Open Hardware Test Automation Platform

Design Document

Team 33
Client: Dr. Geiger
Advisor: Dr. Geiger
Team Members/Roles:

- Antonio Montoya - Team Lead
- Christian Hurst - Software Team Lead & Webmaster
- Ben Wiggins - Communications Head
- Braden Rosengren - Hardware Lead
- Chris Little - Key Concept Holder

Team Email: may1733@iastate.edu
Team Website: http://may1733.sd.ece.iastate.edu/

Revised: 3 Nov 2016

Contents

1 Introduction

1.1 Project statement

We have set out to create an open source platform for hardware test automation and remote test execution and data collection. The platform will allow users access remotely through any web browser, and will facilitate them to upload and execute custom test scripts as well as view and download results. The platform will be built on a Raspberry Pi. The Raspberry Pi will host a web server, and users will interface with the system through a browser based UI/UX. The Raspberry Pi will be able to deploy, at the user's command, user defined and uploaded test scripts, store and report back test results, and allow the user to download the raw test data. The Raspberry Pi will be able to control and query results from common laboratory equipment like power supplies, digital multimeters, signal generators, and oscilloscopes. The platform will also provide the ability to program/configure digital or mixed signal devices under test via SPI, I^2C, or manually controllable digital I/O pins, all with level shifted outputs at the device's operating voltage.

1.2 Purpose

This platform will allow individuals seeking to perform device characterization, or to automate other laboratory tests, to easily design and implement tests that can be launched remotely, as well as facilitating remote data collection. The original concept was conceived with users of wafer probing stations in mind, however the functionality of allowing remote test launch and result reporting has a wide array of usage cases, for example, allowing remote device demonstration for instructive or educational purposes. In addition to allowing for remote access, it provides a cost-effective platform that can be used to implement identical laboratory setups at multiple locations that support uniform test code, which could be of great use to teams working in parallel to do similar characterization or verification on similar or related devices. The platform also offers an excellent opportunity to allow access to test automation to a wider audience, as an open source solution would provide an inexpensive alternative to costly software and hardware that is currently industry standard (e.g. NI Signalexpress).

1.3 Goals

With this project we hope to demonstrate the viability of the concept outlined in the Project Statement. The scope of this project is too broad to expect to be able to deliver a polished and robust platform in only two semesters of work. It is however, completely reasonable to target a demonstration of the utility of the platform in an arbitrary usage case, which at the moment is taking measurements from a device on a wafer prober. Developing a proof of concept will therefore be our primary goal, and expanding/enhancing functionality will be a secondary goal. We will produce supporting documentation outlining potential usage cases and documenting the

additional work required to adapt the platform to be viable for a broad range of usage cases. Pursuant to our primary goal of producing a proof of concept, some elements of the original project concept may be earmarked as stretch goals.

- Primary Goals
    - Due to the broad scope of the project, our primary goal is to demonstrate the viability of the concept outlined in the Project Statement
    - Demonstrate the utility of the platform with an arbitrary usage case.
        - Currently usage case is taking measurements from a device on a wafer prober
        - Second usage case is as replacement test launching platform for academic laboratory exercises that utilize signalexpress currently
    - Produce documentation of additional usage cases and the steps needed to implement them
    -
- Secondary goals
    - Expand upon the list of proposed usage cases and implementation plans
        - Create proposal for new test platform and equipment configuration for EE 201 and/or EE 230
        - Create interface API for hypothetical equipment set for 201/230

2 Deliverables

At the completion of this project, we plan to deliver the following items:

- Breakout board for Raspberry Pi
- Linux-based software package
    - Web Server
    - Test Scripting API
- *Getting Started* manual and *User Guide*
- Demonstration of successful implementation in at least one proposed usage case as proof of concept
- Outline/Documentation of process for adapting the platform to other potential usage cases
- Documentation of Potential usage cases

