Preliminary Design of a Systematic Gain Calibration Tool Using MATLAB

Samuel Fanfan
Cornell University
Department of Electrical & Computer Engineering

in collaboration with
Brookhaven National Laboratory

Samuel Fanfan
sf85@cornell.edu
Advisor: Bruce Land
ECE 4920 Sr. ECE Project

Supervisor: Om Singh
Project Supervisor: Kurt Vetter

Abstract

This paper presents a project done in collaboration with Brookhaven National Lab for the application of Radio Frequency Beam Position Monitors (BPM). These Radio Frequency Beam Position Monitors are to be used in the particle accelerator facility (NSLS-II) post construction. The purpose of the RF BPM is to detect transverse beam position with sub-micron resolution and stability for normal operation conditions. The most crucial parameter of the BPM to accurately measure the position is for the gains of the four channels to be matched. Traditional calibration schemes comprise of manually adjusting a signal generator coupled to the RF BPM unit under test and recording the receiver signal strength for various power level settings. A MATLAB "RF BPM Cal Tool" has been developed to automate the process. The tool controls the signal generator and communicates with the BPM unit, both over TCP/IP, and was developed utilizing the MathWorks Instrument Control Toolbox. Remote control of the signal generator is executed using the VISA-TCP protocol in conjunction with SCPI commands. Communication with the BPM is established using the ModBus-TCP protocol and the raw data is read and handled by the Cal Tool. A created graphical user interface implements the aforementioned functionality. A final post-processing step is implemented in the GUI and exports the calculated systematic gain coefficients to a user-specified filed to be used for further verification. Testing of the tool proved successful, although a ModBus-TCP Slave program was used to simulate the yet-to-be-completed BPM unit under test. Further testing will be required once the final hardware is completed.

Introduction

Brookhaven's current light source – the National Synchrotron Light Source (NSLS) – is one of the world's most widely used scientific facilities. Each year its bright beams of x-rays, ultraviolet light, and infrared light are used by researchers worldwide for experiments in diverse fields such as biology, chemistry, physics, and materials science. To meet the critical scientific challenges surrounding the energy future a new facility – NSLS II – is currently under construction. This new light source will utilize a medium-energy electron storage ring capable of producing x-rays 10,000 brighter than the current NSLS.

The NSLS-II facility produces "bright" light sources by means of a particle accelerator. For the purposes of the various experiments it is necessary to have accurate readings and measurements of the beam (the accelerating particle). Various devices are incorporated into the accelerator system for this purpose, with one being the Radio Frequency Beam Position Monitor. The primary purpose of the RF BPM is to detect the transverse beam position with a resolution of approximately 200nm. The RF BPM will also be required to support turn-by-turn measurements primarily for beam dynamic studies, and single pass beam position.

There are two primary components to channel-to-channel gain variation. The first is time-varying variations induced from factors including temperature variation, externally coupled noise, and intrinsic noise. The second source is systematic gain variations which are defined as the static gain differences of each channel when measured at an operational ambient temperature. Correcting the dynamic gain variations is a task left to the engineers of Brookhaven National Lab (Vetter et all). Systematic gain variations may be removed in a lab setting by calibrating the RF

BPM devices. Traditional calibration schemes comprise of manually adjusting a signal generator coupled to the RF BPM unit under test and recording the receiver signal strength for various power level settings. In order to automate this process, a calibration tool has been proposed, to be designed using the MATLAB design environment. Calibration consists of a Rhode & Schwartz SMA 100A Signal Generator, the proposed "RF Cal Tool," and the RF BPM Unit Under Test (UUT). The RF Cal Tool will utilize a Graphical User Interface (GUI). The following illustration depicts the architecture and functionality of the tool.

