Software Architecture Development and Implementation of a Synchrophasor-based Real-Time Oscillation Damping Control System

Eldrich Rebello
KTH Royal Institute of Technology
Stockholm, Sweden
Email: rebello@kth.se

Luigi Vanfretti
KTH Royal Institute of Technology
Stockholm, Sweden & Statnett SF, Oslo, Norway
Email: luigi.vanfretti@statnett.no

Md. Shoaib Almas
KTH Royal Institute of Technology
Stockholm, Sweden
Email: msalmas@kth.se

***Abstract*—Low-frequency, electro-mechanically induced, inter-area oscillations have in the past, posed a serious threat to the stability of interconnected power systems. Wide Area Monitoring, Protection and Control (WAMPAC) systems based on wide-area measurements can be exploited to address the inter-area oscillation problem. This work develops a modular software architecture for a prototype WAMPAC control system. A Compact Reconfigurable Input Output (cRIO) controller from National Instruments is used to implement the real-time prototype. This paper presents the design process followed for the development of the software architecture. The design method followed a three step process of design proposal, refinement and attempted implementation. Results from each iteration are used to improve the next. The goals of the design, the challenges faced and the refinements necessary are presented. The details of the final implementation in LabVIEW are also documented.**

I. Introduction

The growth of power system interconnections between previously isolated areas has given rise to the phenomenon of inter-area oscillations. These are low frequency (0.1-2 Hz.) oscillations where the generators of one synchronous area oscillate against those of another area. Damping for these oscillations is generally poor and, if allowed to grow, can lead to disconnection of the tie lines or a collapse of the power system. A famous example of the latter was the August 1996 blackout of the WSCC system in the USA [1]. Although the purpose of system interconnection was to increase stability, the present situation of the power system incorporates renewable energy sources and constrained power transfer corridors, both of which impact system stability.

A. Previous Experiences

Locally available signals such as active power can be used in stabilizing devices such as Power System Stabilisers (PSS). While effective at damping intra-area modes with good observability, these may not be as effective at damping inter-area modes [2] [3]. A theoretical analysis of the advantages of using wide-area signals as a damping input is presented in [4]. Field-test experiences with WAPOD controllers from Norway and China are presented in [2] and [5] respectively. Although these initial field tests show promising results, the wide-area control systems tested so far have been implemented by extending the installed control system of an existing device (e.g. an SVC, see [2]) to receive synchrophasor data, process it and to then feed the control algorithm. To the knowledge of the authors, there has not been any reported attempt at designing a general purpose, wide-area control system, starting from specifications and considering different hardware and software constraints. Such an approach is attractive as it can easily be adapted to different controllable elements thereby reducing implementation costs and facilitating straightforward development.

B. Contributions

The goal of this paper is to document the software architecture development process and challenges faced in the real-time implementation of a wide-area control system. The Phasor Power Oscillation (Phasor POD) algorithm [6] was chosen based on the operational requirements and control system design constraints. The developed prototype is termed a Wide Area Power Oscillation Damper [1] (WAPOD). For purposes of comparison, the real-time SIMULINK implementation of the Phasor POD algorithm in [8] is used as a benchmark. The software architecture designed is implemented in LabVIEW and is deployed on a real-time controller from National Instruments. To test the developed controller, the two-area four-machine model developed by Klein, Rogers and Kundur [9] is used as a test-case and is modified to fit the requirements of experimental testing. A Hardware-in-the-loop experiment is constructed around the eMEGASIM real-time simulator from OPAL-RT [10] with the two-area model running in real-time, physical PMUs and a synchrophasor-based Phasor-POD algorithm running on a cRIO (Figure 1).

C. Paper Outline

This paper is organised as follows. Section II outlines the hardware and software components used together with an

[1]Historically, damping stabilizers have been termed WAPOD where the P represents a measurement of active power through the line. In such context, active power is used as a controller input signal. Although this term is not accurate when other quantities are used as control inputs or feedback signals, the term is used here to maintain consistency with existing literature.

