Remote Real-time Testing Tool

Joseph Clifton
Computer Science and Software Engineering
University of Wisconsin – Platteville
Platteville, WI 53818
clifton@uwplatt.edu

Abstract

In the fall of 2008 and the spring of 2009, the students in our yearlong software engineering project course developed a tool that allows remote testing of real-time embedded systems. The tool is a prototype developed for a local software engineering company as a service-learning project. The company has about 150 employees and does contract work for companies throughout the United States. Many of the projects involve real-time embedded systems. Oftentimes, these systems must remain at the customer's site. If a significant amount of testing could be performed from the home site, it would result in less travel, thus saving money and making employees' families happier. This paper describes the details of that project.

1. Course Background

The University of Wisconsin – Platteville has had a BS-SE degree since fall 1999. The capstone for the software engineering major is two-semester course sequence. The entire class works on a single large project. Students are divided into groups and each group is responsible for completion of a portion of the project. Typically, the project has a service-learning component and often has an industry sponsor.

For the fall 2008 / spring 2009 project, the industry sponsor was a local software engineering company, Esterline Control Systems – AVISTA, hereafter referred to as Avista. [1] They have been in business since 1988, employ approximately 150 software engineers, and do contract work for a variety of companies such as Rockwell, Boeing, GE, Honeywell, John Deere, etc. Many of the employees perform system testing of life-critical software for embedded systems in aircraft and medical devices. Oftentimes, these systems must remain at the customer's site. This means employees have to spend a substantial amount of time at the customer's site. If a significant amount of testing could be performed from the home site, it would result in less travel, thus saving money and making employees' families happier. Hence, the desire was for a tool that allows remote testing of real-time embedded systems.

The first semester, groups are chosen based on compatibility of the team member's available time. The justification is that during the early phases of a project, "together time" is needed to flesh out the requirements.

The students started the first semester by eliciting requirements from Avista employees. They produced problem descriptions, use cases and a requirements document. Then they developed an architectural design, created high-level class diagrams, wrote system level tests, and produced a Software Project Management Plan (SPMP). They also spent a significant amount of time prototyping. The desire was to have a tool that was general enough to handle a variety of real-time embedded system platforms. Prototyping helped determine what was feasible; furthermore, it helped the class decide on a general approach to the project.

The second semester, group assignments are made based on the major functionality expressed in the architectural design. Members choose groups based on interest in the type of work involved with a particular set of functionality. Each team member is responsible for the low-level design, implementation, module level testing, and integration for several classes from the high-level class diagrams. A more traditional approach to artifact ownership is taken, where a given class has a single owner and only that person is allowed to make changes to the design, code, and module-level tests for that class. Furthermore, in the second semester, there is a leader designated for each group. We give the students project experience in Extreme Programming [2] in an earlier course. For this course, our SE faculty members believe that an experience in a process-heavy project is important for our graduates, since there are still many companies that follow such practices, particularly those companies developing life-critical software.

Development of the tool turned out to be an excellent project for the students. It was challenging, and for most students, required exploration into new areas. Risks and trade-offs were discussed throughout the project. An object-oriented design and implementation resulted in over 130 source files (not counting the NUnit test files) with three major components. There were enough facets to the project that all sixteen students could find something interesting and remain engaged through the year.

2. Project Description

Based on 2008 market surveys, it is estimated that the total market for Embedded Software Engineering is over $25 Billion, with spending on software and tools representing a large portion of that amount. [3] A quick search of the web reveals many tools for testing real-time embedded systems, with lists of such tools given in [4, 5, 6]. Of the tools listed, only Rational Test RealTime lists test execution remote monitoring, and then only lists start, synchronize, and stop capabilities. [7] In a conference in Madrid in June 2008, Happonen presented a paper on remote embedded testing proposing that Tree and Tabular Combined Notation (TTCN) be used as a standard for remote control of Built-In Tests (BIT) of an embedded system; however, even that is limited in scope. [8]

Given the lack of commercial tools for remote testing of real-time systems, engineers at Avista had developed a few specialized versions of such tools. When we approached them about suitable projects for our yearlong capstone software engineering project course sequence, development of a generalized remote testing tool was high on their list. We believed that this project was the right scope and would be an excellent experience for our students.

Given that the goal was to develop a generalized tool that allowed remote testing of real-time embedded systems, the students along with Avista software engineers defined what the capabilities should be. Desired functionality included the ability of the tester to do the following:

- Reserve a block of time to test a given system
- Connect to a server at the designated time to perform real-time tests
- Reprogram the remote target system with the software to be tested
- Control / manipulate test inputs to the system
- Observe outputs and behavior of the system
- Perform automatic logging and retrieval of test results
- Create test scripts to aid in test automation
- Perform at least a limited amount of target-level debugging

It was also desirable to have tools that allow administration of the testing of multiple real-time embedded systems across multiple sites. This required the following administrative capabilities:

