MAY 2018

UNIVERSITY OF SURREY

RASPBERRY PI

RADIATION EXPERIMENT

GEORGIOS TOUMBAS
ELECTRICAL & ELECTRONIC ENGINEERING
gt00099@surrery.ac.uk

Abstract: Raspberry Pi Compute Module 3 is a powerful small device that can be used as payload computer in the AAReST project, as it meets the processing power needed for the project. The purpose of this experiment is to make sure that it can physically withstand the space environment when it comes to cosmic ray radiation. By the end of the report a conclusion has been made stating that a similar device can functions normally under 130krad of beta ray Total Dose Irradiation. The device had shown no signs of failure but instances of mild degrading performance during the irradiation period.

1 TABLE OF CONTENTS

1.1 TABLE OF FIGURES

1.2 LIST OF TABLES

2 INTRODUCTION

2.1 PURPOSE

Space exploration has attracted the interest of an increasing number of scientists in the last to decades. Space missions and research studies are not only feasible by a couple of organisations around the globe which was the case in the 20th century. Many universities have built departments focused on space research, two of those, the University of Surrey and CalTech space centres came with the idea of AAReST. The idea is to launch CubeSat-like nanosatellites each carrying an electrically actuated adaptive mirror capable of undocking and redocking to a central nanosatellite dock that will house fixed mirrors and a boom deployed focal plane assembly. The payload computer of the CubeSat will be a Raspberry Pi Compute Module 3. It is well known that ionizing radiation can cause parametric degradation and ultimately functional failures in electronic devices. The generation of hole traps and interface states contribute to the overall MOS transistor Vt shifts. Therefore, it is necessary to test whether the payload computer can withstand or how long it can survive space radiation caused by cosmic rays.

2.2 AIMS - OBJECTIVES

The aim of this project is to:

1. Determine the working conditions of the device under cosmic ray's radiation by correlating its performance with beta ray irradiation
2. Explore its limitations and possible ways to extend the life cycle of the device within the space environment.

A fine line between failure and success when it comes to the outcome of the project is set by the time management. Thorough background knowledge of the device characteristics as well as semiconductor behaviour under irradiation is crucial when planning the experiment. There is a need for a detailed and well thought test plan which will determine the effectiveness of the experiment itself, leading to fluent execution and accurate results. A well-presented analysis of data is obligatory such as the outcome of the experiment is clear and easily passed to the reader. Finally, the experiment outcome will correlated to a possible use of the device in space and future notes for a similar improved experiment will be given.

3 LITERATURE REVIEW

3.1 RASPBERRY PI COMPUTE MODULE 3

Figure 1:RPi Front View

Figure 2: RPi Back View

The RPi CM3 is a product of Raspberry Pi Foundation which is a charity that works to put the power of digital making into the hands of people all over the world. The fact that it is a widely available product running on an open source software gives the room for improvement from work done through the community. Leading to a world where space missions will be open to public without the need of expensive equipment.

The board has a Broadcom quad-core processor (BCM2837), as seen on Figure 1, which is a Cortex A53 technology running up to 1.2 GHz using the ARMv7 instruction set. The Broadcom chip accommodates a VideoCore IV processor which is the graphics processor unit of the device. Integrated are a LPDDR2 RAM memory chip of 1GB (left) and a 4GB eMMC Flash memory chip (right) seen on Figure 2. The RAM chip which stands for Random Access Memory is used by the processor to store information of running interfaces. On the other hand, the embedded Multi-Media Controller is the storage device of the RPi where information is stored when offline. As specified the device can run in the temperature range of -25ºC to +80ºC.

There are 200 pins on the 67.6 mm x 31 mm board (Appendix 1) which are in a DDR2 SODIMM configuration. This means that it can easily be mounted on an external circuit using the standard DDR2 SODIMM socket. Many known interfaces are supported by the Raspberry Pi as listed on Table 1.

Types of Interfaces:

NUMBER	INTERFACES
48	General purpose Input-Output (GPIO)
2	I2C
2	SPI
2	UART
2	SD/DIO
1	HDMI 1.3a
1	USB 2 HOST/OTG
1	DPI
1	NAND Interface (SMI)
1	4-lane CSI camera interface
1	2-lane CSI camera interface

Table 1: List of RPi Interfaces

The Compute Module has 6 different supplies which all need to be powered even if an interface is not used. Three different voltage banks are used for GPIO. Banks 0 and 1 are power GPIO0 to GPIO27 and GPIO28 to GPIO45 respectively. Bank 2 is used to connect to the eMMC device and for an on-board I2C bus. All the GPIO interfaces support at least 2 alternative functions within the System on Chip as listed on Appendix 2.

3.2 RADIATION

3.2.1 Strontium-90 (Sr-90)

Sr-90 is a radioactive form of strontium, which is formed in nuclear reactors or during the explosion of nuclear weapons, when uranium and plutonium undergo fission. It has a half-life of 28.90 years and emits Beta particles with mean energy of 195.8 keV, and a maximum of 524 keV. The activity of SR-90 used in this experiment is 2.6 GBq.

3.2.2 Cobalt-60 (Co-60)

Cobalt was discovered in 1735 by Georg Brandt. It is produced by bombarding cobalt-59 or nickel-60 with neutrons. Usually a product of nuclear reactors or weapons. Co-60 emits two high energy gamma rays of 1.33 MeV and 1.17 MeV respectively. A beta energy is also released on average at 95.8 keV that could reach a peak of 318 keV.

3.2.3 Effects on CMOS

When electronics are struck by radiation energy they could start misbehaving. These upsets could occur randomly called single event effects or could be due to long term ionizing called total ionizing effects.

3.2.3.1 Single Event Effects (SEE)

These are effects caused by a single charged particle as it passes through the semiconductor, in other words, a heavy ion that causes direct ionization. When the linear energy transfer (LET) of the particle is higher than the critical charge then soft (temporary) or hard (permanent) errors may occur.

Single event upsets and transients (SEU, SET) fall under the soft error category. SEU occurs when an ionized particle causes a change of state, whereas SET is when a voltage spike is caused. These events can be resolved by resetting or rewriting the device.

Hard errors include single event latch-up (SEL), burnout (SEB) and gate rupture (SEGR). SEL is a condition where high current state causes loss of device functionality. Permanent damage may or may not be caused, but requires power strobing of the device to resume normal operations. SEB is when a device is destructed due to a high current state. Finally, SEGR occurs when an ionized particle creates a conductive path through the gate oxide of a MOS device.

3.2.3.2 Total Ionizing Dose (TID)

It is the damage caused by protons and electrons due to cumulative long-term ionizing. It is usually measured in units of krads and can be reduced by shielding low energy protons and electrons. This long-term exposure can cause threshold shifts, leakage current, timing changes and functional failures.

3.3 SIMILAR EXPERIMENTS AND SPACE MISSIONS USING RASPBERRY PI

3.3.1 Arduino/Raspberry Pi: Hobbyist Hardware and Radiation Total Dose Degradation

This experiment was performed on both an Arduino UNO and Raspberry Pi Model B, however only the RPi part will be in focus. The board was functioning using a BCM 2835 processor at 700 MHz, with a Dual core VideoCore IV Multimedia GPU and 26 GPIOs running a Linux Raspbian OS. The devices under test (DUT) were mounted within a Pb-Al shielding box and tested 2 at a time. The dose rate used was 1 krad (Si)/minute of gamma irradiation. After one irradiation step the DUTs were connected to a HDMI monitor, mouse and keyboard. The OS was installed and run off a non-irradiated micro-SD. Verification was performed by booting and running benchamarking tests to check bus speed, memory speed and drive speed of the device. The results had shown that all devices were able to run normally through 40 krad (Si). From 50-60 krad (Si) all devices lost the ability to detect the attached USB and keyboard preventing login. However, DUTs could fully boot to log-in screen up through 150 krad (Si).

The experiment has not provided enough data about the operation of a raspberry pi under irradiation. Firstly the device had been irradiated without power therefore the results do not represent the operation of a device in space conditions. Secondly no evidence have been provided to suggest the failure of a particular interface other than the USB. The outcome states that the DUIs were able to power up to 150krad but no functionality is proven. Finally the timeline of the experiment suggests that after every irradiation step, irradiation was terminated to check the functionality of the device, which leads to a conclusion that interrupting between irradiation steps could make the device subject to annealing effects.

Table 2 : Hobbyst Experiment Device Parameters

Input Voltage	5V
Input Current	750-1200 mA
Storage	4 GB Micro-SD
SDRAM	512 MB
Clock speed	400MHz

3.3.2 Radiation testing of rad-hard CMOS devices

Vorago Technologies is a leading manufacturer of radiation hardened devices for use in space. Microcontrollers are processed in order to immunize the device from latch-up, such devices are certified to work up to 300krad. An error detection and correction subsystem is included to detect single event upsets in memory.

To certify device use in radiation environments an experiment had been performed at a facility of the University of Massachusetts by VT. Similar to the experiment in 3.3.1 cobalt 60 had been used and four devices had been irradiated at the same time, at a dose rate of 85 rad/s. DUI's had been running an automated test prior, during and after irradiation. Electrical readings had been logged for before and after stages. However irradiation had been halted at equal intervals to ensure proper operation and record standby readings.

After test completion the devices had been packed in dry ice and shipped to VT's facilities for another ATE test and then were baked for 24 hours at 125^{o}C to allow device characteristics to return to pre-irradiation characteristics.

This experiment has been performed in accordance to MIL-STD-883H Method 1019.8 Condition A. This suggest the outcome of the experiment to be valid. According to MIL standards the use of dry ice to avoid annealing effects during transport is an acceptable procedure, however this practise can easily add errors any results captured. Moreover, while irradiating the devices, radiation exposure had to be halted in order to check proper operation of the DUI's. The report suggests that interruptions had been kept to a minimum to avoid annealing, however no number of interruptions had been given and thus annealing effects could have occurred.

3.3.3 AstroPi

The use of a Raspberry Pi in space has been around for quite some time. The Raspberry Pi Foundation built two space hardened units which were sent to the International Space Station. The project started in 2015 and is live today. The devices where upgraded to a B+ version in the meantime for increased functionality. The Raspberry Pi board is mounted inside a space grace aluminium case. Each device is space approved by NASA.

3.4 SUMMARY

By the end of this chapter a strong knowledge foundation had been built to proceed with the experiment. After beta and gamma ray radiation had been compared a conclusion had been reached stating that gamma ray is more penetrative, this suggests that single event upsets are more likely to occur. However it terms of total ionizing dosage the device had been expected to experience similar effects.

Studying similar experiments contributed to a better planning this experiment, referring to the experiment in 3.3.1 the DUI's had been powered off during irradiation steps, as a result the outcome of the experiment does not correspond to a realistic usage of such a device is space. The device should be irradiated both at booting stage and operation stage, in order to determine its behaviour in a similar environment and that had been taken into consideration when during the planning stage. Failure of the USB interface at 50krad is assumed to be caused by an upset due to the high energy of gamma ray. It is expected for the USB interface to withstand higher TID under Sr-90 irradiation.

Vorago Technologies experiment had been performed on rad-hard devices which had not been the case in this experiment. However, approach to irradiation followed had been vital to planning this experiment even though the exact procedure was different to avoid interruptions during the irradiation process.

AstroPi project has been using Raspberry PIs in space for just over 3 years now. The fact though the device is used inside the International Space centre where radiation dose is low, doesn't represent the device's capabilities under irradiation. Details on any process in hardening the devices for space use or radiation effects have not been published.

4 PLANNING - PREPARING

The Experiment will be performed according to the MIL-STD 883H 1019.8 standards as far as possible to ensure correct procedures and reliable results. A flow chart is available in Appendix 3 that shows the steps followed when designing the test plan. The Ionizing radiation test procedure suggests the use of Cobalt-60 (gamma ray) but given no similar previous beta ray test on RPi CM3 and the lack of gamma ray facility on University Site, Strontium-90 will be used as an irradiating source. A flow chart showing the experiment timeline is illustrated in

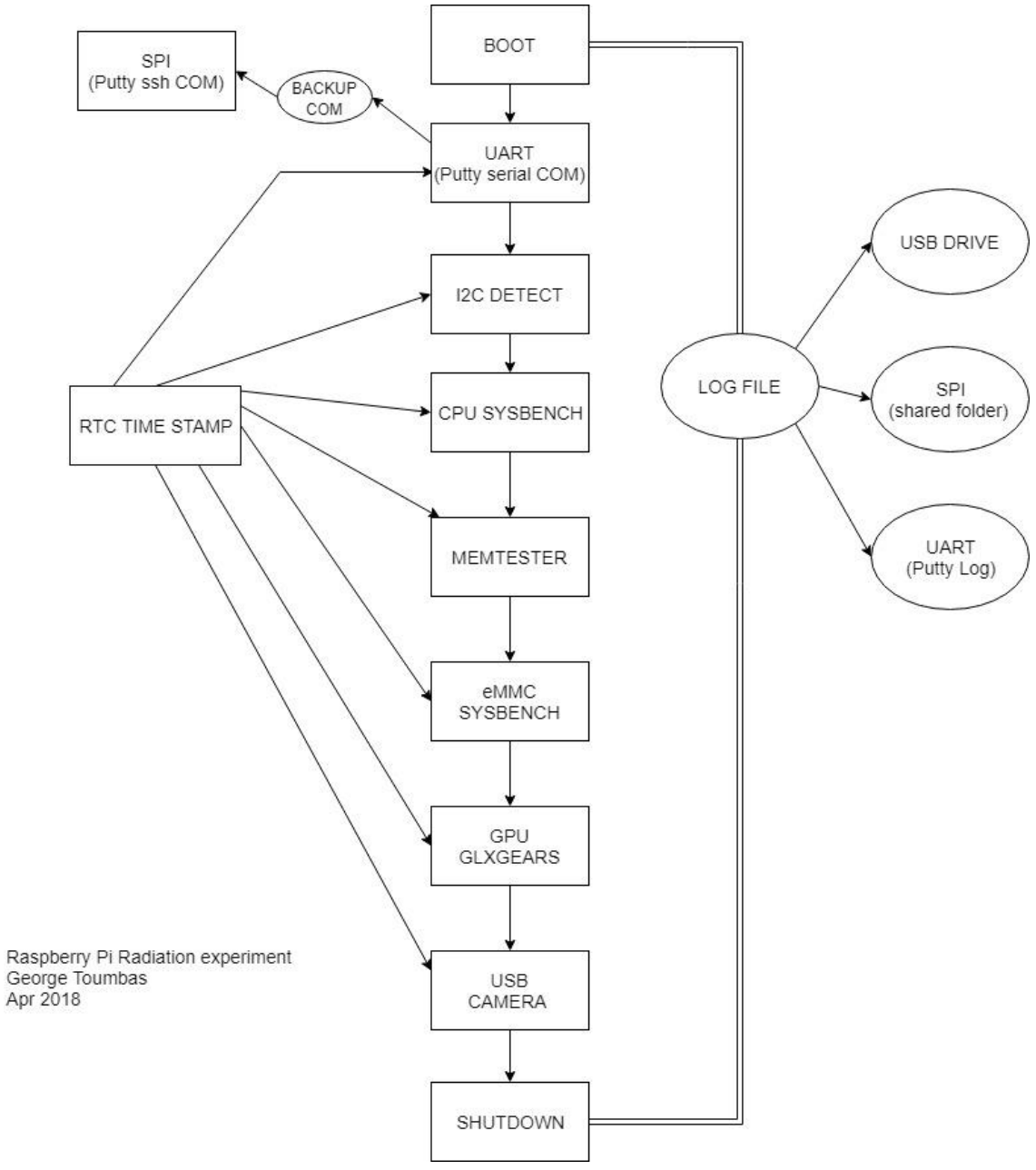


Figure 3: Experiment flow Chart

Figure 10 includes a pin diagram of the complete setup. The USB devices located inside the chamber were connected directly to the I/O board using the installed USB OTG plug through a USB hub which was powered by the DUI. An Adafruit PCF8523 Real Time Clock was connected on the I2C interface which was powered by the 3v3 rail of the DUI. The data and clock pins, SDA and SCL, were mounted to GPIO 2 and 3 respectively. A CR1220 coin battery was installed on the RTC board and was used to provide power to the clock and avoid the time being lost when the DUI was powered off.

