**057-T-20**

**Hardware in the Loop (HIL) for Embedded Code Development in an Introductory Undergraduate Course**

Dale Litwhiler

Penn State, Berks

[dale.litwhiler@psu.edu](mailto:dale.litwhiler@psu.edu)

**Abstract**

Hardware in the Loop (HIL) is a powerful method in the development and testing of embedded processor software. HIL provides a means of simulating a system to be monitored and/or controlled with an embedded device such as a microcontroller. This method allows the embedded device’s code to be developed in parallel with the hardware development of the system thus shortening the overall development time. HIL is also useful when the actual system to be controlled is too complex and/or costly to operate just to verify software during the early stages of development. The ideas of HIL are also very useful in the delivery of undergraduate courses in embedded systems. Using simple computer-controlled hardware, an interface to a simulated system can be created that is much more interesting and insightful than simply blinking an LED or displaying Hello World. The students can also develop the simulator-side HIL hardware and software to further gain understanding of how the overall system operates. This paper presents some of the HIL work developed for and by students in a four-year mechanical engineering program. PIC microcontrollers were used as the target devices for the embedded course. LabVIEW software with associated controllable hardware was used to create the HIL system. Design considerations and practical implementation details are presented and discussed.

**Introduction and Motivation**

Hardware In the Loop (HIL) simulation is a technique where real signals from a controller are connected to a test system that simulates reality, tricking the controller into thinking it is in the actual physical system [1,2]. This technique allows the embedded controller hardware and software to be developed and tested without the need for the actual hardware to be present. This is especially useful in applications where the system to be controlled is complex or costly to operate. Implementing an HIL system also allows for controller development in parallel with that of the system to be controlled [3].

The concepts of HIL can also be applied to academic instruction in an embedded systems course. Here, the desired system to be controlled may not actually exist at the institution but the embedded control algorithms can still be developed and explored. Students can also learn to develop the HIL models to simulate the system to be controlled. This can further enhance their knowledge of how the system works and how it should perform.

Students in the four-year mechanical engineering program at Penn State, Berks may enroll in an elective course that explores the role of hardware and software for measurement and control of electromechanical systems. As a large part of this course, embedded processors (microcontrollers) are used to become familiar with the concepts of making measurements from various sensors. The measurements are then used together with a simple control algorithm to decide what action should be taken. Some sort of control output is then produced to close the loop. In some cases, the HIL hardware and software is used in an open loop manner where it serves more as a simulator that produces a response to the controller output that mimics that of an actual system.

Figure 1 illustrates the basic idea behind hardware in the loop simulation. Students are first introduced to the concepts of HIL by presenting examples of common machinery that contain embedded processors to control their motion. Examples include automobiles and farm/construction equipment. The complexities of the machinery’s motion are then discussed to help the students appreciate the role of embedded control.