3 Design

3.1 System Specifications

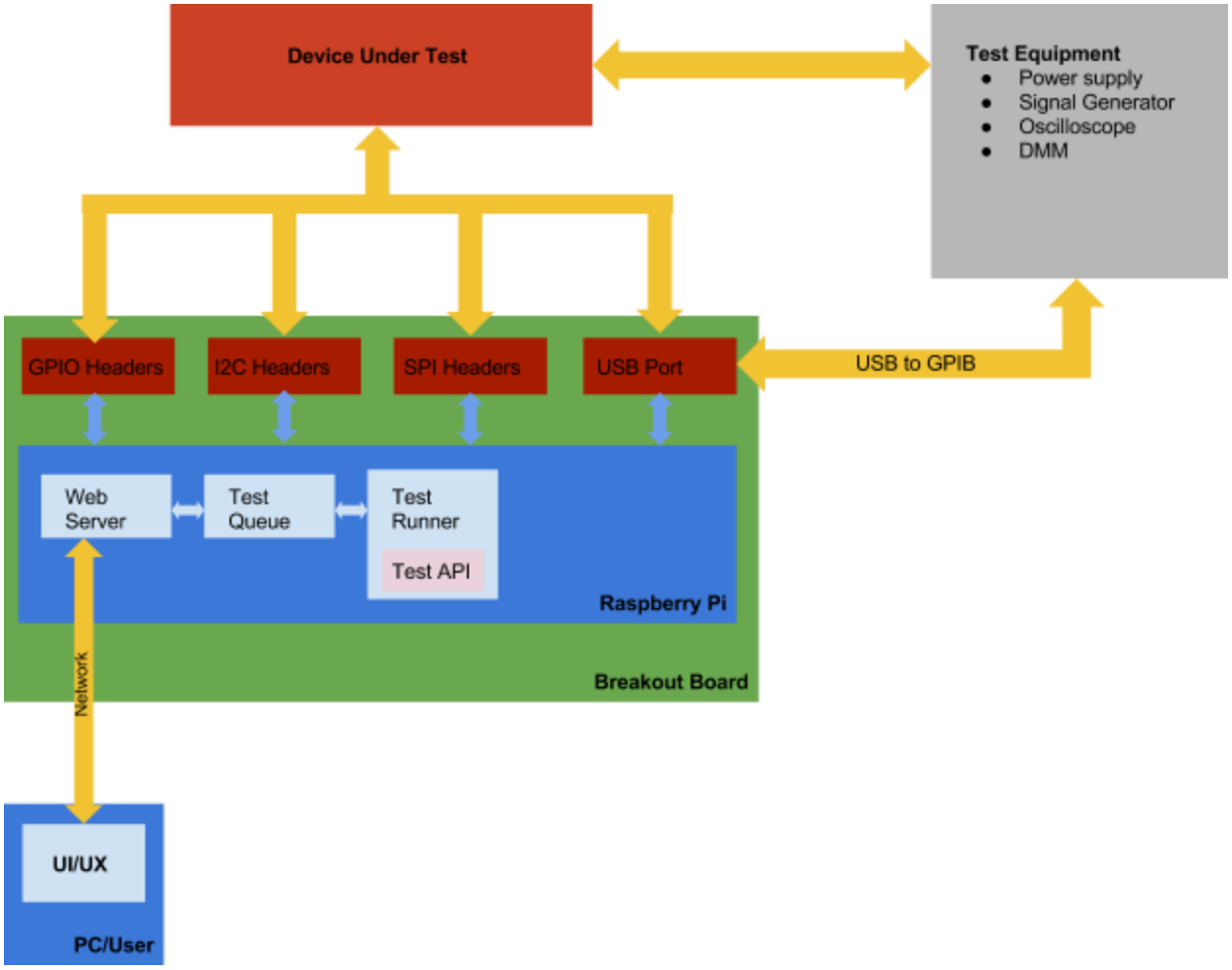

Figure 1

3.1.1 Non-Functional Specifications

3.1.1.1 Software Requirements

- Web interface is well organized and self explanatory
- Create instruction manual for building software package