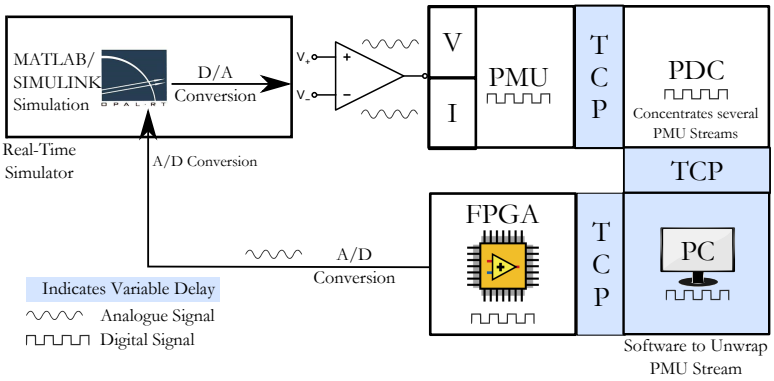

Fig. 1. Data flow path with digital and analogue components indicated

overview of the Phasor POD algorithm implemented. Section III examines the details of the architecture developed for this controller and the various stages of refinement and changes made to it. Section V details the different sections of LabVIEW code written and finally conclusions are drawn in Section VI.

II. Background

A. Phasor POD Algorithm

This algorithm [6] was selected due to its wide applicability and the fact that it does not depend on the network model or updated knowledge of the network topology. Typical model-linearisation based damping algorithms rely on computer-intensive calculations and are often valid for a particular network operating point. In contrast, the Phasor-POD algorithm only requires knowledge of the inter-area oscillation frequency, which is generally known from system studies or can be determined from synchrophasor measurements [7]. Consider Equation (1), which represents a general signal $s(t)$, as an average valued component and an oscillating component.

$$s(t) = s_{avg} + \mathrm{Re}\left\{ \vec{s}_{ph} \cdot e^{jwt} \right\} \tag{1}$$

where $\vec{s}_{ph}$ is a complex phasor, rotating at the frequency ω [6]. The Phasor-POD algorithm uses the known oscillation frequency to set up a co-orindate system rotating at this frequency and continuously extracts a phasor representing the oscillation [6]. The implementation of the algorithm used here is based on the SIMULINK implementation in [8].

B. Hardware

1) Real-Time Controller: The real-time implementation of the Phasor-POD algorithm in this work is specific to the Compact Reconfigurable Input Output (cRIO) controller platform from National Instruments. Specifically, the cRIO 9081 is used to implement the algorithm. It has an on-board Field Programmable Gate Array (FPGA) in addition to a 1.06 GHz. Intel Celeron processor for real-time control applications [11].

2) Phasor Measurement Units: The inputs to the real-time controller were taken from an IEEEC37.118-compliant synchrophasor data stream. Measurement data for this stream was generated by two PMU's which monitor three-phase currents and voltages at different points on the power network. In this work, two cRIO9076 real-time controllers [11] were deployed as PMUs [12].

3) Real-Time Simulator: The eMEGASIM real-time simulator from OPAL-RT [10] was used for simulating the two-area network in real-time. Figure 1 presents the entire signal path including the external, closed-loop controller.

III. Architecture Development Process

Before beginning the development process, a software development methodology was adopted with the aim of streamlining the process[2]. The approach followed here is based on the Waterfall method detailed in [15]. The approach broadly includes the four steps from [15]

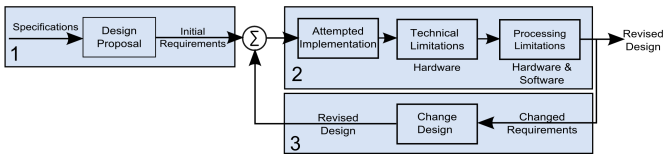

Fig. 2. General Iterative Revision Flow Diagram

1) Initial Investigation
2) Requirements Definition
3) Architecture Design
4) Coding and Implementation
5) Experimental Testing to validate requirements

An additional fifth step could be avoided thanks to the availability and use of HIL testing. The linear waterfall method was modified to be iterative so as to account for various constraints