- User addition, removal, and modification
- Management of user data and user rights
- Management of servers
- Administration of sign-up times for multiple test stations
- User access control
- Management of test-related files such as scripts and logs

To do testing, a host computer is generally connected to the target microcontroller of the embedded system via a programmer and/or debugger interface. In addition, the host computer is also connected to testing equipment (controller hardware) that is connected to the target system. The testing equipment allows specification of inputs to the real-time system and monitoring of outputs from the real-time system. It is desirable to automate as much of the test setup as feasible and to make that setup available in a generic format to the testing tool. This includes additional tools that could:

- Create script files that express the capabilities of the embedded system's microcontroller(s), such as discrete inputs and outputs (DIO), Analog-to-Digital Converter (ADC) inputs, Digital-to-Analog Converter (DAC) outputs, etc.
- Create script files that express the physical connections between the inputs and outputs of the test equipment and those of the embedded system

2.1 Architecture

Based on the above desires, the class decomposed the tool into three major components:

- Administrative Website – runs on a single server and is accessed by everyone in the company
- Hardware Server – runs on each host computer and allows remote control of the programmer /debugger and test equipment
- Testing Client – runs on the tester's computer and allows access to the hardware server

In addition, the class developed a few smaller tools to help automate some of the script generation as specified above.

Figure 1 depicts the architecture with a sample setup for a single hardware server hosting an embedded system based on the PIC16F877A microcontroller. A Microchip MPLAB ICD 2 is used as the Programmer / Debugger. A National Instruments USB-6008 Multifunction Digital Acquisition device is used as the Controller and provides analog inputs, analog outputs, and digital I/O. [9, 10, 11]

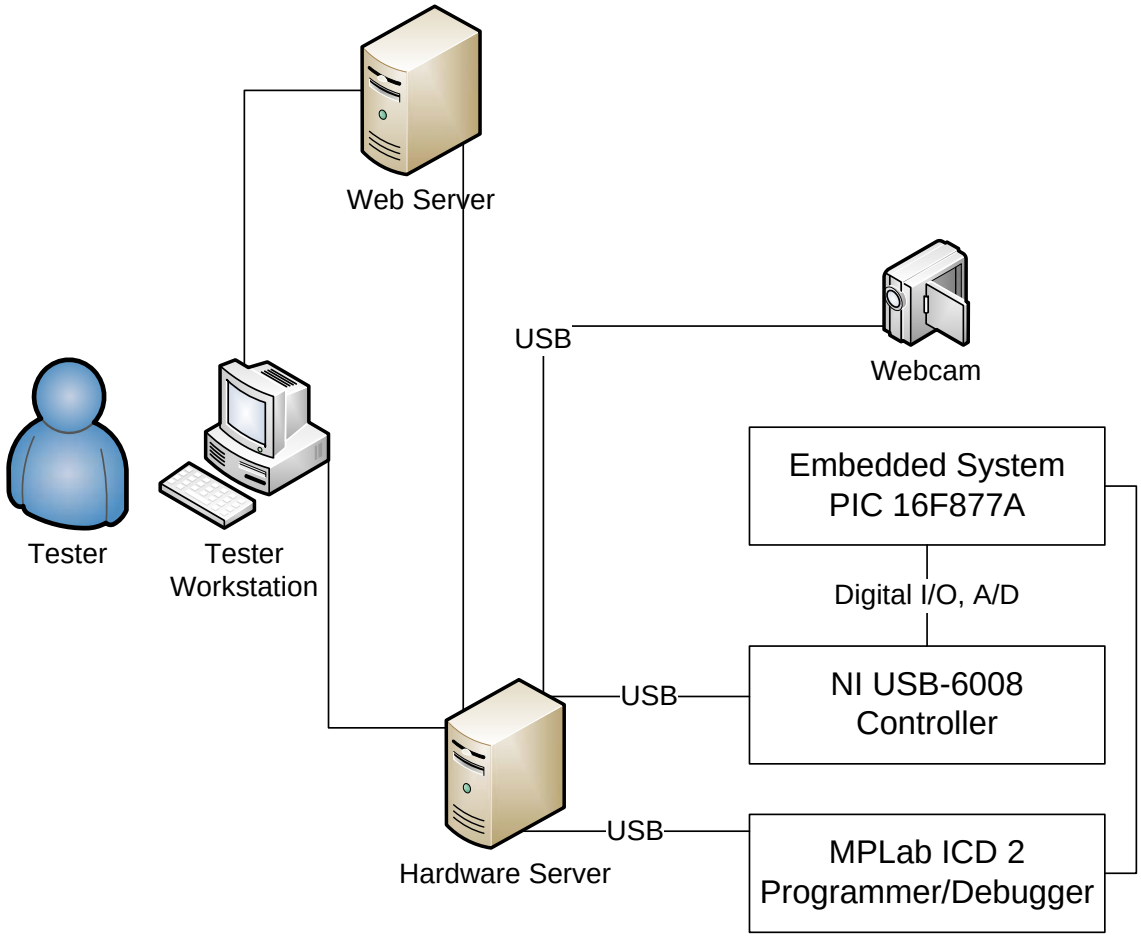

Figure 1: Architecture

2.2 Administrative Website

The website allows administration of multiple real-time projects across multiple sites. Each project is associated with a hardware server that provides access to the real-time embedded system to be tested. Aspects for each project, such as users and their access rights; the hardware server name and IP address; and test scheduling, can be administered.