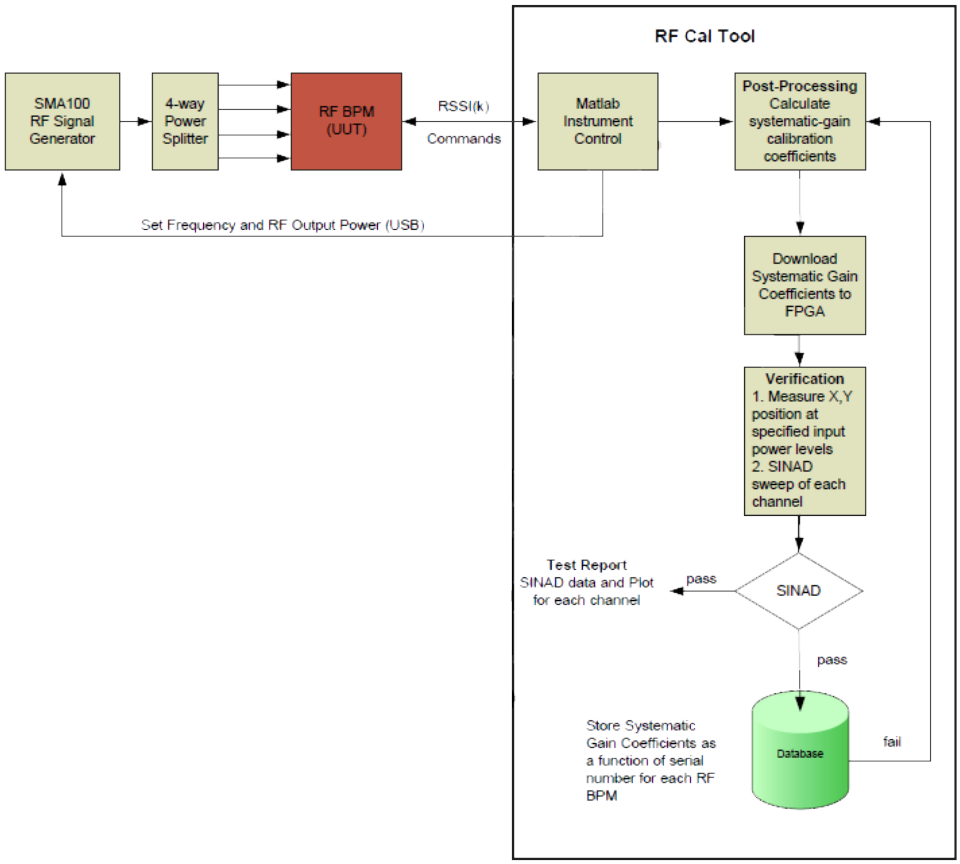

Figure 1. High level architecture of RF Cal Tool (Vetter et all)

The RF Cal Tool will be designed using the MathWorks Instrument Control Toolbox software package to communicate with the instruments. Communication with the R&S SMA 100A signal generator will be done remotely via a Local Area Network (LAN) interface. The Cal Tool will also communicate to the RF BPM via the RF BPM Slow Acquisition port over TCP/IP. The Slow Acquisition port is being designed to use the ModBus TCP/IP protocol. The MATLAB-developed tool will request an averaged value of the Receiver Signal Strength (RSSI) from the RF BPM for each of the four channels under various input settings. After completing acquisition of RSSI data the RF Cal Tool will calculate the systematic gain coefficients. Once all the coefficients have calculated they are to be downloaded to an FPGA for further verification. To achieve this end the calculated coefficients shall be stored in a data file following a format specified by Brookhaven National Laboratory.

The RF Cal Tool architecture is detailed before describing the design process.

Program Architecture

i. VISA-TCP

The Rhode & Schwartz SMA100A signal generator is equipped with several interfaces for remote control. The Network Interface Card for the LAN interface uses 10/100/1000Mbps Ethernet IEEE 802.3u standard and provides supports for the VISA protocol. The Virtual Instrument Software Architecture (VISA) is a widely used I/O API in industry for communicating with instruments from a PC. One of VISA's advantages is that it uses many of the same operations to communicate with instruments regardless of the interface type. This allows for interchangeability among instruments.

Since the VISA protocol is being used, a VISA-TCP object is constructed in MATLAB. For this to work an active installation of the VISA API from any of the vendor companies; the National Instruments API was throughout the design process. In addition to specifying the API vendor one must also specify the resource string in the command. The format of the *visa* command is as follows:

obj = visa('vendor', 'resourcename', …),

where the ellipse indicates the option of additional property names and values that may be initialized. The resource name provides the created object with the information necessary for locating and connecting to the instrument. For a TCP/IP interface the resource name is a string with the format

TCPIP::ipaddress::inst0::INSTR

It should be noted that the "TCPIP" header of the resource name is all that lets the object know the VISA protocol will be used over TCP/IP. Specifying use of VISA over a different interface is simply a matter of choosing the appropriate header. After substituting the instrument's IP address into the resource name, the VISA-TCP object is ready for communication. Once the MATLAB host computer is connected to the signal generator via Ethernet, the connection can be opened using the *fopen* command.

Though the connection with the instrument has been established an instruction signaling to switch to remote control mode must be sent before the instrument can be controlled remotely. Following the standardized VISA protocol for instrument automation is the Standard Commands for Programmable Instruments (SCPI) command set, an industry standard for controlling programmable test and measurement instruments. SCPI specifies a common syntax, command structure, and data formats, to be used with all instruments (R&S SMA100A manual). SCPI commands are ASCII textual strings and may comprise a series of one or more keywords, many of which take parameters. Commands are sent to the instrument over the physical layer. In the MATLAB workspace this is done by creating the string command and sending using either the *fprintf* or *query* functions. The *fprintf* function is used to write text to the instrument; *query* is used to read from the instrument. Query string commands must end with '?' to designate the instruction as a request for information from the client (MATLAB). Responses to queries may be stored to a variable name in the MATLAB workspace for future use (i.e. *iden = query(obj1, '*idn?');*). Similar to all commands, responses to queries are formatted as ASII textual strings. These strings may be reformatted upon receipt for proper use.

