UNIVERSITY OF CAMBRIDGE
Department of Engineering

Compiling Physical Invariants to Hardware for a Secure & Private Sensor Interface

Author Name: Gregory Brooks

Supervisor: Phillip Stanley-Marbell

Date:

I hereby declare that, except where specifically indicated, the work submitted herin is my own original work.

Signed _____ *date* _____

IIB Project Report: Compiling Physical Invariants to Hardware for a Secure & Private Sensor Interface

Gregory Brooks, gb510, Christ's College

29 May 2019

Contents

1 Technical Abstract

Compiling Physical Invariants to Hardware for a Secure & Private Sensor Interface

Gregory Brooks, gb510, Christ's College

⟨ *TODO: write this last*⟩

This project investigates the idea of using the Newton [1] physics description language as part of a multi-sensor embedded local differential privacy system implemented on an iCE40 FPGA. ⟨ *TODO: tidy up the above and expand on it if necessary*⟩

1.1 System Specification

⟨ *TODO: move to technical abstract?*⟩

1.2 System Diagram

⟨ *TODO: insert block diagram of system and describe which parts have been the focus of this project*⟩

2 Introduction and Background

2.1 Differential Privacy

Differential privacy can be thought of as a constraint applied to queries for information from a database whereby information about individual members of the database is obscured whilst still releasing useful aggregate information about a population/demographic as a whole. A simple method of implementing this would be the addition of zero mean noise to database entries — whilst the noise would obscure the *true* value of individual data points, the mean value of the dataset would remain intact. More formally, ϵ-*differential privacy* is defined by Dwork and Smith [2] as follows:

"A randomized function K gives ϵ-*differential privacy* if for all data sets x and x' differing on at most one element, and all $S \subseteq Range(K)$,

$$Pr[K(x) \in S] \leq exp(\epsilon) \times Pr[K(x') \in S], \tag{1}$$

where the probability space in each case is over the coin flips of the mechanism K."

Throughout this project, the chosen randomised function K is the Laplace mechanism i.e. the addition of zero mean Laplace distributed random noise to *private* data such as a measurement from a sensor[1] to produce a *noised output* that can be released to untrusted observers (the outside world). This technique provides privacy by ensuring that every possible measurement value (i.e. any value within the sensor measurement range) has a similar posterior probability of being the *true private* value given the observed *noised output*. An outside observer can use the *noised output* to estimate the value of *private* data with some uncertainty but should never be able to deduce it with complete certainty.

The Laplace distribution used by this method has parameter $\frac{\lambda}{\epsilon}$. λ refers to the *global sensitivity* of the database query function; in this scenario this is simply the maximum difference between possible sensor measurements i.e. a sensor's measurement range. ϵ is a privacy scaling parameter, where a smaller ϵ value results in greater privacy by applying noise with a greater variance, thereby reducing the amount of information each *noised output* value reveals about the *private* data. For a particular application, deciding on an ϵ value is

[1] The *databases* in this scenario are simply single sensor measurements hence any two *databases* will always differ by at most a single element.

a tradeoff between utility and privacy since, in the limiting case, an ϵ value of zero means that the system outputs no useful information (just random noise), consequently keeping the *private* data completely private.

Within the context of embedded electronic systems, recent work (for example, this project builds heavily on the work of Choi et al. [3]) has investigated the implementation of differential privacy techniques on low power hardware such as the processors found in smartphones and Internet-of-Things devices. In contrast to the conventional differential privacy model, where a trusted database stores *private* data collected from local hardware, a local differential privacy architecture involves *private* data being masked at the source (e.g. an embedded sensor system) before being sent to an untrusted database [4]. This local approach can result in a more secure system since *private* data does not need to be recorded anywhere nor transmitted from local hardware to a remote server.

One of the key advantages of generating random noise with known parameters is that it allows a quantitative *privacy loss* [3] to be calculated for each piece of information revealed to the outside world (e.g. with the release of each *noised output* value). Privacy losses can be deducted from a total *privacy budget* allocation — once the budget has been depleted, a differential privacy system will not release further information to the outside world. An intuitive justification for such a system comes from the fact that if an attacker were allowed to obtain an unlimited number of noised values, they would be able to take the mean of these responses to gain an arbitrarily accurate estimate for the private data value being masked by zero-mean random noise.

One challenge associated with implementing local differential privacy is that the randomised function K used to mask *private* data can be difficult to implement on low power hardware — one of the findings presented by Choi et al. [3] is that a finite precision implementation of the Laplace mechanism can lead to infinite privacy loss, since certain specific values for *noised output* data can reveal the *true* values for *private* unnoised data with complete certainty. This problem is most apparent on low power hardware utilising low precision fixed point arithmetic. This problem can be overcome by ensuring *noised output* remains within a safe range so that privacy loss can never be infinite; this can be achieved by truncating values if they fall outside the range or by resampling the random noise if it would result in a *noised output* value outside the acceptable range [3].

Existing research regarding embedded systems has focused on single sensor systems [3]. Introducing multiple related sensors introduces complications when quantifying privacy loss, since releasing a noised value of one particular measurement can indirectly reveal information about others. For example, if a particular embedded system contains a GPS and accelerometer, accelerometer measurements can be integrated over time to gain an estimate for GPS position — some privacy loss quantity would have to be deducted from the privacy budget allocated to the GPS as a result of releasing noised acceleration values. One of the aims of this project was to analyse this problem in detail — this work is presented in Section 3.2, with detailed mathematical analysis in Appendix B.

2.2 The Newton Language

Newton is a physics description language developed by Jonathan Lim and Phillip Stanley-Marbell [1]. A Newton description can specify dimensionally annotated *signals* i.e. physical quantities such as measurements from a sensor, and invariants/constraints relating these signals (such as the physical laws governing a particular system). Figures 1 and 2 and Equation 2 illustrate a basic example of the langage being used to describe the physical resonance of a circuit board.

```
include "NewtonBaseSignals.nt"
board : invariant(Q:dimensionless, d:distance, f:frequency, a:acceleration, t:time) = {
  a ~ Q*d*(2*kNewtonUnitfree_pi*f)**2*((2*kNewtonUnitfree_pi*f*t)
      - (2*kNewtonUnitfree_pi*f*t)**3/6 + (2*kNewtonUnitfree_pi*f*t)**5/120)
}
```

Figure 1: Example of a Newton description for a physically resonating circuit board (modelled in two dimensions) governed by Equation 2. Figure sourced from a poster produced in parallel to this project [5].

$$a = Q_{bd} d_{bd} \omega^2 sin(\omega t), \tag{2}$$

where:

- a represents board acceleration,
- Q_{bd} represents quality factor,
- d_{bd} represents board displacement amplitude,

- ω represents angular resonant frequency,
- t represents time.

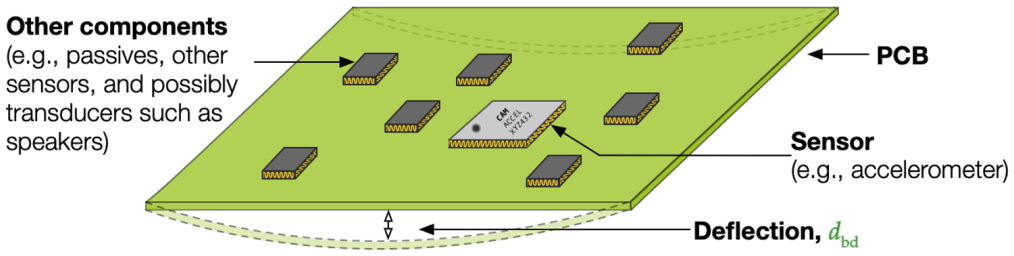


Figure 2: The system modelled by Equation 2. Figure sourced from a poster produced in parallel to this project [5].