[2]There are many software development methods that could be adopted. However, for this specific application (i.e a PMU-based WAPOD), none of them have been reported and used in available literature. The process here was favoured over the others due to the availability and limitations of the hardware-in-the-loop experimental testing facilities available at the SmarTS Lab [14].

and to also account for revisions in the initial definition caused by these constraints. This iterative process is illustrated in Figure 2 and is realised for the specific application here, as illustrated in Figure 3. Several reasons were behind the choice of an iterative design approach. Chief among them was the cRIO hardware itself. As an implementation on this hardware had not been attempted earlier, the limitations of the hardware were not fully understood. Based on the limitations faced during an implementation attempt, the design and features incorporated would be revised to fit within the limitations of the hardware.

A. Constraints

Before defining requirements, the possible constraints at each stage of the test set-up were considered. The test set-up shown in Figure 1 is used to illustrate the constraints faced.

Differing Loop Rates: The power system simulation running on the real-time target was executed at a loop rate of 50 μs. This meant that the simulator generated new values for currents and voltages every 50 μs. and would also expect data from the HIL system every 50 μs. The constraint here was the data reporting rate of the PMU's used which was a maximum of 50 samples per second or one sample every 20 ms. This was much slower that the 50 μs. loop rate of the real-time simulator.

FPGA Accuracy: While the exact role of the FPGA in this implementation was not known in advance, it brought with it some limitations. Chief among these was the limitation on data accuracy due to the fact that all calculations together with circuit logic would be implemented in hardware.

IV. Architecture Implementation and Refinement

Figure 2 shows the three-stage design process with design proposal, implementation and revision. To begin with, only the controller specifications were available. These were used to draft a design proposal. An implementation attempt was made using this draft proposal. When the additional limits imposed by the software and hardware platforms were included, some goals of the original implementation needed to be revised. With these limitations, the original design was modified to generate a revised design. An attempt was then made to implement this revised design. As further limitations were encountered, the design was further modified. This iterative process was repeated till a working implementation was derived.

1) Initial Design: Initially, an autonomous, independent controller was envisaged. Such a controller would be able to receive IEEE C37.118 synchrophasor data directly over the communication network and extract measurement data. The architecture block diagram is shown in Figure 3. This controller was completely autonomous, with automated signal selection and processing. This design also incorporated two control functions, a wide area control function and a local control function. The wide area control function selected for implementation was the phasor-based oscillation damping control algorithm [6]. The local control function would use locally available data in a manner similar to a Power System Stabiliser (PSS). Automatic switching between the two functions and automated input signal selection was also considered. If the signal to noise ratio in the wide area signal deteriorated or if the signal delay became prohibitively high, the controller would automatically switch to the local signal or another wide-area signal. The principal consideration in this design was the limited resources available on the FPGA and the complexity of the algorithms to be implemented. This required implementation of all algorithms and computation sections on the RT section of the cRIO with the FPGA being used for interpolation only.

Challenges: This design was faced with a problem of differing loop rates. The real-time simulator runs at a 50 μs time step while the PMUs reported data every 20 ms. The fastest that the RT controller could run was 1ms. Data would thus be generated faster than the RT controller could process. The second problem was producing control output at the rate expected by the real-time simulator. The speed constraint of the RT controller meant that it would not be possible to use it to generate a control signal. To address this issue, this architecture envisaged using the FPGA to perform interpolation between successive data points. The FPGA would run at a loop rate of 50μs, to match the real-time simulator. The region between successive data points would be interpolated using a suitable interpolation algorithm. Data generated by the FPGA would be sent over the communication network back to the real-time simulator for use in the simulation. This design needed to be changed due to limitations of different components.