3.1.1.2 Hardware Requirements

- Connectors are spaced so that they leave room to easily connect peripherals

- Demonstration of some potential usage cases and proof of concept

3.1.2 Functional Specifications

3.1.2.1 Software Requirements:

- Web server and browser-based UI/UX
  - Allow user to define and run tests remotely
  - Documented API allows user to define new tests
- Raspberry Pi must be able to interface with the test equipment
  - API for GPIB
  - API for the SPI, I2C and ADC
- System should have the ability to start all required processes on power up without human intervention
- Ability to interface with GPIB devices over multiple types of connections (LAN, USB, IEEE-488)

3.1.2.2 Hardware Requirements:

- PCB breakout board for computer to chip communication
  - SPI and I^2C protocols supported
  - PCB should implement level shifting of digital control signals to acceptable range for use on wafer
  - Provide outputs via SMA connectors to allow for use with industry standard probe card leads
- Ability to interface with test equipment over GPIB
  - GPIB over LAN
  - GPIB over USB
- Create a variable level logic shifter for SPI, I2C, and GPIO

3.2 Proposed Design/Method

The project will be split into hardware and software components to be handled by their respective subteams. The software subteam will handle the implementation of the web user interface using the Apache web server on the Raspberry Pi. The browser-based UI/UX will allow the users to upload custom test scripts, schedule and execute tests, and easily collect and view test results, with the option to download the results directly to their personal computer using Apache's file IO system.

The test runner software will be programed in Python and will store tests as Python modules. Additionally, schedules and results can be stored in JSON files and these files can be accessed by the test runner process. Additionally, the server and the test runner can communicate through use of a Unix Socket. In addition, this test runner needs to be able to time the tests in order to ensure proper scheduling of the tests. Currently, we are planning on implementing this in the test runner,

however we also explored implementing it with the help of the native cron utility. The test runner software needs to be implemented to allow interfacing with the testing equipment through GPIB interface which will be interpreted by the Raspberry Pi as a virtual serial port. Once basic functionality has been confirmed, we will look into implementation with interfacing with GPIB over ethernet and GPIB over USB to USB in order to support newer models of lab equipment. The test runner will also need to interface with I^2C and SPI busses in order to program any on chip digital test circuitry.

3.3 Design Analysis

The design of our project was driven primarily by the functionality outlined in the specifications above. Since our project is based around being able to control various test equipment we needed to support GPIB, the bus over which most lab equipment can be controlled. Additionally, since I2C and SPI are commonly used on more complex ICs, having support for these interfaces was also necessary.

Additionally, we needed a remotely accessible user interface that was capable of rendering content dynamically depending on the tests uploaded and scheduled by the user. For this we decided to implement our interface using Python CGI scripts and the Apache web server.

Our choice of language was driven primarily by our desire for an easily extensible system. Python was chosen as the language of our implementation since it is easy for new users learn. Additionally, its straightforward approach to module support makes it ideal for defining tests.

4 Testing & Development

4.1 Interface Specifications

GPIB Communication:

- Communication between Raspberry Pi and test equipment requires a GPIB module in older devices
- Newer devices communicate using GPIB commands over USB or Ethernet

Device Under Test Communication:

- Breakout board will have I2C, SPI and GPIO ports
    - Plan to implement variable level shifting for these ports later on

4.2 Hardware/Software

Apache Server with CGI

- Can serve dynamic content (such as current test queue) over http
- Can communicate with test runner process

    - Request information about and results from tests
    - Upload and schedule tests
- Can send commands over GPIB through testing API

Test Runner Process

- Can execute tests specified by users
- Implements testing API to communicate over the Raspberry Pi's busses and GPIB
- Can store results of tests
- Can receive commands from the user interface (Apache server)

