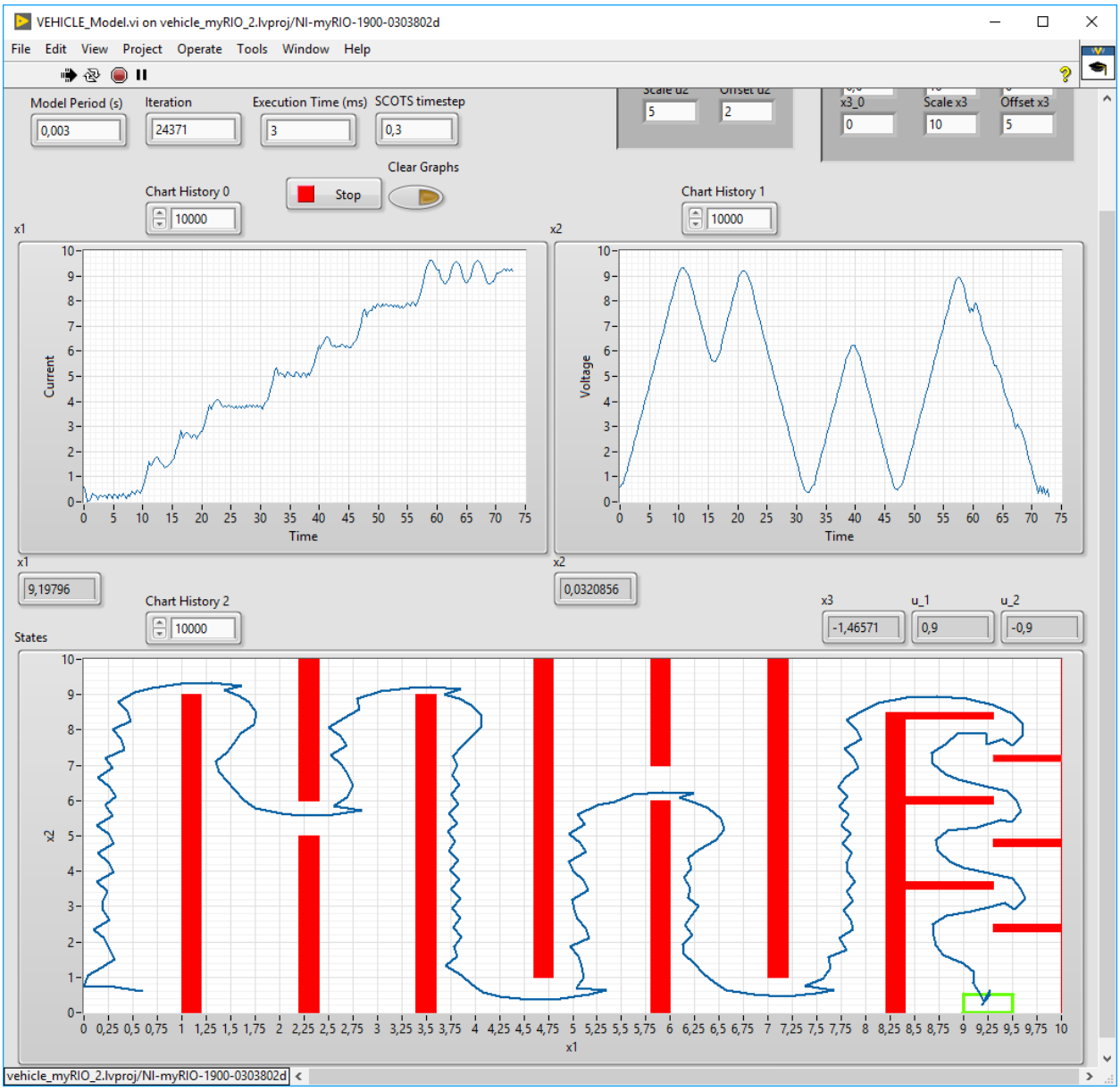
The next image will be the controller running on the first myRIO FPGA:



On the other myRIO, with the help of the FPGA, we will read the inputs and write the outputs of the plant:



And on the CPU side, the simulation will produce:



Please go to the GitHub page of the project to take a deep look at the examples: `https://github.com/ruedales/Thesis_Project`(It is still private).