For the SPI interface testing, the ENC28J60 Ethernet Development Board was used. Power was again provided by the 3v3 rail. GPIO 8, 9, 10, 11 and 25 where connected to the Ethernet board to ensure proper functionality. Pin 8 provided the chip select option, pins 9 and 10 were used as MISO and MOSI (Master In Slave Out, Master Out Slave In) and pin 11 was the clock. Pin 25 provided an interrupt. An Ethernet cable established the connection from the Ethernet board to the laptop's Ethernet port.

For the UART to USB serial communication a FTDI Chip USB to UART Cable was used. Connection to the DUI was established by using GPIO 14 as TX, GPIO 15 as RX and a common ground. The USB cable connected to the laptop. A simple Veroboard design was implemented to provide pins of RX, TX and GND outside of the chamber as well as rails for 5v0, 3v3 and 1v8 for voltage readings to be taken.

Digital multimeters were connected to the laptop via an individually powered USB Hub. Four RS232 converters were used to provide compatibility from the multimeter's RS232 port to the laptop where readings would be logged.

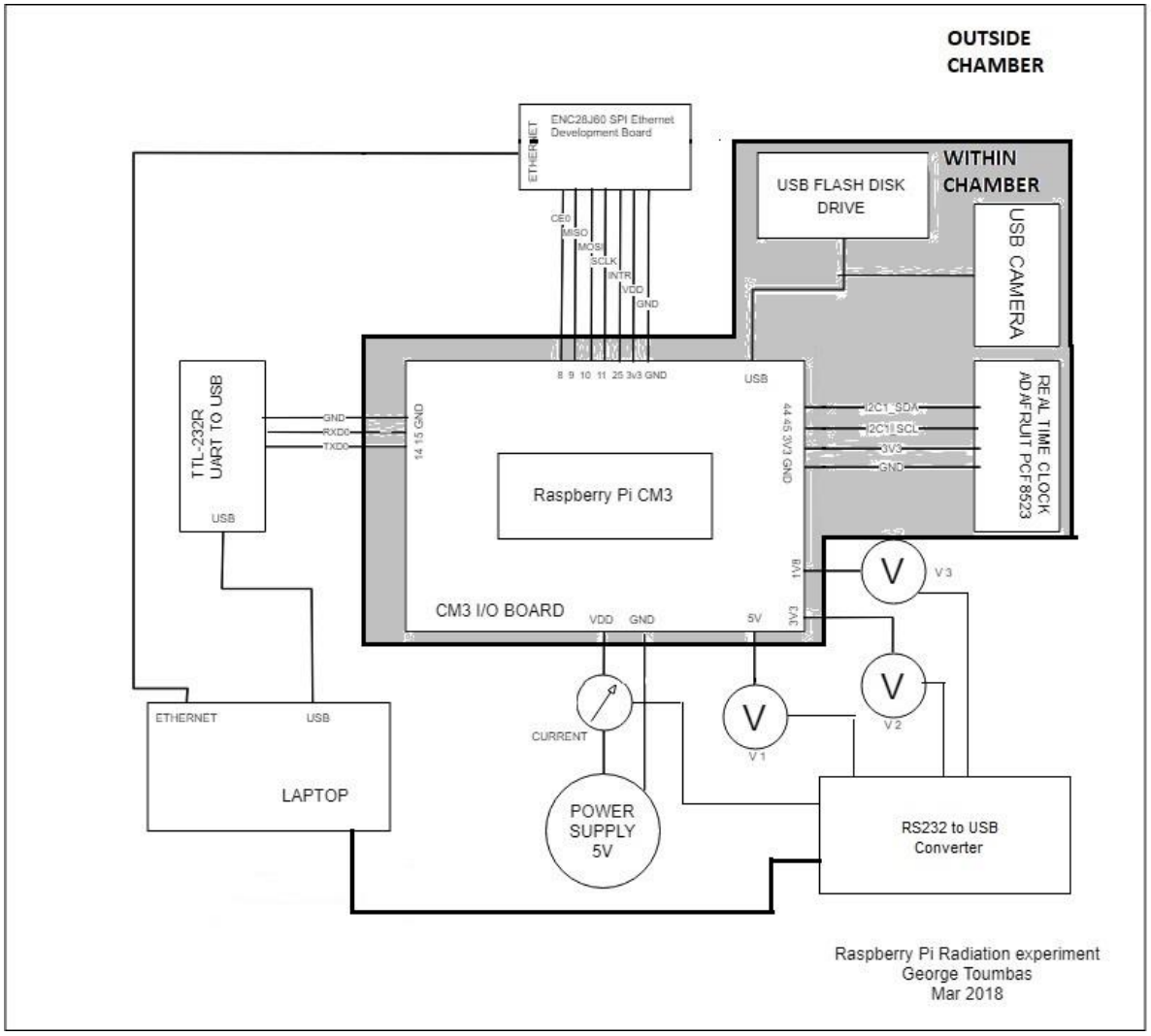


Figure 10: Wiring

4.1 SOFTWARE SETUP

4.1.1 Software Installation

A series of steps and installations had to be performed in order to prepare the device for this experiment. A detailed description will follow under this section.

First of all an operating system had to be installed. An official unmodified version was preferred, therefore the only option was Raspbian Stretch lite 4.14 April 2018. The Desktop version was not compatible since more than the available 4GB of storage are required. The installation was performed on a Linux machine using rpiboot software to power the device's flash memory and write the OS image.

The drivers for all peripherals had to be enabled. This was executed by editing the boot configuration file and setting dtoverlay parameters for the Ethernet board the VideoCoreIV and

real time clock driver. An error given by the real time clock at random times after each reboot had been fixed by only enabling the RTC interface in the boot configuration file but loading the pcf8523 driver when the booting process had already finished. Therefore an explicit command to load the PCF drivers was written inside the rc.local file which executes towards the end of the booting process rather than right after the kernel loads which had been the case with configuration file. While editing the configuration file the I2C, UART and SPI interfaces were enabled.

Once the Ethernet board was fully working a connection to a router had been established via an Ethernet cable to proceed with downloading the additional software. At this stage i2c-tools were installed which included the i2cdetect software. Next was to install sysbench which was the software to run for both the CPU and eMMC tests. Installing from the APT library gave room for errors since the 0.5 version was available. The validate option of the fileio test which was the function testing the eMMC and validating reads and writes was crashing the program. The problem was resolved by downloading the 1.0 version from github and compiling.

A similar approach had to be followed when downloading the memory tool, memtester, from the APT library. The output file of this test was eleven pages long and was the result of each individual function within the test passing characters (“-/|\-“) to the output stream for each memory write performed. As a result an enormous volume of data would have gathered and would be challenging to analyse. The program had to be downloaded from github and the source file had been altered to remove the unwanted set of characters from the test result.

The most challenging part of the device preparation was getting the VideoCore IV to render images. Initially the software to be used had to be installed. Glxgears had been downloaded through mesa utilities from the APT library. At the time the only way to launch the application was through an SSH connection and using an x-server application to transfer the rendered images over to the laptop connected. However it was soon realised that the particular approach was using software acceleration rather that hardware acceleration. One solution would be to follow the Virtual GL path, which is an open source toolkit that allows remote display software to launch applications with 3D hardware acceleration. However it had been ruled out since the kernel would have to be modified and only remote interfaces that support xserver communication would work, therefore a test over UART would be impossible. The solution to this problem came with the installation of the PIXEL GUI. In this way the glxgears would run on the GUI interface using the GUI as its Display and therefore rendering would be executed

by the on-board GPU. A total of four applications were installed from the APT libraries including a display server, the desktop environment, a window manager and a login manager. Once the installation was completed glxgears could run in the background on the GUI interface and the frames per second were past to the terminal.

Finally capturing images from the USB camera had been handled by fswebcam which is an application available on APT libraries. Jpeg pallets had already been installed as part of the mesa drivers. Prompting for user credentials prior to login had been disabled and the device automatically logged in to tty0 session as user pi.

4.1.2 Test Script

Bash scripts had been drafted in order to run all the necessary instructions automatically. Description of the scripts is included in this section. All scripts used are available in the Appendices. Individual instructions had been performed by different scripts which were all called by a main test script.

Firstly a script had been created to read the time from the real time clock, called time.sh Appendix 5. In order to account for the i2c bus being busy a delay of 1 second was held before each access to the clock. The script consists of a loop that checks whether a communication to the clock had been established. In case the i2c failed 5 times a time stamp would still be provided for timing purposes by the time stored locally by the DUI.

The CPU test had been called by cpu.sh. A loop had been used inside this script to check whether sysbench had executed without error, and in case the CPU test failed sysbench would be called again.

Memory.sh Appendix 7 is the script responsible for the memory test to run. A variable equal to the size of memory to be tested is set and memtester is called once. The script check whether the memory test had been given the amount of memory requested from the OS and if that is not the case a loop runs the test again for half the amount of memory. The process continues until a threshold of 50MB is reached.

Appendix 8 shows the implementation of the script running the storage test. The processes of prepare run and cleanup are run sequentially. The exit status of each process is recorded and in case of error the process is repeated.

The GPU test script exports a few parameters before executing the test, Appendix 9. Firstly the Display number is set to 0 which directs the output of glxgears to GUI interface. Secondly

vblank mode is se to 0 which turns Vertical Sync off and the calls glxinfo to pass to the output a list of data about the GPU, which include the driver's version, rendered used and video memory available. Then glxgears is executed which is parallelised over 4 different samples, a string counting the FPS count is passed to the terminal. A loop is used to retry calling the test if the first attempt fails.

Images captured through the USB camera are instructed by camera.sh, Appendix 10. The script calls the sleep function for 30 seconds to avoid errors accessing the camera. Then fswebcam is called three times to store 3 different pictures on the eMMC. The captured images are then copied over to the USB drive and the shared folder on the laptop.

Appendix 11 shows the test.sh which is the script that calls each test individually. An argument is passed at runtime which determines the test number under execution. It should be noted for time stamp purposes time.sh is called after each process.

Finally the script called at boot, Appendix 12: Test_main.sh, is the script that called for test.sh and mounted all the peripheral devices as well as setup the log files. Once called the USB flash drive and shared folder on the laptop are mounted. Then two loops are performed to check whether test logs exist on either device in order for the number of test to be decided. Dmesg and the boot.log file are both listed on putty terminal and both log locations. Next the test begins which is again logged in two locations. After the test is completed the USB flash drive and shared folder are unmounted and the device calls the sleep function for 30 minutes. Once the time has passed a reboot is called and the test process starts from the beginning.

4.2 DATA DECODING

Data from multimeter readings had been logged throughout the experiment on the connected laptop. Readings had been passed through a serial communication from the RS232 port of the TTI 1604 digital multimeters to the USB port of the computer device using the Startech converter in between. The serial terminal captured data every 400ms and stored them as ASCII characters in log files. Since data from the RS232 port are passed as hexadecimal words, Aim-TTi provides a software to handle remote data logging from their devices. However it had not been available at the time of Data Analysis which meant a C program had to be implemented for the conversion.

Firstly an open source software called bin2hex written by Tomasz Ostrowski had been used to convert ASCII characters to hexadecimal. Then a conversion had to be made from hexadecimal

to decimal and using the table shown in Appendix 13, the exact voltage and current readings had be interpreted. The source code of program used to make the conversion is illustrated in Appendix 14.

The program used consists of three functions. Function hexconv takes as argument a pointer to an array where a hexadecimal number is stored and use a for-loop and a series of if statements to calculate the decimal value. The function then returns the decimal value. The function voltage takes as an argument the decimal value return from hexconv and using a switch statement decides which set of integers should be printed on the output stream. Once the program is executed the main function opens the file where the hexadecimal words are stored. The file is scanned and each hexadecimal is passed to hexconv and voltage respectively. Each hexadecimal word starts with an identifier number which is used to decide when a new reading is being read. A sample list of the three stages of conversion can be seen in Figure 11, as captured and converted from a 5 volt source.

An Excel file had been prepared to store the outcome of the all the interface tests. Electrical readings of voltage and current will be processed by Matlab.

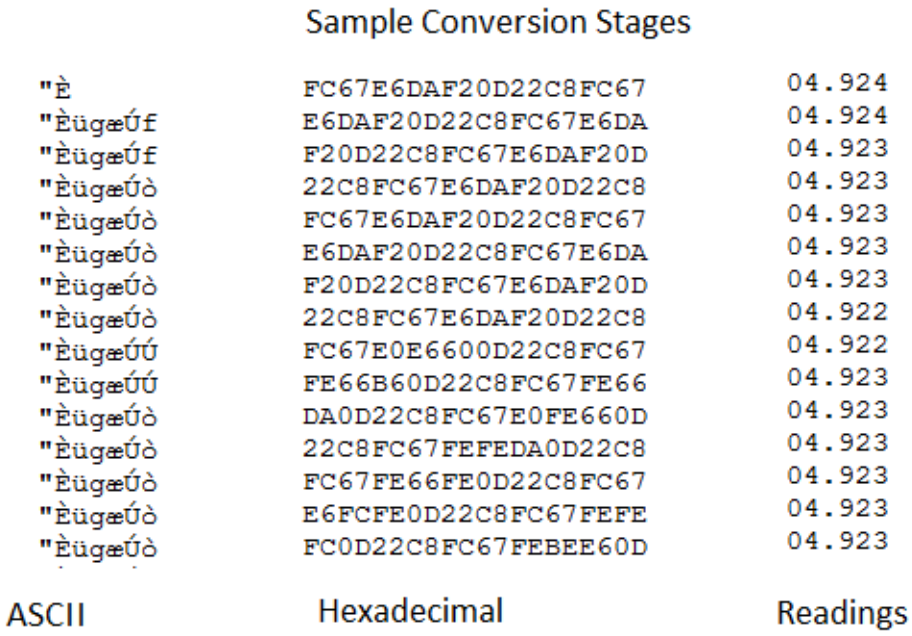


Figure 11: Sample Conversion Stages Once the DUI had been powered putty and termite logs were already running and logging multimeter readings and serial communication session. At boot Test_man.sh is called which establishes a connection to a shared folder over the laptop via the SPI interface and the USB flash drive is mounted. At this point the boot log is passed

to the log file. Then i2c, CPU, Memory, Storage and USB are called sequentially time stamping after each tests is complete. When designing the test sequence a low to high risk failure order has been kept. Vital interfaces of the system which had a lower risk of failure had been tested first and interfaces with a higher risk of failure had been tested towards the end of the test. The reasoning behind such practise had been to avoid missing vital test information. In case where the USB interface had been tested before the CPU and a system stall had been caused then no information would have been captured about the CPU status.