The class debated among several options for the website. The students constructed prototypes for a few alternatives. There were many passionate discussions about what platforms, languages, and tools would be the most appropriate. Alternative prototypes included use of Java, C#, Servlets, JSP, PHP, JavaScript, AJAX, and ASP .NET. The architecture that was finally agreed upon by a majority of the students in the class was to use ASP .NET and C# with a SQL Server 2005 database.

Figure 2 shows the main page of the administrative website that comes up after a successful login, assuming the user is an admin. The left side displays system messages and any pending testing sessions for the current day.

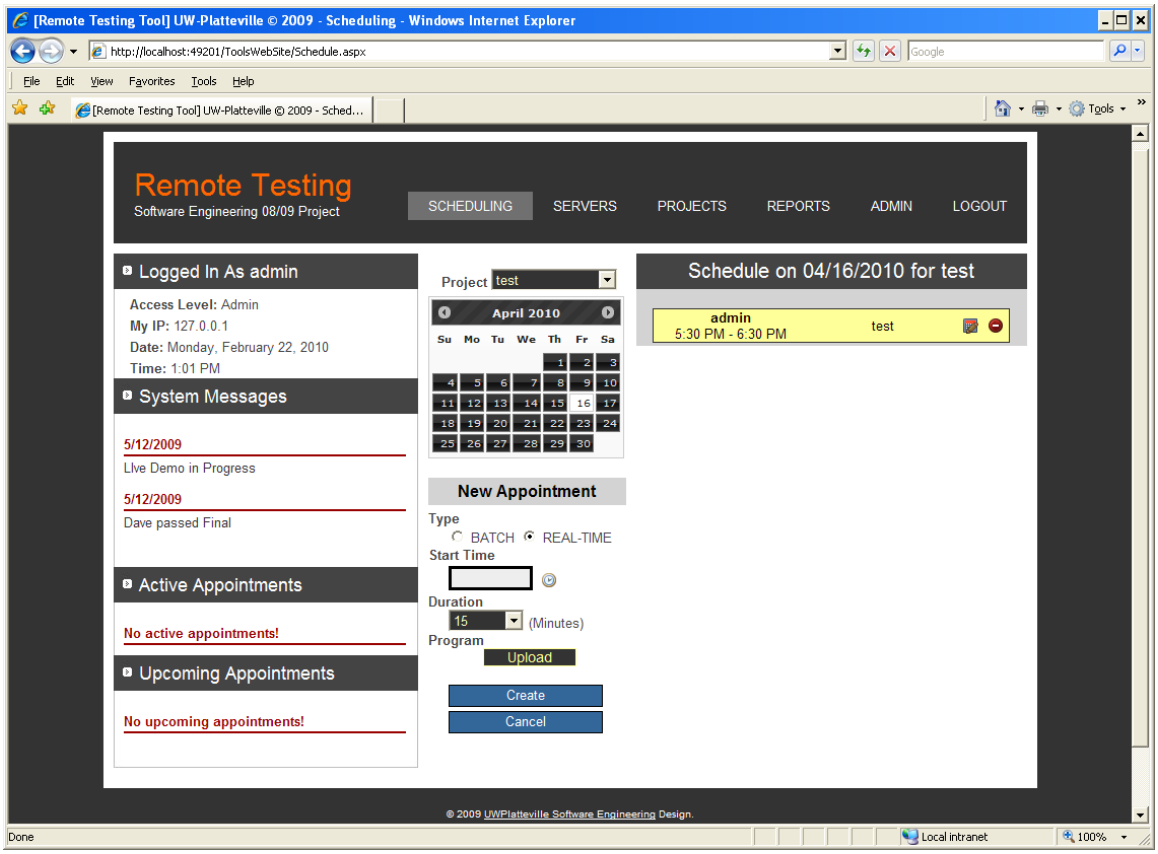

Figure 2: Administrative Web Site – Scheduling

The “Scheduling” tab, which is the default tab, allows the user to schedule a real-time testing session. Besides specifying the date, start time, and duration, the user specifies the program to be tested, for example, a hex file.

Figure 3 shows the "Servers" tab after a server has been selected from a list. It allows the details of a hardware server to be viewed or modified, depending on the user's rights. The IP Address and the Port number define a unique hardware server. It is possible that a given host machine can run more than one hardware server, provided it has enough physical hardware connections to attach to multiple programmers/debuggers and testing equipment. In addition, the port for the webcam and hardware server configuration file is specified.

