PHAS0097: MSci Project Progress Report

Design of a Range Calorimeter for Proton Beam Therapy Quality Assurance

Fern Pannell

Supervisors: Dr Simon Jolly & Prof. Ruben Saakyan

Department of Physics and Astronomy

University College London

January 18, 2021

Word Count: 1969

1 Hardware: Detector Development

The UCL Proton Beam Therapy (PBT) group is upgrading the current prototype range calorimeter from a CMOS sensor-based set-up to a photodiode-based system, where the photodiodes are coupled directly to plastic scintillator sheets. An FPGA configures an analogue-to-digital converter connected to the photodiodes, processes the data and sends this to a PC.

1.1 FPGA

The initial FPGA used to communicate with the analogue-to-digital converter was a Xilinx Zynq-7000 on a Digilent Zybo Z7-10 development board. A serial terminal emulator, CoolTerm, saves data that has been sent from the FPGA via UART. However, with the intention of sending data from the FPGA at a rate of 1 kHz, the relatively slow data transference of the UART protocol proved to be a significant limiting factor. Therefore, the UART protocol has been upgraded to a complete USB 2.0 interface, which is expected to provide a substantial increase in transfer speeds (up to 480x faster).

The USB interface on the Zybo Z7-10 board is tied to the ARM processor system, therefore increasing the complexity of the set-up. A new development board, the Digilent Nexys Video was purchased due to its comparatively simplistic design. Adjustments were made to the development board in the form of soldering and cabling, to prevent cable sharing for the data transfer and debugging of the FPGA. The soldering of additional pins into the port allowed for the separation of these channels. An FTDI chip on the Nexys Video board facilitates the transference of data via the USB interface using the libFTDI C++ library. This library allows for the detection of FTDI chips, the configuration of their ports, as well as both sending and receiving data [1]. The USB interface with the Nexys Video is currently functional, sending the correct data at an optimised rate.

1.2 Raspberry Pi

It is planned that a Raspberry Pi will receive data from the FPGA instead of a PC to provide a compact solution. Taking this into account, a Raspberry Pi4 has been set up with the appropriate software, including the Apache web server, PHP extensions and XRDP for remote access.

Before the USB interface was considered, CoolTerm was installed on the Raspberry Pi and a test with the FPGA transferring data to the Raspberry Pi via UART was performed at UCL. It is known that the Linux and Raspberry Pi versions of CoolTerm are not well supported, and results

from the test proved this; CoolTerm failed to keep up with the configuration data. However, transferring to the USB interface eliminates the need for CoolTerm altogether.

It was discovered that the CPU of the Raspberry Pi reaches very high temperatures when under heavy workload. To prevent incidences of thermal throttling, resulting in diminished graphics and frame rates, various cooling solutions were explored [2]. The use of the ARGON ONE Raspberry Pi case and heat sink, combined with a heat sink compound, has so far proven to be efficient in dissipating thermal energy. The performance of this setup will need to re-checked when the Raspberry Pi is receiving data from the FPGA and hosting the webpage, as this is likely to be heavy-duty work.

2 Software: Graphical User Interface Development

The range calorimeter aims to reduce the intricacy of quality assurance in PBT and a Graphical User Interface (GUI) displaying real-time measurements contributes to the work towards this objective.

2.1 Data Visualisation: JavaScript Charting Libraries

Before deciding that the GUI should be web-based, LabView, a graphical programming tool provided by National Instruments, was considered for parsing data. It was quickly determined that in order to ensure the GUI was more future proof, the photodiode graphing should be implemented as a webpage. This would also for allow user control on a variety of devices, including handheld. Consequently, JavaScript was selected as the programming language for the GUI and various charting libraries to visualise the detector output were researched.

The three JavaScript libraries explored were CanvasJS, C3/D3 and Chart.js. Static bar graphs were constructed in each library by directly inputting arbitrary data into the code. CanvasJS had substantial online documentation relative to the other two libraries, but disfavoured aesthetics. Chart.js appeared to have the most appropriate commercial license package; well-aligned with the intended use of the GUI [3]. Therefore, it was decided that the initial development would be in CanvasJS for a more supported development process, before switching over to Chart.js to display the final product as desired.

2.2 Parsing CSV Data

The FPGA generates a one-line CSV file of 16 values, representing the light output of each photodiode. Therefore, the initial task was to parse a CSV file of the same format, log the results to the browser's JavaScript console and plot these values on a bar chart. This task was broken down into the following steps:

1. Parse a one-line CSV file and log the values to the JavaScript console.

2. Reproduce the CanvasJS graph, replacing the arbitrary data with the parsed CSV values.

3. Edit the function to repeatedly parse the same one-line CSV and log the values to the JavaScript console. This will result in logging the same array of values at a fixed rate (chosen as once per second, 1 Hz, for easier debugging).

4. Update the CanvasJS function to graph the latest line of data, rendering at 1 Hz.

5. Repeatedly parse a one-line CSV that is being constantly overwritten by an external Java program and log the values to the console. This will result in logging different arrays at 1 Hz.