The signal generator can be switched from manual to remote control by sending the *'&GTR'* string over an open connection (using *fprintf*). From here on all of the signal generator's many functions maybe be invoked or statuses checked by use of the proper SCPI command/query. The R&S SMA100A product manual details all the allowed functions. Once a remote control session is completed, the instrument can be switched to manual control from MATLAB using the *'GTL'* command. It is recommended that the device be reset before switching back to local mode.

After successfully establishing the VISA-TCP connection through MATLAB, all that remains is developing a script for proper automation per the requirements set forth by Brookhaven National Laboratory. This entails setting the signal generator to a specified frequency, setting the output waveform to continuous mode, and sweeping RF output power in 1 dB increments from -100 dBm to +10 dBm (decibels (dB) of the measured power referenced to one milliwatt). Using the signal generator's instruction manual this simply becomes a matter of sending the appropriate commands in the correct order to achieve desired functionality. A script was developed (see Attached) for experimenting with various SCPI commands to achieve the desired functionality. The results of this experiment are described later on.

ii. ModBus-TCP

Following the specifications outlined by Brookhaven National Laboratory, the RF Cal Tool will communicate with the RF BPM via the RF BPM Slow Acquisition port using the TCP/IP interface. The BPM Slow Acquisition Port will use the ModBus TCP/IP Protocol (Vetter et all). The RF Cal Tool will request an averaged value of the Receiver Signal Strength for each of the four channels for each input setting. The MATLAB Instrument Control Toolbox software does not provide direct support for the ModBus-TCP protocol so much time was spent understanding the dynamics of the standard.

ModBus is an application layer messaging protocol intended for supervision and control of automation equipment (Swales). Published as a 'de-facto' industry standard, ModBus is a vendor neutral communication protocol most commonly found in use with Programmable Logic Controllers, Input/Output modules, and gateways to other simple field buses or I/O networks. By definition ModBus is a request/reply messaging system and offers services specified by function codes.

ModBus-TCP/IP (MB-TCP) is a variation of the ModBus family of protocols for use over a TCP/IP interface. The listening TCP port 502 is reserved for MB-TCP communications, although another port may be dedicated to ModBus over TCP based on the specific application. The ModBus messaging service is provided using a Client/Server model to facilitate communication between devices connected on an Ethernet network (Swales). The original ModBus protocol defines a simple Protocol Data Unit (PDU) independent of the underlying communication layers. The mapping of ModBus protocol on specific buses or networks introduces additional fields on the Application Data Unit (ADU). The figure below depicts the structure of a typical ModBus frame.

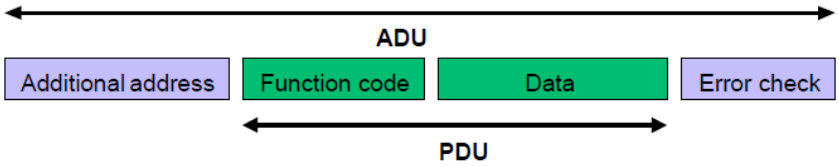

Figure 2. Standard ModBus frame structure (The ModBus Organization)

The client that initiates the ModBus transaction builds the ModBus ADU before communicating with the ModBus device. The function code indicates to the server which kind of action to perform. ModBus uses 'big-endian' representation for addresses and data items so that when a numerical quantity larger than a single byte is transmitted, the most significant byte is sent first.

ModBus-TCP exhibits near-identical functionality as the standard ModBus protocol except for changes to the frame structure. For ModBus-TCP, the ModBus frame is encapsulated in an Ethernet frame's Data/Payload field before transmission. Since the TCP/IP protocol incorporates its own means for error correction and the Ethernet frame contains a destination address, the Application Data Unit of the ModBus frame contains a different structure, depicted in the following figure.