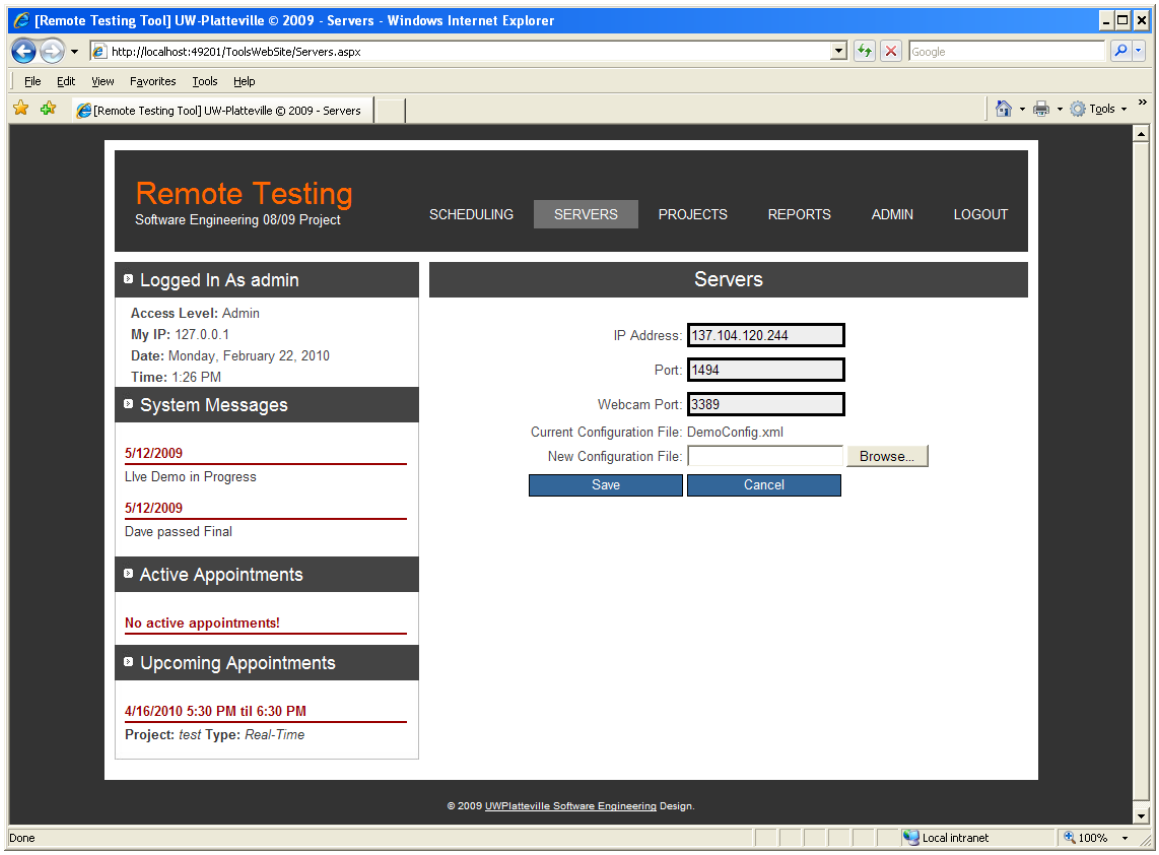

Figure 3: Administrative Web Site – Servers

The "Projects" tab displays the details about the different testing projects. The "Reports" tab displays reports related to the tests and projects. The reports were left as “future enhancements” since there wasn’t enough time to implement everything.

Figure 4 shows the “Admin” tab. The top two buttons allow hardware servers to be created and edited. These buttons take you to the same web page as displayed in Figure 3. The next set of buttons allows the admin to create projects, view projects, and assign testers and other users to a project. A project must have a name and associated hardware server (IP Address and Port number). In addition, it may have a description and the appointment length interval time can be specified (default is 15 minutes). The third set of buttons allows the admin to add or edit users. The admin must specify the user’s login name, given name, email, password, phone number, and rights/privileges. Current choices for privileges are Tester, Admin, Setup, and None. “Setup” privileges allow the user to set up a hardware server. “None” is there to disable a user. Users are never deleted, since for an audit, the tester who performed a given test must be available in an audit report, even if the tester has left the company. The bottom button is used to send global messages to users, such as a hardware server is down.