4.3 TYPES OF TESTING

To ensure the proper operation of the DUI it is essential to perform a series of hardware and software testing that will either monitor or stress the device. The type of testing to be conducted depends on the vitality of operation of a specific interface or the need of an interface for the devices intended use. Using this guideline should be noted that the testing does not include CSI, DSI, SD/SDIO, DPI and SMI.

4.3.1 UART

The UART is used as the default method to communicate with the DUI from the laptop. A serial communication is established through the putty terminal and all the data passed through to putty are logged and failure of the interface can easily be identified from logged data. It is predicted that this interface will function as long as the CPU does, no clock is used therefore failure risk is low. In case of failure the test will continue using an SSH connection to the DUI if SPI is live.

4.3.2 I2C

In order to test the functionality of the I2C interface a real time clock had been used. Since the DUI does not have an RTC on board it would also provide the functionality of time stamping during the test. At the beginning of each test cycle i2cdetect is used to check the I2C bus and return the state of each address. The outcome is shown on Figure 4 the note UU describes that a device is recognised on address 0x68 and its drives have been loaded successfully. Time stamping is also a mean of confirming the proper functionality of I2c over the course of each test. I2C data lines are pulled up to the 3,3v rail and the i2c clock is used to transmit data between devices therefore it is categorised in the low risk of failure interfaces.

```
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- UU -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Figure 4: i2cdetect

4.3.3 SPI

The SPI interface will be tested using an Ethernet development board. The board provides an interface for the DUI to communicate with the laptop. By establishing a connection the DUI will send log files and Images captured to the laptop. This serves as a backup logging location but also a test that ensures proper operation of the SPI interface when a log file exists. In case the UART fails the SPI interface can provide communication to the pi in order for the test to continue. Even though it can be used as backup in case UART fails, the SPI is more likely to fail before UART does.

4.3.4 USB

The CM 3 I/O board accommodates one USB type a port that gives access to the USB OTG interface of the PI. For testing as well as logging purpose two devices where connected through a hub to the USB port. A USB flash drive was used to as a logging device as well as a reference in case of interface failure, in that case the last output file would indicate the point in time where the USB interface failed. An alternative test had been the use of a USB camera, which was used to capture images and store them on the eMMC, SPI log folder and USB flash drive. Again any failure of the interface would result in the DUI failing to communicate with the Camera. The interface has a high risk of failure and is likely to fail first, this had been the case is the experiment performed in 3.3.1.

4.3.5 CPU – Stress Test

Sysbench is an open source software available on Github by Alexander Kopytov. One of the functionalities included is a benchmark test for the CPU which was used for the purpose of the experiment. The test runs for a given period of time which for the purpose of this test was set to 3 minutes. Using all four threads of the CPU it generates prime number up to a limit of 10000. The outcome of the test, as shown in Figure 5, is the CPU speed in event per second,

the total number of events executed and statistics of the CPU latency. This software provides a strong representation of the CPU speed and responsiveness throughout the experiment. It is expected that as TID increases the total number of events executed in 3 minutes and in turn eps will reduce.

```
CPU speed:
    events per second:   217.54

Throughput:
    events/s (eps):                      217.5419
    time elapsed:                        180.0159s
    total number of events:              39161

Latency (ms):
         min:                                    16.66
         avg:                                    18.39
         max:                                    43.83
         95th percentile:                        20.37
         sum:                                 719977.86
```

Figure 5: CPU Test

4.3.6 Memory Test

For this part of the experiment a software called memtester 4.3 was used. The software is also publicly available on Github and was written by Charles Cazabon. The program runs at every cycle of the test, asking for a certain amount of memory from the operating system. For this experiment 300 MB was used to balance time taken for the memory test to complete and a sufficient amount of memory to be tested. Once a block of memory is secured, the software executes a series of 18 different memory tests and return the status of each test. The individual tests performed are illustrated in Figure 6. In case of failure while running a memory process the outcome of will be a non ok result, though the memory is considered a low failure risk interface.

```
want 300MB (314572800 bytes)
got  300MB (314572800 bytes), trying mlock ...locked.
Loop 1/1:
  Stuck Address       : ok
  Random Value        : ok
  Compare XOR         : ok
  Compare SUB         : ok
  Compare MUL         : ok
  Compare DIV         : ok
  Compare OR          : ok
  Compare AND         : ok
  Sequential Increment: ok
  Solid Bits          : ok
  Block Sequential    : ok
  Checkerboard        : ok
  Bit Spread          : ok
  Bit Flip            : ok
  Walking Ones        : ok
  Walking Zeroes      : ok
  8-bit Writes        : ok
  16-bit Writes       : ok
```

Figure 6: Memory Test

4.3.7 Storage Test

The eMMC can be tested using the fileio test of sysbench. In order to run a test the prepare command should be executed first, this creates a defined number of files that occupy a given amount of eMMC space. Once all the files are created sysbench is called with the run argument which executes sequential or random reads and writes to the files created. A validate option is available which uses a checksum created inside the files to confirm that each read and write is valid. The test returns the throughput of each operation as shown in Figure 7. After the test is complete a cleanup argument is called to delete the files. The input output operations per second for read and write are expected to degrade over time. During the control test, the fileio test caused the device to freeze while preparing the files for the test. Hence the decision had been made to remove the test from the experiment to avoid any complications in the automation process.

```
Throughput:
        read:  IOPS=123.83 1.93 MiB/s (2.03 MB/s)
        write: IOPS=82.60 1.29 MiB/s (1.35 MB/s)
        fsync: IOPS=66.02