At the time of writing the Newton language and compiler is still under development by the Physical Computation Laboratory [6]; the Newton compiler frontend is able to transform a human-readable Newton description into an intermediate representation, a tree structure that can be easily traversed by a computer. The motivation for this project was the idea that this information could be used by a compiler backend to generate a hardware description (e.g. Verilog).

Within the context of this project, Newton provides the ability to describe the relationships between the sensors in a multi-sensor embedded system. This information is necessary to be able to quantify privacy losses in a multi-sensor system.

2.3 Random Number Generation

One important component of a local differential privacy system is a true random number generator (TRNG) to generate the random noise used to obscure *private* data[2]. The Laplace mechanism requires random noise to be Laplace distributed, introducing additional complexity compared to uniform or Gaussian distributed RNGs. The resource constraints imposed by the iCE40 FPGA dictated the need to implement a hardware efficient RNG to generate this Laplace distributed noise. A research article by De Schryver et al. [7] outlines a possible architecture for such a system, based on a lookup table indexed by a floating point representation of the output of a uniform random number generator. This architecture was implemented in Verilog and analysed as part of this project (see Section 4.5).

[2]In a security related application, pseudorandom number generators such as linear feedback shift registers are unsuitable since an attacker would be able to predict their output

3 Theoretical Development

The early stages of the project involved brainstorming and investigating potential research problems that would involve compiling Newton descriptions into a hardware description language. It soon became apparent that the Newton language was well suited to describing the relationships between sensors in a multi-sensor embedded system, thereby providing a basis to build upon the single-sensor state-of-the-art [3] [8].

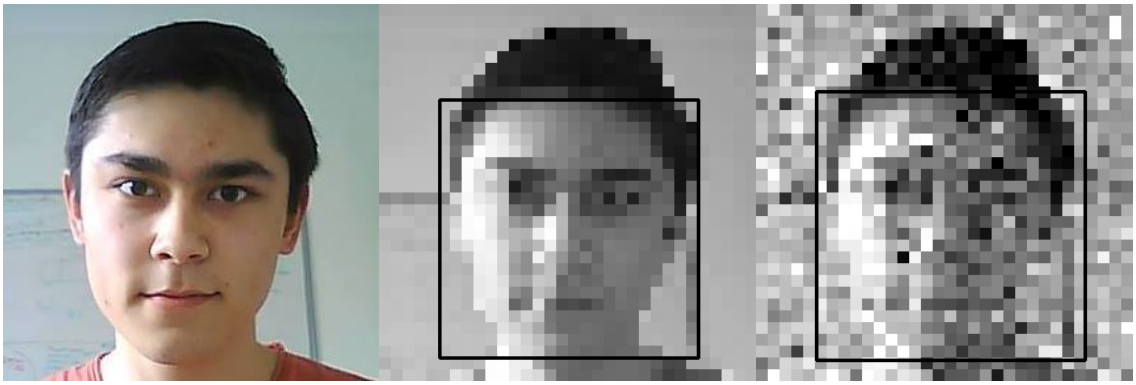
3.1 Brainstorming Example Applications

Below are some examples of security/privacy related applications of compiling a Newton description into Verilog that eventually led to the project's focus on differential privacy for multi-sensor systems.

3.1.1 Digital Camera/Viola-Jones Facial Detection

iCE40 FPGAs are able to drive digital cameras and perform facial recognition, as illustrated by the iCE40 Ultraplus Mobile Development Platform [9]. In this context a differential privacy system might aim to apply random noise to an image in such a way that the identity of an individual can be obscured whilst still preserving enough information for facial detection algorithms to function correctly. A Python script available in this report's GitHub repository[3] demonstrates this idea using OpenCV [10] — the script converts a photo of a face to greyscale and subsamples to 32x32 size (effectively low pass filtering to remove high spatial frequencies) before Laplace distributed random noise ($\lambda = 25$, mean $= 0$) is added to the pixel intensity values, similarly to the Laplace method described in Section 2.1. The script then uses a Viola-Jones [11] classifier to detect faces in the images before and after the application of noise, demonstrating that a face can be detected in both cases provided the variance of the noise distribution is not too great to drown out all relevant information (analogous to the ϵ value in Section 2.1 being too small). The results of running this script can be observed in Figure 3; the addition of random noise somewhat obscures the identity of the subject in the photo whilst still allowing a Viola-Jones facial detection to function. The demonstration featured here is a simple proof-of-concept; random distortion could be applied to the image in many different ways e.g. random displacement of pixels.

[3] See Section 5.1.



(a) Original photo (b) Subsampled greyscale image (c) Random noise added

Figure 3: Proof-of-concept demonstration of Laplace distributed random noise being used to obscure an individual's identity whilst preserving a detectable facial structure. A square has been drawn around the faces detected by the pre-trained Viola-Jones classifier bundled with OpenCV. The photo features, and was taken by the author.

The Viola-Jones classifier works by applying rectangular filters to detect facial features. A Newton description could contain invariants to describe these filters in terms of pixel values; this information could, for example, be used by a noising system to ensure that random noise is applied in a way that does not significantly impact an image's response to the facial detection filters (i.e. if an image contains a face, then the Viola-Jones classifier described in Newton should still detect the face after the noising process, and vice versa if no face is present in the un-noised *private* image).

3.1.2 Microphone

In a similar manner to the camera example above, noise could be added to audio data in such a way that a person's voice can be masked whilst still preserving features, such as mel-frequency cepstral coefficients, that allow speech recognition algorithms to function. The Newton language would be used to describe invariants in the time and/or frequency domains that must be held during the noising process.

3.1.3 *Intelligent* Noising

This project investigates a system which allows a user to specify which measurement signals they wish to add random noise to. It is possible to generalise this idea to a hypothetical system which has control over how it applies random noise to measurement data in order to mask

a more abstract form of information. One concrete example of this would be when working with images (e.g. the Viola-Jones example above) — given the requirement that individuals' identities should be obscured whilst still providing enough information for facial detection algorithms to work reliably, a system might choose to apply noise using, for example:

- Random noise added to pixel intensities.
- Convolution with a random filter.
- Random rearrangement of pixels.
- A combination of multiple techniques.

For such a system, a Newton description could be used to define constraints on the noised image's response to the Haar-like features used by a particular Viola-Jones facial detection algorithm; the system would then randomise the image in such a way that the constraints are not violated i.e. both the unnoised and noised images exhibit similar features, producing similar results when the face detection algorithm is applied.

3.2 Multi-Sensor Differential Privacy Loss

4 System Design and Implementation

4.1 Privacy Budget Management System Architecture

The following section proposes a privacy management system architecture that could be used to manage multi-sensor privacy on an FPGA. Due to project time constraints, the system has not been completely implemented, since this would divert time and attention away from investigating the more important research questions posed by the project. The progress made so far by the author is documented in sections 5 and 6.
⟨ *TODO: describe what will have been implemented by the submission date e.g. use privacy yaml file and Newton AST to generate some of the logic implementation by filling in a Verilog template*⟩
The proposed system architecture is as follows:

- The FPGA acts as a sensor interface, allowing external circuitry to request noised sensor values. Noised measurement signals are referred to as *protected*, as their true value has been masked by the addition of zero-mean Laplace distributed random noise

to preseve privacy; the system also allows *unprotected* signals to be forwarded to the outside world with no random noise added.