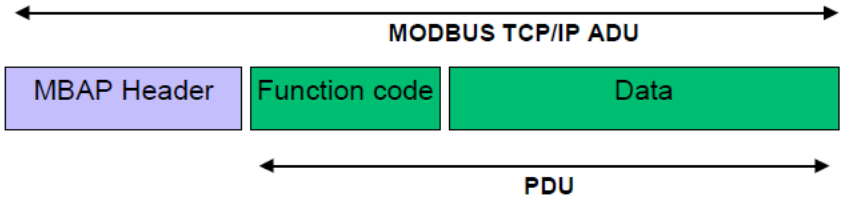

Figure 3. ModBus-TCP frame structure (The ModBus Organization)

When ModBus is carried over TCP, additional length information is carried in the prefix to allow the recipient to recognize message boundaries even if the message had to be split into multiple packets for transmission. The existence of explicit and implicit length rules, and use of a CRC-32 error check code (on Ethernet) results in an infinitesimal chance of undetected corruption to a request or response message. A dedicated header is used on TCP/IP to identify the ModBus Application Data Unit(ADU). The actual byte structure of the ADU is described as follows:

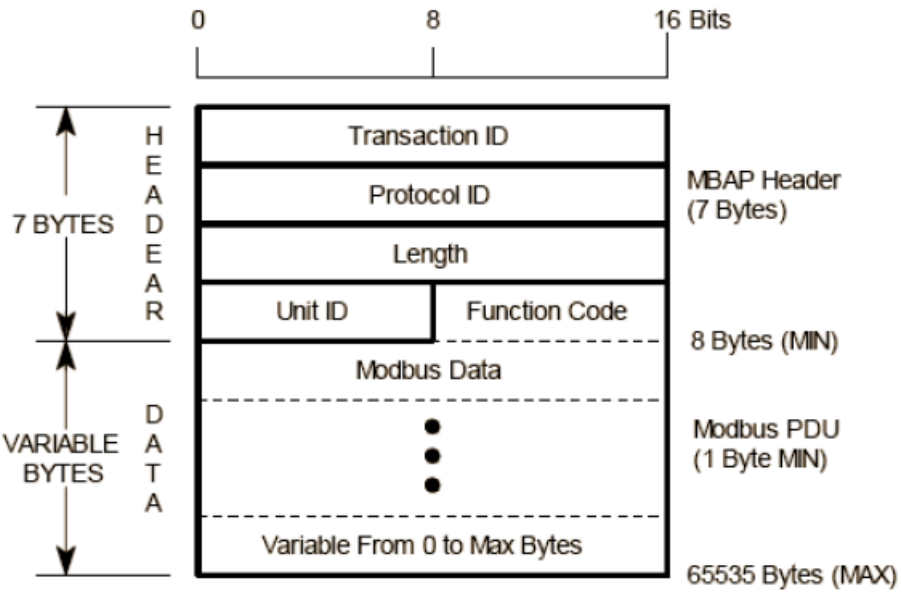

Figure 4. Byte structure of ModBus-TCP frame (Vetter et all)

- 2 Byte Transaction Identifier used to identify a ModBus request/response action. This field is initialized by the client in a request and copied by the server for the response. Comparing the transaction IDs of the response to a specific request can be used for error checking.
- 2 Byte bytes for the Protocol Identifier. Per the ModBus standard this field is always initialized to 0 by the client and copied by the server for the response message.
- 2 Bytes (upper and lower) denoting the length field that represents the number of bytes following (i.e. Unit ID + Function Code + Data). This field is initialized by client for a request and by the server for a response. Some functions utilize a variable data field while others do not.
- 1 Byte specifying the Unit Identifier used to identify the remote server device. This field is initialized by the client and recopied by the server. Request made where the Unit Identifier field does not match that of the slave device are ignored by the server.
- The 1 byte Function Code field specifying the function to be executed. This field is also initialized by the client and recopied by the server.
- The variable-byte Data field contains information related to a request or response. The structure of this filed may be dependent on the function being used.

ModBus bases its data model on a series of tables which have distinguishing characteristic (The ModBus Organization). The four primary tables are input discretes (single bit, read only), output discretes (single bit, alterable by an application program with read/write capabilities), input registers (16-bit quantity, read only) and output registers (16-bit quantity, alterable by an application program). For each of the primary tables the protocol allows individual selection of 2^{16} = 65536 data items. Read/write operations of those items are designed to span multiple consecutive data items up to a data size limit which is dependent on the transaction function code.

One potential source of confusion is the relationship between the reference numbers used in ModBus numbers were expressed as decimal numbers with a starting offset of 1. However ModBus uses the more natural software interpretation of an unsigned integer index starting at zero.

It should be noted that MATLAB has limitations with the ModBus-TCP protocol. The Instrument Control Toolbox only includes support for client functionality, so trying to develop a ModBus server using the MATLAB environment is not doable.

The process for establishing communication with a ModBus-TCP device using MATLAB is now detailed. Since the BPM units that will need calibration are still being manufactured, a ModBus-TCP simulator program was used to emulate the BPM interface. *"ModBus Slave"* is a program used for simulating up to 32 slave/server devices using a variety of interface options (Serial, GPIB, TCP/IP, etc). It was specifically made for the purpose of programming and testing while the actual ModBus device is still being manufactured. The program provides support for the most common ModBus functions, mainly register and input read/write functions.