4.3 Process

The original plan for generating the Kernel, filesystem and other programs on the raspberry pi that would run the webserver and the test scripts was to use a tool called Buildroot. However, there were a couple issues that arose when attempting to use this software. In order to use Buildroot to create a Linux system, our team needed root access on a Linux box, which is not given to students. To get around this, we set up virtual Linux machines in order to gain root access. Once this was done, we were successfully able to load a version of Linux on the Raspberry Pi. Unfortunately, we encountered a glitch with that prevented us from logging in to the Pi. After researching and implementing several documented fixes we concluded that it was unprofitable to continue to use Buildroot. We then decided to switch to an Raspberry Pi-oriented Linux distribution called Raspbian. While it has none of the basic http web server framework and other packages that attracted us to build root, there were fewer reported issues with Raspbian usage.

Once the decision to use Raspbian was made, a new framework had to be decided on. A common web server application to use on Raspberry Pi is called Apache. This sets up a basic framework on the Raspberry Pi that allows the users to create a website using an html file stored on the Raspberry Pi. In order to test Apache's usability in the scope of our project. We created a simplistic IO system that could be used to write information into a file stored on the Raspberry Pi and then displayed the information on the website. Because the test scripts will be stored in files, this test results suggest that this process can be expanded upon to allow users to create more complicated tests.

During this time, we also researched the programing languages in order to decide the one that best fit our needs. The two main contenders were Perl and Python. In the end, we concluded that while Pearl has amazing features for parsing text that make it more efficient than Python, Python has a fairly readable language that most of the users of this project could easily pick up and debug if necessary.

We then proceeded to research how to implement communication between the Raspberry Pi and test equipment over GPIB. After some research, we discovered it simply opened a virtual serial port on the Raspberry Pi that corresponded to the GPIB connection and could be controlled through simple commands through the command line. In order to confirm this, we attached the

Raspberry Pi via GPIB to a signal generator in the lab and made a script to change the magnitude, frequency and waveform of the signal generator's output.

5 Results

To verify that the GPIB communication worked with the lab equipment, we plugged in the GPIB module into the lab equipment and attached it to one of the Raspberry Pi's USB ports. We then observed that we could change the output waveform, frequency and amplitude using the terminal. In the future we plan to implement this as a script that passes user input to the bus to control the test equipment.

To verify that our Apache server was set up to support CGI files we used a simple Python script as index.cgi. When we navigated to the server from an external computer we observed the expected output which confirmed that we were properly set up to serve dynamic content using Python scripts.

6 Conclusions

We have set out to create an open source platform for hardware test automation and remote test execution and data collection. The platform will allow users access remotely through any web browser, and will allow them to upload and execute custom test scripts as well as view and download results. The platform will provide the ability to program/configure digital or mixed signal devices via SPI, I^2C, or through manually controllable digital I/O pins, all with level shifted outputs at the device's operating voltage. It will also be capable of controlling a wide array of test equipment through several common implementations of the GPIB bus.

This platform will allow individuals seeking to perform device characterization, implement testing automation, and facilitate remote data collection. The original concept was conceived with users of wafer probing stations in mind, however the functionality of allowing remote test launching and result reporting has a wide array of usage cases such as allowing remote device demonstration for instructive or educational purposes. In addition to allowing for remote access, it provides a cost-effective platform that can be used to implement identical laboratory setups at multiple locations that support uniform test code, which could be of great use to teams working in parallel to do similar characterization or verification on similar or related devices.