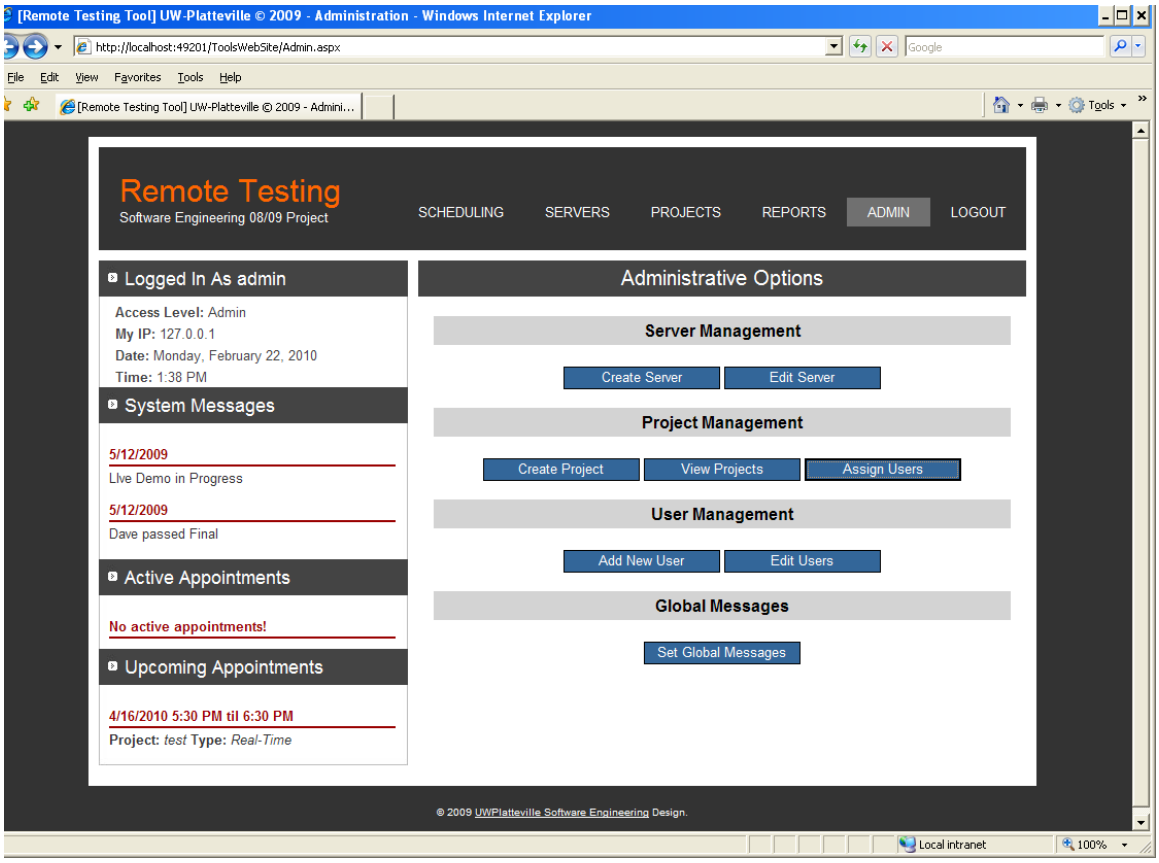

Figure 4: Administrative Web Site – Admin

2.3 Hardware Server

The hardware server provides access to the real-time embedded system. It receives commands and sends status to the testing client in real-time via sockets. It also provides a real-time webcam stream to the testing client.

The connection to the microcontroller is abstracted into Controller access and Programmer/Debugger access. A Controller is used to manipulate the inputs and report the outputs of the embedded system. A Programmer/Debugger allows program downloading and target debugging. The Programmer/Debugger can be connected to the host PC in various ways, such as USB, serial or JTAG.

An XML configuration file is used to specify the properties of the microcontroller, the Programmer/Debugger, and the Controller. In addition, the configuration file specifies how the microcontroller and controller are interconnected. Figure 8 shows a sample configuration file. The students also wrote tools to help produce configuration files. These are discussed further in section 2.5.

Abstract interfaces are provided for the Controller and Programmer/Debugger. Concrete Controller and Programmer/Debugger instances that implement the interfaces are provided via Dynamic Link Libraries (DLLs). This allows different Controller and Programmer/Debugger devices to be used without the need for recompilation of the project. The particular DLLs used for a given hardware server are specified in the configuration file.

2.4 Testing Client

The stand-alone testing client provides access to a hardware server. This client allows the tester to control the real-time tests remotely. The user can specify values (on, off) for discrete inputs to the embedded system. Furthermore, values can be specified for ADC inputs. The values for discrete and DAC outputs can be monitored. A web camera window allows the tester to view the system while tests are being performed. A scripting language is provided to allow automation of tests.

Figure 5 shows the testing client after a successful login. A successful login requires the tester to specify a name, password, and project. Furthermore, the tester, using the administrative web server, must have reserved the current block of time. The setup for the embedded system shown in Figure 5 is the same as that depicted in Figure 1.

The menu allows for erasing and programming the microcontroller. It also allows the target program to be started, stopped, and paused. The students did not implement target debugging (a feature for a future release). The right side of the menu shows the status, in this case, “Running”.

On the left of the testing client are controls to manipulate the target inputs. In the sample, there are four digital inputs and two analog inputs. On the right of the testing client are controls to monitor the target outputs. In the sample, there are four digital outputs and one analog output.

The top middle panel displays the output from a real-time webcam. The bottom middle panel allows scripts to be typed or read from a file. The script in the “Script Input” text area is executed when the “Send Commands” button is clicked, and the output is displayed in the bottom text area. The scripting language is described in Section 2.5.