- A *privacy file* (e.g. a YAML file) is used to specify protected measurement signals for a particular embedded system.
- In order to apply random noise to a particular signal, a range of parameters must be defined for each protected signal:
  - Range of possible sensor outputs.
  - *Privacy budget* value allocated to the sensor.
  - *Privacy budget* replenishment rate.
  - Value of ϵ, a measure of privacy where a smaller value results in greater privacy by increasing the variance of applied noise [3].

  These values can be provided for each protected signal within the *privacy file*.
- The FPGA maintains a differential privacy *budget* for each protected signal. Querying a value for a protected signal (i.e. requesting a noised measurement from the FPGA) causes that signal to incur a privacy loss. The FPGA quantifies this loss (a function of the random noise sample added to the raw measurement [3]) and subtracts it from the signal's privacy budget. Once the budget is depleted, further queries of the signal in question are ignored — a nonzero budget replenishment rate can be specified in the privacy file so that the signal can eventually be queried again once the budget has replenished sufficiently.
- Invariants in the Newton description define relationships between signals. This is important as privacy is lost if a measurand's value is calculated by taking measurements from related sensors, without directly measuring the measurand. The system must account for this *indirect* privacy loss:
  - Each Newton invariant containing protected measurement signals has an associated bitfield register on the FPGA. Each bit in the register acts as a read flag for each measurement signal contained in the associated invariant — the flag is set when the sensor is queried (i.e. when a measurement is taken and the noised measurement value becomes *known* to the outside world) and is not cleared until the sensor's privacy budget is completely replenished some time after a query (at

which point the measurement value can be considered *unknown* to the outside world).

  - As long as any two flags remain unset, there are two variables in the equation that are *unknown* to an attacker. The equation cannot be solved and so no information[4] can be gained about the *unknown* values.
  - If only one bit remains unset, then the remaining *unknown* value can be calculated using the other *known* values with set flags — the *unknown* signal incurs a privacy loss. If all flag bits are set then responding to a measurement query results in a privacy loss for every other signal in the invariant. These scenarios can be summarised by saying that, for signals contained within a particular Newton invariant, a signal experiences an indirect privacy loss if any of the other signals in the invariant are queried *and* all other signals are *known* i.e. have been recently queried.
  - If responding to a query would exceed the privacy budget for any protected signal (not just the one being queried) then a response should not be granted.

- The mathematics behind the calculation of a quantitative value for privacy loss is investigated in Appendix B; whilst calculation of an exact value for privacy loss (Equation 6 in Appendix B) requires the use of a mathematical optimisation technique such as Lagrange multipliers, a simpler calculation can be used to obtain an upper bound on the value. This simpler calculation can be performed more easily on a small FPGA such as the iCE40, and can even be pre-calculated during compile-time since it is a function of the random noise output only.

4.2 FPGA

This system is based on the iCE40 UP5K FPGA [12], selected for its small size and low power consumption. These qualities make it ideal for inserting into an embedded device without significantly impacting the size or power consumption/battery life. Using FPGA hardware rather than a software implementation running on a processor has several advantages:

- Increased speed, to the point where data rate is limited by the SPI/I2C bus clock rather than the privacy system.

[4] This assumes that the probability distributions of sensor outputs are independent, see Appendix B for an investigation into correlated measurements.

- Increased energy efficiency.
- Potential for integration into a sensor's package (due to the FPGAs small size).
- Security - a hardware implementation is difficult for an attacker to manipulate or modify.

4.3 Hardware Entropy Source

As discussed in section 2.3, one of the key components of a security/privacy application is a true random number generator (TRNG). Unlike a pseudorandom number generator (PRNG), which produces a predictable deterministic outcome, a true random number generator's output cannot be anticipated by an attacker, preserving system security and users' privacy. The output bit rate of TRNGs is often limited so can be used to seed a cryptographically secure PRNG (CSPRNG) algorithm to increase output bit rate without significantly compromising security. This extra step was not required for this project, since the rate at which random numbers are consumed is relatively low.

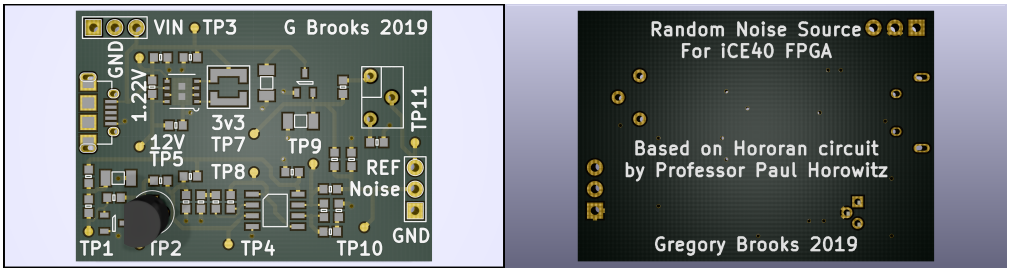
Initial investigations focused on using the iCE40 FPGA's differential I/O hardware to generate random bits which could be fed into a TRNG to generate random samples from an arbitrary distribution. Initially, it was hoped that simply leaving two comparator inputs floating would cause the output to fluctuate randomly — in practice this did not occur as the comparator hardware requires one of the inputs to be biased within a narrow common mode voltage range (roughly half the I/O supply voltage, allowing the other input to swing about this reference voltage) [13]. The author decided to design a PCB to set this bias voltage, as well as provide an alternative entropy source in the form of a reverse biased avalanche diode producing avalanche noise. In theory, this circuit would provide a better i.e. less predictable source of entropy since, unlike the floating input pin, the avalanche noise (and therefore random number output) produced is independent of the device's external environment[5].

The circuit used in this noise generator is based on a design by Professor Paul Horowitz [14, p. 984], modified to operate at 3.3V and 0V power rails (rather than $\pm 5V$). Avalanche noise produced by a reverse biased base-emitter junction of a 2N4401 transistor is amplified through a dual op-amp amplifier stage before being modulated about a 1.65V DC bias $\left(\frac{V_{cc}}{2}\right)$;

[5] A floating input pin can easily be influenced by its external environment e.g. picking up 50Hz noise from nearby electrical mains, resulting in a non-uniform frequency spectrum for random output.

this output can then be fed into one of the iCE40's comparator inputs. A second output on the PCB comes from a simple potentiometer circuit to set the common mode reference voltage on the second comparator input (with series resistors to allow for fine control about $\frac{V_{cc}}{2}$). Since the comparator output depends on whether the instantaneous voltage of the noise waveform lies above or below this reference voltage, this reference point can be adjusted to set the ratio of ones to zeros in the random comparator output.

The Art of Electronics [14, p. 984] describes the noise produced by the circuit as being spectrally white up to around 50MHz, with a 100mVrms output signal.



(a) Front (b) Back

Figure 4: 3D renders of PCB (in KiCad).

The full circuit schematic and PCB layout can be found in Appendix C, as well as the GitHub repository listed in Section 5.1.