Latency (ms):
        min:                                    0.05
        avg:                                   58.63
        max:                                 5160.75
        95th percentile:                       64.47
        sum:                                2887899.26
```

Figure 7: Storage Test

4.4 HARDWARE SETUP

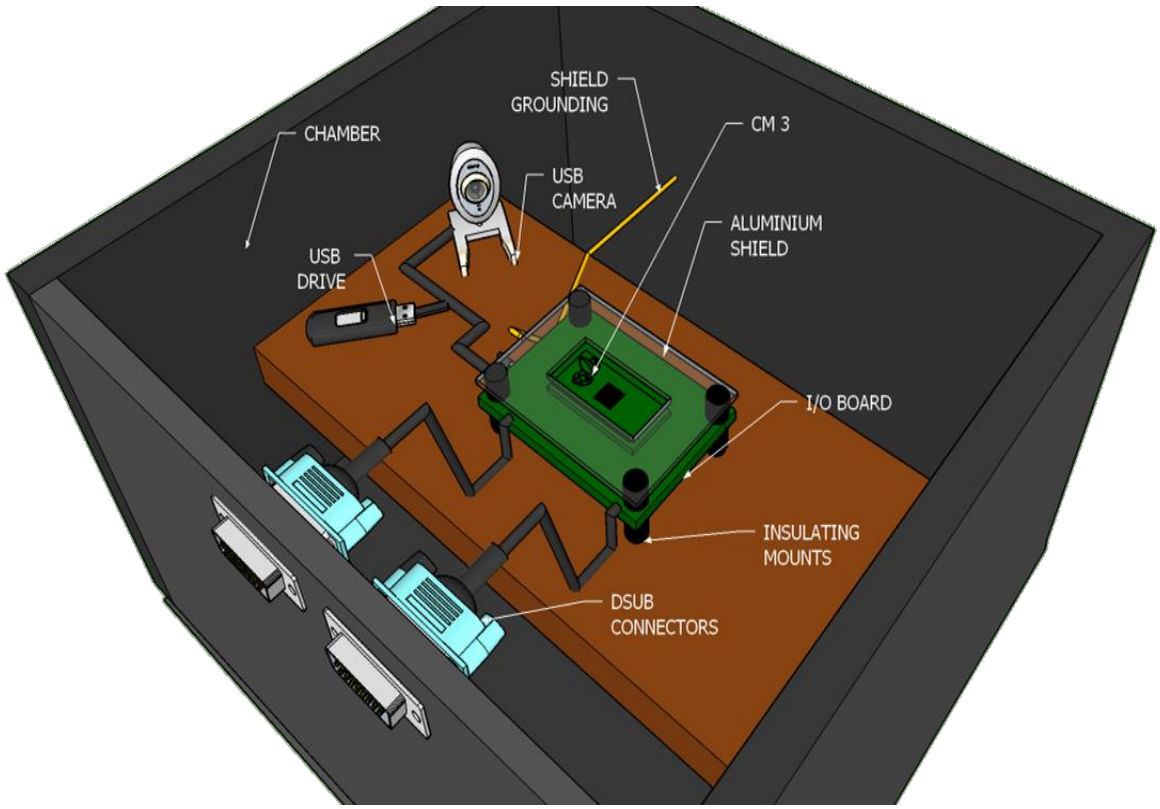


Figure 8: Radiation Chamber

The DUI (device under irradiation) was set inside a sealed chamber as illustrated in Figure 8. Spacing between radiation source and device had been measured to 2 cm. The CM3 was attached to the I/O board which was shielded using a 2mm thick aluminium sheet, exposing only the Broadcom Processor chip to radiation. Kapton®, a polyimide film made by Dupont,

had been used on the bottom side of the shield, the bottom side of the Compute module I/O board as well as the real time clock. This practise ensured that any exposed pins were covered and no short circuits could occur since the film is a good insulator. In addition charge discharge from any build up on the shield to the DUI was avoided, a picture of the DUI is shown in Figure 9. Teflon mounts were used to secure the shield onto the I/O board. A USB drive along with a USB camera were both connected to the I/O board within the chamber through a USB hub powered directly from the DUI. The device was powered using a desk power supply through the microUSB port. The remaining peripherals used were installed outside the chamber to avoid irradiation and wiring connection was provided by the d-sub plugs already installed on the chamber. A wiring diagram is provided in Figure 10. Alterations to the plan had to be made to resolve and issue caused to the real time clock when connected via the D-type cabling to the device. After observations the decision had been made to keep the real time clock inside the chamber and connected directly to the I/O board pins, since the length of the d-type cabling and consequently its impedance was causing the I2C to fail.

Table 3: Equipment List

No.	Name	Serial No.	Quantity
1	Raspberry Pi	Compute Module 3	1
2	Raspberry Pi Development board	I/O board CM3	1
3	Adafruit Real Time Clock board	PCF8523	1
4	ENC28J60 Ethernet Development Board	ENC28J60	1
5	USB Flash Drive		1
6	USB Camera		1
7	FTDI Chip USB to UART Cable for Raspberry Pi	TTL-232R-Rpi	1
8	Digital Multimeter	AIM-TTi 1604	4
9	HP Laptop Windows 10	Envy 15	1
10	Power supply		1
11	USB 2.0 to RS232 Serial Converter	Startech ICUSB2321F	3
12	USB 2.0 to RS232 Serial Converter		1
13	USB hub		2

A list of equipment used is provided in

Table 3. A laptop with Windows 10 64-bit installed was used to communicate with the DUI, Putty release 0.70 is the software used to provide serial or SSH communication to the device. Wireshark 2.4.4 had been already installed to detect the IP address of the DUI for SSH communication through the SPI interface to be established had the serial communication through UART interface failed. A total of two devices were installed outside the chamber a SPI Ethernet board and a FTDI UART to USB cable, connections to the board can also be seen in Figure 10. Voltage readings were captured from the 5V, 3V3 and 1V8 rails respectively from three individual digital multimeters and were logged to the laptop using the serial console Termite. Similarly a digital multimeter had been connected in series with the power supply to capture current readings which were also logged. The power supply used was set prior to testing to 5.05 volts, just above 5V to account for the voltage drop across the wiring. Current limit was set to 2 A which was more than enough for the operation of the device, almost double the amount drawn during control testing, but still within safe limits.

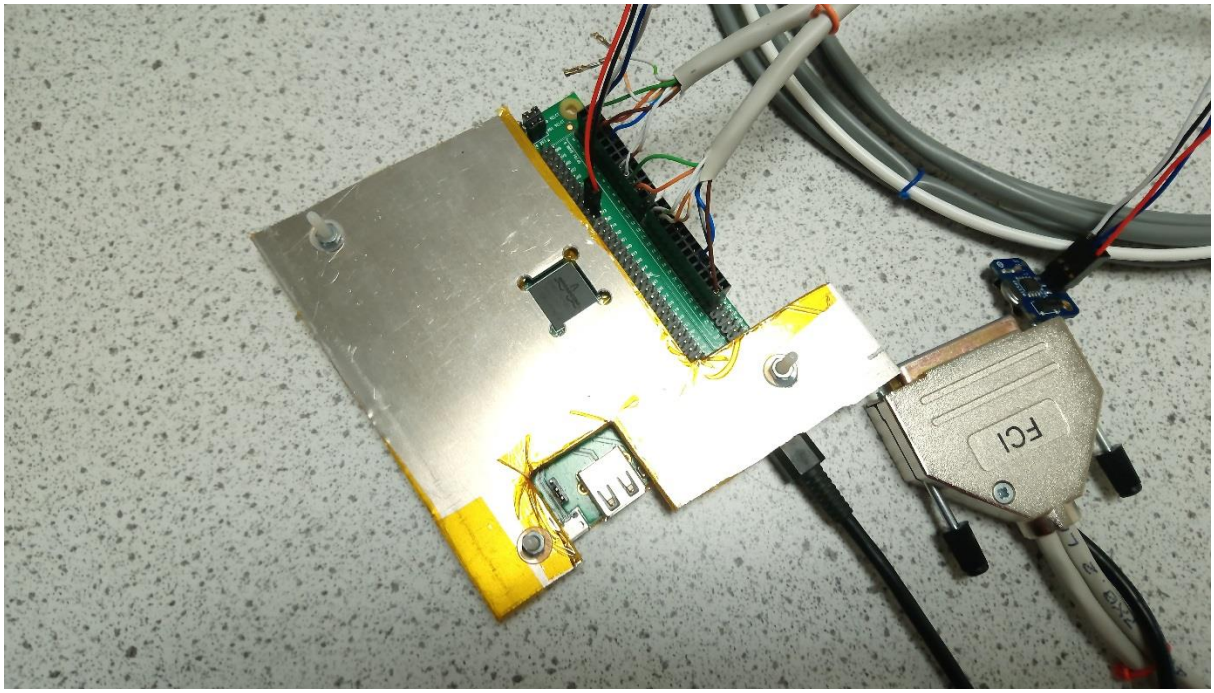


Figure 9 : DUI with shield, mounts and Kapton film

Figure 10 includes a pin diagram of the complete setup. The USB devices located inside the chamber were connected directly to the I/O board using the installed USB OTG plug through a USB hub which was powered by the DUI. An Adafruit PCF8523 Real Time Clock was connected on the I2C interface which was powered by the 3v3 rail of the DUI. The data and clock pins, SDA and SCL, were mounted to GPIO 2 and 3 respectively. A CR1220 coin battery

was installed on the RTC board and was used to provide power to the clock and avoid the time being lost when the DUI was powered off.

For the SPI interface testing, the ENC28J60 Ethernet Development Board was used. Power was again provided by the 3v3 rail. GPIO 8, 9, 10, 11 and 25 where connected to the Ethernet board to ensure proper functionality. Pin 8 provided the chip select option, pins 9 and 10 were used as MISO and MOSI (Master In Slave Out, Master Out Slave In) and pin 11 was the clock. Pin 25 provided an interrupt. An Ethernet cable established the connection from the Ethernet board to the laptop's Ethernet port.

For the UART to USB serial communication a FTDI Chip USB to UART Cable was used. Connection to the DUI was established by using GPIO 14 as TX, GPIO 15 as RX and a common ground. The USB cable connected to the laptop. A simple Veroboard design was implemented to provide pins of RX, TX and GND outside of the chamber as well as rails for 5v0, 3v3 and 1v8 for voltage readings to be taken.

Digital multimeters were connected to the laptop via an individually powered USB Hub. Four RS232 converters were used to provide compatibility from the multimeter's RS232 port to the laptop where readings would be logged.

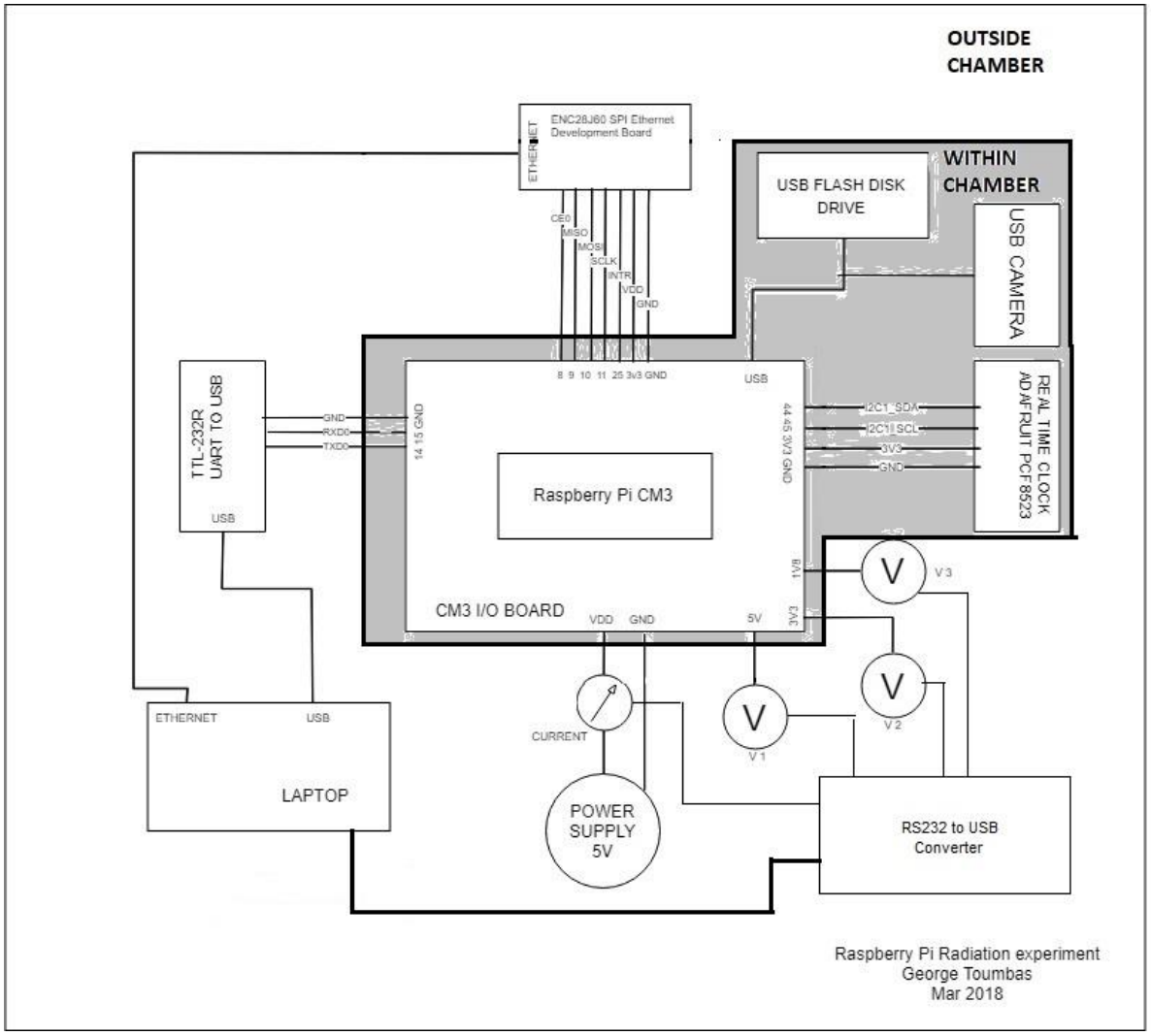


Figure 10: Wiring

4.5 SOFTWARE SETUP

4.5.1 Software Installation

A series of steps and installations had to be performed in order to prepare the device for this experiment. A detailed description will follow under this section.

First of all an operating system had to be installed. An official unmodified version was preferred, therefore the only option was Raspbian Stretch lite 4.14 April 2018. The Desktop version was not compatible since more than the available 4GB of storage are required. The installation was performed on a Linux machine using rpiboot software to power the device's flash memory and write the OS image.

The drivers for all peripherals had to be enabled. This was executed by editing the boot configuration file and setting dtoverlay parameters for the Ethernet board the VideoCoreIV and

real time clock driver. An error given by the real time clock at random times after each reboot had been fixed by only enabling the RTC interface in the boot configuration file but loading the pcf8523 driver when the booting process had already finished. Therefore an explicit command to load the PCF drivers was written inside the rc.local file which executes towards the end of the booting process rather than right after the kernel loads which had been the case with configuration file. While editing the configuration file the I2C, UART and SPI interfaces were enabled.

Once the Ethernet board was fully working a connection to a router had been established via an Ethernet cable to proceed with downloading the additional software. At this stage i2c-tools were installed which included the i2cdetect software. Next was to install sysbench which was the software to run for both the CPU and eMMC tests. Installing from the APT library gave room for errors since the 0.5 version was available. The validate option of the fileio test which was the function testing the eMMC and validating reads and writes was crashing the program. The problem was resolved by downloading the 1.0 version from github and compiling.

A similar approach had to be followed when downloading the memory tool, memtester, from the APT library. The output file of this test was eleven pages long and was the result of each individual function within the test passing characters (“-/|\-“) to the output stream for each memory write performed. As a result an enormous volume of data would have gathered and would be challenging to analyse. The program had to be downloaded from github and the source file had been altered to remove the unwanted set of characters from the test result.

The most challenging part of the device preparation was getting the VideoCore IV to render images. Initially the software to be used had to be installed. Glxgears had been downloaded through mesa utilities from the APT library. At the time the only way to launch the application was through an SSH connection and using an x-server application to transfer the rendered images over to the laptop connected. However it was soon realised that the particular approach was using software acceleration rather that hardware acceleration. One solution would be to follow the Virtual GL path, which is an open source toolkit that allows remote display software to launch applications with 3D hardware acceleration. However it had been ruled out since the kernel would have to be modified and only remote interfaces that support xserver communication would work, therefore a test over UART would be impossible. The solution to this problem came with the installation of the PIXEL GUI. In this way the glxgears would run on the GUI interface using the GUI as its Display and therefore rendering would be executed

by the on-board GPU. A total of four applications were installed from the APT libraries including a display server, the desktop environment, a window manager and a login manager. Once the installation was completed glxgears could run in the background on the GUI interface and the frames per second were past to the terminal.

Finally capturing images from the USB camera had been handled by fswebcam which is an application available on APT libraries. Jpeg pallets had already been installed as part of the mesa drivers. Prompting for user credentials prior to login had been disabled and the device automatically logged in to tty0 session as user pi.