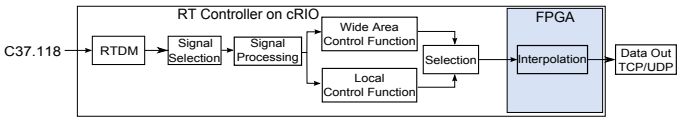

Fig. 3. Initial Software Architecture

One, no software was available to receive a synchrophasor stream and extract measurement data directly on the RT controller. This process had to be performed on a desktop computer running a real-time data mediator (Statnett's Synchrophasor Software Development Toolkit, S^3DK [16]) and LabVIEW. Once measurement data from the synchrophasor stream was available, it could be streamed to the real-time controller over the TCP/IP network using LabView's Shared Network Variables. Two, data generated by the FPGA could not be sent to the real-time simulator directly over the communication network. Though theoretically possible, further work is required. Also, for each successive data point received by the real-time controller, the FPGA would generate 400, interpolated, data points. Synchronising the process of generating the control signal and the use of the generated data on the real-time simulator is a far too complex task. Three, the process of data interpolation on the FPGA is computationally intensive. To achieve results better than with a simple linear

interpolation, a history of past data points is required. This process is, in itself, complex and introduces further delay. It is, however, simpler than implementing the damping control algorithm on the FPGA. At this stage, FPGA limitations were the principal factor behind choosing to implement the Phasor-POD algorithm on the RT controller.

Four, an algorithm for automated signal selection based on observability indices had not been developed. In principle, this too, is possible and can be implemented on the real-time controller in the future when such an algorithm is developed and verified. The Phasor POD algorithm input parameters also change depending on the input used. An approach to determine these parameters iteratively or calculate them from input data is required.

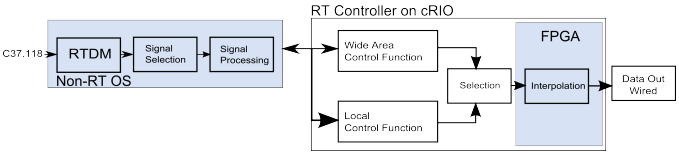

Fig. 4. Revised Architecture

2) Development and Revision: With the limitations of the initial architecture in mind, the architecture was refined. The process of extracting data from the synchrophasor stream was shifted to a workstation computer along with the processes of signal selection and processing. Once data was available, it was sent to the RT controller, over the network. The damping control was kept on the RT controller with the FPGA performing the interpolation required to match the read-interval of the real-time simulator. Besides the damping control algorithm, a local control function was included on the RT controller. This revision is shown in Figure 4. Here, the complexity of implementing an interpolation algorithm on the FPGA was examined in detail. An implementation was also attempted. It was determined that a simple linear interpolation algorithm would not be sufficiently accurate. An implementation of the damping control algorithm on the RT controller was also tested. The fastest loop rate that could be achieved was always greater than 25 ms. This was slower than the reporting rate of the PMUs. Communication between the cRIO and the RT simulator using TCP/IP was also abandoned. Analogue output signals of the FPGA were instead hard-wired to the real-time simulator's analogue inputs.

3) Final Implementation: At this stage, the damping control algorithm was moved to the FPGA with the RT controller being used only for network communication and data monitoring so as to achieve an acceptable loop rate. The Phasor POD algorithm parameters and monitoring data would be sent over the network to the RT section of the controller. The local control function was also discarded as it was found to reduce the response speed of the RT controller. If needed and if space permits, the local control function can be implemented on the FPGA. The desired 50μs. data output rate could also be maintained as the FPGA was capable of response times of this order. The problem of differing data and loop rates was solved by implementing a basic sample-and-hold algorithm on the FPGA. Data would still be received from the RT controller every 20 ms. but the output would be held constant till the next data point was received and processed. With the Phasor POD algorithm implemented on the FPGA, resource utilization stood at 78%. As with the previous design, the process of extracting data from the synchrophasor stream was done on a workstation computer. This (Figure 5) was the final architecture as implemented.