4.4 Uniform Random Number Generator

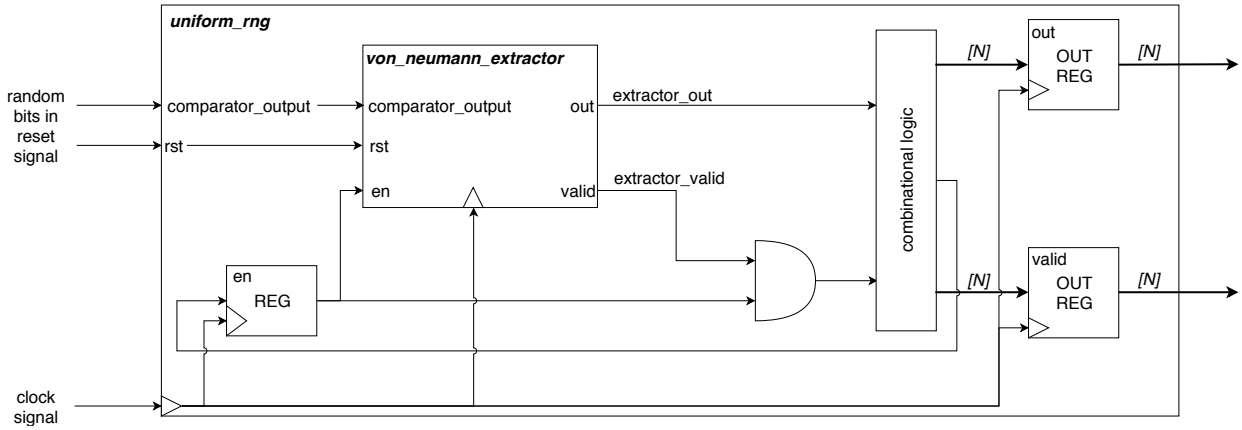


Figure 5: Block diagram of URNG logic circuit.

Figure 5 shows the logic circuit for the uniform random number generator described in Verilog as part of this project (see Section 5.1 for the GitHub repository containing the code). Random bits are supplied to this logic circuit, such as those produced by the hardware entropy source described in Section 4.3. These are fed into a von Neumann debiasing module that uses a simple algorithm, invented by John von Neumann, to remove bias from the input bits, so that the probability of reciving a 1 bit from the debiasing module's output is equal to the probability of receiving a 0. This is achieved by taking two input bits at a time — if both bits are equal (received sequence 0,0 or 1,1), they are discarded; if the received sequence is 1,0 then the output bit is 1; if the received sequence is 0,1 then the output bit is zero. The module's behaviour can be described more formally: for random input bits modelled by random variable X with a distribution $p()$ such that $p(X = 1) \neq p(X = 0)$ but $p(X_i = 1, X_{i+1} = 0) = p(X_i = 0, X_{i+1} = 1)$, the debiasing module outputs random bits modelled by random variable Y with distribution $q()$ where $q(Y = 0) = q(Y = 1)$.

The additional logic in this URNG module is used to store n bits output by the extractor in an n bit output register (providing a fixed point n bit uniform random number). Each bit in the register is associated with a valid signal (stored in the *valid* output register) that goes high once the associated output bit takes on a value from the extractor output.

The circuit runs until the output register is filled, at which point the internal enable (*en*) register is set from 1 to 0; to generate a new random number the module must be reset.

The GitHub repository for the URNG contains a Verilog testbench that can be run with the command `make simurng` to successfully simulate and test this logic circuit. The circuit has also been successfully synthesised for an iCE40 UP5K FPGA; a detailed analysis is presented in Section 6.3.

4.5 Inversion Method Random Number Generator

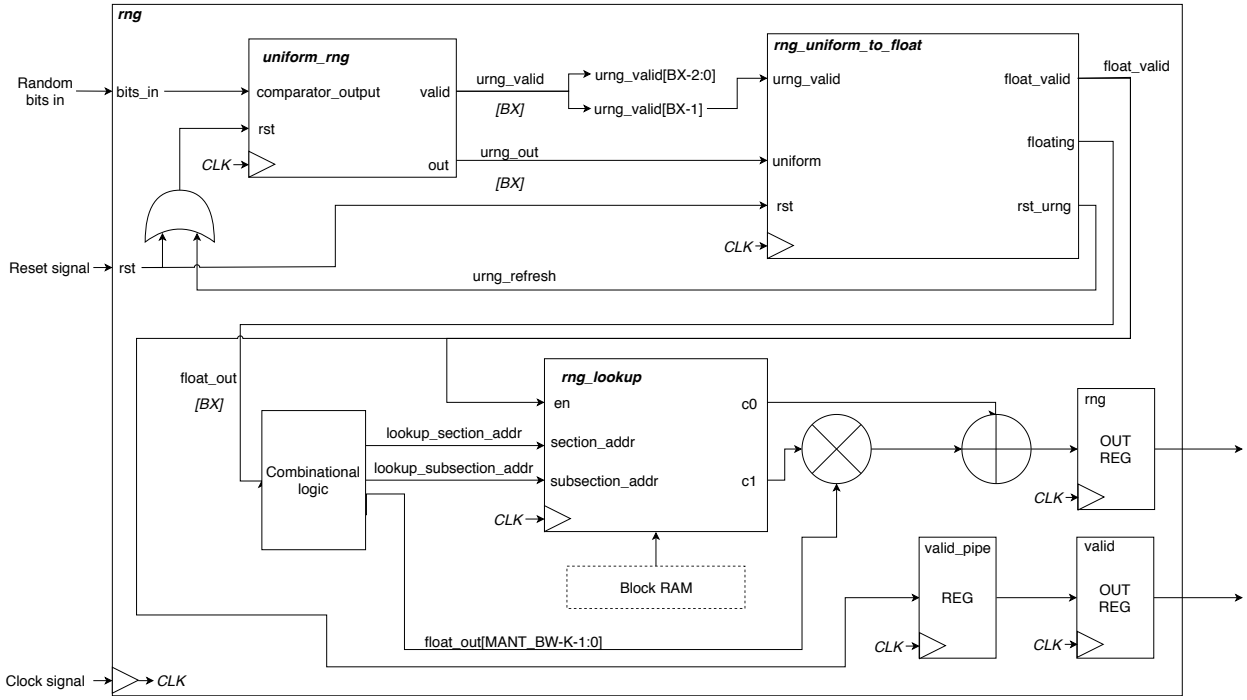


Figure 6: Block diagram of RNG logic circuit.

This module contains the URNG described in Section 4.4, transforming uniformly distributed random numbers into random numbers distributed according to an arbitary (e.g. Laplace) distribution. This is achieved by implementing the hardware efficient system described by De Schryver et al. [7].

The module *rng_uniform_to_float* takes the uniform random numbers produced by the URNG and transforms them into a floating point representation using the algorithm described by De Schryver et al. [7]. The production of a single floating point number may require more than one uniform random number to be consumed, so this module is capable of resetting the URNG to generate as many uniform random numbers as it needs to produce a floating point output number. This module requires an internal block of combinational logic to count the number of leading zeros within a particular region of the bit vector coming from the URNG; the author implemented this logic module using a hardware-efficient recursively[6] instantiated tree structure proposed by Oklobdzija [15].

[6]The finite recursion depth when instantiating this module means that the resulting logic circuit is still synthesisable i.e. can be realised as a physical circuit on an FPGA.

The floating point number produced by the *rng_uniform_to_float* module is used as the input address to a block RAM lookup table, allowing two coeffients (c0 and c1) to be read out of the RAM. The contents of this RAM is generated by a C program written by the author (described in Section 5.3) which obtains c0 and c1 coefficients by taking discrete samples of a continuous probability distribution such as the Laplace distribution. The final output number is calculated at $c0 + c1 \times m$ where m consists of a certain number[7] of the least significant bits from the mantissa part of the floating point number produced by *rng_uniform_to_float*. This final random number output is accompanied by a valid flag signal; this signal must be delayed using the *valid_pipe* register to synchronise it with the associated random number at the output. As with the URNG, the logic circuit runs until a valid random number appears at the output; it must then be reset to produce a new random number.

[7]Details of this random number generator architecture can be found in the paper [7] on which this implementation is based; they are not repeated in this report.