The instrument control toolbox is designed to handle TCP/IP communication, so interfacing with a ModBus-TCP is boils down to properly parsing and interpreting the ModBus frame in the data field of the received Ethernet frame. For requests, the frame must be created in MATLAB before being encapsulated in a TCP/IP message and sent to the ModBus device.

A TCP/IP object is created using the *tcpip* function, making sure to specify the remote host address and port number (default port for ModBus-TCP is 502). Next the ADU header field is constructed - segments of the header field are initialized as a vector in the MATLAB workspace. By default variables declared in the MATLAB environment are of type double, with each element requiring 8 bytes for representation. For proper communication over ModBus-TCP, each element of a vector must occupy 1 byte. This is achieved by use of MATLAB's *uint8* function which converts any type to an unsigned 8-bit (1 Byte) integer. So after the request has been constructed in MATLAB it is converted to an unsigned 8-bit type to assure proper reception on the server end.

Construction of the data field is dependent on the function being performed. For the output register read function (FC = 3), the data field need only contain two bytes denoting the reference address to be read and a following two bytes denoting the word count (number of registers to be read). Once the data field is constructed, the header, function code, and data field are concatenated into the variable *req* which is the completed request. The request is transformed using the *uint8* command before being written to the *ModBus Slave* program

The TCP/IP object is opened using the *fopen* command to establish a connection with the simulator. A connection timeout is typically due to either an incorrect remote host address or firewall settings (when running the slave program on a remote workstation). The *fwrite* command is used to send the request as binary data to the server. A 1 second pause is implemented before reading the server's response to ensure ample time for creation and receipt of the response. The *fread* command is used to read the server's response stored in the TCP/IP

object's *BytesAvailable* parameter. The *'uint8'* format option is used to ensure the incoming binary data is read properly. This raw response is the actual ModBus frame from the server. The header of the response is removed before handling response's data field.

After successfully receiving the response the connection is closed before parsing the binary data into decimal format. Handling of the response may be dependent on the given function, so separate scripts may need to be implemented. The above processed may be altered in future implementations to improve efficiency handling the ModBus response.

iii. Post Processing

Upon successfully requesting the Receiver Signal Strength information from the BPM for various settings of the signal generator, a post-processing step is implemented to calculate the systematic gain coefficients. For a given setting, the coefficients are calculated by normalizing each of the channel readings by the value of the first channel. The reading of multiple channels is simulated by reading multiple registers of the ModBus Slave program. The systematic gain coefficients are thus relative.

In the MATLAB environment the Receiver Signal Strength readings are stored as a matrix in a global variable *PP*. Each row of this matrix contains channel readings for a given signal generator setting. The structure of reach row adheres to the following format:

$$PP(i) = \{Setting_i, Channel\ A, Channel\ B, Channel\ C, Channel\ D\}$$

Calculation of the systematic gain coefficients is handled by a *for* loop that operates on each row of the matrix. For a given row, a temporary variable takes the value of the second element (Channel A) and divides each channel reading by that value. The result is then stored to a user specified file for future use, a process detailed in the proceeding section.

iv. GUI Development

The final requirement of the RF Cal Tool is to have all the aforementioned functionality accessible by way of a Graphical User Interface (GUI). MATLAB provides the option of creating a GUI programmatically or by using their GUIDE (GUI Design Environment) tool, an interactive GUI construction kit. Though use of GUIDE simplifies the visual layout process, M-file source code generated by the tool possesses a cumbersome and sloppy format. Designing the GUI programmatically takes a bit more time but offers the user more control of the overall design and insight on the functionality of the GUI components.

The typical process for GUI design is followed. After sketching the desired design on paper, the GUI is programmed one component at a time with constant references to the MATLAB GUI documentation. Several user input fields are implemented in the layout design to give the RF Cal Tool varying degrees of freedom (and in turn more practicality & control over the instruments).

After successfully laying out the structure of the GUI, labels are added to each different control object. Spacing/positioning of the items is handled using the *Position* property for each

of the different user interface controls/panels. A simple menu toolbar is implemented with “File” as the only selectable menu option.

The source code for the GUI is divided into 5 sections (using MATLAB’s cell break feature): Pre-initialization Tasks, Construction Tasks, Post-initialization Tasks, Callbacks, and Utility Functions. The pre-initialization tasks section initializes global variables that are used throughout the GUI layout process. The construction tasks area contains code responsible for the physical appearance of the GUI. Post-Initialization Tasks initializes variables that will be used in subsequent callback and subfunction definitions. The Callbacks section contains code for the given callbacks implemented. Lastly the Utility Functions region contains subfunctions.