7 References

[1] Linux GPIO Sysfs Interface: https://www.kernel.org/doc/Documentation/gpio/sysfs.txt
[2] Raspbian Download and Installation: https://www.raspberrypi.org/downloads/raspbian/
[3] Buildroot Documentation: https://buildroot.org/downloads/manual/manual.html
[4] Dynamic Content with CGI: https://httpd.apache.org/docs/2.4/howto/cgi.html
[5] Agilent 90xxx GPIB Programming manual:
https://d3fdwrtpsinh7j.cloudfront.net/Docs/document/90000_Q-Series_prog_ref.pdf
[6] Bi-Directional Level Logic Converter Overview:
https://learn.sparkfun.com/tutorials/bi-directional-logic-level-converter-hookup-guide
[7] Bi-directional level shifter for I²C-bus and other systems. (APPLICATION NOTES):
http://cdn.sparkfun.com/tutorialimages/BD-LogicLevelConverter/an97055.pdf
[8] Raspberry Pi Pinout Interactive: https://pinout.xyz/
[9] BSS138 IC for Level logic operations:
http://www.mouser.com/ProductDetail/Fairchild-Semiconductor/BSS138/?qs=sGAEpiMZZMvlOED0T0kTWl5MKyjMqq%2fg
[10] BSS138 Datasheet: http://www.mouser.com/ds/2/149/BSS138-1006682.pdf