5 Technologies and Techniques Used

5.1 Git/GitHub

Below is a list of GitHub repositories containing work done as part of this project:

- Main project repository containing Python simulations/tests, Verilog templates, Verilog hardware descriptions (e.g. the nonuniform random number generator) and C code:
  `https://github.com/Gregox273/Newton-Verilog-Compiler`
- Verilog repository for the iCE40 differential I/O based uniform random number generator:
  `https://github.com/physical-computation/iCE40-LVDS-RNG`
- Schematic and PCB design for hardware random noise generator:
  `https://github.com/Gregox273/hardware-rng`
- Physical Computation Laboratory measurement repository, containing measurements taken when characterising the hardware random number generator (Sections 6.1 and 6.2):
  `https://github.com/physical-computation/measurement-data`
  Measurement Number: 33 Branch: issue-33

Over the course of this project, GitHub provided a remote location to store/backup work and transfer it between computers. Git's version control functionality facilitated development of software, Verilog firmware and documentation by allowing work to be divided up into commits that could be merged/rolled back/compared when necessary. The Git commit history also provides a timeline of project progress — an example is shown in Figure 7.

```
commit 24ebf237082df41ef72321b00d26b4ceeab103e4
Author: Gregory Brooks <gregox273@gmail.com>
Date:   Mon Feb 4 21:52:41 2019 +0000

    Begin rng testbench & document clz modules

commit 95f9c3d059be53ccfb0b9234f7a5dd222218de64
Author: Greg Brooks <gregox273@gmail.com>
Date:   Sat Feb 2 22:51:29 2019 +0000

    Begin adding testbenches (clz testbenches)

commit 915af56dd5276e0b3893286984e1087056f6360c
Author: Greg Brooks <gregox273@gmail.com>
Date:   Fri Dec 28 21:44:18 2018 +0000

    Add examples of hardware LUT4s being used as programmable delay lines (for uniform TRNG)

commit 704fb18cdca2f4c0e92dce716eebbb948ca0d2cc
Author: Greg Brooks <gregox273@gmail.com>
Date:   Tue Dec 25 16:55:54 2018 +0000

    Update Python script to match lookup generation algorithm used in C code

commit d282e5796ed1e389b5e7e7fb4e76fa57d3e3217d
Author: Greg Brooks <gregox273@gmail.com>
Date:   Sun Dec 23 21:04:55 2018 +0000

    Add type aliases to help debug/fix in event of overflow

commit c14812dceefaf52619859d4a082eb8972363f83b
Author: Greg Brooks <gregox273@gmail.com>
Date:   Sun Dec 23 02:59:03 2018 +0000

    Add to compiler so that it autogenerates some .vh files and lookup table values

commit 3dfe2b486f849a9d2c59065d7d9212d5b45d2bc7
Author: Greg Brooks <gregox273@gmail.com>
Date:   Fri Dec 21 13:08:56 2018 +0000

    Begin Newton compiler backend (in C)

commit 5af1b34e1fa9560fb6be94f7817e210506306fd7
Author: Greg Brooks <gregox273@gmail.com>
Date:   Tue Dec 18 16:53:14 2018 +0000

    Create directory for c compiler code

commit 456140cf474682fc7d8e156da219e3b87a2bf07c
Author: Greg Brooks <gregox273@gmail.com>
Date:   Tue Dec 18 16:39:06 2018 +0000
```

Figure 7: Example of the commit history for the main project repository (Newton-Verilog-Compiler).

5.2 FPGA Synthesis Toolchain

The author used the open source Project IceStorm [16] toolchain to synthesise Verilog code and to program the iCE40 FPGA. The following tools in the toolchain were used over the course of the project:

- Yosys: synthesises Verilog to a logic circuit representation.
- Arachne-pnr: place and route tool that maps a logic circuit to physical hardware within the FPGA.

- Icepack: tool for converting ASCII format file (e.g. the arachne-pnr output) to a binary file (bitstream) for programming the iCE40 FPGA.
- Icetime: tool used to analyse signal timings in the circuit to be programmed. One of the useful parameters output by this tool is a value for the maximum clock speed a circuit can be expected to operate at.
- Iceprog: a program used to program an iCE40 FPGA over USB, using an FTDI based programmer present on the board to be programmed.

5.3 Main Project Repository Structure and Build System

This section describes the layout of the *Newton-Verilog-Compiler* main project repository, as well as the method used to compile/synthesise code contained within. This repository contains a significant amount of the work carried out as part of this project, but not all of it (see Section 5.1).

At the start of the project, the author decided to use Python scripts to prototype algorithms that would later be implemented in C or Verilog. This was done to aid debugging and testing due to the ease with which a Python script can be paused, inspected and restarted during execution (though it could be argued that a debugger such as gdb allows compiled C programs to be manipulated in a similar way). This Python script can be found in the `python-diff-priv` directory within the main repository.

All Verilog modules written over the course of this project are parameterised so that wire/register bit widths can be easily changed without having to rewrite any code. In this repository, parameter values are taken from `.vh` Verilog header files which can be supplied manually or generated using the C program in the `c_compiler` directory — this program takes the privacy file `privacy.yaml` (a partial implementation of the *privacy file* described in Section 4.1) as an input and uses it to generate Verilog header files, by populating a template file. The program also generates the data to be stored in the FPGA's block RAM as part of the random number generator's BRAM lookup table; this data is stored in `.mem` files that are read in Verilog using the `$readmemh` command.

The `c_compiler` directory also contains the Verilog code for the nonuniform random number generator.

Throughout this project, the author made extensive use of the `make` tool for Linux, simplifying the compilation/synthesis process:

- The C program is compiled by running `make` in the `c_compiler` directory. It can then be run with `build/c_compiler`.
- Once the C program has been run to generate the necessary Verilog header files and `.mem` BRAM initialisation value, the Verilog code for the nonuniform random number generator can be synthesised by running `make synthrng` in the `c_compiler/verilog` directory. Alternatively `make simrng` can be used to simulate it, running a Verilog testbench. Similar commands (`make synthclz`, `make simclz`) can be used to synthesise/test the leading zero counter logic in isolation.

6 Evaluation and Characterisation

6.1 Evaluation of Hardware Entropy Source

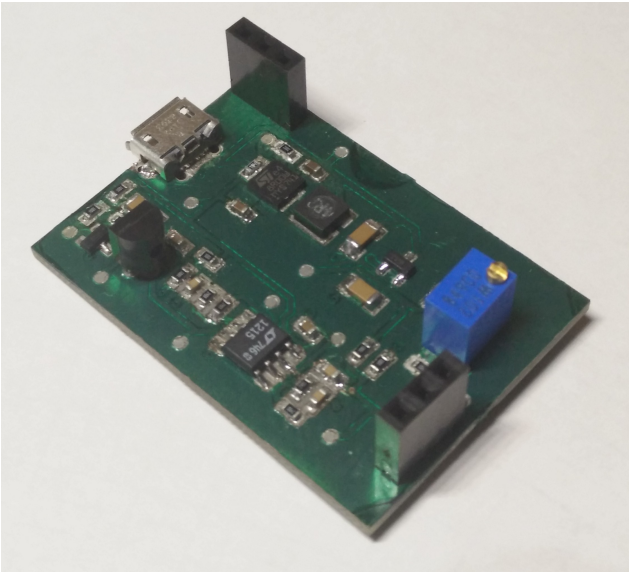


Figure 8: The physical PCB, assembled by the CUED Dyson Centre's Electronics Development Group.