Method

The following screenshot is of the most current version of the RF Cal Tool designed using MATLAB. A text string is placed at the bottom of the program window to specify the current version of the program.

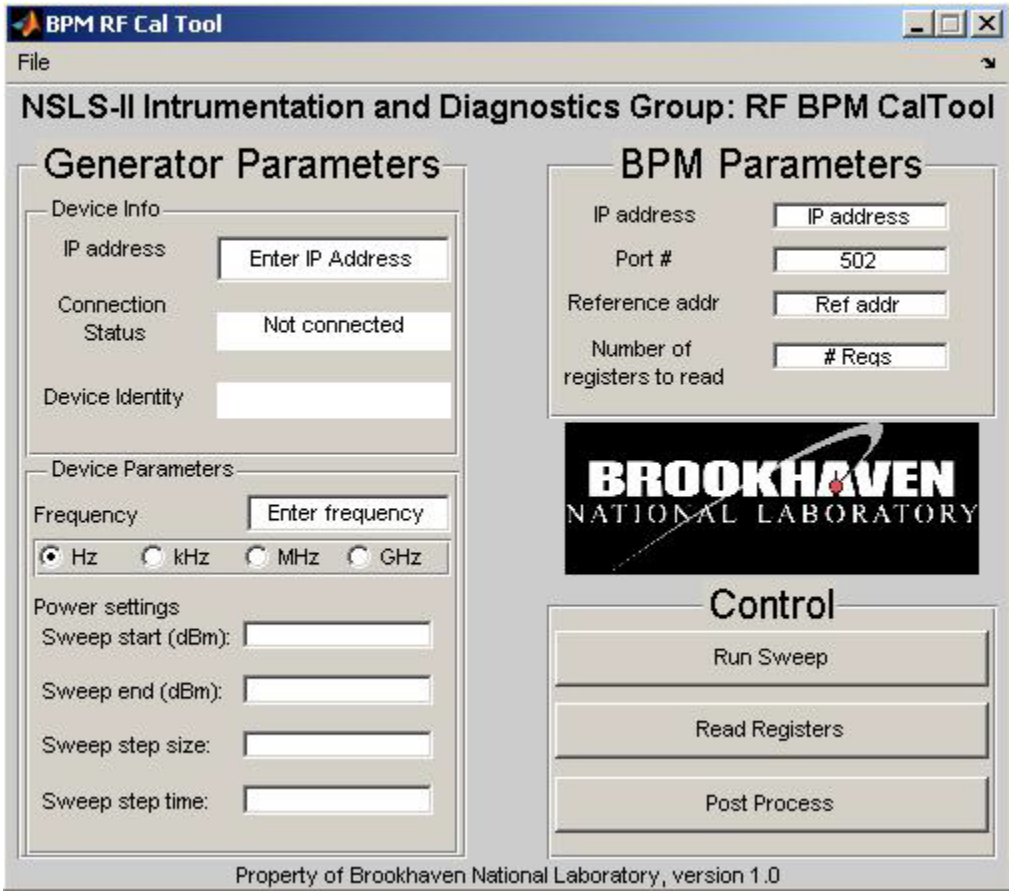

Figure 5. Screenshot of the developed RF Cal Tool Program

As can be seen in the screenshot, the GUI is separated into three regions. The *Generator Parameters* region provides options for control of the signal generator. The user may specify the desired frequency they wish to set the RF output to, identify their level sweep start and end

points along with the step size and time for the sweep. The *Connection Status* and *Device Identity* fields are static text fields altered by the RF Cal Tool and not the user. When a successful connection has been opened with the signal generator for user-denoted IP address, the *Connection Status* parameter will change from the "Not connected" text string with a white background to a "Connected" text string with a green background. Termination of the connection results in a "Disconnected" text string with a red background. The "Not connected" option is set as the program default and should only be displayed upon initial launch of the RF Cal Tool program. The *Device Identity* field displays the device information when connected by use of the SCPI command **idn?* for querying. The radio buttons located in the *Device Parameters* button group are programmed with callbacks. Depending on the radio button selected, an internal string variable is adjusted and assures that the multiplier parameter of the SCPI command for setting the frequency is correct.

The *BPM Parameters* region allows the user to specify the appropriate information for connecting to the BPM unit under test and also denoting which registers to read (by reference point and number). To reiterate, the default listening port for the ModBus-TCP protocol is 502 (although it may be altered depending on the specific application) so the default program value for this field is initialized appropriately.

The *Control* region of the program contains pushbuttons that allow the user to carry out some specified command. The three available options are "Run Sweep," "Read Registers," and "Post Process". Clicking a given pushbutton will execute a script coded into the "Utility Functions" section of the source code. Before describing functionality it should be noted that an additional utility function for reading the inputs is utilized by the "Run Sweep" and "Read Registers" controls.