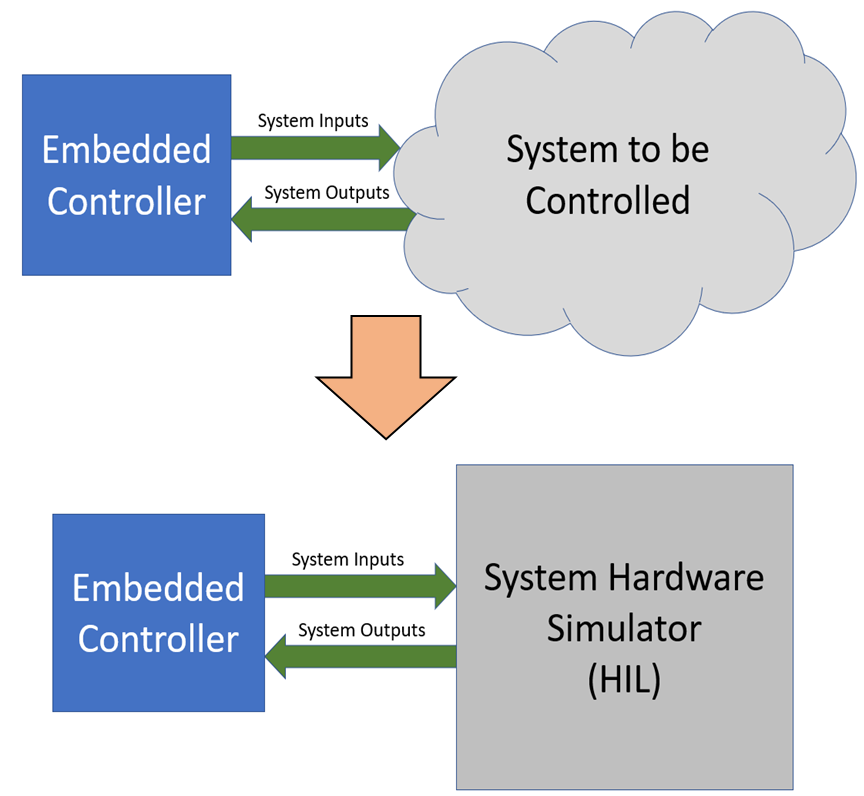


Figure 1. The concept of Hardware in the Loop simulation

Many students are not aware of how ubiquitous embedded processors have become. The motivation for developing the HIL models is to provide the students with an experience that demonstrates the types of systems that *could* be controlled with the embedded processor. Although it is initially somewhat satisfying to blink an LED with embedded code, being able to demonstrate that the same process can be used to control more useful hardware helps to keep students interested and engaged.

**Examples**

Throughout a semester, as the students become more familiar with the embedded processes and the concepts of HIL, the features of the HIL simulators become more sophisticated. For most examples, a National Instruments USB-6003 USB Data Acquisition (DAQ) device is used as the hardware interface. The USB-6003 has several analog inputs, analog outputs, digital input/outputs, and counter inputs [4]. Custom LabVIEW code is used to control the DAQ.

Example 1: Residential Thermostat

The first example is that of a thermostat to control a residential heating system. The controller uses a simple hysteretic (bang-bang) control scheme as would be found in a commercially available thermostat. Interestingly, designing the actual code to realize hysteretic control is challenging to many undergraduate students.

The “house” is modelled to have a first order temperature response. The first order differential equation is discretized with a 50ms Δt time step. The maximum and minimum temperature asymptotes can be adjusted to set “furnace on forever” and “furnace off forever” (outside temperature) temperatures. The house time constant can also be modified to help speed up the simulation. The default time constant is 30 seconds just to keep things moving along but still allowing the user to easily observe the response.

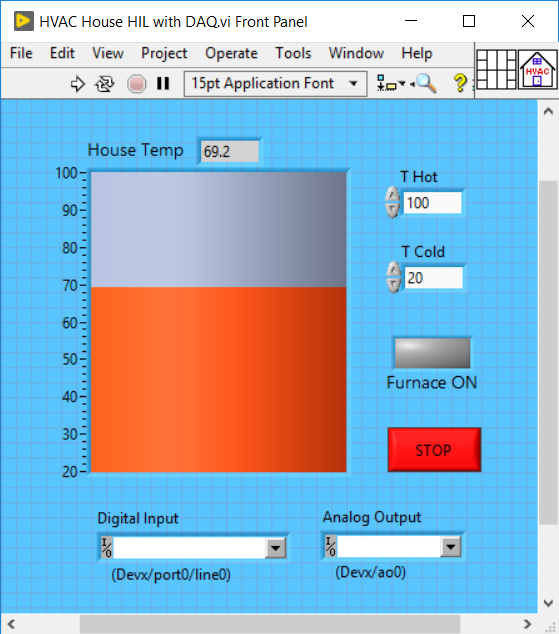


Figure 2. User interface for residential heating system HIL example

Figure 2 shows the user interface (LabVIEW Front Panel) for the HIL house simulator. A flowchart for the LabVIEW code is shown in Figure 3. (For those familiar with LabVIEW code, the block diagram is provided in Appendix A.) One digital input of the DAQ unit is used to determine if the furnace is on or off. One analog output is used to simulate the output from a temperature sensor with a sensitivity of 50mV/°F. A connection diagram for the HIL system is shown in Figure 4.