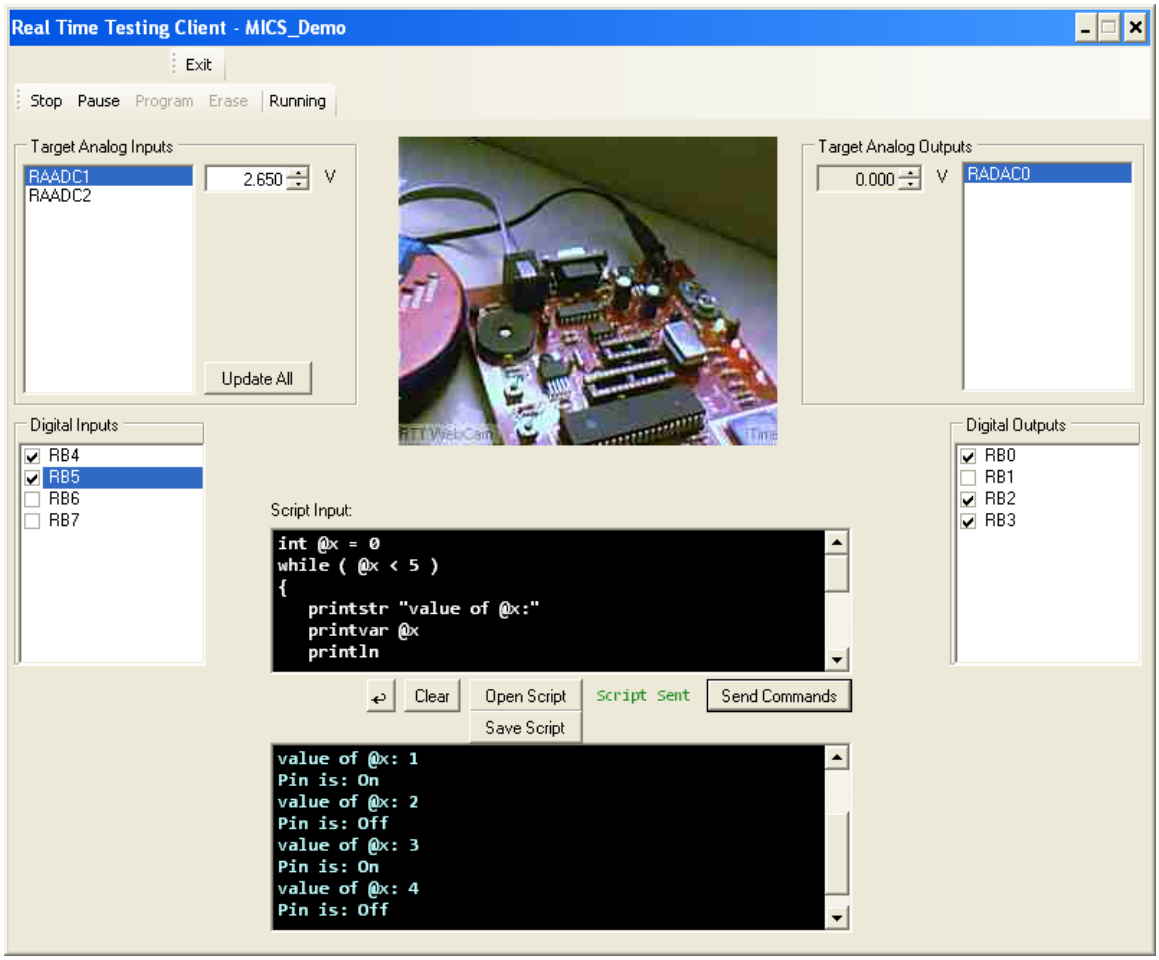

Figure 5: Testing Client

2.5 Scripting Language

A scripting language was developed to allow automation of tests. Two students worked for most of the second semester developing the language and interpreter. They created a BNF for the desired language, which included block statements as well as "if", "while", and "for" constructs. They successfully implemented the interpreter and performed considerable testing. There are still a few bugs when trying to interpret some complicated sequences of instructions; however, it performs well for normal usage.

Variables are prefixed with the "@" character while hardware, such as digital I/O pins, are prefixed with the "$" character. Three versions of print statements are provided for output. A wait instruction is provided for timing. Furthermore, instructions to erase and program the microcontroller as well as start, stop, and pause tests are provided.

Below is a sample script that illustrates many of the language features. The script together with the last eight lines of output it produced are shown in Figure 5. The

program in the PIC16F877A microcontroller “blinks” RB0 (which is connected to the right-most LED) when the RB4 input is high, which is verified by the displayed output.

```
int @x = 0
while ( @x < 5 )
{
   printstr "value of @x:"
   printvar @x
   println
   printstr "Pin is: "
   if ( $RB0 == false )
      printstr "Off"
   else
      printstr "On"
   println
   @x = @x + 1
   wait 500
}
```

2.6 Scripting Tools

The students wrote two tools to aid in the creation of XML configuration files. Figure 6 shows the tool used to create a target hardware description XML file. It allows specification of analog and digital ports and the individual pins within the port. For digital pins, the directionality (input or output or bidirectional) can be specified. For analog, properties such as minimum value, maximum value, precision, and sampling rate can be specified. Ideally, the tool should have provided specification for more microcontroller capabilities; however, the students ran out of time.