4.5.2 Test Script

Bash scripts had been drafted in order to run all the necessary instructions automatically. Description of the scripts is included in this section. All scripts used are available in the Appendices. Individual instructions had been performed by different scripts which were all called by a main test script.

Firstly a script had been created to read the time from the real time clock, called time.sh Appendix 5. In order to account for the i2c bus being busy a delay of 1 second was held before each access to the clock. The script consists of a loop that checks whether a communication to the clock had been established. In case the i2c failed 5 times a time stamp would still be provided for timing purposes by the time stored locally by the DUI.

The CPU test had been called by cpu.sh. A loop had been used inside this script to check whether sysbench had executed without error, and in case the CPU test failed sysbench would be called again.

Memory.sh Appendix 7 is the script responsible for the memory test to run. A variable equal to the size of memory to be tested is set and memtester is called once. The script check whether the memory test had been given the amount of memory requested from the OS and if that is not the case a loop runs the test again for half the amount of memory. The process continues until a threshold of 50MB is reached.

Appendix 8 shows the implementation of the script running the storage test. The processes of prepare run and cleanup are run sequentially. The exit status of each process is recorded and in case of error the process is repeated.

The GPU test script exports a few parameters before executing the test, Appendix 9. Firstly the Display number is set to 0 which directs the output of glxgears to GUI interface. Secondly

vblank mode is se to 0 which turns Vertical Sync off and the calls glxinfo to pass to the output a list of data about the GPU, which include the driver's version, rendered used and video memory available. Then glxgears is executed which is parallelised over 4 different samples, a string counting the FPS count is passed to the terminal. A loop is used to retry calling the test if the first attempt fails.

Images captured through the USB camera are instructed by camera.sh, Appendix 10. The script calls the sleep function for 30 seconds to avoid errors accessing the camera. Then fswebcam is called three times to store 3 different pictures on the eMMC. The captured images are then copied over to the USB drive and the shared folder on the laptop.

Appendix 11 shows the test.sh which is the script that calls each test individually. An argument is passed at runtime which determines the test number under execution. It should be noted for time stamp purposes time.sh is called after each process.

Finally the script called at boot, Appendix 12: Test_main.sh, is the script that called for test.sh and mounted all the peripheral devices as well as setup the log files. Once called the USB flash drive and shared folder on the laptop are mounted. Then two loops are performed to check whether test logs exist on either device in order for the number of test to be decided. Dmesg and the boot.log file are both listed on putty terminal and both log locations. Next the test begins which is again logged in two locations. After the test is completed the USB flash drive and shared folder are unmounted and the device calls the sleep function for 30 minutes. Once the time has passed a reboot is called and the test process starts from the beginning.

4.6 DATA DECODING

Data from multimeter readings had been logged throughout the experiment on the connected laptop. Readings had been passed through a serial communication from the RS232 port of the TTI 1604 digital multimeters to the USB port of the computer device using the Startech converter in between. The serial terminal captured data every 400ms and stored them as ASCII characters in log files. Since data from the RS232 port are passed as hexadecimal words, Aim-TTi provides a software to handle remote data logging from their devices. However it had not been available at the time of Data Analysis which meant a C program had to be implemented for the conversion.

Firstly an open source software called bin2hex written by Tomasz Ostrowski had been used to convert ASCII characters to hexadecimal. Then a conversion had to be made from hexadecimal

to decimal and using the table shown in Appendix 13, the exact voltage and current readings had be interpreted. The source code of program used to make the conversion is illustrated in Appendix 14.

The program used consists of three functions. Function hexconv takes as argument a pointer to an array where a hexadecimal number is stored and use a for-loop and a series of if statements to calculate the decimal value. The function then returns the decimal value. The function voltage takes as an argument the decimal value return from hexconv and using a switch statement decides which set of integers should be printed on the output stream. Once the program is executed the main function opens the file where the hexadecimal words are stored. The file is scanned and each hexadecimal is passed to hexconv and voltage respectively. Each hexadecimal word starts with an identifier number which is used to decide when a new reading is being read. A sample list of the three stages of conversion can be seen in Figure 11, as captured and converted from a 5 volt source.

An Excel file had been prepared to store the outcome of the all the interface tests. Electrical readings of voltage and current will be processed by Matlab.

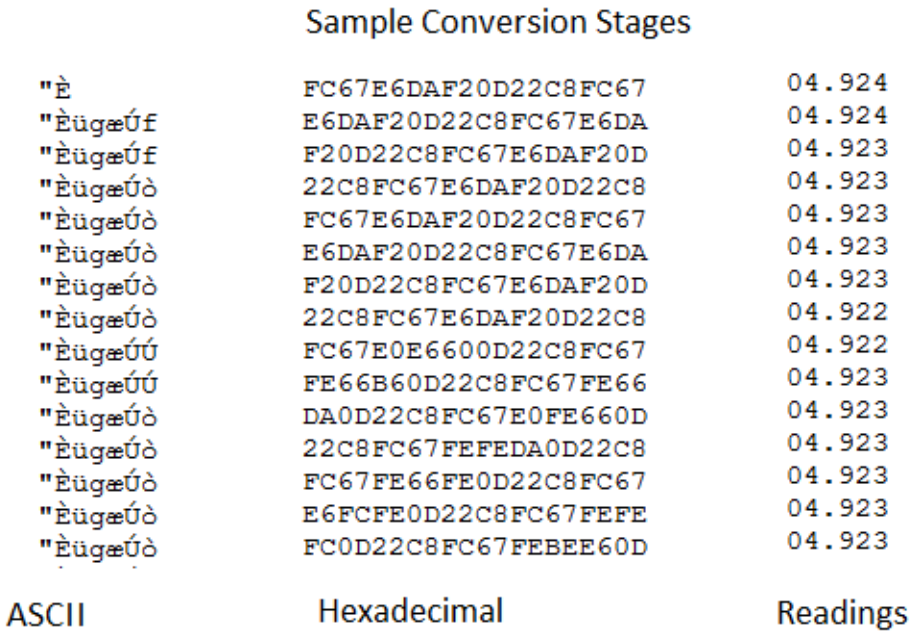


Figure 11: Sample Conversion Stages

4.7 SUMMARY

By the end of this chapter the experiment had been ready to be executed. All interface tests had been carefully picked and designed to ensure result validity. All scripts have already been implemented and revised to ensure no errors had been missed. Wiring required had been prepared in EEE Undergraduate Labs using components already stocked. Components including the d-type back shells, RTC and Ethernet Boards had been ordered from RS-components and Mouser respectively using the £100 budget provided by the project. Software used to interpret electrical readings had been tested to function properly.

5 THE EXPERIMENT

5.1 CONTROL EXPERIMENT

Prior to irradiation a control experiment had to be performed. Data gathered have been used to compare the behaviour of the DUI before and while under irradiation. The experiment had been performed within the Undergraduate Laboratory. All wiring used were the same with those used on the day of the experiment. In order to replicate the d-type connections on the radiation chamber, d-type male plugs had been soldered together and were used to connect the inner and outer wiring together. Results taken from the experiment were confidently accurate and no external errors had been introduced since the setup had been the same. The outcome of the experiment is illustrated on Table 4. The 1.8 V rail had not been measured due to a failure of the RS232 to USB adapter.

For all three tests the maximum current drawn at boot ranged from 0.955 to 0.975 A. During the CPU test events per second were fluctuating at 219 eps mark, which kept the current drawn to fairly equal levels. Memory tests 1 and 3 had drawn an equal amount of current whereas test 2 had drawn 20 mA more. When it comes to the GPU test average FPS where close to 500 and were arguably stable, however Current drawn had increased 70 mA between test 1 and 3. Contrariwise the current spike during the image capture of the USB camera decreased my 100mA from test 1 to test 3. No significant fluctuation had been seen between experiments during the sleep period, in all three cases current was kept below 0.5A. The average voltage reading on the 5v rail was 4.8V. Finally the 3.3 V rail had been kept to a constant level of 3.29 fluctuating during reboots to 3.295.

Test No	Max Boot Spike (A)	CPU		Memory		GPU		Camera Spike	Sleep	Average Voltage 5v0
		EPS	I (A)	S	I (A)	FPS	I (A)	I(A)	I(A)	v (V)
1	0.961	220	0.5860	ok	0.5790	493	0.5210	0.829	0.4580	4.839
2	0.955	218	0.5760	ok	0.6080	521	0.5770	0.779	0.4740	4.851
3	0.974	219	0.5840	ok	0.5750	512	0.5930	0.686	0.4020	4.845

Table 4: Control experiment Results *1. I = Average Current during test, EPS events per second, FPS frames per second*

5.2 Sr-90 Experiment

The experiment started on Day 1 at 11:30 am. One and a half hours had been used upon execution to allow for the setup to be complete Firstly the aluminium shield had been prepared with the Kapton film and secured to the DUI. Wires had been connected to the DUI and placed inside the chamber along with the real time clock and the USB devices. The complete setup inside the chamber can be seen in the picture in **Error! Reference source not found.**. On the right corner of the image the USB camera can be seen facing the DUI. In the middle of the image the DUI is placed under the radiation source, marked with the radiation sign. Secondly

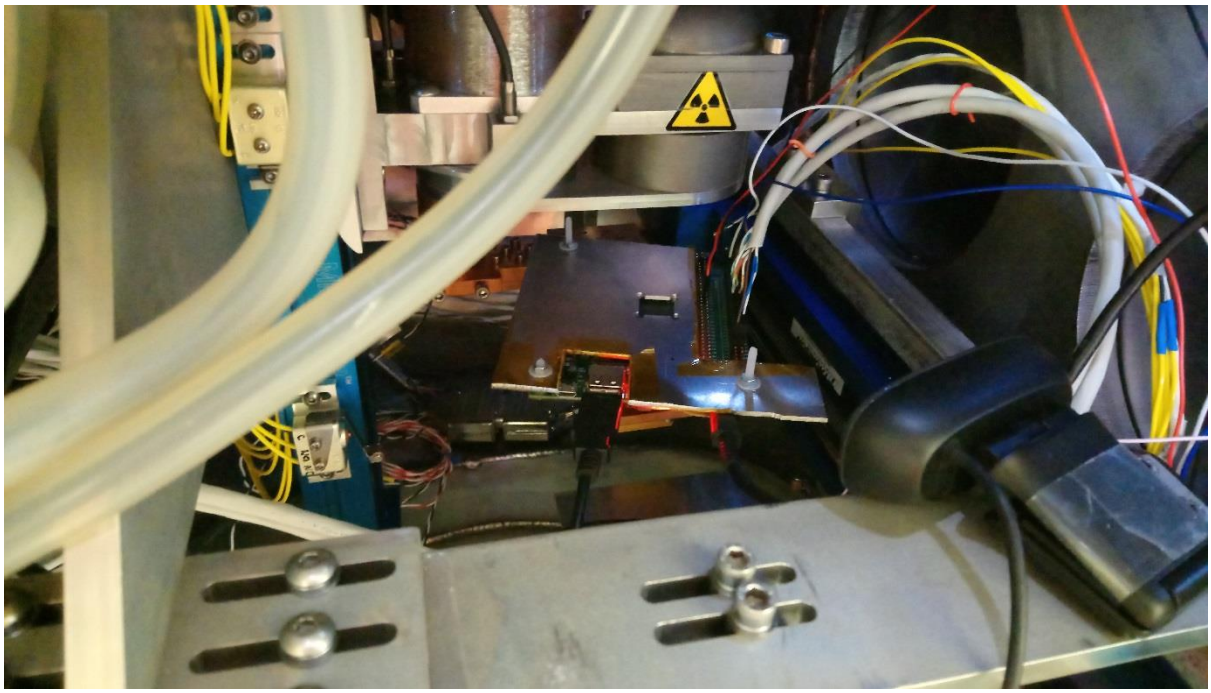


Figure 12 : Chamber Setup

the Laptop and multimeters had

been placed on a bench outside the chamber and tested for correct logging, it should be noted at this point that the test bash script had been edited to remove the eMMC test.

