

Automating the PTSM system

Jason R. Heimann

August 22, 2004

1 Overview

Even a simple experiment, such as measuring the length of a piece of string, is not without several sources of error. There is the error introduced by the measurement device, and that caused by the device reading the measurement (i.e. the person performing the experiment). As an experiment becomes increasingly complicated, so does the opportunity for introducing error. Systematic errors can be greatly reduced in any experiment by strictly regulating the manner in which the experiment is performed. In the development of the PTSM system, we use PC-based automation to “regulate” our experiments.

1.1 What is automation?

Automation is used throughout industry to perform many tasks that would otherwise require human intervention. The task could be as simple as opening a valve when a tank is full, or it can be as complicated as producing finished silicon die from raw materials. An automated process will include subsystems which perform measurement, analysis, communication and control tasks. These subsystems can exist in hardware or software; the automation of a particular process depends on the well-defined interactions of these subsystems. Using automation ensures that one’s instructions are carried out to the letter, every time the process is run.

1.2 Automation and PTSM

Two common measurements that characterize the PTSM system are automated: calibration and noise measurement. Automation is achieved by harnessing the GPIB capabilities of existing test equipment, and by incorporating additional hardware to control the flow of signals. In the process of calibration, we automate the injection of signals and the collection of data thereafter to achieve precise, low-noise measurements. Noise measurement experiments are automated to achieve reliable results within well-defined time intervals. Separately, data analysis is automated (using ROOT scripts) to ensure proper calculations are performed each time an experiment is run. A block diagram of the automation hardware is provided in figure 1, below.

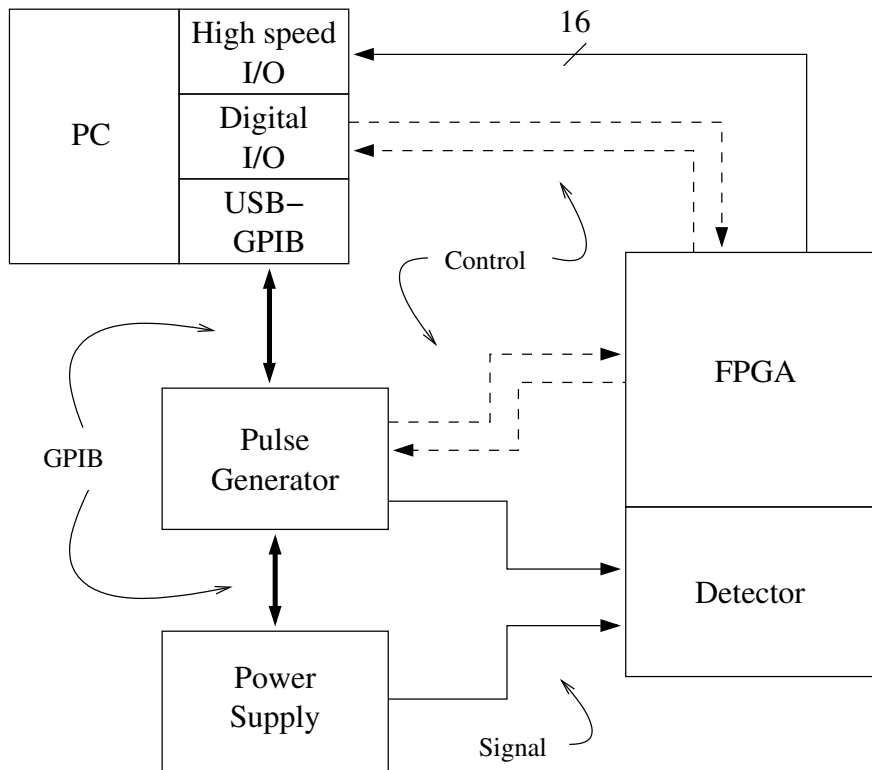


Figure 1: PTSM automation block diagram

2 GPIB

The GPIB (General Purpose Interface Bus) was developed to allow computers and programmable instruments to interconnect. Some years later, the IEEE488 standard was released, which further defined the physical interface and communication protocol used to support GPIB. With this technology, we are able to communicate with our test instruments in software, giving us “virtual instruments” with which we can make programmed measurements under various configurations.