Figure 7 shows the tool used to create a hardware server configuration file. The user first selects the target hardware description file (created using the tool described above). In addition, the user selects the DLLs for the Controller and Programmer/Debugger. After that, the user can drag a controller pin from the right panel to associate it with a microcontroller pin in the left panel.

Figure 8 shows a sample configuration file produced by the tool. Note that the target device as well as the controller and debugger DLLs are specified. Also, the pin associations between the controller and device are specified, for example, target pin RB0 is connected to controller pin Dev2/port0/line0.

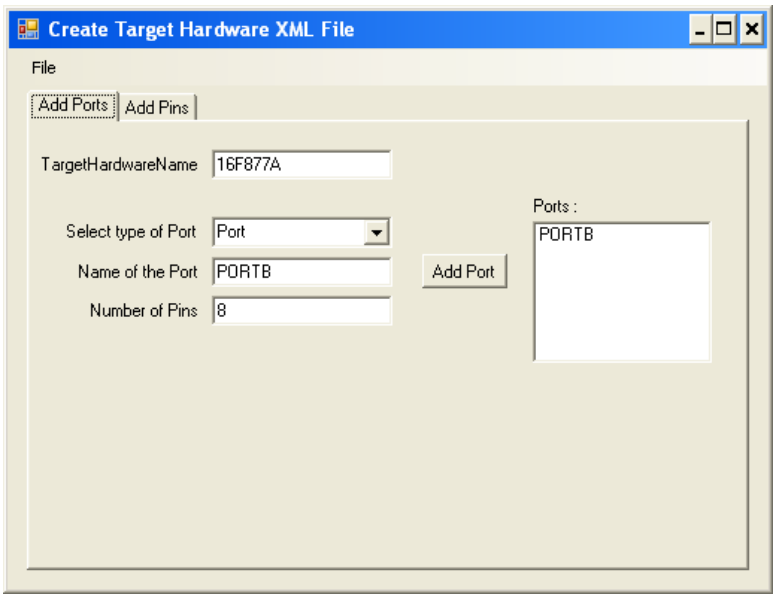

Figure 6: Tool to Create a Target Hardware Description File

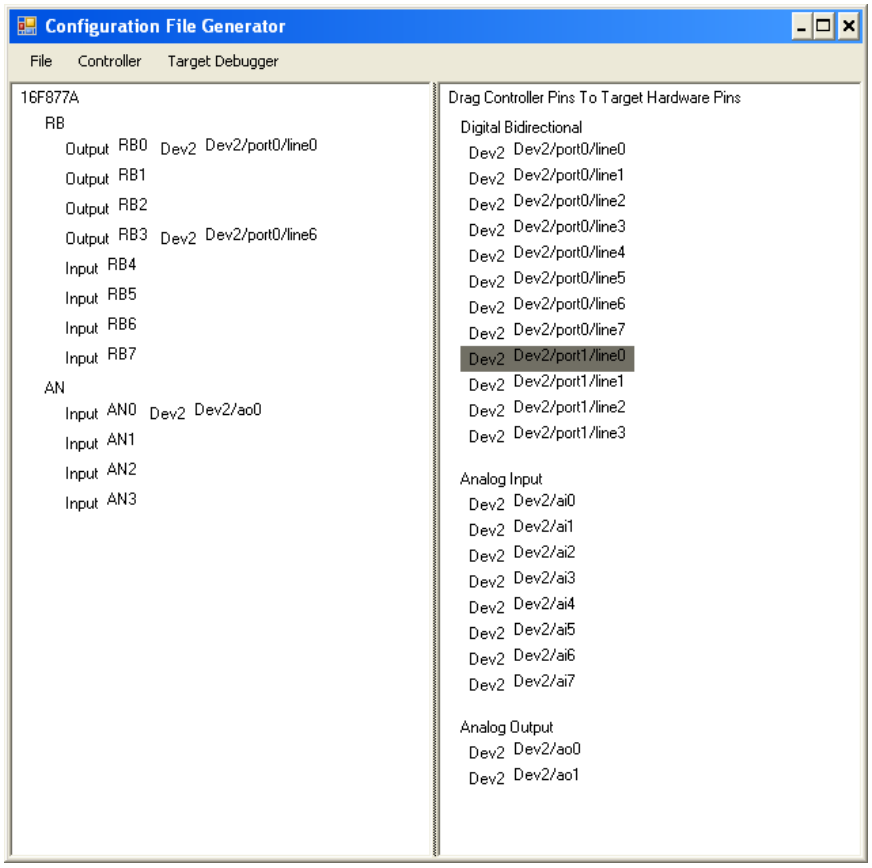

Figure 7: Tool to Create a Hardware Server Configuration File

```
<?xml version="1.0" encoding="utf-16"?>
<HardwareServerSettings>
   <TargetDebugger dll="DriverDLLs\Debugger_ICD2_Programmer.dll">
      <Interface>usb</Interface>
      <DeviceName>16F877A</DeviceName>
      <FirmwareDirectory>DriverDLLs\ICD2</FirmwareDirectory>
   </TargetDebugger>
   <Controller dll="DriverDLLs\Controller_USB-6008.dll">
      <id>1</id>
      <Interface>usb</Interface>
   </Controller>
   <Resources>
      <ADC name="RAADC">
         <Width>10</Width>
         <SamplingRate>1</SamplingRate>
         <Precision>0.001</Precision>
         <MinVal>0</MinVal>
         <MaxVal>5</MaxVal>
         <Pin Position="1" Direction="Input" TargetPinName="RAADC1">
            <Controller>1</Controller>
            <ControllerPin>Dev2/ao0</ControllerPin>
         </Pin>
      </ADC>
      <Port name="RB">
         <Width>8</Width>
         <Pin Position="0" Direction="Output" TargetPinName="RB0">
            <Controller>1</Controller>
            <ControllerPin>Dev2/port0/line0</ControllerPin>
         </Pin>
         <Pin Position="7" Direction="Input" TargetPinName="RB7">
            <Controller>1</Controller>
            <ControllerPin>Dev2/port0/line7</ControllerPin>
         </Pin>
      </Port>
   </Resources>
</HardwareServerSettings>
```

Figure 8: Sample Hardware Server Configuration File

3. Conclusion

The remote real-time testing tool was an ambitious project. It was poorly defined at first, but the students helped define reasonable functionality. The resulting tool is far from production quality; however, it can be a prototype for a future Avista endeavor. The

architecture and implementation were kept general so that it should be extensible. Having the students create clean components helped. The weakest link turned out to be the testing client, which the author thought would be one of the more straightforward components given the prototyping done during the first semester. The resultant testing client ended up with very tight coupling that severely hindered integration. The testing client, as implemented, required all pieces to be fully functioning before it could do anything.