6. Render the CanvasJS graph for each new set of data, resulting in a dynamically updating bar chart that reflects the latest line of data in the console.

Steps 1 and 2 were completed relatively quickly, however, a number of issues occurred beyond this point. Initially, when repeating the parsing function as specified in step 3, the values would append onto the previous array as opposed to replacing it, subsequently failing to plot on the 16-bin bar chart. This was resolved by resetting the length of the array to 0 after each iteration. Steps 5 and 6 involved running a Java file that wrote 16 dummy data values to a CSV, displaying the constantly changing data in the JavaScript console and plotting these values. After identifying issues with cache clearing in certain browsers, this was largely successful.

When increasing the data-logging rate from 1 Hz to 10 Hz, performance issues became apparent. The webpage issued warnings of heavy memory use within seconds of running the JavaScript, the graph would not render correctly and the laptop viewing the webpage began to exhaust its CPU fans. With the end goal of utilising a Raspberry Pi to display the photodiode data on a GUI rendering at 50Hz, it became apparent that the current manual parsing was not only an issue at this stage, but an impractical design for the future. Thermal throttling had already been identified as a potential issue when using the Raspberry Pi and was deemed inevitable if a high-quality laptop struggled to regulate its temperature. Therefore, it was necessary to investigate alternative means of parsing CSV data rapidly in JavaScript, without impacting the performance of the web page or the hardware displaying it.

Papa Parse, an open-source, powerful CSV parser, was implemented into the JavaScript. The appendix shows the JavaScript used when manually parsing the CSV values and when utilising the Papa Parse library. Papa Parse removed the issue of significant memory usage within the webpage and the graphs rendered successfully at 10 Hz. To improve the general aesthetics of the GUI and adopt the library with the appropriate commercial license, the necessary adjustments to the JavaScript were made to utilise Chart.js going forward. This was relatively transparent and mainly involved adapting to the syntactic differences and distinct configuration objects.

The current prototype GUI is shown in Fig. 1. A CSV file on the UCL High Energy Physics

(HEP) server is repeatedly overwritten at a specified rate. The webpage, hosted by the HEP server, displays the data in real-time, updating 10 times per second.

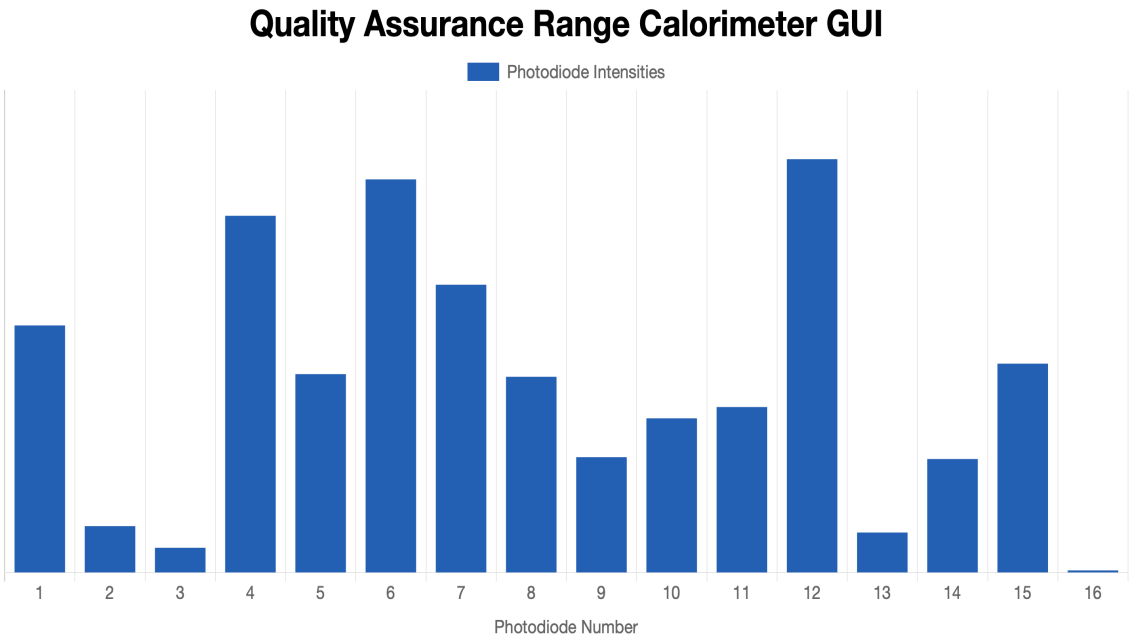

Figure 1: The current GUI prototype utilising the Chart.js library. Arbitrary data is displayed at an update rate of 10 Hz.

2.3 Testing the GUI with Live Photodiode Data (4th January 2021)

After working with arbitrary data written to the CSV file by the Java program, the GUI was tested with real photodiode data, processed by the FPGA. The webpage was loaded on multiple devices of varying widths to ensure the GUI rendered appropriately across a range of devices, as seen in Fig. 2.