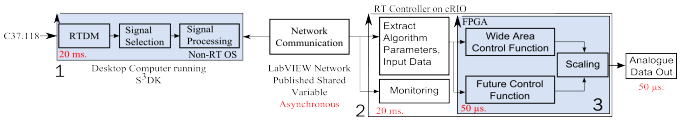

Fig. 5. Final Controller Architecture as Implemented

V. Architecture Implementation using the LabVIEW Platform

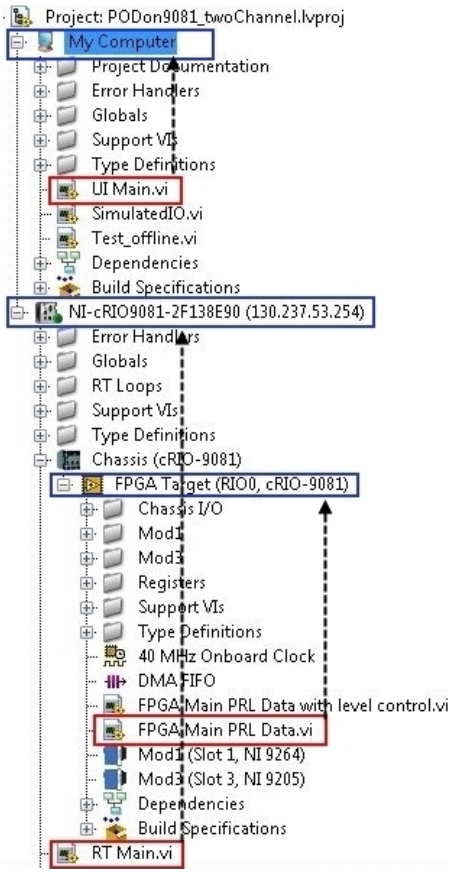

Fig. 6. LabVIEW Project explorer showing where different sections of code are run

The central part of implementing the architecture described here was the code written for the cRIO. This was written using the real-time and FPGA toolboxes available in LabVIEW [11]. Broadly speaking, three layers of code were written

corresponding to labels 1, 2 and 3 in Figure 5. As mentioned before, code here was written with modularity and future expansion in mind. Sections of code in LabVIEW are referred to as Virtual Instruments (VIs) and essentially consist of code with certain inputs and outputs [11]. It is important to note that Figure 6 is the actual implementation of Figure 5 with three, distinct layers each of which is described below.

A. UI Main.vi

The primary VI, called UI Main, corresponds to Label 1 in Figure 5. A screenshot of the interface of this VI is shown in Figure 7. This serves are the main interface to the POD algorithm and allows for monitoring algorithm inputs, outputs and the performance of the cRIO. This VI is responsible for handling communication between the host computer (My Computer, in Figure 6) and the cRIO (NI-cRIO9081 in Figure 6). Data extracted from the PDC stream is read in this VI and is sent over the network to the cRIO. Errors generated by the POD algorithm are also received and stored here. Parameters required for the POD algorithm are entered and displayed here. Presently, the selection of the input signal for the POD algorithm is also performed here. This can be automated in future. Note that this is the only VI with a user interface that can be interacted with. T

B. RT Main.vi

This VI runs on the real-time section of the cRIO at a loop rate of 5ms. It has no user interface as data is transmitted over the network and displayed on the remote real-time target. The primary function of this VI is to act as an intermediary between the POD algorithm running on the FPGA and the user interface on the remote computer. The input to the POD algorithm is received over the network and passed to the FPGA. As such, this VI is designed to be 'headless' and does not require the user interface to be active. In the present implementation, this cannot be achieved as the input to the POD algorithm is generated on the remote computer. Further development is needed to be able to unwrap the PMU data stream on the RT controller itself. Default values of the POD control parameters are stored and the algorithm itself can run without any action from the user. Network errors and latency are also handled here. Logging information and errors can be buffered temporarily before transmission over the network.

C. FPGA Main.vi

This section of code runs on the FPGA and primarily consists of the POD algorithm itself. Additionally, debug, data logging and error checking code is written to ensure that the algorithm responds only to user input. The code here is optimised and compiled for the limited resources available on the FPGA. This is also the only section of code that is directly realised in hardware. The output of the POD algorithm is directly written to the analogue output module in the cRIO. The loop rate here is $50\mu s$. To account for the difference in loop rates between the FPGA and the RT section, a sample and hold algorithm is also implemented here.