After completing the preparation stage, the DUI had been powered up and all the testing had been allowed to run once for verification purpose. Since everything had been connected and a verification test was successful, the experiment was ready to start. The chamber had been sealed and the radiation source moved to 4 cm from the DUI. Everything had been set and thus the source shutters had been instructed to open and begin the irradiation process. shows and image of the complete setup outside the chamber. As seen the irradiation chamber is in the left corner of the room, the laptop is setup on top of a bench running putty terminal and three termite

terminals logging the experiment process. A stack of multimeters can be seen as well which are used to measure the appropriate electrical readings.

Results included in the report account for the outcome of six days of straight testing, and repeating after roughly every 50 to 55 minutes. The total amount of test cycles performed is 165. Dosage level inside the silicon SoC had been calculated to be at 0.875 krad/hr which add to a total irradiation dosage of 126krad. The calculations had used the assumption of DUI thickness to be 1mm and it is mainly made out of fused silica, Mullasis had been used for the calculations

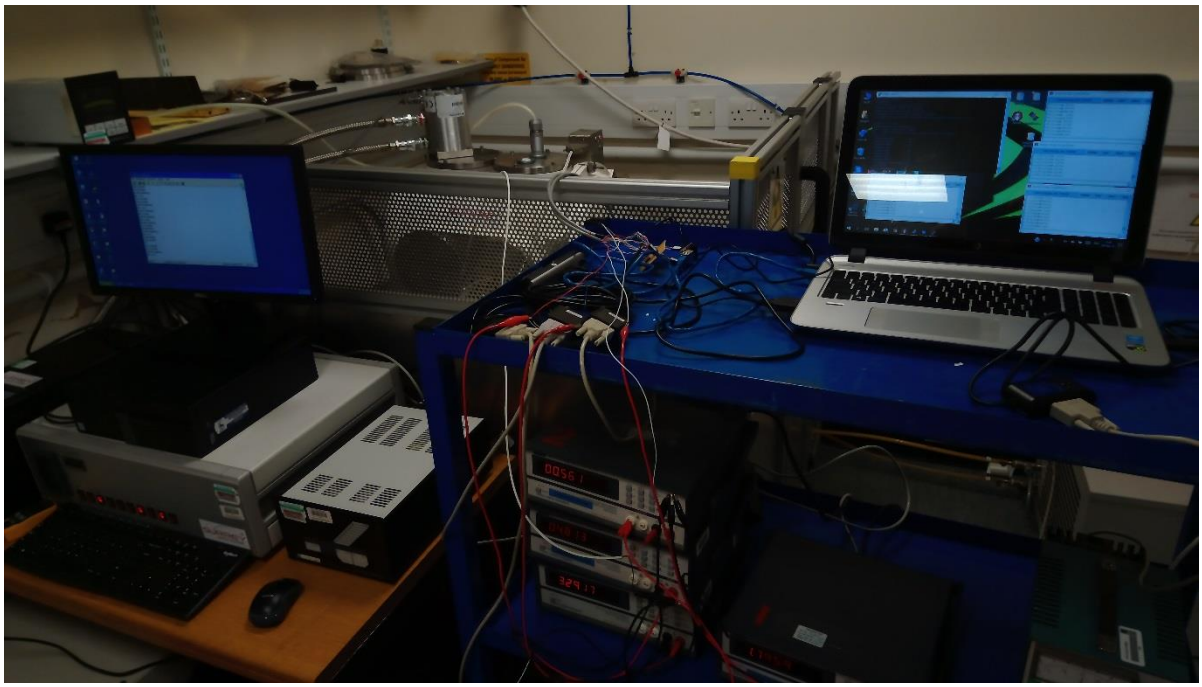


Figure 13: Experiment Setup

Since the experiment had been performed automatically and access to the facility had not been allowed past office hours, the laptop had to be access remotely. A series of data transfers had been performed daily in order for data to be analysed while the experiment was running. Through Teamviewer the laptop was accessed regularly and the normal operation of the experiment had been verified.

At 19:36 of Day 2 a remote connection had been established and evidence suggested that the use of a faulty RS232 to USB converter let to a system restart which terminated the logging process. At this point the termite consoles had to be reset and a new set of log files had been used, the same applies for putty. These had cause a loss of electrical readings for test numbers 37 to 39 as well as the UART and SPI logs for the mentioned tests. However the incident occurred at an early stage of the experiment and enough data had been gathered afterwards that

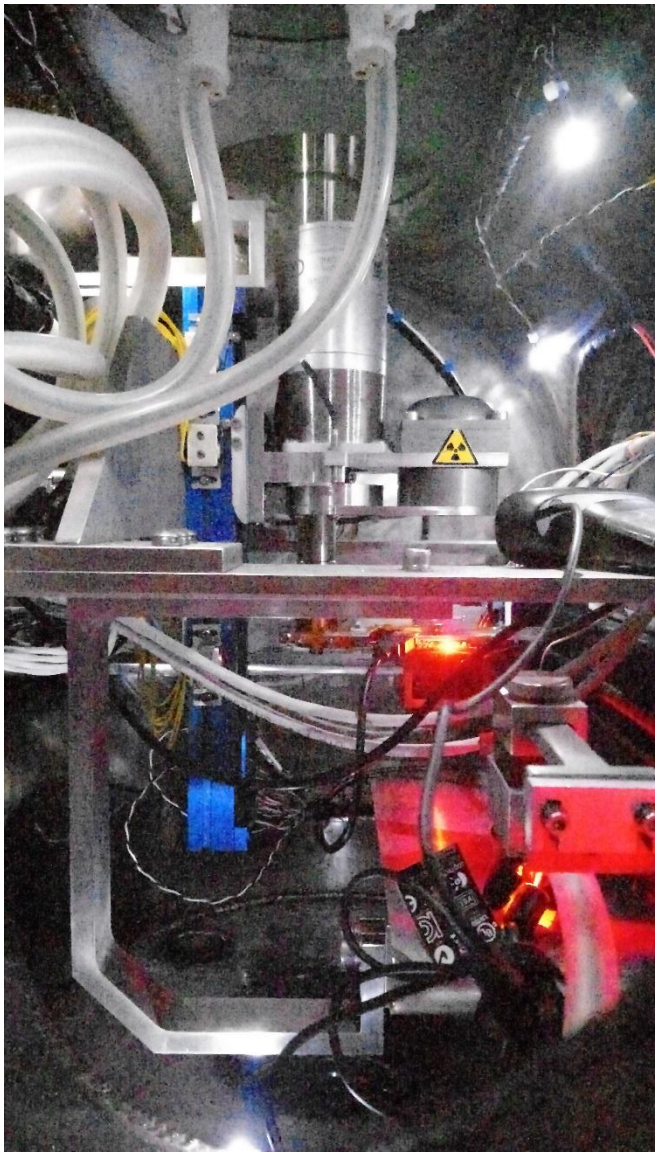


Figure 14: Day 3 chamber status

lead to the conclusion that the loss of data had not been significant enough to jeopardise the experiment outcome. Figure 14 shows an image captured during Day 3 of the experiment through the glass windows of the chamber.

The log terminal (termite and putty) had been reset every day creating new set of files for each day in order to parallelise data analysis while the experiment was running. On Day 6, while performing test number 162 and in between the memory test execution the DUI had lost connection to the laptop via the SPI interface, as a result the log file and image captured were not received inside the SPI_LOG folder. However connection had been established normally for the next test.

5.3 DATA ANALYSIS

Data presented under this section have been plotted using Matlab and Microsoft Excel. After calculating the capture intervals of each reading from the multimeters, a plot of current drawn by the DUI for one test cycle had been drawn. Knowing the capture intervals allowed for each reading to be mapped to a specific test. Figure 15 illustrates the current profile for test number 1, red notes have been used to separate the curve for each case.

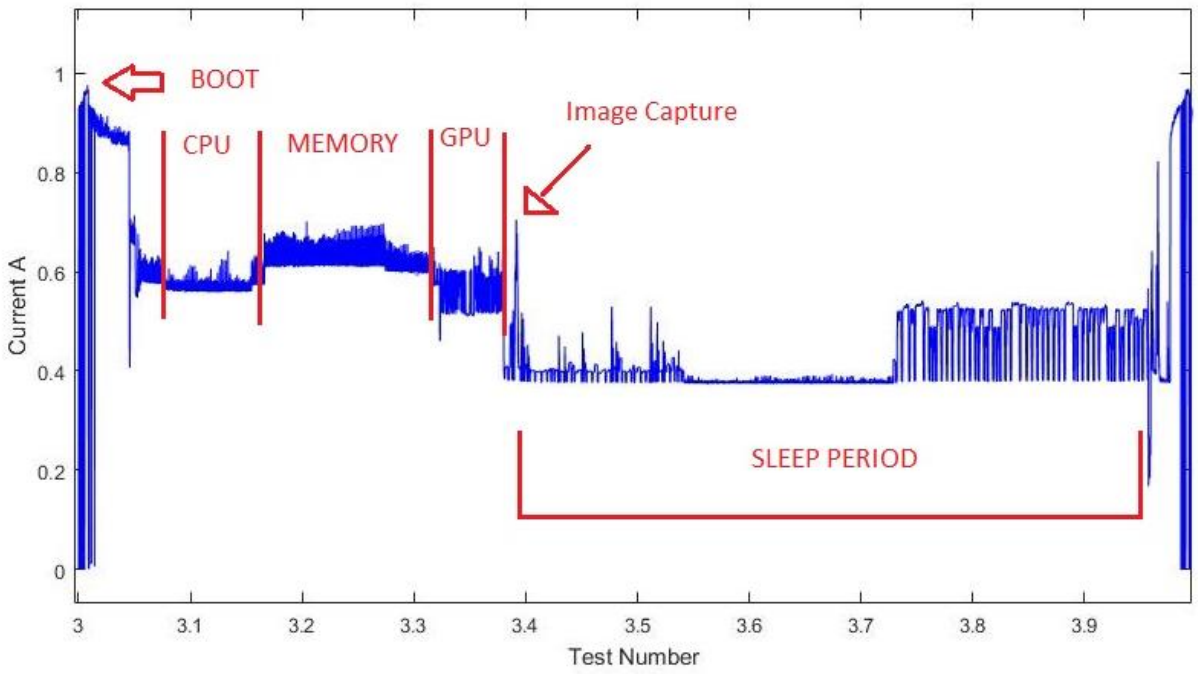


Figure 15 : Current profile of Test no 1

The highest amount of current drawn by the DUI is at boot, reaching a peak of just under 1 A. The CPU test causes the device to draw around 0.6 A, increasing by around 300ma for the memory test. During the rendering period of the GPU test the current drawn fluctuates about just under 0.6 A. A current spike can be seen of around 0.7 A when the device is instructed to capture an image from the USB camera. Finally the terminal goes to a sleep mode where the

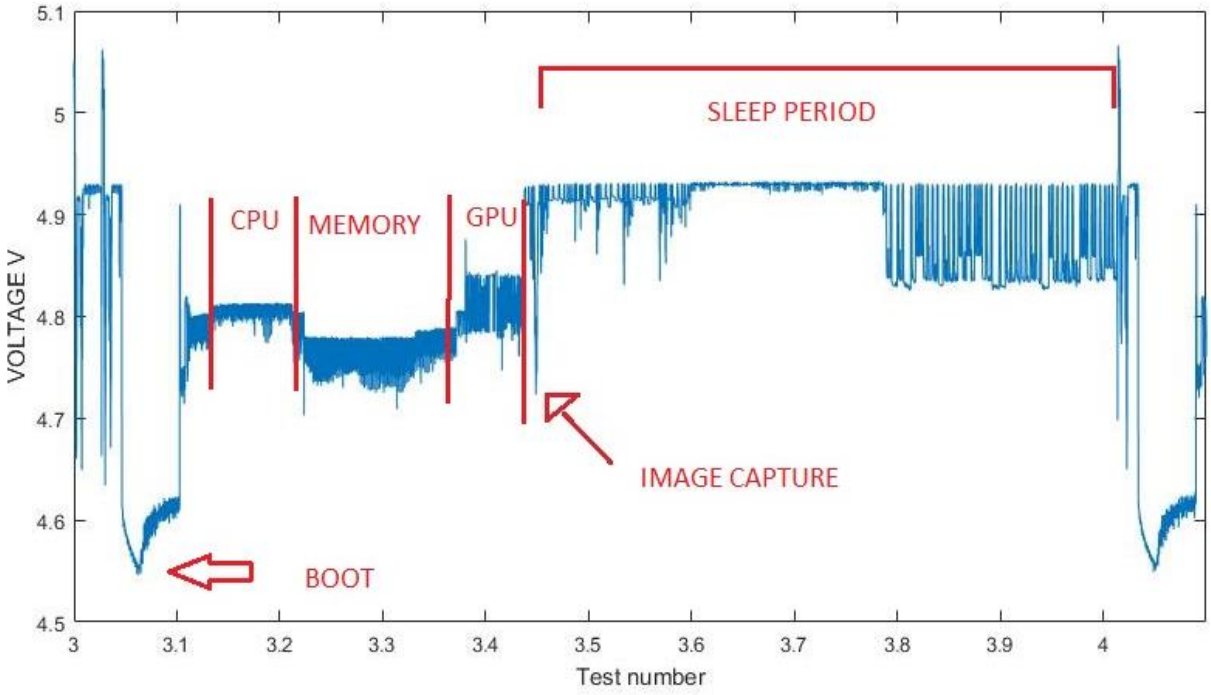


Figure 16 : Voltage profile for test no 1

current is around 0.4 A, until reboot. The average current drawn for each test had been logged and the results for every 20 tests up to 160 are available in Table 5.

The voltage profile of test number 1 is an inverted image of the current. As shown in Figure 16 the lowest voltage capture is at boot, reaching a trough of just under 4.6V. The CPU test causes the 5v rail to remain constant at around 4.8V, decreasing by around 100mV for the memory test. During the rendering period of the GPU voltage of the 5v rail fluctuates about just the 4.6 V mark. A voltage drop to almost 4.7 V is recorded when the device is instructed to capture an image from the USB camera. Finally the terminal goes to a sleep mode where the voltage is just above 4.9 V, until reboot. The average voltage level for each test cycle had been calculated and shown in Table 5.

A plot had been drawn for the 3.3 volt rail, however the fluctuation between readings had been from 3.290 V to 3.295 V. A difference of 5 mv is not enough, taking into account the precision of the digital multimeters, to not significant changes over the course of the experiment.

Table 5 : Experiment outcome 1. EPS = events per second, FPS = frames per second

Test No	Max Boot Spike (A)	CPU		Memory		GPU		Camera Spike	Sleep	Average Voltage 5v0	TID
		EPS	I (A)	S	I (A)	FPS	I (A)	I(A)	I(A)	v (V)	krad
0	0.976	218	0.5490	ok	0.6080	465	0.6170	0.704	0.4060	4.856	0
20	0.985	211	0.5810	ok	0.7480	469	0.5230	0.745	0.4080	4.851	15.8
40	0.996	208	0.6190	ok	0.6730	372	0.5450	0.851	0.5650	4.819	31.5
60	1.027	196	0.5970	ok	0.5960	547	0.5750	0.838	0.511	4.831	47.3
80	1.049	188	0.5960	ok	0.6160	523	0.6030	0.705	0.4070	4.821	63
100	1.053	184	0.6110	ok	0.6850	532	0.5960	0.683	0.4030	4.883	78.8
120	1.056	179	0.6600	ok	0.6070	540	0.5830	0.742	0.4400	4.954	94.5
140	1.073	168	0.6480	ok	0.7190	527	0.5530	0.699	0.39	4.9270	110.3
160	1.092	167	0.6430	ok	0.7370	518	0.6690	0.698	0.3890	5.1290	126

Throughout the experiment none of the interfaces tested had shown failure signs. Both SPI and UART had functioned properly and all logging were captured. A loss of operation had been recorded for SPI interface but regained communication after reboot. The USB interface worked as planned, both in capturing and storing a log file. CPU, Memory and GPU tests had not failed. Table 5 summarises the outcome of this experiment.

As seen from the table current drawn during while the device had been booting is increased with TID. From the beginning of the experiment to the 160th test the current drawn at boot has raised over 100mA. Investigating the current drawn for each individual test corresponds the trend noted for the current at boot. For each test the pattern of a total increase in current drawn of over 100ma is true. Current drawn during the terminal sleep period had increased by 100mAduring the period of TID from 15.6 krad to 63krad. However, for the remaining test cycles the sleep current returned to its original value and even dropped by 20mA. The average voltage reading of the 5v rail for one test had increased by 300mV from the start of the experiment. These statements are valid when comparing the results with the control tests.

The performance of CPU suggests that the speed of operation is degrading. As seen in the number of events per second, the average number of calculations per second during the CPU

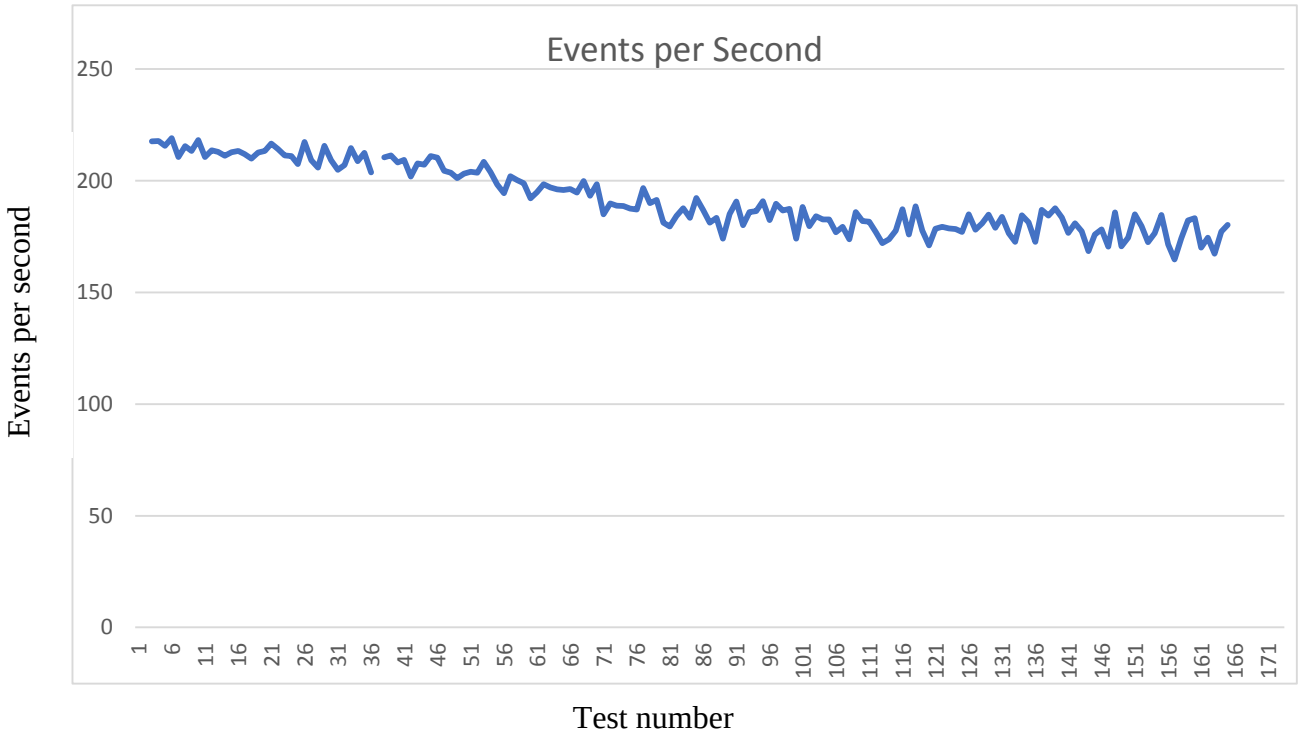


Figure 17: Events per second

test. The number of eps is dropping as the tests progress and ultimately as the total irradiation dose increases. A steep fall from above 200 to around 180 eps can be seen from test 60 to test 100 when TID was between 45 and 80 krad. The curvature for the remaining tests is kept constant fluctuating about 170 eps.

Data gathered from each GPU test had been gathered and an average number of FPS had been calculated. Data have been plotted in Figure 18. As seen the average number of FPS is fluctuating about 470 FPS. Behaviour can not be correlated to radiation.

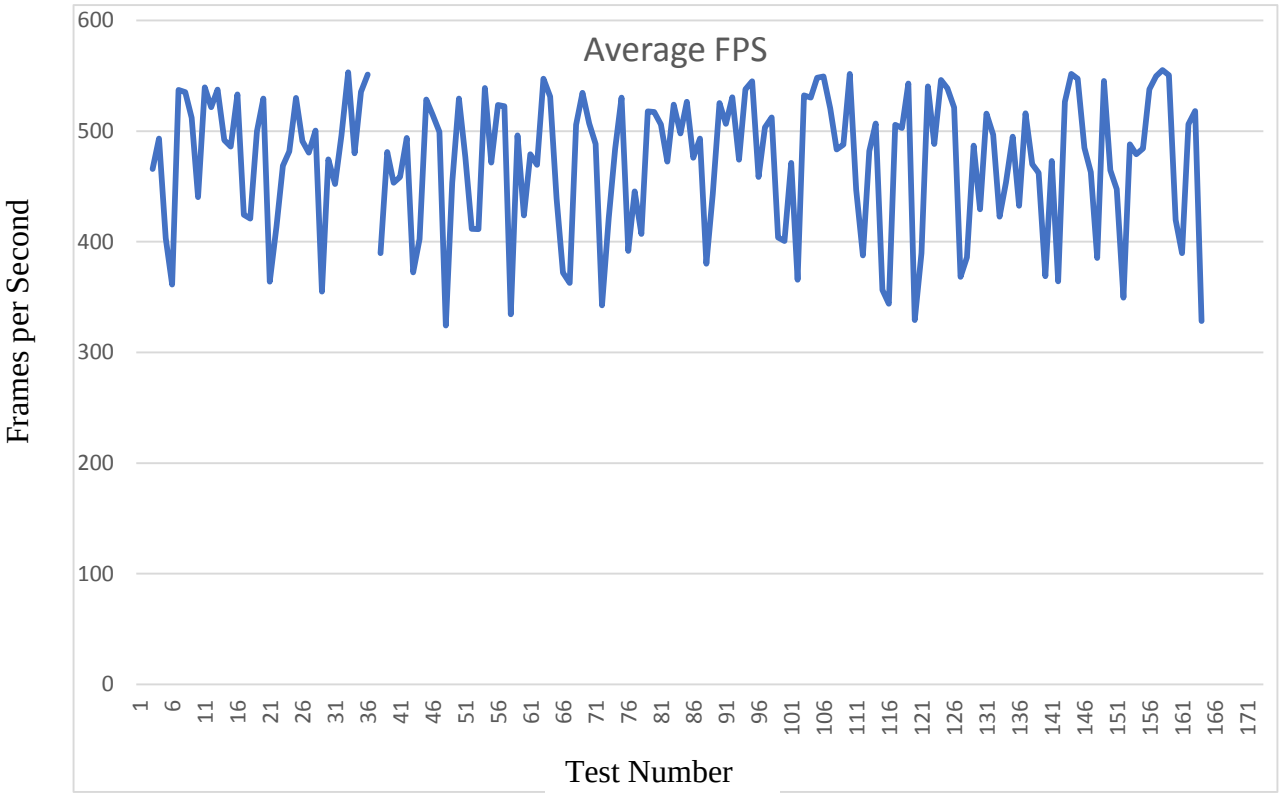


Figure 18: Frames per Second

5.4 DISCUSSION

From data analysed in section 5.3 conclusions have been made that in cases agree with the literature review and the predictions made.

By investigating the available data, an increasing amount of current, of 100mA, had been drawn from the device after about 60krad of exposure. This behaviour has been predicted and had been known that threshold currents increase with irradiation. A significant finding is the correlation of irradiation exposure with the number of events calculated in each CPU test. The decreasing number of eps as the TID is increased suggests that the CPU itself is degrading. However past the mark of 80 krad, eps are fluctuating about a constant point. The curvature of the GPU performance over TID does not illustrate evidence of effects of radiation on the VideocoreIV processor.

Calculations of dose rate reached within the Broadcom chip assumes the chip is 85% silica. However for an accurate result the architecture of the chip architecture needs to be studied and the portion of radiation that penetrates all the way through the chip calculated.

A theory suggesting the findings of this experiment follows the penetrative properties of Sr-90 and beta rays. Assuming electrons bombarded on the chip in the form of beta ray can not penetrate and irradiate the full volume of the SoC could explain the result of no radiation effects on the GPU if it is physically located at the bottom of the chip.

The same applies for the CPU tests and event count, the test runs on four threads which use all four cores of the SoC. However if not all arithmetic logic units of each core experience the same radiation level then a decrease in performance is seen by the affected core but the rest remain functioning causing the eps to reach a plateau, and not drop anymore. Similar behaviour apply for all interfaces that haven't experience failure or decreased performance.