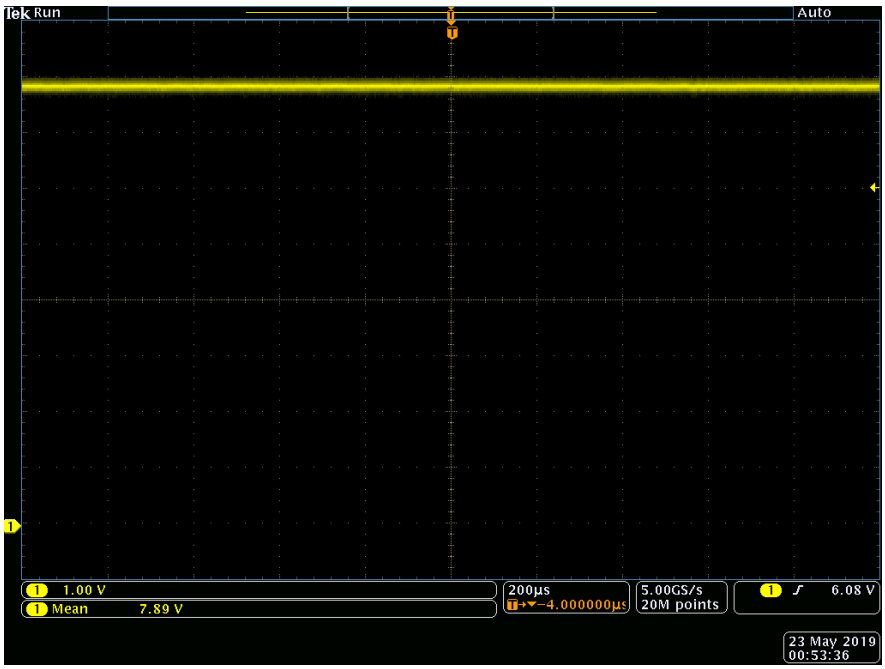
6.1.1 Method

This section presents measurements taken at several test points in the assembled circuit, analysing the noise signal at each stage in the amplifier circuit. A schematic is available in Appendix C, showing the locations of these test points within the circuit. The author measured and recorded voltage waveforms at each of the testpoints listed below using a Tektronix MDO4104C oscilloscope with both AC and DC probe coupling — DC coupling was used for the waveform screenshots shown below, (as well as for taking the average voltage measurements shown in the screenshots), whilst AC coupling was used for V_{RMS} measurements. When displaying the FFT of signals on the oscilloscope, the author used a rectangular window function since the noise spectrum was predicted to be fairly wide (white up to around 50MHz [14, p. 984]).