VI. Conclusion & Results

Figure 7 shows an outline of the HIL test constructed to verify the working of the developed controller. The results are not examined in detail here due to constraints of space. The reader is, however, referred to [17] for a presentation of the results.

A hardware prototype of a real-time power oscillation damping control system was developed and tested. The developed prototype uses a real-time implementation of a wide-area control system. Issues and challenges faced in this implementation are documented and examined. The architecture development & refinement process for this implementation is also examined in detail. The final software implementation resulting from this process was coded in LabVIEW and is presented. The success of the tests performed indicates that PMU-based wide area controllers can be exploited for multiple control applications in power system control and monitoring.

References

[1] North American Electric Reliability Council *Review of selected 1996 Electric System Disturbances in North America,* August 2002. Available Online: http://www.nerc.com/pa/rrm/ea/System\%20Disturbance\%20Reports20DL/1996SystemDisturbance.pdf

[2] K. Uhlen, L. Vanfretti, M.M. De Oliveira, A.B. Leirbukt,V. H. Aarstrand and J. O. Gjerde *Wide-Area Power Oscillation Damper implementation and testing in the Norwegian transmission network* in Power and Energy Society General Meeting, 2012 IEEE pp. 1-7

[3] M. E. Aboul-Ela, A. A. Sallam, J. D. McCalley and A. A. Fouad, *Damping Controller Design for Power System Oscillations Using Global Signals*, IEEE trans. on Power Systems, Vol. 11, No. 2, May 1996, pp. 767-773

[4] L. Vanfretti, Y. Chompoobutrgool, and J.H. Chow, *Chapter 10: Inter-Area Mode Analysis for Large Power Systems using Synchrophasor Data*, Book Chapter, in Coherency and Model Reduction of Large Power Systems, Joe H. Chow (Ed.), Springer, 2013.

[5] Li Peng and Wu Xiaochen and Lu Chao and Shi Jinghai and Hu Jiong and He Jingbo and Zhao Yong and Aidong Xu *Implementation of CSG's Wide-Area Damping Control System: Overview and experience* in Power Systems Conference and Exposition, 2009. PSCE '09. IEEE/PES pp. 1-9

[6] L. Ängquist and C. Gama *Damping Algorithm based on Phasor Estimation* in Power Engineering Society Winter Meeting, 2001. IEEE, Volume 3, pp. 1160 - 1165

[7] M.Crow, M. Gibbard, A. Messina, J. Pierre, J. Sanchez-Gasca, D. Trudnowski, D. Vowles *Identification of Electromechanical Modes in Power Systems* IEEE Task Force Report, Special Publication TP462, June2012.

[8] M. Shoaib Almas and L. Vanfretti, *Implementation of Conventional PSS and Phasor Based POD for Power Stabilizing Controls for Real-Time Simulation*, IEEE IES IECON14, 29 Oct-1 Nov, 2014, Dallas, USA.

[9] M. Klein, J. G. Rogers and P. Kundur *A fundamental study of inter-area oscillations in Power Systems* IEEE Trans, PWRS, no. 6, pp. 914-921, 1991.

[10] eMEGAsim PowerGrid Real-Time Digital Hardware in the Loop Simulator Opal-RT, [Online]. Available: http://www.opal-rt.com/

[11] *Operating Instructions and Specifications Compact RIO NI cRIO-9075/9076 & NI cRIO-9081/9082*, National Instruments, Available Online at http://www.ni.com/

[12] M. Paolone, A. Borghetti, C. A. Nucci, *A Synchrophasor Estimation Algorithm for the Monitoring of Active Distribution Networks in Steady State and Transient Conditions*, Proc. of the 17th Power Systems Computation Conference (PSCC 2011), Stockholm, Sweden, Aug. 22-26, 2011

[13] Kamwa I. (Hydro-Quebec) "Performance of Three PSS for Interarea Oscillations" Available Online at : http://www.mathworks.se/help/physmod/sps/examples_v2/performance-of-three-pss-for-interarea-oscillations.html