Nevertheless this experiment has proven that a Raspberry Pi Compute Module 3 and specifically the Broadcom BCM2837 can safely operate without any function loss, under Sr-90 radiation of activity 2.6GBq, for 6 days , experiencing TID of estimated at 130 krad.

6 CONCLUSION – FUTURE NOTES

This experiment has offered a lot of knowledge throughout the academic year. Good time management and solid background knowledge had been the main contributing factors that led to a good experiment planning. Preparation had been a key for a smooth experiment run without external factors and errors jeopardizing the results. Finally data analysis had been crucial in interrupting the data outcome of the experiment and understudying DUI's behaviour.

Completing this experiment and analysing data logged has led to a conclusion that a similar device can safely operate in a cosmic radiation environment, where levels are similar to the activity of source used. Clearly radiation causes increased current drawn of 100mA when the device is operating as well as a higher voltage reading on the 5v rail by 300mV. However throughout the experiment the device had been functioning properly without severe signs of decreased performance. Evidence suggest that beta ray irradiation is not penetrative enough to cause dysfunctionalities up to levels of 130 krad. The only sign of decreased performance had been recorded during the CPU calculation tests where fewer calculations were made as TID increase, performance drop was limited and no further degrading occurred. All other interfaces tested functioned properly at all times.

A set of notes had been kept though the experiment for improving a future experiment. A factor that has not been tested during the experiment is the event of power loss of the device. This could have contributed to the limited evidence of performance degrading since the power rails are not drained and repowered which gives a higher risk upsets – latchups. A modification to a future experiment would be to measure current drawn directly from the DUI and not the CM I/O board, in order for the true current drawn from the DUI to be measured. Moreover analysis of the DUI architecture, through X-ray scanning or Electron Microscopy, result to accurate dosage calculations which in turn would help understand the behaviour of the device in greater depth. An experiment of gamma ray radiation according to MIL-STD 833H can ideally be made in the future to certify the device for space use.

Overall evidence suggests the experiment had been successful and has proven the operation of a Compute Module 3 under beta ray irradiation of up to 130krad is possible without failure.

7 ACKNOWLEDGMENT

This project has been performed under the supervision of Dr Christopher Bridges. I would like to acknowledge Dr Bridges support throughout the academic year regarding this project. Dosage level measurements had been calculated by Dr Alexander Dyer. At this point I would like to acknowledge Dr Dyer for the help in setting up the experiment inside the REEF and providing information about the radiation facility throughout the course of planning and preparing.

8 REFERENCES

Bannatyne, R., n.d. *Radiation testing of CMOS device required for space use.* [Online]
Available at: http://mil-embedded.com/articles/radiation-devices-required-space-use/
[Accessed 25 11 2017].

Department Of Defence, USA, 2010. *Test Method Standard Microcircuits, MIL-STD-883H,* s.l.: s.n.

Honess, D., 2015. *Raspberry Pi.* [Online]
Available at: https://www.raspberrypi.org/blog/astro-pi-tech-specs/
[Accessed 15 12 2017].

LaBel, K., 2004. *Radiation Effects on Electronics 101,* s.l.: s.n.

O'Brien, T., Jassin, L., Horwitz , P. & McAlister, D., 2012. *The Rapid Determination of Strontium-89 and Strontium-90 in Environmental Samples,* Kailua-Kona: s.n.

Raspberry Pi Foundation, n.d. *Raspberry Pi.* [Online]
Available at: https://www.raspberrypi.org/about/
[Accessed 2 January 2018].

Raspberry Pi Foundation, n.d. *Raspberry Pi.* [Online]
Available at: https://www.raspberrypi.org/documentation/hardware/computemodule/cm-emmc-flashing.md
[Accessed 1 3 2018].

Raspberry Pi Foundation, October 2016. *"Raspberry PI Compute Module 3" datasheet, version1.0,* s.l.: s.n.

Salzman, J., 2013. *Total Ionizing Dose and Single Event Effect Test Report,* s.l.: Texas Instruments.

The VirtualGL Project, n.d. *Virtual GL.* [Online]
Available at: https://virtualgl.org/About/Introduction
[Accessed 1 3 2018].

Underwood , C. & Pellegrino, S., 2013. *Autonomous Assembly of a Reconfigurable Space Telescope (AAReST).* s.l., s.n.

Violette, D., 2014. *Arduino/RPi: Hobbyist Hardware and Radiation Total Dose Degradtion,* Greenbelt, MD: s.n.

Washington State Department of Health, 2002. *Cobalt-60 Fact Sheet,* Washington: s.n.

9 APPENDICES

CM1	CM3-Lite	CM3	PIN		PIN	CM3	CM3-Lite	CM1
GND			1		2	EMMC_DISABLE_N		
GPIO0			3		4	NC	SDX_VDD	NC
GPIO1			5		6	NC	SDX_VDD	NC
GND			7		8	GND		NC
GPIO2			9		10	NC	SDX_CLK	NC
GPIO3			11		12	NC	SDX_CMD	NC
GND			13		14	GND		NC
GPIO4			15		16	NC	SDX_D0	NC
GPIO5			17		18	NC	SDX_D1	NC
GND			19		20	GND		NC
GPIO6			21		22	NC	SDX_D2	NC
GPIO7			23		24	NC	SDX_D3	NC
GND			25		26	GND		
GPIO8			27		28	GPIO28		
GPIO9			29		30	GPIO29		
GND			31		32	GND		
GPIO10			33		34	GPIO30		
GPIO11			35		36	GPIO31		
GND			37		38	GND		
GPIO0-27_VDD			39		40	GPIO0-27_VDD		
			KEY					
GPIO28-45_VDD			41		42	GPIO28-45_VDD		
GND			43		44	GND		
GPIO12			45		46	GPIO32		
GPIO13			47		48	GPIO33		
GND			49		50	GND		
GPIO14			51		52	GPIO34		
GPIO15			53		54	GPIO35		
GND			55		56	GND		
GPIO16			57		58	GPIO36		
GPIO17			59		60	GPIO37		
GND			61		62	GND		
GPIO18			63		64	GPIO38		
GPIO19			65		66	GPIO39		
GND			67		68	GND		
GPIO20			69		70	GPIO40		
GPIO21			71		72	GPIO41		
GND			73		74	GND		
GPIO22			75		76	GPIO42		
GPIO23			77		78	GPIO43		
GND			79		80	GND		
GPIO24			81		82	GPIO44		
GPIO25			83		84	GPIO45		
GND			85		86	GND		
GPIO26			87		88	HDMI_HPD_N_1V8		GPIO46_1V8
GPIO27			89		90	EMMC_EN_N_1V8		GPIO47_1V8
GND			91		92	GND		
DSI0_DN1			93		94	DSI1_DP0		
DSI0_DP1			95		96	DSI1_DN0		
GND			97		98	GND		
DSI0_DN0			99		100	DSI1_CP		
DSI0_DP0			101		102	DSI1_CN		
GND			103		104	GND		
DSI0_CN			105		106	DSI1_DP3		
DSI0_CP			107		108	DSI1_DN3		
GND			109		110	GND		
HDMI_CLK_N			111		112	DSI1_DP2		
HDMI_CLK_P			113		114	DSI1_DN2		
GND			115		116	GND		
HDMI_D0_N			117		118	DSI1_DP1		
HDMI_D0_P			119		120	DSI1_DN1		
GND			121		122	GND		
HDMI_D1_N			123		124	NC		
HDMI_D1_P			125		126	NC		
GND			127		128	NC		
HDMI_D2_N			129		130	NC		
HDMI_D2_P			131		132	NC		
GND			133		134	GND		
CAM1_DP3			135		136	CAM0_DP0		
CAM1_DN3			137		138	CAM0_DN0		
GND			139		140	GND		
CAM1_DP2			141		142	CAM0_CP		
CAM1_DN2			143		144	CAM0_CN		
GND			145		146	GND		
CAM1_CP			147		148	CAM0_DP1		
CAM1_CN			149		150	CAM0_DN1		
GND			151		152	GND		
CAM1_DP1			153		154	NC		
CAM1_DN1			155		156	NC		
GND			157		158	NC		
CAM1_DP0			159		160	NC		
CAM1_DN0			161		162	NC		
GND			163		164	GND		
USB_DP			165		166	TVDAC		
USB_DM			167		168	USB_OTGID		
GND			169		170	GND		
HDMI_CEC			171		172	VC_TRST_N		
HDMI_SDA			173		174	VC_TDI		
HDMI_SCL			175		176	VC_TMS		
RUN			177		178	VC_TDO		
VDD_CORE (DO NOT CONNECT)			179		180	VC_TCK		
GND			181		182	GND		
1V8			183		184	1V8		
1V8			185		186	1V8		
GND			187		188	GND		
VDAC			189		190	VDAC		
3V3			191		192	3V3		
3V3			193		194	3V3		
GND			195		196	GND		
VBAT			197		198	VBAT		
VBAT			199		200	VBAT		

Appendix 1:RPi CM3 Pin Assignment

Appendix 2:Alternative GPIO Functions

GPIO	Default Pull	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5
0	High	SDA0	SA5	PCLK	-	-	-
1	High	SCL0	SA4	DE	-	-	-
2	High	SDA1	SA3	LCD_VSYNC	-	-	-
3	High	SCL1	SA2	LCD_HSYNC	-	-	-
4	High	GPCLK0	SA1	DPI_D0	-	-	ARM_TDI
5	High	GPCLK1	SA0	DPI_D1	-	-	ARM_TDO
6	High	GPCLK2	SOE_N	DPI_D2	-	-	ARM_RTCK
7	High	SPI0_CE1_N	SWE_N	DPI_D3	-	-	-
8	High	SPI0_CE0_N	SD0	DPI_D4	-	-	-
9	Low	SPI0_MISO	SD1	DPI_D5	-	-	-
10	Low	SPI0_MOSI	SD2	DPI_D6	-	-	-
11	Low	SPI0_SCLK	SD3	DPI_D7	-	-	-
12	Low	PWM0	SD4	DPI_D8	-	-	ARM_TMS
13	Low	PWM1	SD5	DPI_D9	-	-	ARM_TCK
14	Low	TXD0	SD6	DPI_D10	-	-	TXD1
15	Low	RXD0	SD7	DPI_D11	-	-	RXD1
16	Low	FL0	SD8	DPI_D12	CTS0	SPI1_CE2_N	CTS1
17	Low	FL1	SD9	DPI_D13	RTS0	SPI1_CE1_N	RTS1
18	Low	PCM_CLK	SD10	DPI_D14	-	SPI1_CE0_N	PWM0
19	Low	PCM_FS	SD11	DPI_D15	-	SPI1_MISO	PWM1
20	Low	PCM_DIN	SD12	DPI_D16	-	SPI1_MOSI	GPCLK0
21	Low	PCM_DOUT	SD13	DPI_D17	-	SPI1_SCLK	GPCLK1
22	Low	SD0_CLK	SD14	DPI_D18	SD1_CLK	ARM_TRST	-
23	Low	SD0_CMD	SD15	DPI_D19	SD1_CMD	ARM_RTCK	-
24	Low	SD0_DAT0	SD16	DPI_D20	SD1_DAT0	ARM_TDO	-
25	Low	SD0_DAT1	SD17	DPI_D21	SD1_DAT1	ARM_TCK	-
26	Low	SD0_DAT2	TE0	DPI_D22	SD1_DAT2	ARM_TDI	-
27	Low	SD0_DAT3	TE1	DPI_D23	SD1_DAT3	ARM_TMS	-
28	None	SDA0	SA5	PCM_CLK	FL0	-	-
29	None	SCL0	SA4	PCM_FS	FL1	-	-
30	Low	TE0	SA3	PCM_DIN	CTS0	-	CTS1
31	Low	FL0	SA2	PCM_DOUT	RTS0	-	RTS1
32	Low	GPCLK0	SA1	RING_OCLK	TXD0	-	TXD1
33	Low	FL1	SA0	TE1	RXD0	-	RXD1
34	High	GPCLK0	SOE_N	TE2	SD1_CLK	-	-
35	High	SPI0_CE1_N	SWE_N	-	SD1_CMD	-	-
36	High	SPI0_CE0_N	SD0	TXD0	SD1_DAT0	-	-
37	Low	SPI0_MISO	SD1	RXD0	SD1_DAT1	-	-
38	Low	SPI0_MOSI	SD2	RTS0	SD1_DAT2	-	-
39	Low	SPI0_SCLK	SD3	CTS0	SD1_DAT3	-	-
40	Low	PWM0	SD4	-	SD1_DAT4	SPI2_MISO	TXD1
41	Low	PWM1	SD5	TE0	SD1_DAT5	SPI2_MOSI	RXD1
42	Low	GPCLK1	SD6	TE1	SD1_DAT6	SPI2_SCLK	RTS1
43	Low	GPCLK2	SD7	TE2	SD1_DAT7	SPI2_CE0_N	CTS1
44	None	GPCLK1	SDA0	SDA1	TE0	SPI2_CE1_N	-
45	None	PWM1	SCL0	SCL1	TE1	SPI2_CE2_N	-

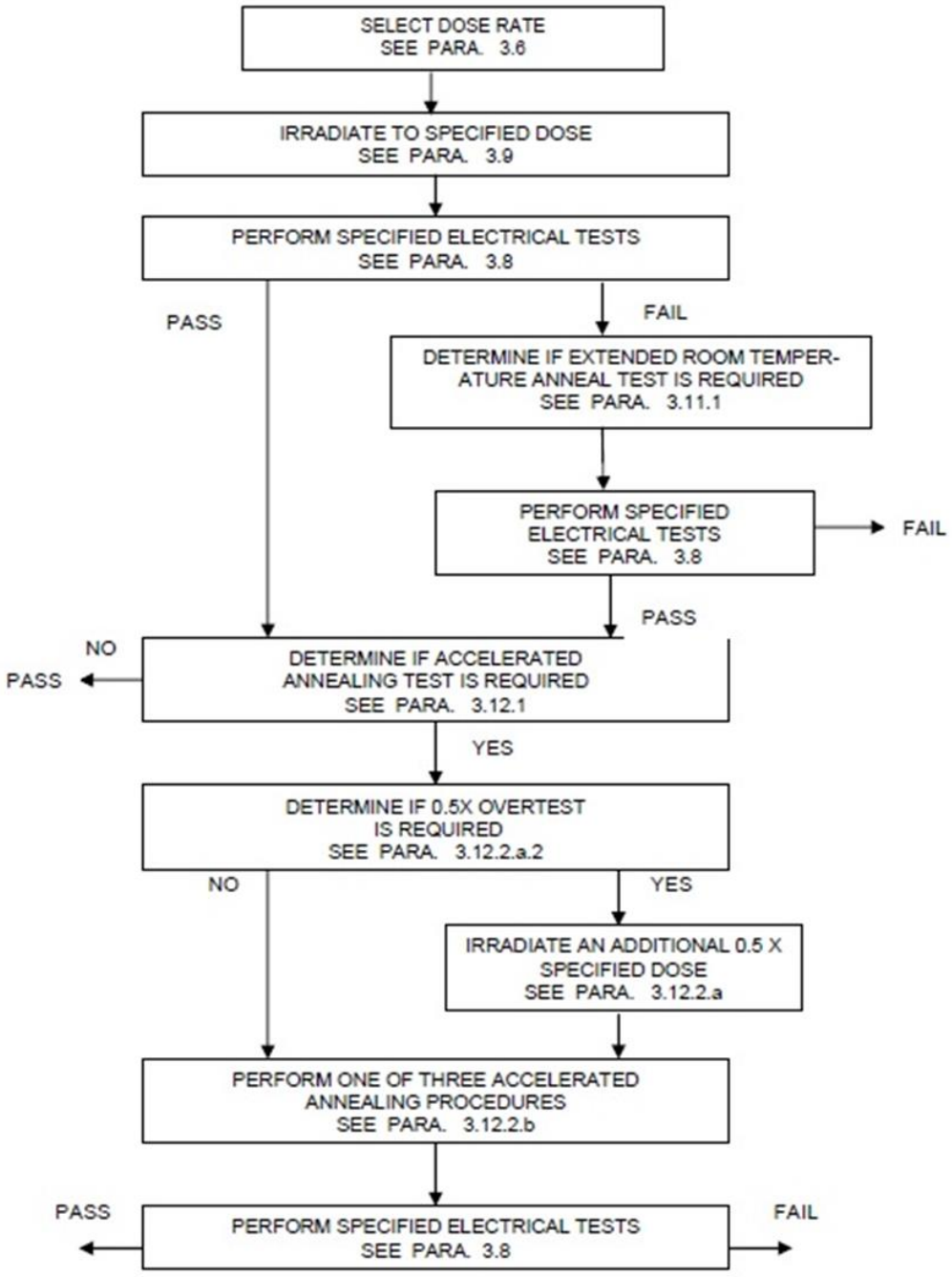
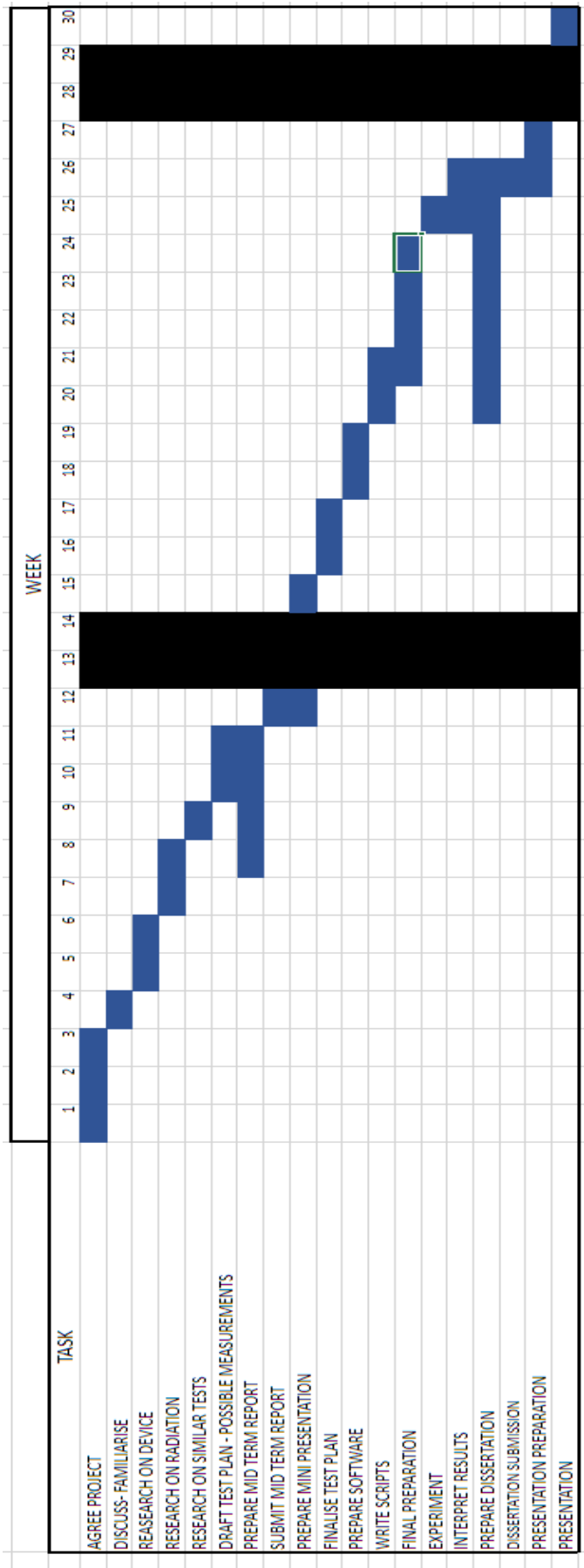


FIGURE 1019-1. Flow diagram for ionizing radiation test procedure for MOS and digital bipolar devices.

METHOD 1019.8

Appendix 3: Test procedure according to MIL STD 883H



Appendix 4 : Gant Chart

time.sh