Essential to the operation of a GPIB-enabled system is the **controller**. The controller’s role is analogous to that of the telephone operators of the late 1800’s: to route communications between a **talker** and one or more **listeners** (devices/instruments with the ability to send or receive data). A USB to GPIB interface was purchased from National Instruments (NI) which allows our PC to act as a controller, and as a talker / listener. While the controller functionality is mostly transparent to our implementation, the talker and listener capabilities are harnessed to achieve part of our automated solution.

As a talker, our PC can send commands to various instruments to affect

their operations. We can configure power supply voltages, signal generator amplitudes and bias current limits, to name a few possibilities. We can also send triggers to an instrument to initiate a predefined response such as sending a pulse or making a measurement. To pull information from our instrumentation, we can send a query: a command that directs the instrument to respond with specific information. Query responses can include the make and model of an instrument, values of various configuration parameters, or the value of a previous measurement.

Boldface words are terms defined in the IEEE488 standard

2.1 Installation

With the NI USB-GPIB-B we've purchased, installation is as simple as plugging the interface into the PC's USB port. After the NI-DAQ and NI-GPIB software (included) are installed, the interface is ready to operate. We use a "daisy chain" configuration to connect our test equipment together: each instrument is connected to its neighbor and the final connection is made to the interface. A unique GPIB address (generally from 0 to 15) has to be assigned to each instrument to ensure proper operation; the software relies on these addresses to identify each instrument.

3 Hardware

3.1 Instrumentation

Specific test equipment is required to operate and "debug" the PTSM system. We harness the functionality of these instruments by using their GPIB interface(s). The GPIB capabilities of each instrument are detailed in their respective user manuals, and often these capabilities mirror those that are available from the front-panel controls. Older GPIB instruments simply implement a unique command string to represent an action on each button, knob or switch on the front panel. Newer instruments use more intuitive commands that allow the configuration of several parameters in one concise string.

At the present time, three instruments have been automated: two power supplies and a pulse generator. Threshold and bias voltages are automatically set and applied, and bias current is monitored. The pulse generator is configured to a known state through its GPIB interface, however, we declined to use the same interface to trigger pulses. Existing trigger delays in the GPIB hardware and software imposed a ceiling on our trigger rate at about 10 Hz. The pulser is instead triggered through control hardware in the FPGA - see section 3.2 for more information.

3.2 Control

Separate PC hardware purchased from NI gives us a number of digital lines which can be used to input or output TTL-level signals. A PC DAQ card

(the NI 6703) has been implemented to provide several control and feedback lines to the FPGA. Four lines are used to select the calibration bus(es) used during a calibration experiment. Two additional lines are used to implement handshaking with the pulse handler hardware: a section of the FPGA used to control pulse injection and readout. All of these lines are connected through hand-made TTL to LVDS converters, which are in turn directly connected to the FPGA.

These digital lines are controlled and monitored in the software (see section 4). Their behavior depends on the “cal_jumper” setting at the FPGA. If the calibration jumper is installed on the board, the PTSM system is in *source measurement mode*. In this mode, the “start” line is used to enable data collection from within the software. The “done” line is used to indicate the presence of data in the system: initially low, done transitions to high (from logic 0 to 1) when the system’s buffers are empty. Monitoring the “done” line ensures the completeness of the data set from a given experiment.

If the calibration jumper is not present on the board, the PTSM system is in *calibration mode*. When the start line (see Figure 2) transitions to high, the FPGA sends a trigger pulse to the “external trigger” input of the pulse generator. A delay occurs after each trigger while data is read out, ensuring the system’s buffers are emptied before sending the next pulse. After a preselected number of pulses are sent and all data is read, the done line goes high and the system is inhibited until the start line is cycled low and high again.

