Progress for the Design and Testing of Photon-based Hardware Random Number Generator

Nikolas Thornton*, Dmitriy Beznosko, Keith Driscoll, Fernando Guadarrama , Steven Mai

March 27, 2024, CIMS SYMPOSIUM

*presenter, NikolasThornton@clayton.edu

INTRODUCTION

Hardware random number generators (HRNG) are widely used in the computer world for the security purposes as well as in the science world as a source of the high-quality randomness for the models and simulations. Currently existing HRNG are either costly or very slow and of questionable quality. This work proposes a simple design of the HRNG based on the low-number photon absorption by a detector (a photo-multiplier tube of a silicon-based photodetector) that can provide a large volume of high-quality random numbers [1]. The different options of processing and the testing of quality of the generator output are presented.

PSEUDO-RANDOM NUMBER GENERATORS

Seed – a number that PRNGs use to derive their randomness.

Derived from unpredictable or difficult to replicate factors, such as the user's mouse movement or the machine's uptime.

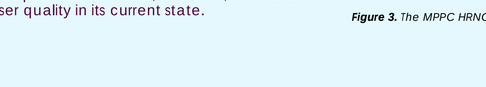


Figure 1. *The Minecraft seed input for creatin a new world*

Pros:

- Easily replicable results
- No external hardware necessary

Cons:

- Easily replicable results
- Not random enough for many use cases

PHOTON-BASED SOLUTION TO RANDOM NUMBER GENERATION

Photomultiplier Tube Solution

A photomultiplier tube (PMT) detects photons in sync with an LED pulse from an external trigger and measures their amplitudes as shown in Figure 2. The main downside of this setup is that it requires a blackout-box for the PMT and the LED to be cased in. The PMT is also only capable of detecting 100 amplitudes per second.

Figure 2. *The PMT HRNG setup*

Multi-Pixel Photon Counter

A Multi-Pixel Photon Counter (MPPC) [2] is connected to an Arduino and detects photons in sync with an LED pulse from an internal trigger, but only as a prototype setup with some drawbacks. The LED is powered by the Arduino's default Pulse Width Modulation (PWM) solution to low-voltage output. The sensor also needs a proper enclosure, currently a dark tube is attached to it, and the LED is taped to the end.

This prototype setup has shown enormous potential despite the current setup, generating about 6000 random amplitudes per minute, which contrasts starkly with the PMT's 100 per amplitudes per second. As will be shown in Experimental Results, however, the data is arguable of lesser quality in its current state.

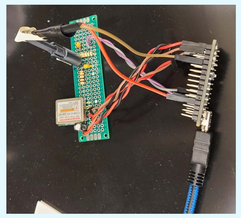


Figure 3. *The MPPC HRNG setup*

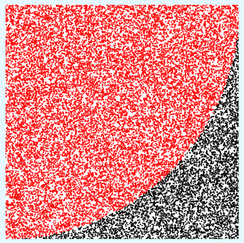
TESTING METHODS

Monte Carlo Pi Estimation Test

The Monte Carlo Pi Estimation Test (MCPE Test) is done by randomly selecting points within a square and counting each time a point falls within the quarter-cross-section of a circle, and then dividing that total by the total number of points everywhere, pi can be estimated, as shown in Figure 4.

By turning every 8 bits into a byte, turning every byte into a float between 1 and 0, and then pairing every byte to form a coordinate, they are processed as shown in the pseudo code in Figure 5.

The usefulness of this test relies on having enough coordinates to accurately estimate pi in the first place. To aid in this, pi is calculated the same way using Python's Random library using the same number of coordinates as the bitstream. Comparing both helps determine whether a poor pi calculation is due to too few coordinates or poor quality data.



$$4 \times \frac{RED\ DOTS}{ALL\ DOTS} = \sim 3.14159265\ldots$$

Figure 4. *Monte Carlo Pi Estimation*

```
all_dots += 1
if X^2 + Y^2 <= 1: red_dots += 1
pi = 4*(red_dots/all_dots)
```

Figure 5. *Monte Carlo Pi calculation pseudo-code*

Fractional Line Symmetry Test

The Fractional Line Symmetry Test, or FLS Test, is a test ***developed specifically for this project*** that compares how frequently bits appear back-to-back horizontally and vertically when stacked and visualized. The folding done to stack and visualize the data is naturally random, therefore the number of lines found horizontally and vertically should be relatively equal. This test is sensitive to poor quality data that otherwise passes tests like the standard deviation and average of the bitstream.

Figure 6 is an example of a bitstream that has been stacked left-to-right. The grey bits at the end are "nothing bits" that are added only to make the bitstream long enough to form a perfect square

0	0	1	0
1	1	1	0
0	1	0	1
0	1		

Figure 6. *Bitstream stacking and visualization*

Figure 7 is an example of counting horizontal lines with a detect length of 2 for zeros. All lines are highlighted in blue. This means that the image is read left-to-right for all instances of 2 back-to-back black pixels. ***Notice that lines are counted even if they wrap around the borders of the image.***

0	0	1	0
1	1	1	0
0	1	0	1
0	1		

Figure 7. *Horizontal line counting searching for zeros with a detect length of 2*

Figure 8 is an example of counting vertical lines with the same detect length of 2 for zeros. All lines are highlighted in red. In this case, the image is searched top to bottom, starting from the right-most column and moving to the left. ***Notice that the "nothing bits" are ignored if they interrupt a line that wraps around the borders of the image.***

0	0	1	0
1	1	1	0
0	1	0	1
0	1		

Figure 8. *Vertical line counting searching for zeros with a detect length of 2*

The FLS Test also counts fractional lines as well. That means if there are additional bits attached to an otherwise complete line, then they are added fractionally to the line total. For example, with a detect length of 2 searching for zeros, the bitstream in Figure 9 has a total of 1.5 horizontal lines.