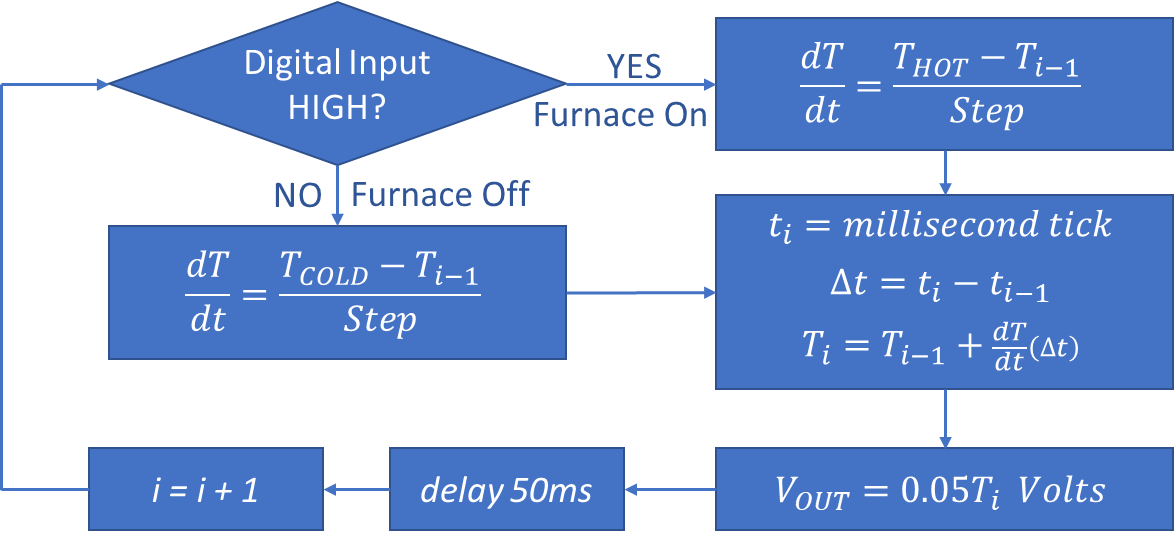


Figure 3. LabVIEW code flowchart for residential heating system HIL example



Figure 4. Connection diagram for residential heating system HIL example

Example 2: Water Flow/Level Control Trainer

The concepts and usefulness of HIL simulation become very evident to the students in this example. The system to be controlled is a water flow/level trainer, *Basic Process Rig model 38-100*, manufactured by Feedback [5]. Figure 5 shows a photograph and functional block diagram of the system [6]. The *interface unit* was constructed to house the solenoid driver and a connector adapter for the water level sensor.

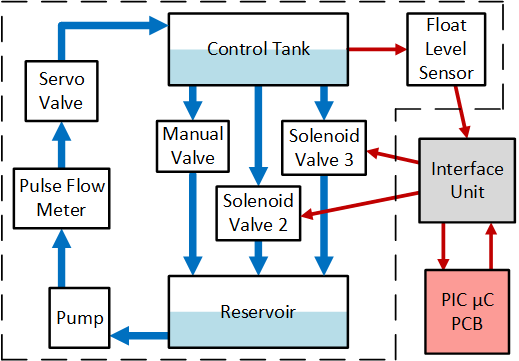
 

Figure 5. Water flow/level control trainer photograph and functional block diagram

The laboratory contains four of the Feedback Basic Process Rig units for students to use. However, due to scheduling constraints in the shared laboratory space and the time required to setup and breakdown the system connections, students needed another means for testing and debugging their microcontroller code. This also provided a perfect situation to emphasize the HIL concept. Therefore, a HIL simulator was developed.

Students were tasked with making measurements of water flow and transducer outputs and control input responses to be used in the HIL model. Again, a USB DAQ device and LabVIEW were used to develop the HIL system. A connection diagram for the HIL hardware is shown in Figure 6.