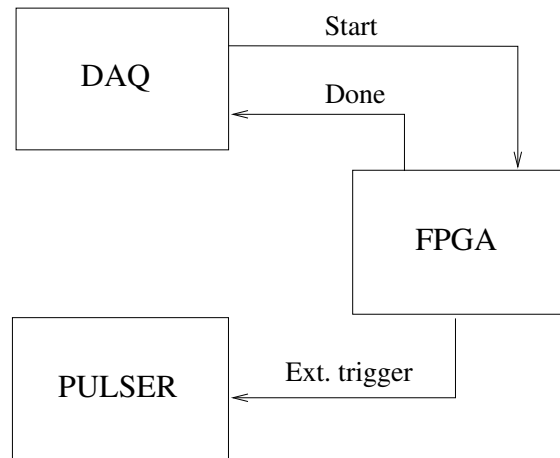


Figure 2: PTSM handshaking diagram

3.3 Communication

Another PC card (the NI 6534) is implemented to facilitate high speed digital read-out. The PTSM system outputs event data through 16 digital lines (output as LVDS, converted to TTL at an external board). Several additional lines are

used to provide synchronous handshaking capabilities: REQuest, ACKnowledge and a clock input. With these additional lines, the PC card operates in *burst mode*: data is transferred in “bursts” defined by the state of the ACK and REQ lines. The FPGA outputs one word of data for each clock cycle during a burst; the typical data clock rate is 10 MHz. See the 6534 User Manual (available from NI) for a detailed timing diagram for this mode.

4 Software

Prior to automation, the PTSM system included read out software that was developed to facilitate the use of the NI 6534. The software would simply channel data from the PC card to a file on disk. Implementing automation in this system required compiling additional libraries into the existing software. These libraries are provided by NI along with sample code demonstrating their use. With the new libraries we can communicate with our GPIB-enabled instruments and control the NI 6703.

The additional control provided by our automation hardware isn’t very useful until our it’s interactions are well defined. Having done several calibration and measurement experiments “by hand,” we know what steps to take to achieve each result. We also know the approximate timing required to complete each step. The final step in achieving automation is developing the software that controls each step in the process with the proper timing. Each part of the system must be considered individually, as some instruments take longer than others to respond to commands (from milliseconds to tens of seconds). Also some steps are more critical than others: the bias current must be carefully monitored, while the threshold current is insignificant and can be ignored.

As the software was developed for PTSM, there were many iterations of development and experiment. Using Windows-based PCs, we can not always achieve repeatable behavior in our software. Delays between subsequent commands in the software varied from fractions to tens of milliseconds, and before we understood this, we had some difficulty achieving our intended results. We achieved our best results using FPGA hardware to control our timing-critical tasks (such as triggering and handshaking), and PC software to control everything else. It must be noted that FPGA hardware is much more difficult to develop than most PC software, hence the sparse use of hardware in our automation scheme.

5 Further automation

Software routines developed by Brian Keeney use ROOT to analyze and present the results of calibration experiments. The raw output from a calibration experiment measures several megabytes in size. Analyzing this output channel by channel is tedious and prone to error. Even worse, errors in analysis are not quickly apparent. Brian’s routines make quick work of analysis; by “previewing”

the data to be analyzed, his software knows the parameters of the experiment performed (i.e. threshold voltages used, number of calibration pulses). The routines then perform a standardized analysis on each channel for each parameter that is varied in the experiment, and aggregates the results for concise presentation.

Hardware and scintillating detectors assembled by John Wray are currently being used to measure efficiencies of SSDs. John's hardware incorporates a (manual) micrometer controlled XY table to position his scintillators with respect to the SSD under test. With PC (or FPGA) controlled stepping motors, one can automate the positioning of the scintillators. Performing a channel-by-channel efficiency scan by hand could take quite some time! (64 channels * 10 minutes = about 6 hours) Even larger amounts of data could be collected with an automated system, requiring minimal effort.