```
#!/bin/bash


ex_status=0
sleep 1
while [ $ex_status -le 4 ]
do
    ((ex_status++))
    echo "TIMESTAMP"
    sudo hwclock
    ret_Val=$?
    if [ "$ret_Val" -eq 0 ]
        then
        ex_status=0
        exit $ex_status
    fi
    echo "ERROR GETTING TIME"
    echo "RETRYING..."
    sleep 1
done

echo "UNABLE TO ACCESS RTC"
sudo date

exit $ex_status


```

Appendix 5: Time.sh

cpu.sh

```
#!/bin/bash



counter=0
num_threads=4
prime_num=10000
time=180
minutes=$((time / 60))
while [ "$counter" -lt 2 ]
do
    printf "RUNNING CPU TEST \nGENERATING PRIME NUMBERS FOR %s  MINUTES\n" "$minutes"
    sudo sysbench cpu --cpu-max-prime=$prime_num --threads=$num_threads --time="$time" run
    ex_status=$?
    if [ "$ex_status" -eq 0 ]
    then
        echo "CPU TEST COMPLETE"
        exit $ex_status
    fi
    echo "ERROR RUNNING CPU TEST"
    echo "RETRYING ... "
done

echo "UNABLE TO RUN CPU TEST"

exit $ex_status



```

Appendix 6:CPU.sh

memory.sh

```
#!/bin/bash


size=300

while [ "$size" -gt 50 ]
do
    printf "RUNNING MEMORY TEST FOR %s MB\n" "$size"
    sudo memtester $size 1
    ret_Val=$?
    if [ "$ret_Val" -eq 0 ]
    then
        ex_status=0
        echo "MEMORY TEST COMPLETE"
        exit $ex_status
    fi
    echo "ERROR RUNNING MEMORY TEST"
    size=$((size - 100))
    echo "RETRYING ... "
done

echo "UNABLE TO RUN MEMORY TEST"

exit $ex_status
```

Appendix 7: Memory.sh

Appendix 8: Storage.sh

storage.sh

```
#!/bin/bash



counter=0
size=254
time=180

sysbench fileio cleanup

sleep 10

while [ "$counter" -lt 3 ]
do
    printf "RUNNING eMMC TEST FOR 3 MINUTES \n"
    printf "\nCREATING TEST FILES...\n"
    sysbench fileio --threads=16 --file-total-size="$size"M --file-num=32 --file-test-mode=rndrw --validate --verbosity=2 prepare
    ex_status=$?
    if [ "$ex_status" -eq 0 ]
    then
        sysbench fileio --threads=16 --file-total-size="$size"M --file-num=32 --file-test-mode=rndrw --time="$time"  --validate run
        ex_status=$?
        if [ "$ex_status" -eq 0 ]
        then
        break
        fi
    fi
    echo "ERROR RUNNING eMMC TEST"
    echo "RETRYING ... "
    sysbench fileio --threads=16 --file-total-size="$size"M cleanup
    size=$((size / 2))
done

if [ "$ex_status" -ne 0 ]
then
echo "UNABLE TO RUN eMMC TEST"
fi

sysbench fileio --threads=16 --file-total-size="$size"M cleanup

exit $ex_status



```

GPU.sh

```
#!/bin/bash

ex_status=0

while [ $ex_status -le 2 ]
do
        ((ex_status++))
        export DISPLAY=:0
        export vblank_mode=0
        echo "RUNNING GPU TEST - 4 SAMPLES FOR 3 MINUTES"
        glxinfo -B
        timeout 181s glxgears -samples 4
        ret_Val=$?
        if [ "$ret_Val" -eq 124 ]
        then
                ex_status=0
                echo "GPU TEST COMPLETE"
                exit $ex_status
        fi
        echo "ERROR RUNNING GPU TEST"
        echo "RETRYING..."
        sleep 1
done

echo "UNABLE TO RUN GPU TEST"
exit $ex_status


```

Appendix 9 : GPU.sh

camera.sh

```
1  #!/bin/bash
2
3  test=$1
4
5  sleep 30
6
7  sudo fswebcam -r 1280x720 ~/IMAGES/image_"$test".jpg
8  sudo fswebcam -r 1280x720 ~/IMAGES/image_"$test"a.jpg
9  sudo fswebcam -r 1280x720 ~/IMAGES/image_"$test"b.jpg
10
11
12 sudo cp ~/IMAGES/image_"$test".jpg ~/USBdrv/IMAGES
13 sudo cp ~/IMAGES/image_"$test"a.jpg ~/USBdrv/IMAGES
14 sudo cp ~/IMAGES/image_"$test"b.jpg ~/USBdrv/IMAGES
15
16 sudo cp ~/IMAGES/image_"$test".jpg ~/SPI_LOG/IMAGES
17 sudo cp ~/IMAGES/image_"$test"a.jpg ~/SPI_LOG/IMAGES
18 sudo cp ~/IMAGES/image_"$test"b.jpg ~/SPI_LOG/IMAGES
19
20
21
22
```

Appendix 10: Camera.sh

test.sh

```
#!/bin/bash

test_no=$1

printf "RUNNING AUTOMATED INTERFACE TEST NO: %s\n" "$test_no"

./SCRIPTS/time.sh

i2cdetect -y 1

./SCRIPTS/time.sh

./SCRIPTS/cpu.sh

./SCRIPTS/time.sh

./SCRIPTS/memory.sh

./SCRIPTS/time.sh

./SCRIPTS/storage.sh

./SCRIPTS/time.sh

./SCRIPTS/GPU.sh

./SCRIPTS/time.sh

./SCRIPTS/camera.sh $test_no

./SCRIPTS/time.sh

printf "\nEND OF TEST %s\n" "$test_no"


exit 0

```

Appendix 11: Test.sh

Appendix 12: Test_main.sh

TEST_main.sh

```
#!/bin/bash
sleep 30

test_no=1

sudo mount /dev/sda1 USBdrv -o uid=pi,gid=pi

sudo mount.cifs //169.254.59.251/SPI_LOG /home/pi/SPI_LOG/ -o user=gat19_94@hotmail.com,password=[redacted]

while [ -e USBdrv/LOGS/TEST_OUTPUT_"$test_no".txt ]
do
        ((test_no++))

done
while [ -e SPI_LOG/TEST_OUTPUT_"$test_no".txt ]
do
        ((test_no++))

done


printf "\nOUTPUT FILE:  TEST_OUTPUT_%s\n" "$test_no"


echo -e "\ LISTING BOOT MESSAGE: \n " |& sudo tee -a USBdrv/LOGS/TEST_OUTPUT_"$test_no".txt SPI_LOG/TEST_OUTPUT_"$test_no".txt
./SCRIPTS/time.sh |& sudo tee -a USBdrv/LOGS/TEST_OUTPUT_"$test_no".txt SPI_LOG/TEST_OUTPUT_"$test_no".txt
sudo dmesg |& sudo tee -a USBdrv/LOGS/TEST_OUTPUT_"$test_no".txt SPI_LOG/TEST_OUTPUT_"$test_no".txt
sudo cat /var/log/boot.log |& sudo tee -a USBdrv/LOGS/TEST_OUTPUT_"$test_no".txt SPI_LOG/TEST_OUTPUT_"$test_no".txt

./SCRIPTS/test.sh $test_no |& sudo tee -a USBdrv/LOGS/TEST_OUTPUT_"$test_no".txt SPI_LOG/TEST_OUTPUT_"$test_no".txt

sudo umount USBdrv
sudo umount SPI_LOG

echo -e "\ SLEEPING FOR 30 MINUTES \n"


sleep 1800

./SCRIPTS/time.sh

sudo reboot

exit 0


```

Char(4),Char(5),Char(6),Char(7),Char(8) are the display digits and are translated as follows:

```
Ch=#252 then Ch = ‘0’
Ch=#96   then Ch = '1'
Ch=#218 then Ch = '2'
Ch=#242 then Ch = '3'
Ch=#102 then Ch = '4'
Ch=#182 then Ch = '5'
Ch=#190 then Ch = '6'
Ch=#224 then Ch = '7'
Ch=#254 then Ch = '8'
Ch=#230 then Ch = '9'
Ch=#238 then Ch = 'A'
Ch=#156 then Ch = 'C'
Ch=#122 then Ch = 'D'
Ch=#158 then Ch = 'E'
Ch=#142 then Ch = 'F'

Ch=#140 then Ch = 'R'
Ch=#30   then Ch = 'T'
Ch=#124 then Ch = 'U'
Ch=#28   then Ch = 'L'
Ch=#0     then Ch = ' '
Ch=#2     then Ch =' '
```

Appendix 13: Decimal Representation

HEX to Reading Value converter

```
#include <stdio.h>

#include <math.h>
#include <string.h>
long long hexconv (char*);
void voltage (long long);

int main()
{
    FILE * pFile;
    char mystring [3];
    long long value;
    char valid[3] ={"0D"};
    int stop = 0;

    pFile = fopen ("test.txt" , "r");
   if (pFile == NULL) perror ("Error opening file");
   else {

        if (fgets (mystring , 3 , pFile) == NULL)
          stop = 1;

     while( stop != 1 ){
            if (strcmp(mystring,valid) == 0){
                 if (fgets (mystring , 3 , pFile) == NULL)
                    stop = 1;
                while (strcmp(mystring,valid) != 0 && stop != 1){
                    value = hexconv(mystring);
                    voltage(value);
                    if (fgets (mystring , 3 , pFile) == NULL)
                        stop = 1;
                }
                printf("\n");
            }

     }
   }
 fclose (pFile);


}
```

```
long long hexconv(char* hex) {
    /*char hex[17];*/
    long long decimal, place;
    int i = 0, val, len;

    decimal = 0;
    place = 1;

    /* Input hexadecimal number from user */
    /*printf("Enter any hexadecimal number: ");
    gets(hex);*/

    /* Find the length of total number of hex digit */
    len = strlen(hex);
    len--;

    /*
     * Iterate over each hex digit
     */
    for(i=0; hex[i]!='\0'; i++)
    {

        /* Find the decimal representation of hex[i] */
        if(hex[i]>='0' && hex[i]<='9')
        {
            val = hex[i] - 48;
        }
        else if(hex[i]>='a' && hex[i]<='f')
        {
            val = hex[i] - 97 + 10;
        }
        else if(hex[i]>='A' && hex[i]<='F')
        {
            val = hex[i] - 65 + 10;
        }

        decimal += val * pow(16, len);
        len--;
    }

    return decimal;
}
```

```
void voltage (long long decimal){
    switch (decimal){
    case 252 :
        printf("0"); break;
    case 253:
        printf("0."); break;
    case 96 :
        printf("1"); break;
    case 97 :
        printf("1."); break;
    case 218 :
        printf("2"); break;
    case 219 :
        printf("2."); break;
    case 242 :
        printf("3"); break;
    case 243 :
        printf("3."); break;
    case 102 :
        printf("4"); break;
    case 103 :
        printf("4."); break;
    case 182 :
        printf("5"); break;
    case 183 :
        printf("5."); break;
    case 190 :
        printf("6"); break;
    case 191 :

printf("6.");
   case 224 :
        printf("7");
   case 225 :

printf("7.");
   case 254 :
        printf("8");
    case 255 :

printf("8.");
    case 230 :
        printf("9");
        break;
    case 231 :

printf("9.");
        break;

    defaut: break;

    }
}
```

Appendix 14: Conversion software