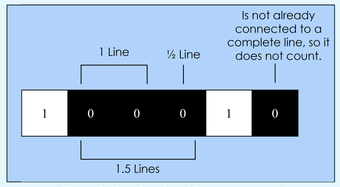


Figure 9. *How fractional lines are counted*

The number of lines that are expected to appear for any size bitstream can be estimated using the equation in Figure 10.

There is not yet a full mathematical proof for this equation, as it was derived through the observation of clear patterns between the number of lines found across various detect lengths for a large bitstream.

$$\frac{n+1}{(2^{n+1})n} \times bitstream\ length$$

Where n is the detect length

Figure 10. *Equation used to estimate the number of lines that should be found by the FLS test*

PROCESSING AMPLITUDES

High/Low

```
if amp[i] > amp[i-1]: bitstream += 1
if amp[i] < amp[i-1]: bitstream += 0
else: skip
```

Figure 11. *High/Low amplitude processing pseudo-code*

Even/Odd

```
if amp[i] % 2 == 0: bitstream +=1
else: bitstream += 0
```

Figure 12. *Even/Odd amplitude processing pseudo-code*

Von Neumann's Procedure

For the coin:
If $P(heads) = x$ then $P(tails) = 1 - x$ and in general $x \neq 1 - x$
Look at pairs: heads + tails (ht) and tails + heads (th). Probability is a product of individual ones.
$P(ht) = x(1 - x)$ and $P(th) = (1 - x)x$
$P(ht) = P(th)$, QED

EXPERIMENTAL RESULTS

PMT Setup

Using 290,000 amplitudes

High/Low

```
Monte Carlo Pi Estimation:
PRE-NEUMANN:  3.45441
POST-NEUMANN: 3.09256

Average:
PRE-NEUMANN:  0.49992
POST-NEUMANN: 0.49872

Standard Deviation:
PRE-NEUMANN:  0.4999999942
POST-NEUMANN: 0.4999983711
```

Even/Odd

```
Monte Carlo Pi Estimation:
PRE-NEUMANN:  3.13644
POST-NEUMANN: 3.14502

Average:
PRE-NEUMANN:  0.49946
POST-NEUMANN: 0.49947

Standard Deviation:
PRE-NEUMANN:  0.499999711
POST-NEUMANN: 0.499999723
```

The FLS Test

Detect length of **4**.

Estimated line count for 290,000 bits using Figure 10: **11328.125**

High/Low

```
Horizontal lines:
PRE-NEUMANN:  2042.25
POST-NEUMANN: 11981.25

Vertical lines:
PRE-NEUMANN:  11359.75
POST-NEUMANN: 10495.25
```

Even/Odd

```
Horizontal lines:
PRE-NEUMANN:  11337.5
POST-NEUMANN: 10990.0

Vertical lines:
PRE-NEUMANN:  11291.75
POST-NEUMANN: 11006.25
```

Figure 13. *Sample of high/low pre-Neumann processed PMT horizontal lines*

Figure 14. *Sample of even/odd pre-Neumann processed PMT horizontal lines*

MPPC Setup

Using 290,000 amplitudes

High/Low

```
Monte Carlo Pi Estimation:
PRE-NEUMANN:  3.47953
POST-NEUMANN: 3.07141

Average:
PRE-NEUMANN:  0.49971
POST-NEUMANN: 0.50052

Standard Deviation:
PRE-NEUMANN:  0.4999999136
POST-NEUMANN: 0.4999997340
```

Even/Odd

```
Monte Carlo Pi Estimation:
PRE-NEUMANN:  3.08083
POST-NEUMANN: 3.11290

Average:
PRE-NEUMANN:  0.50791
POST-NEUMANN: 0.50059

Standard Deviation:
PRE-NEUMANN:  0.499937370
POST-NEUMANN: 0.499999653
```

The FLS Test

Detect length of **4**.

Estimated line count for 290,000 bits using Figure 10: **11328.125**

High/Low

```
Horizontal lines:
PRE-NEUMANN:  1415.0
POST-NEUMANN: 11143.75

Vertical lines:
PRE-NEUMANN:  10563.5
POST-NEUMANN: 9683.25
```

Even/Odd

```
Horizontal lines:
PRE-NEUMANN:  10647.5
POST-NEUMANN: 11004.75

Vertical lines:
PRE-NEUMANN:  10545.75
POST-NEUMANN: 11002.5
```

Figure 15. *Sample of high/low pre-Neumann processed MPPC vertical lines*

Figure 16. *Sample of even/odd pre-Neumann processed MPPC vertical lines*

CONCLUSION

The photon-based solutions explored in this research have shown great potential for being high-speed sources of high-quality random number generation.

It was found as shown in Experimental Results that the current best way to process data is the even/odd method shown in Figure 12. The PMT appears to outperform the MPPC in the MCPI Test, but the MPPC is still in its prototype stage.

By use of the FLS Test, it was revealed that the high/low method of processing is far worse than the even/odd method. The FLS Test successfully revealed this in spite of the high/low algorithm otherwise passing the other tests relatively well.

Current objectives for the future of this project are to sync the MPPC Arduino with an LED that is voltage controlled through means besides PWM. A proper enclosure for the board and sensor is also necessary, as well as a faster board to increase the number of amplitudes per second.

References

[1] D. Beznosko et al., "Random Number Hardware Generator Using Geiger-Mode Avalanche Photo Detector", e -Print: 1501.05521, DOI: 10.22323/1.252.0049, PoS PhotoDet2015 (2016), 049

[2] Hamamatsu Photonics K.K., 325-6, Sunayama-cho, Naka-ku, Hamamatsu City, Shizuoka Pref., 430-8587, Japan