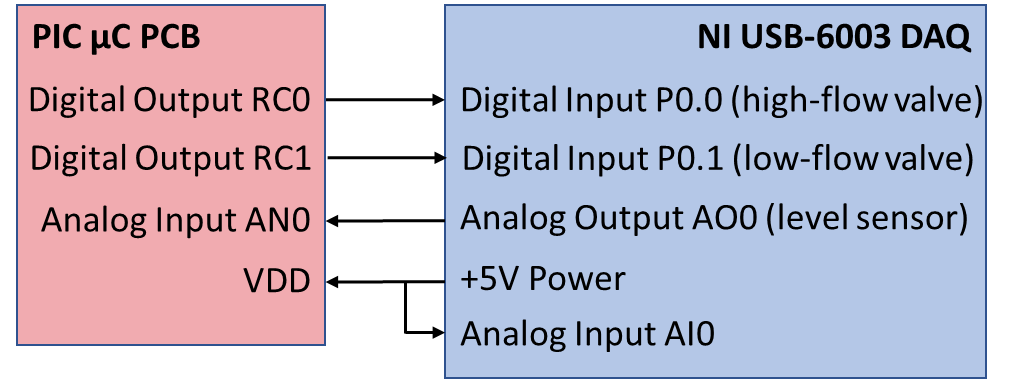


Figure 6. Connection diagram for water level control HIL example

The objective of this example is to control the water level in the *control tank* as set by the user. Water is pumped from the *reservoir* to the *control tank* at a constant rate. The pump flow can be adjusted with a manual valve. The water level in the tank is measured by a float connected to a potentiometer. The control tank can be drained via two solenoid valves of different diameter (*high-flow* valve, SV2, and *low-flow* valve, SV3) and a manual valve. Again, a “simple” hysteretic control algorithm was to be used. The user would set the desired water level as an integer percentage of full (via the serial port). The controller used a ±2 dead band around the desired value. In normal operation, only one solenoid valve was controlled (SV2) however, if the water level exceeded +2 of the desired value, the other solenoid valve would also be opened to arrest the rising water level. Figure 7 shows the LabVIEW user interface for the water level control example.

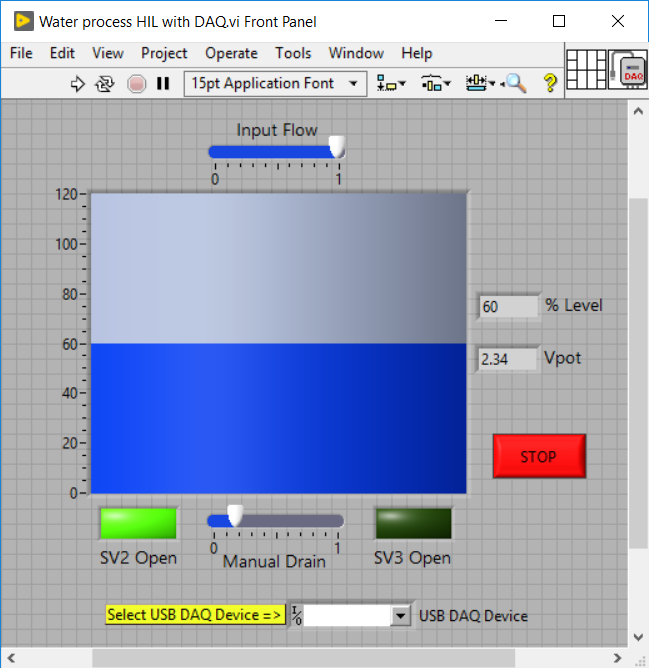


Figure 7. User interface for water level control system HIL example

Table 1. Measured control tank fill/empty rates

|  |  |
| --- | --- |
| **Test Conditi**on | **Water Level Rate of Change (Percent/second)** |
| Only Input Valve Open | 2.466 |
| Only SV2 Open | -1.324 |
| Only SV3 Open | -0.517 |
| Only Manual Drain Valve Open | -3.488 |

The results of measurements on the control tank input and output flows are shown in Table 1. For each test, only one valve was open, and the water level change was timed. The water level was measured using a percentage of full scale as indicated on tank markings. Despite the change in head pressure, the drain rates were essentially constant. These rates were then used in the HIL model equation:

|  |  |
| --- | --- |
|  | (1) |

Where,

*Fin* = Analog position of manual input flow valve, (0.00 through 1.00)

*SV2* = Boolean state of solenoid valve 2, (0 if closed, 1 if open)

*SV3* = Boolean state of solenoid valve 3, (0 if closed, 1 if open)

*MV* = Analog position of manual drain flow valve, (0.00 through 1.00)

As a further exercise in the ways of HIL, the subtle difference between the water level sensing potentiometer output and the DAQ analog output was also accounted for. The potentiometer output voltage from the wiper to either end of the element is *ratiometric* with the power supply voltage. The DAQ analog output voltage, however, is derived from an internal *fixed* voltage reference and is therefore not ratiometric. (The microcontroller A2D converter uses the power supply voltage as its reference, so it is ratiometric.) To correct for this difference, the USB-derived +5V power supply is *measured* with a DAQ analog input channel and the analog output is scaled to mimic ratiometric behavior. The scaling and offset values were again determined by measurements on the actual control tank level measuring potentiometer. For a typical control tank, the analog output voltage as a function of the water level, H, and the power supply voltage, Vcc, is given by equation 2:

|  |  |
| --- | --- |
|  | (2) |

Where,

*Vcc* = Measured power supply voltage (nominally 5V)

*H* = Control tank water level in percent of high mark (0 – 120)

The flowchart for the LabVIEW code used for the water level control system HIL simulator is shown in Figure 8. For those familiar with LabVIEW, the block diagram is provided in Appendix B.