The photodiodes were exposed to varying levels of brightness by the removal and replacing of a light shield. The delay between the adjustment of the light levels and the results being displayed on the webpage was minimal at less than 1 second. Devices running old browsers, such as the 2009 Mac Pro in Fig. 2. could still render the graph and see the dynamic updates, although removed the colours, which are defined with modern JavaScript syntax.

Figure 2: Testing the GUI with live photodiode data, across a variety of device widths.

3 Next Steps

A draft of the final report should be in development from now until the end of term 2. This should be reviewed and discussed prior to finalising the report. The project presentations will be held in term 3, and preparation for this should be made alongside the report.

3.1 Raspberry Pi & the USB Interface

The Raspberry Pi will either run the Apache web server or send data to the UCL HEP web server via SCP (secure copy). A hosted webpage will act as the front-end display of the detector, allowing the live photodiode data to be viewed in a web browser. A C++ library, libFTDI, allows for communication with the FTDI chip on the Nexys Video board, facilitating the USB connection. The installation of libFTDI has been successfully tried and tested on both Windows and MacOS for data transfer to a PC. To enable data transfer to a Raspberry Pi this must be installed and tested in Linux, creating the need for the following tests:

- **Test 1**
  - Compile a basic C++ test script on the Raspberry Pi.
  - Add the standard #include statements into the test script and recompile.
  - Successful compilation indicates that the preprocessor will not have issues with these standard headers in the future.
- **Test 2**
  - Install libFTDI on the Raspberry Pi.
  - Compile the FTDI C++ code on the Raspberry Pi.
  - Expected result: Pass the first *if* condition. Fail at the second *if* condition as it is not connected to the device.
- **Test 3**
  - Connect the Nexys Video to the Raspberry Pi via the USB cable.
  - Ensure the C++ compiler is pointed towards the headers and libraries in the g++ line given in the @usage.
  - Run the same C++ code and both conditions should be passed as the code should open the connection with the Nexys Video.

3.2 Expanding the Capabilities of the GUI

The current version of the GUI meets the initial criterion set out at the beginning of the project: the live data from the photodiodes can be visualised anywhere on a browser in real-time. There is room for improvement in the following areas, which should be explored during the remainder of this project:

- Curve fitting, in the form of reconstructed Bragg curves and approximations that account for scintillator quenching effects, form part of the proton range data analysis [4]. This fitted data should be parsed by the JavaScript and displayed over the live photodiode values. If the exact procedure for the curve fitting is not decided before the end of the project, dummy data should be used to show a proof of concept.

- Performance enhancement of the display. Currently updating at 10Hz, the JavaScript should be stripped of any unnecessary code and methods to increase render performance should be investigated. The GUI should be tested with the photodiodes for increasing Hz, ideally culminating in the successful operation of the display at 50Hz.

- Time permitting, additional features, namely the saving of experimental runs and the analysis of individual photodiodes via the GUI, should be investigated.

References

[1] (Jun. 25, 2016). "Fast USB connection on the Nexys Video using FT2232H - Denis Steckelmacher," [Online]. Available: https://steckdenis.be/post-2016-06-25-fast-usb-connection-on-the-nexys-video-using-ft2232h.html#the-ft2232h-chip (visited on 01/17/2021).

[2] (Nov. 28, 2019). "Thermal testing Raspberry Pi 4," Raspberry Pi, [Online]. Available: https://www.raspberrypi.org/blog/thermal-testing-raspberry-pi-4/ (visited on 01/17/2021).

[3] (May 1, 2019). "Comparison with Other Libraries | Chart.js," [Online]. Available: https://chartjs.org/docs/master/notes/comparison (visited on 01/10/2021).

[4] L. Kelleter, R. Radogna, L. Volz, D. Attree, A. Basharina-Freshville, J. Seco, R. Saakyan, and S. Jolly, "A scintillator-based range telescope for particle therapy," *Phys. Med. Biol.*, vol. 65, no. 16, p. 165 001, Aug. 2020, ISSN: 0031-9155. DOI: 10.1088/1361-6560/ab9415.

Appendix

a)

```
105  //function to fetch the one line csv values of Photodiode data
106  //and log to console
107  async function getData() {
108    //const response = await fetch('photodiodeValues.csv'); //java written
109    const response = await fetch('photodiodeValues.csv');
110    const data = await response.text();
111    //recongise each data point is separated by a comma
112    const allElements = data.split(',');
113    //for each element in the row
114    allElements.forEach(element => {
115      //append the element to an array called intensities,
116      //and convert element from string to integer
117      intensities.push(parseInt(element));
118    });
119    console.log(intensities);
120  }
```

b)

```
9   //PARSES LOCAL CSV DATA WITH PAPAPARSE
10  function parseData(createGraph) {
11    Papa.parse("photodiodeValues.csv", {
12      //required for webserver data
13      download: true,
14      complete: function(results) {
15        //1st line of file is PD data, isolate this
16        //console.log(results.data[0]);
17        //will be carried into createGraph()
18        createGraph(results.data);
19      }
20    });
21  }
```

Figure 3: JavaScript code for manually parsing CSV values (a) and employing the Papa Parse library to parse CSV values (b).