8 Appendices

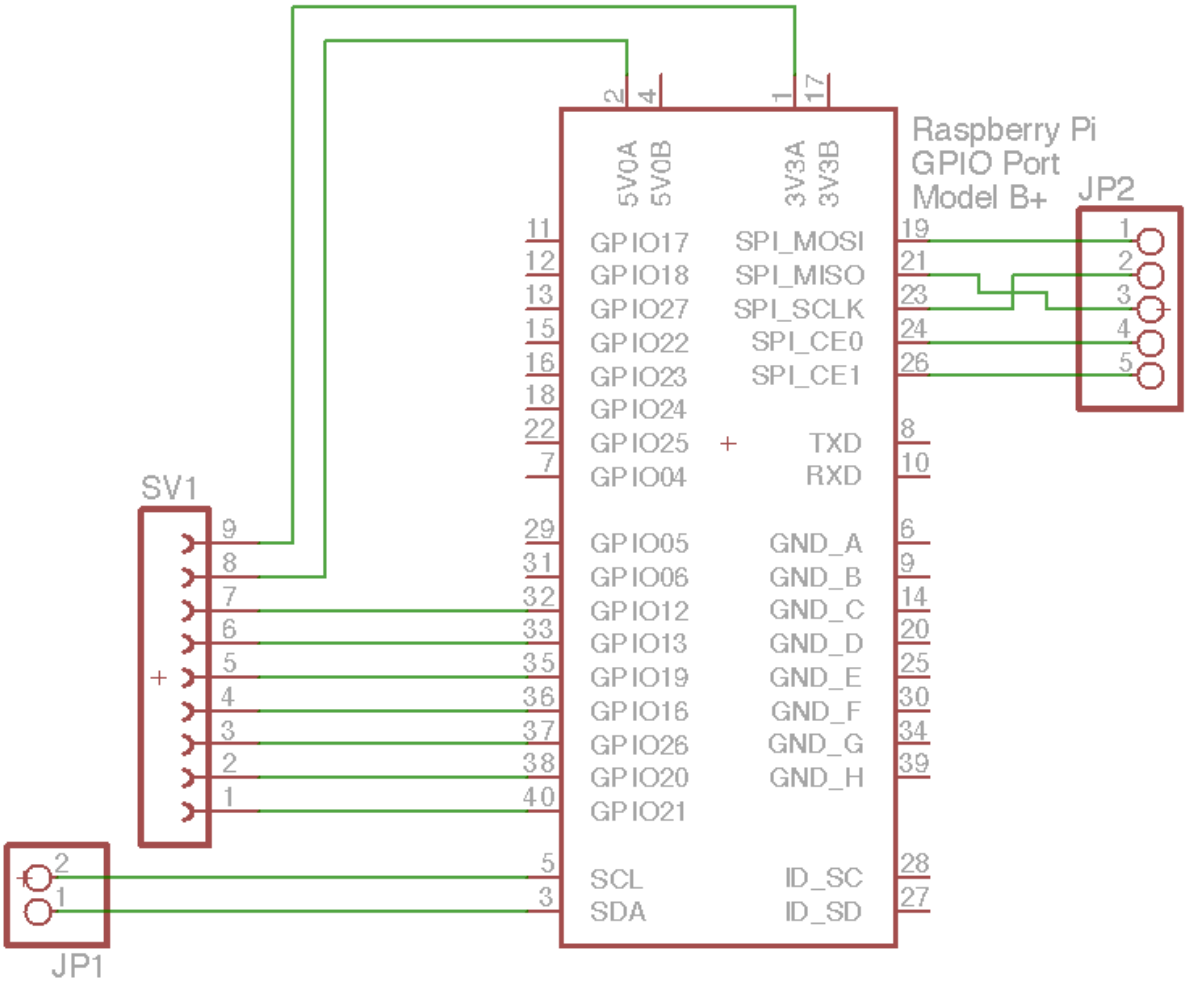

Figure 2. PCB v1.0 schematic

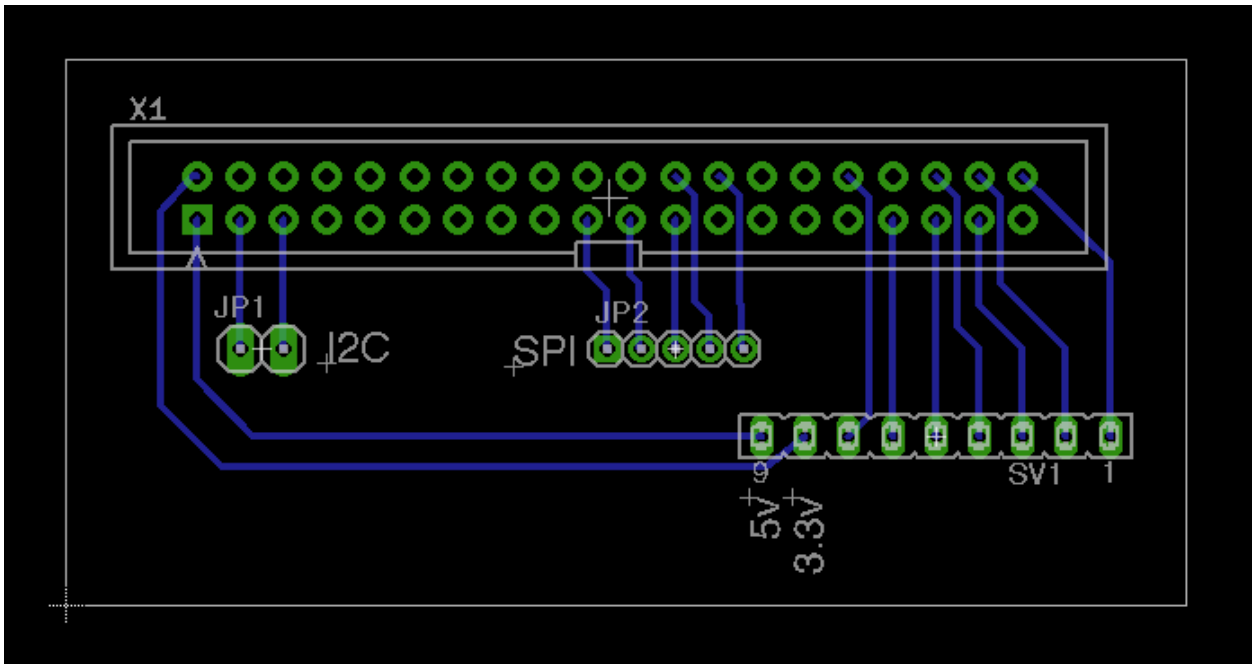

Figure 3. PCB v1.0 layout