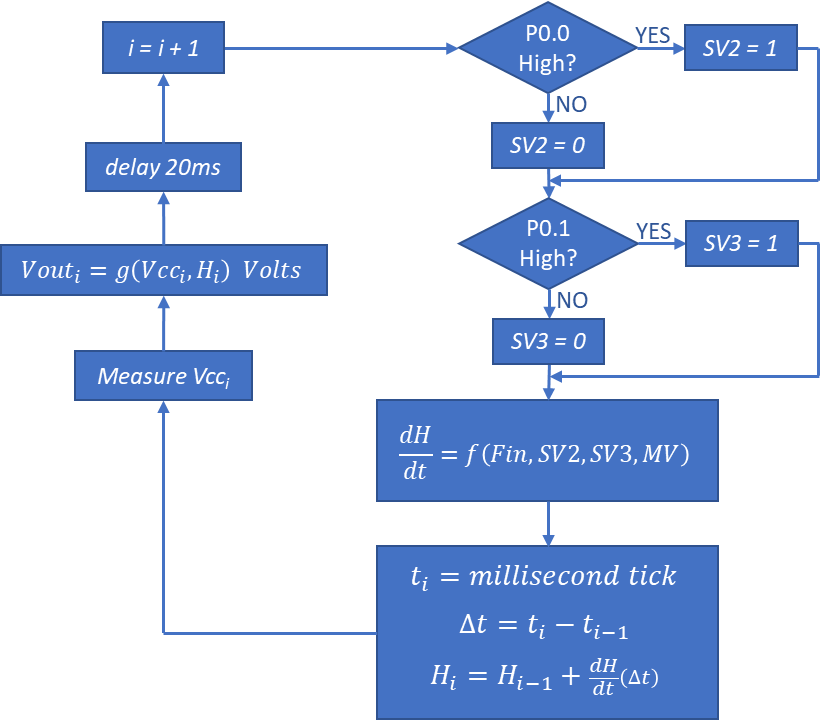


Figure 8. LabVIEW code flowchart for water level control system HIL example

**Conclusions**

The examples presented here, introduce to the concepts of hardware in the loop simulation. By keeping the simulated systems simple and recognizable, the students could easily understand and relate to the expected performance. This helped with their ability to also develop the models for the simulation.

After the students became comfortable with developing code to blink an LED and read the status of an external pushbutton switch, they were quickly ready for something more substantive. Integrating the concepts of HIL, which were already part of the course, provided a means for keeping the coding tasks more interesting.

The residential heating HIL simulation exercise was the students’ first exposure to the concepts required for both the simulation and the embedded processor code. Although both seem simple at first glance, the students were sufficiently challenged and seemed intrigued by the concepts.

The water flow control trainer hardware allowed the students to have a hands-on experience with the actual system that they modelled to help “close the loop” on the ideas. The associated scheduling and setup issues also gave them a small taste of the motivation behind HIL simulation.

The hardware, software, and concepts presented here have also been used to develop other HIL systems to be controlled with an embedded processor in the course. Household appliances such as clothes washers and dishwashers lend themselves to HIL simulation to be controlled by an embedded processor running as a state machine. Stepper motors and DC motors are also good candidates for HIL simulations.

**References**

[1] Retrieved August 2 from <https://www.ni.com/en-us/innovations/white-papers/17/what-is-hardware-in-the-loop-.html>

[2] Retrieved August 2 from <https://www.mathworks.com/help/physmod/simscape/ug/what-is-hardware-in-the-loop-simulation.html>

[3] Ellis, G. (2012). *Control System Design Guide; Using Your Computer to Understand and Diagnose Feedback Controllers*. (4th ed.). Oxford, UK: Elsevier Inc.