The function *ReadInputs*, written as a subfunction of the GUI M-file, reads the user inputs from the graphical interface and stores them in variables within the source code for use in other subfunctions. Values that are to be used within SCPI string commands are stored as character strings. The input values for *Sweep start*, *Sweep end* and *Sweep step size* are stored as integers using MATLAB's *str2num* command for use as indexes for loops described in the following subfunctions.

Another subfunction, *MB_RegRead*, is used communicate with the ModBus device. Note that the ModBus-TCP object must be open before calling this function or else an error will occur. Each time it is called, the *MB_RegRead* function sends the pre-constructed request to the ModBus-TCP device and receives the response. A one second pause is implemented using the *pause* function to ensure ample time for reception of data. An error checking feature is also implemented to further ensure proper reception of the data. After receiving the response from the BPM, the received ModBus frame is parsed and the headers of the given request and response are compared. If the two are the same, functionality continues on. If the two are different, an error message is triggered and an error dialog appears on screen to notify the user. The epilogue of this subfunction converts the raw binary data received from the BPM unit into decimal units for viewing and analysis and returns these values.

The *Run Sweep* control is used to collect Receiver Signal Strength samples from the BPM unit under test for a sweep of the signal generator specified by the start/end points and the sweep step size. Clicking *Run Sweep* pushbutton executes the subfunction *ReadInputs* which reads the user inputs using the predefined subfunction. Device objects are then created for the signal generator's VISA-TCP and the BPM unit's ModBus-TCP protocols using the inputted IP addresses. The buffer input size parameter for the ModBus-TCP object is increased to facilitate any needs for a large transfer of data. Before opening either of the interface connections the ModBus frame for the request is constructed using more of the input data read by means of the *ReadInputs* function. Once construction of the frame is completed the communication channels are opened and the signal generator is set to remote control mode. The frequency and sweep settings of the instrument are set according to the user inputs before sending the first request to the ModBus device. To achieve proper sweep functionality, a loop is used. The declaration of the loop follows:

for i = Sweep_start : Sweep_step_size : Sweep_end

Within the body of the loop the *MB_RegRead* function is called and the request/response transaction occurs. After reading the RSSI data for the initial setting, the current value of the signal generator's level setting is adjusted by the sweep step size, the request is resent, and new RSSI data for the new setting are received. This loop continues until the sweep end value is reached. The *PP* matrix stores successive RSSI readings by declaring it as *PP = [PP; MB_RegRead]*(note that *PP* is initialized as an empty array) within the body of the loop. Upon exiting the loop the signal generator is reset, switched back to local mode, and both channel connections are closed

The *Read Registers* control functions similarly to the *Run Sweep* function, except it only collects one sample of RSSI data. The signal generator is set to the level specified by *Sweep start* and the ModBus response is retrieved for this one setting before closing the interface connections and parsing the received data.

The *Post Process* pushbutton runs the script (subfunction) for calculating the systematic gain coefficients from *PP* and saves the results to a user specified output file. Upon execution, the user is prompted with a question dialog warning that collected data will be deleted after saving to the output file. This is done to reset the *PP* matrix in preparation of a new collection of data. If the user selects "No" when prompted to save, the *PP* variable is maintained at its current state. When the "Yes" option is selected for saving, the *PP* matrix is copied to a local variable *output* to ensure that no data is lost should an error occur during execution. The function proceeds to operate on *output* and normalize the channel readings, row by row, by the reading for the first channel. Once completed, a save dialog box (identical to the typical system file save prompt) prompts the user for a location and filename save the output file as. The default file type for the output file is set .dat (data file), but the user may alter this simply by including the file type extension when naming the output. The filename and save location are stored into the MATLAB environment. The *fopen* function is used to create a new file in the location specified using the write-mode option ('*w*') to denote we will be writing to this created file. A textual header (format provided by Brookhaven) is first written to the file. Following this a for loop is used to write the calculated gain coefficients are written row by row to the output file, making use of a line break between each write for legibility. Once the final line is written the loop exits

and the file is closed. The *PP* matrix is reinitialized as an empty matrix to collect new data and the function exits.

The final steps of the GUI design entailed adding features for ease of use. This is implemented by the use of tool tip strings. Whenever the user positions the mouse over a given input field or interactive user input control (radio buttons) a message appears above the mouse cursor providing some information or insight regarding what should be entered. Additionally a simple menu is added to the design. The only parent element in the menu is the "File" label with two submenus: "File → Help" and "File →Close." Selecting the help menu displays a message box containing brief information about the purpose and creation of the tool. Selecting the close menu option prompts the user with a question dialog. Using this dialog the user can choose to terminate the program or continue execution.