[14] M.S. Almas, M. Baudette, L. Vanfretti, S. Lovlund and J.O. Gjerde "*Synchrophasor network, laboratory and software applications developed in the STRONg^{2}rid project*," PES General Meeting, Conference Exposition, July 2014 IEEE pp 1 - 5

[15] James Taylor *Managing Information Technology Projects : Applying Project Management Strategies to Software, Hardware, and Integration Initiatives*, AMACOM Books, September 2003 pp 237-239
[16] L. Vanfretti, V.H. Aarstrand, M. Shoaib Almas, V. S. Perić and J.O Gjerde *A Software Development Toolkit for Real-Time Synchrophasor Applications*, IEEE PES Grenoble PowerTech, 2013
[17] E. Rebello, L. Vanfretti and M.S Almas *PMU-based Real-Time Damping Control System Software and Hardware Architecture Synthesis and Evaluation* IEEE PES GM, Denver, Colorado, July 2015,

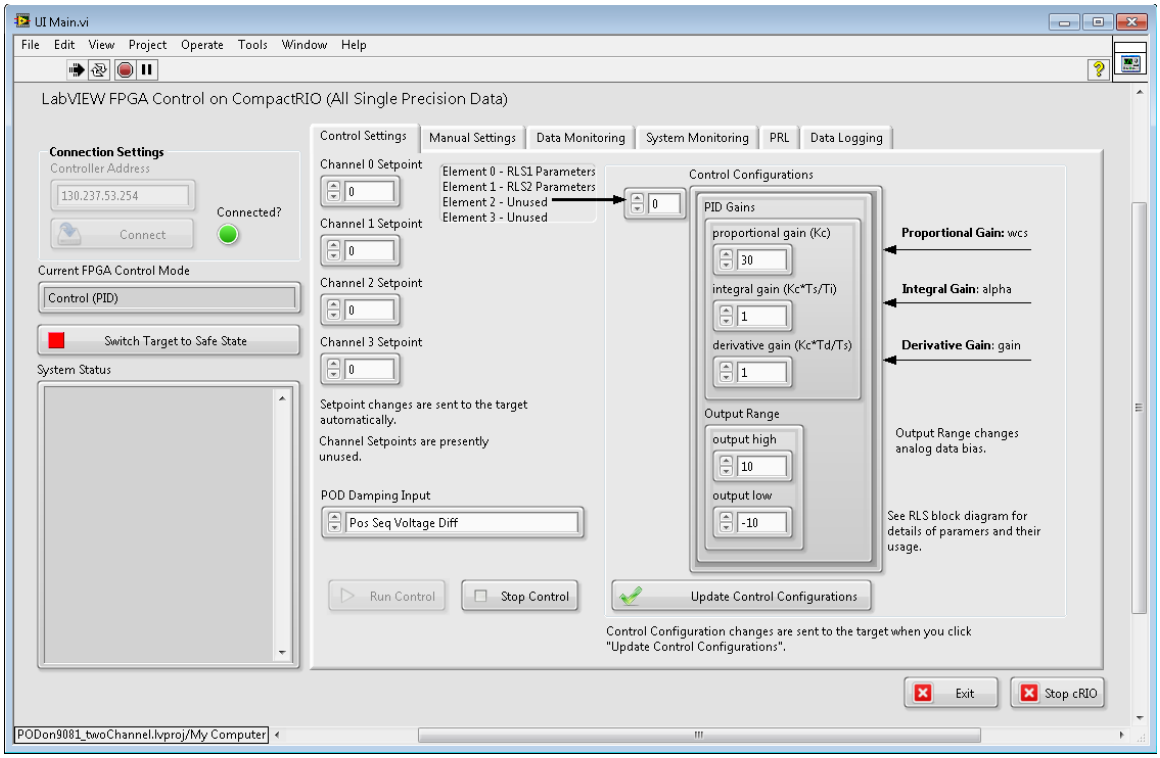

Fig. 7. User interface of *UI Main.vi* showing control buttons, parameter setting, tabbed sections and damping input selection box.