[4] Retrieved August 2 from <https://www.ni.com/pdf/manuals/374372a.pdf>

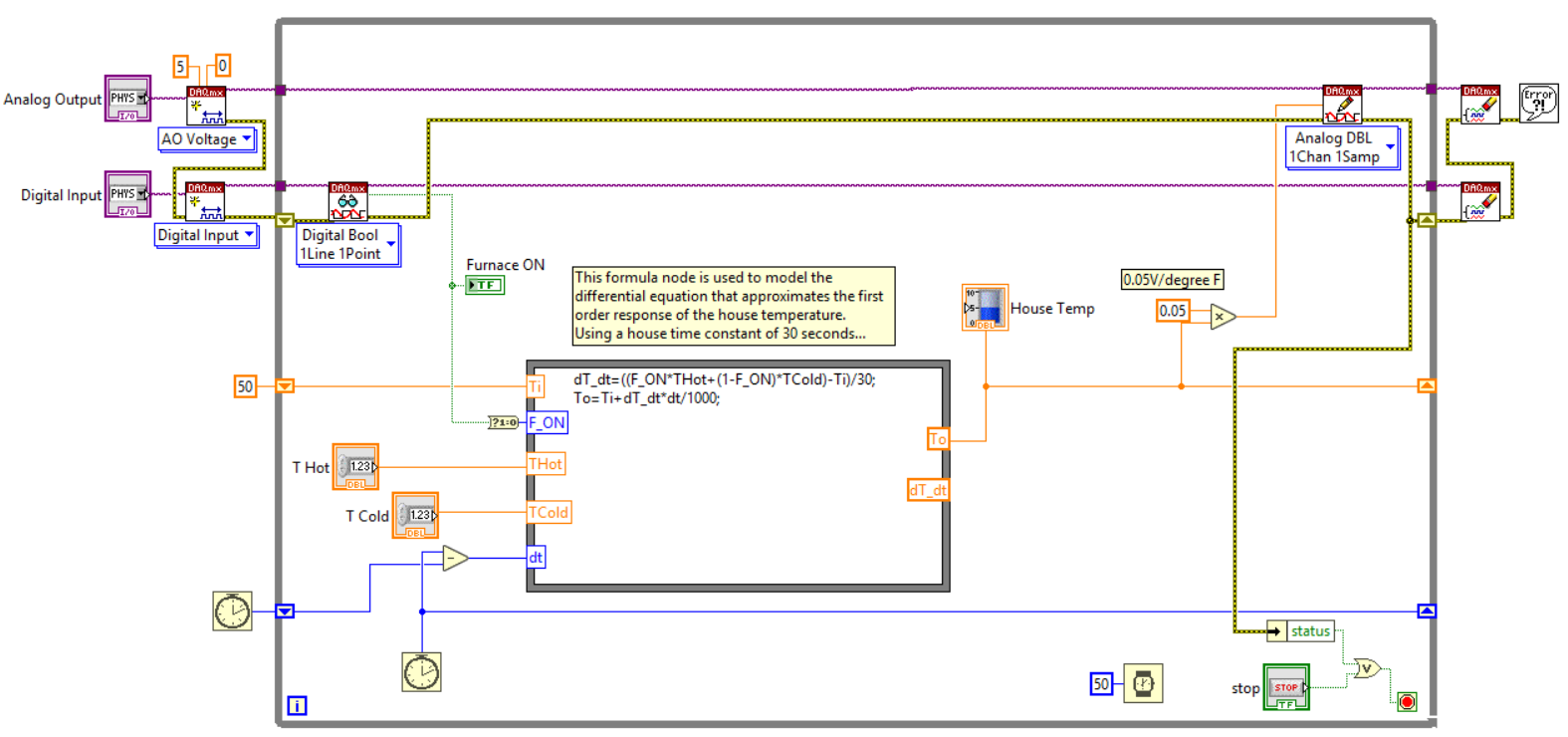
[5] Retrieved August 2 from <http://www.feedback-instruments.com/pdf/brochures/38-001_datasheet_level_flow_control_ESPIAL_10_2013.pdf>

[6] Litwhiler, D. (2012). New Life for Process Control Trainers in a Microcontroller Course. *Proceedings of the 2012 ASEE Annual Conference*, San Antonio, TX.

**Biography**

DALE H. LITWHILER is an Associate Professor of electrical engineering at Penn State, Berks Campus in Reading, PA. He received his B.S. from Penn State University, M.S. from Syracuse University, and Ph.D. from Lehigh University all in electrical engineering. Prior to beginning his academic career, he worked with IBM Federal Systems and Lockheed Martin Commercial Space Systems as a hardware and software design engineer.

Appendix A. LabVIEW block diagram for residential heating system HIL example



Appendix B. LabVIEW block diagram for water level control HIL example