The students were required to log and document all time spent on the project. The second semester, students averaged about ten hours per week, which we believe is about right for a three-credit capstone course.

Overall, the author (instructor) is very proud of what the students accomplished. Since we take turns teaching this course, the author teaches it every other year. Particularly the second semester, the author has relearned the following lessons:

- Assure consistency and thoroughness in module tests, especially for the most critical components
- Require virtual versions of the components interfacing to hardware and insist that major components be able to function with these versions
- Require that the major components provide threads of functionality in the absence of some components
- Do a final check to assure the setup and usage documentation is detailed and correct
- Spend more time spot-checking key components and try to catch potential problems early
- Check for strong coupling, early and often, requiring redesign and refactoring when necessary
- Don’t allow a group leader to give an introverted student responsibility for a major component!

Acknowledgement

The author would like to acknowledge the students in the software engineering capstone project courses that worked on the tool: Bryan Bannach, Eric Bracke, Jason Bulgrin, Blake Devcich, Matthew Dolfin, Samuel Eiring, Michael Hagel, Catherine Huttenhoff, David Manske, Joseph Nowak, Kyle Rafac, David Rulseh, Michael Shesta, Joseph Sims, Jeffrey Theusch, and Jason Watson.

References

[1] Esterline Control Systems – AVISTA, Available at http://www.avistainc.com/

[2] Wells, D., "Extreme Programming: A Gentle Introduction", Available at http://www.extremeprogramming.org/

[3] On Target: Embedded Systems, VDC Research Measuring Total Size of Embedded Sofware Engineering Market, February, 2009, Available at http://ontargetembedded.blogspot.com/2009/02/over-25-billion-vdc-research-releases.html

[4] Test Coverage Tools, Available at http://www.testingfaqs.org/t-eval.html

[5] Embedded Testing, Available at http://www.iturls.com/English/TechHotspot/TH_et.asp

[6] Software QA Testing and Test Tool Resources, Available at http://www.aptest.com/resources.html

[7] How Embedded Systems Issues Affect Testing Process and Technology, Available at http://www.ibm.com/developerworks/rational/library/459.html

[8] Happonen, T., "TTCN-3 Applicability to Remote Embedded Testing of Electronics", TTCN-3 User Conference, Madrid, Spain, June 2008.

[9] Microchip PIC16F87XA Data Sheet, Available at http://ww1.microchip.com/downloads/en/devicedoc/39582b.pdf

[10] MPLAB ICD 2 In-Circuit Debugger/Programmer Data Sheet, Available at http://ww1.microchip.com/downloads/en/devicedoc/51264b.pdf

[11] National Instruments USB-6008 12-Bit, 10 kS/s Low-Cost Multifunction DAQ Data Sheet, Available at http://sine.ni.com/nips/cds/print/p/lang/en/nid/14604