Results

Initial testing of the RF Cal Tool proved successful. It should be noted that the signal generator and BPM functionality were tested at different times on an individual basis due to the signal generator being located at Brookhaven National Laboratory. When tested in January, the script for VISA-TCP commutation successfully programmed the Rhode & Schwartz signal generator and desired functionality was displayed. The script utilized in the current version of the GUI mimics the structure of the tested script, so it is assumed that correct functionality will be achieved. Further testing in June will shed light on this matter. Since the BPM units are still being manufactured the *ModBus Slave* program is used to simulate and test ModBus-TCP functionality. Testing shows that RF Cal Tool exhibits proper communication utilizing the protocol and the ModBus data received is identical to the data in the ModBus request after parsing. This occurs both when the *ModBus Slave* program is executed on the local computer (in which case the local host IP is used) and on a remote machine (utilizing the remote host IP is used). Additionally, the output file is created and formatted as specified by the source code and contains the correct data. Furthermore the RF Cal Tool has been tested on several different MATLAB platforms (versions 7.4, 7.7, 7.8, and 7.9) and proper functionality is achieved. It is not certain, however, if the tool may work with older (versions predating 7) software packages.

An analysis of timing/data speeds is not included since the given application is exhibits no sensitivity to time. If future testing proves the opposite, the proper analysis will be included.

Conclusion

The purpose of the RF Beam Position Monitor is to detect transverse beam position with sub-micron resolution and stability for normal operation conditions. The most crucial parameter of the BPM to accurately measure the position is for the gains of the four channels to be matched. It has been shown that the preliminary design of the RF Calibration Tool provides proper functionality for calibrating these devices. Remote control of the signal generator has been demonstrated through testing and communication using the ModBus-TCP protocol has proven successful. Using the simulated results, systematic gain coefficients are calculated correctly and stored in a file to be downloaded to an FPGA for further verification. Although the source code for the developed graphical user interface may need to be augmented once testing with the actual hardware has been realized, the foundation for any future work has been established by this project.

Future Work

Possible revisions and additions to the RF Cal Tool may include:

- Ease of use
    - Incorporating additional error checking features and feedback to mitigate confusion should a problem arise.
    - Adding additional details to the help menu on how to troubleshoot some of the most common errors when using the program.
- More ModBus Functionality
    - Implementing multiple ModBus functions
- More control of Signal Generator
    - Allowing control to more of the Signal Generator's many functions could increase the longevity and use of the Cal Tool
- Improved GUI design
- Optimize code
    - The current source code focuses on functionality. It is certain that different areas of the code may be optimized for performance.

Acknowledgements

I'd like to extend special thanks to Bruce Land for believing in me and offering tons of support when things difficult or dead ends were reached. Also many thanks for the MATLAB tricks; to Clifford Pollock, the MEng Board, and the ECE Department for allowing this project to happen; to Kurt Vetter and Om Singh for giving me this opportunity and the ongoing education in the field of Particle Accelerator design; to Terrence Buck for his support and representation, working behind the scenes to make this collaboration with Brookhaven possible; to the GEM National Consortium for my placement with Brookhaven Lab, without whom none of this would be possible; to Brookhaven National Laboratory for providing me with an opportunity to apply skills I've learned in the classroom to a practical design problem; and to the MathWorks for providing lots of documentation online and also being available via their Customer Support hotline to clarify a few misconceptions about the Instrument Control Toolbox.

References

Vetter, Kurt. *NSLS-II RF BPM Systematic Gain Calibration – "RF BPM Cal Tool"*. Brookhaven National Laboratory (March 1, 2010). PDF file.

ModBus Application Protocol Specification V1.1b, The ModBus Organization (December 28, 2006). <www.modbus-IDA.org>

ModBus Messaging on TCP/IP Implementation Guide. The ModBus Organization (October 24, 2006). <www.modbus-IDA.org >

Swales, Andy. *Open ModBus/TCP Specification*. Schneider Electric Company (March 1999). PDF file

Rohde & Schwarz GmbH & Co. KG. (2008). Publication Manual for the Rhode & Schwartz SMA100A Signal Generator. Munich.

The MathWorks. *MATLAB: Creating Graphical User Interfaces*. PDF File. < http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/buildgui.pdf

The MathWorks. *MATLAB: Instrument Control Toolbox 2 User's Guide*. PDF File. < http://www.mathworks.com/access/helpdesk/help/pdf_doc/instrument/instrument.pdf>

The MathWorks. *MATLAB 7: Programming Fundamentals*. PDF File. < http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/matlab_prog.pdf>

The MathWorks. *MATLAB 7: Programming Tips*. PDF File. <http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/programming_tips.pdf>

Appendix

Available upon request.