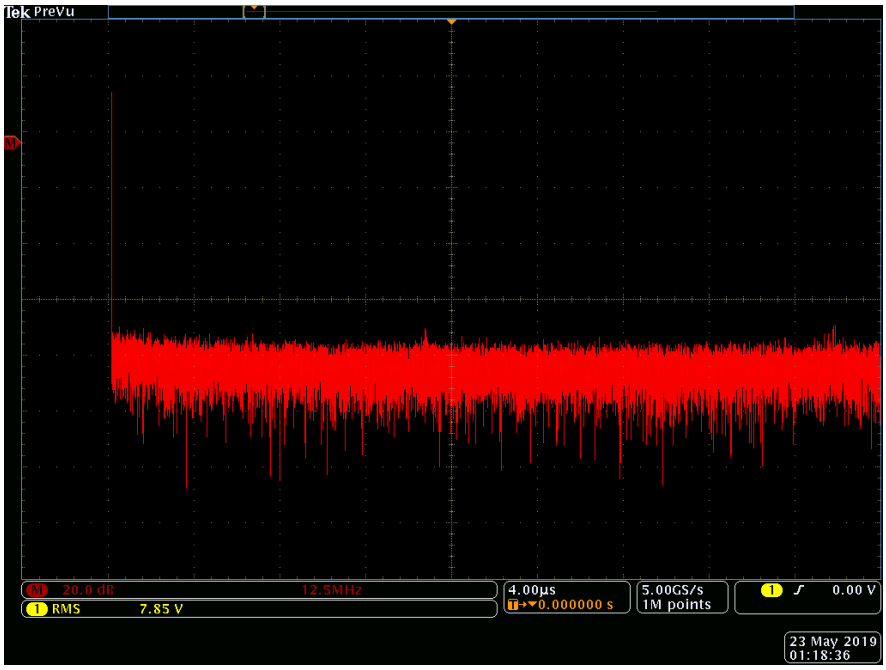
6.1.2 Results

TP2: Avalanche breakdown noise source

$$V_{RMS} = 8.79mV.$$



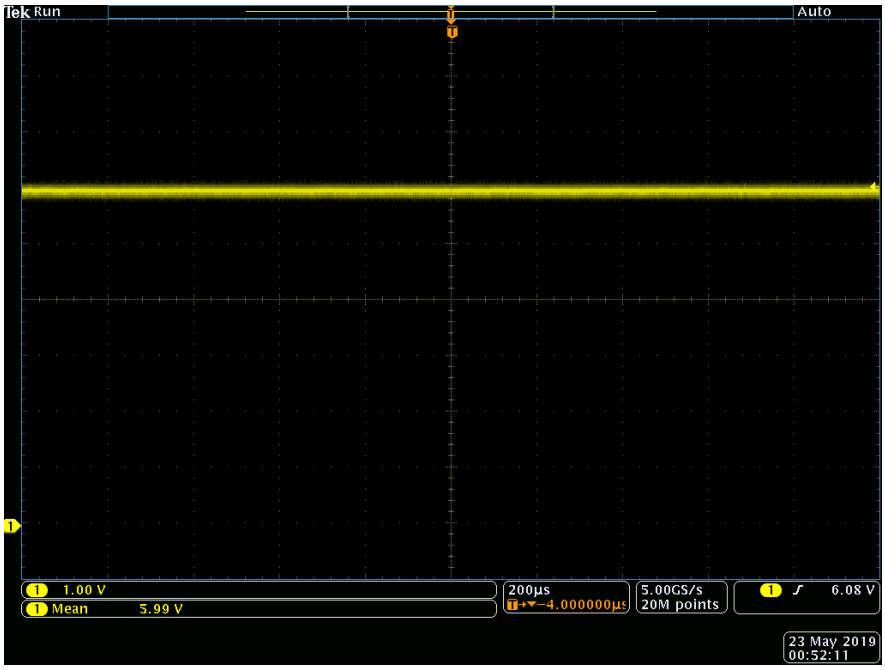
(a) Time domain



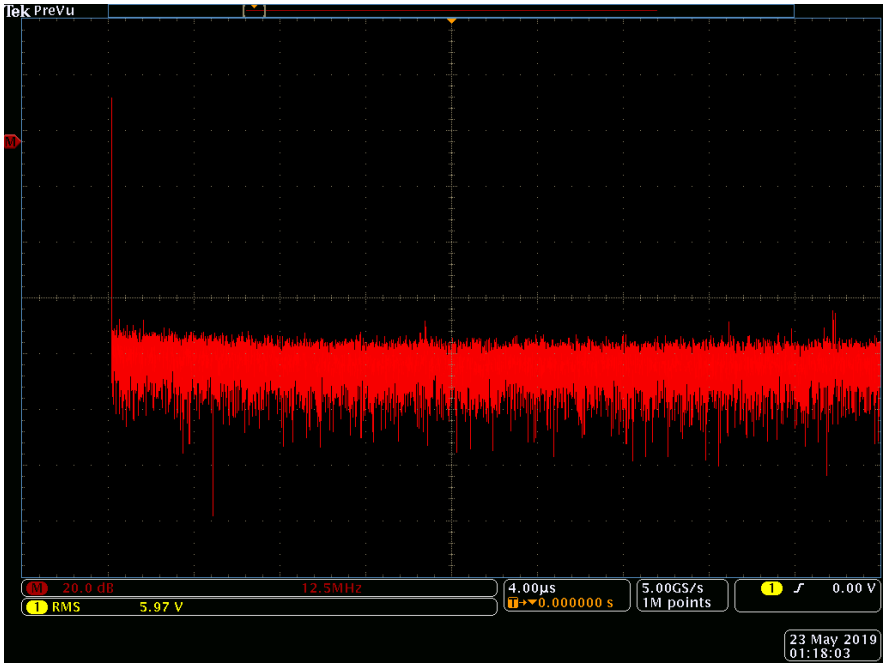
(b) Frequency domain

TP4: Input to first op-amp

$$V_{RMS} = 8.92mV.$$



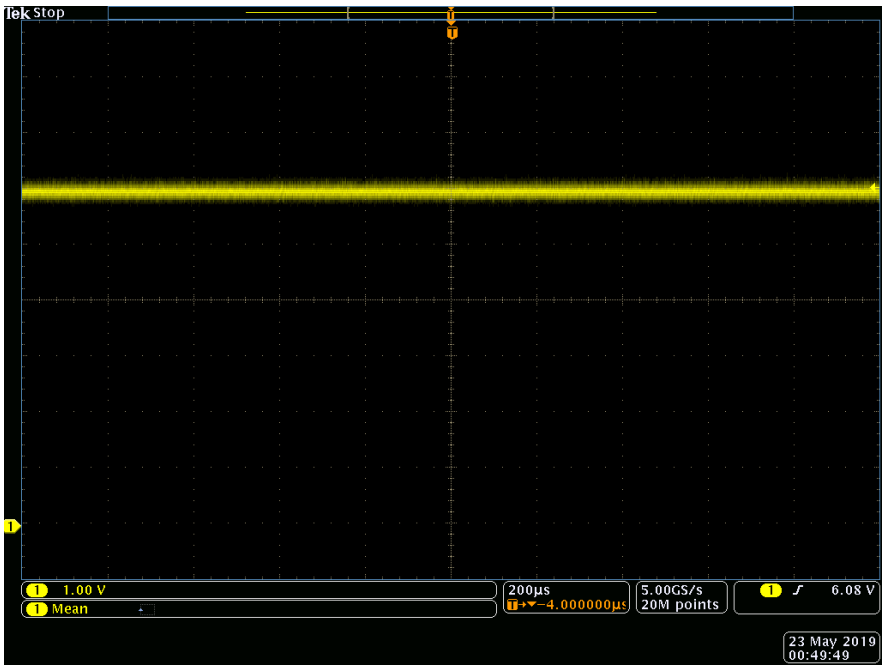
(a) Time domain



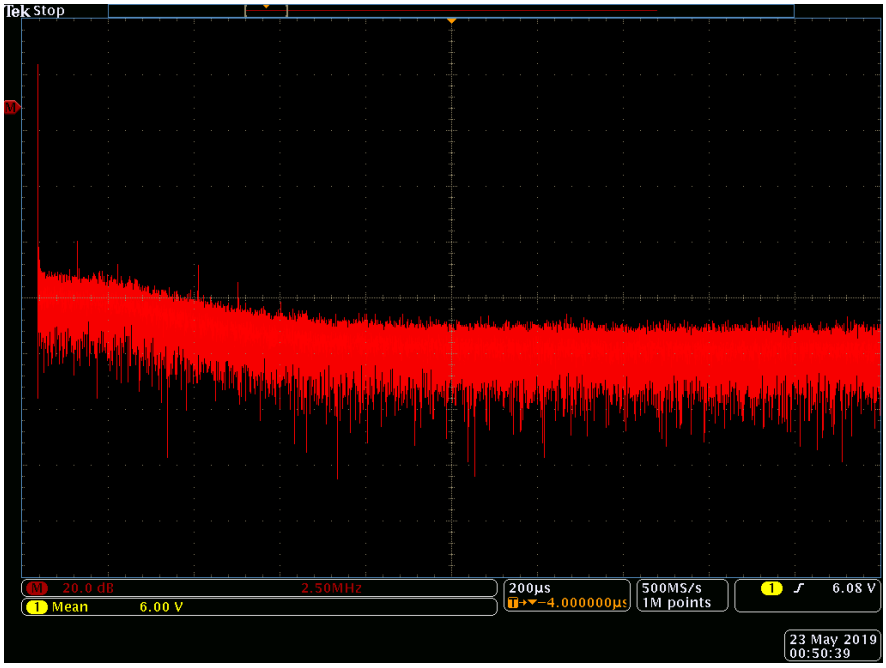
(b) Frequency domain

TP8: Output of first op-amp/input to second op-amp

$$V_{RMS} = 38.4mV.$$



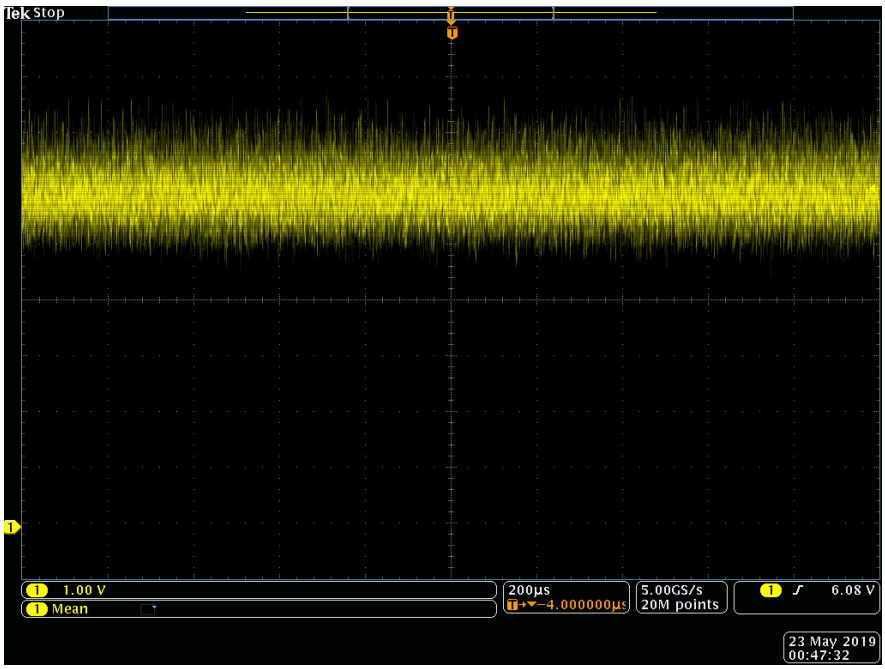
(a) Time domain



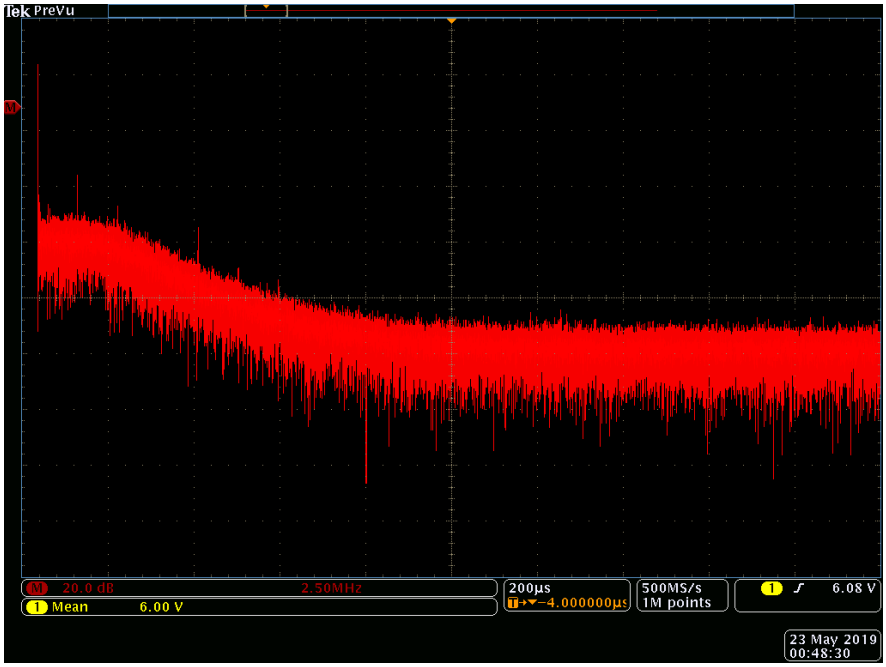
(b) Frequency domain

TP9: Output of second op-amp

$$V_{RMS} = 363mV.$$



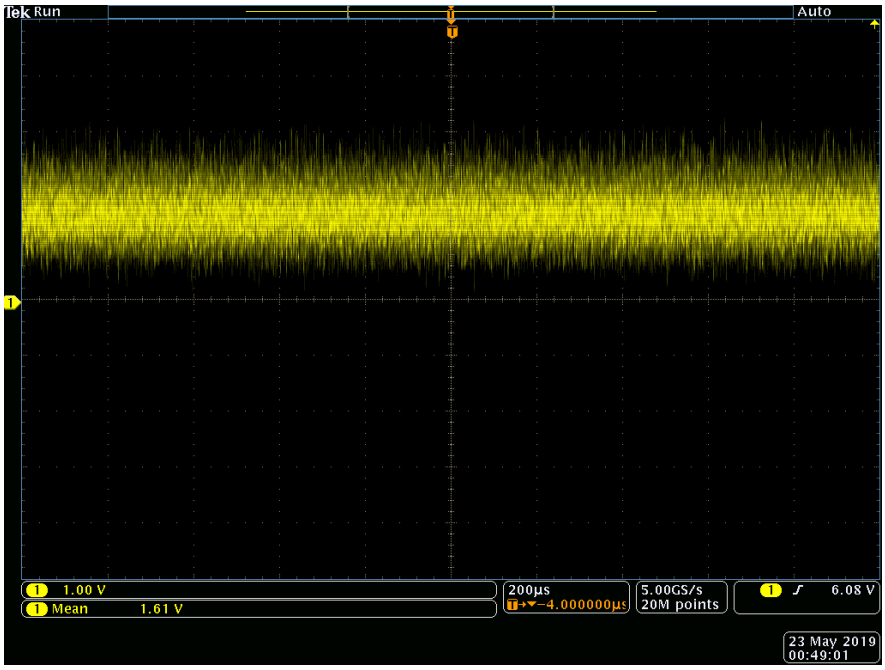
(a) Time domain



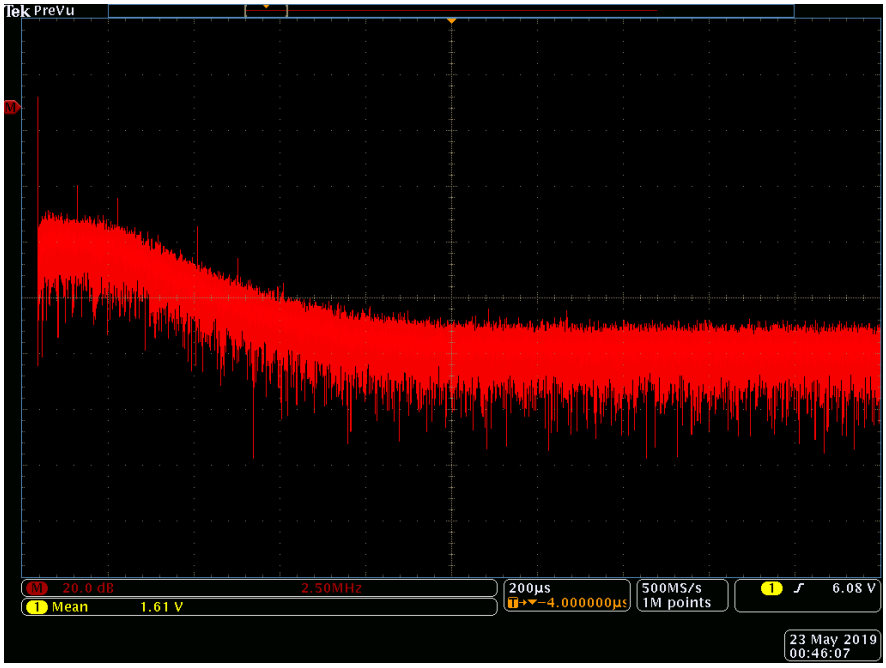
(b) Frequency domain

TP10: Circuit noise output

$$V_{RMS} = 343mV.$$



(a) Time domain



(b) Frequency domain

6.1.3 Discussion

These measurements suggest that at lower frequencies, the circuit behaves as intended:

-

6.2 Randomness Testing

The author used an improved version of the NIST Statistical Test Suite, developed by Cisco [17], to assess the randomness of the bits produced when the hardware entropy source was connected to a comparator input on an iCE40 FPGA (on a Warp-FPGA board [18]). The comparator output was sampled using a Digilent Analog Discovery 2, allowing 2 million random bits to be recorded (at 10kilobits per second, clocked by the iCE40's low frequency oscillator) for analysis.

When running the test suite [19], the input data is split into i *iterations* of length S bits, where a greater i value leads to a more accurate result[8], but also a smaller S value for a given amount of sample data (which can prevent some tests from running; a recommended [17] S value is 2^{20}). Each test run as part of the suite consists of two *analyses*; a test is said to have passed if:

- Proportion analysis: the proportion of the i iterations which pass the test falls within a certain confidence interval.
- Uniformity analysis: the distribution of test P-values for each of the i iterations is sufficiently uniformly distributed.

Different i, S combinations were used to obtain the following results:

6.2.1 $i = 80$, $S = 25000$

The following tests were not run due to S being too small: Rank, Overlapping Template, Universal, Approximate Entropy, Random Excursions, Random Excursions Variant, Serial, Linear Complexity.

150/155 tests passed both analyses:

[8] The accompanying NIST report [19] suggests at least 55 iterations.

Test Failed	Proportion	Uniformity
Runs	X	
3/148 of Non-overlapping Template Matching	X	
1/148 of Non-overlapping Template Matching	X	X

Table 1: Tests failed when using a valid number of iterations. Failures are denoted by an X.

6.2.2 $i = 2$, $S = 1000000$

155/188 tests passed both analyses:

Test Failed	Frequency	Proportion
Frequency	X	
Cumulative Sums (forward)	X	
Cumulative Sums (backward)	X	
Runs	X	X
3/148 of Non-overlapping Template Matching	X	
8/8 of Random Excursions	X	X
18/18 of Random Excursions Variant	X	X

Table 2: Tests failed when using enough bits per iteration to run the entire test suite. Failures are denoted by an X.

A detailed analysis of these results, and the mathematics behind the test suite, is beyond the scope of this report; the fact that the majority of tests appear to have passed suggests that the hardware entropy source is likely to be sufficient for the requirements of this project. Additionally, as i (and hence test accuracy) increased, the proportion of tests passed increased.

One area for further investigation would be to gather a larger sample of random bits ($\gtrsim 60$ Megabits[9]), both with and without the hardware entropy source attached to the FPGA. This would allow for comparison of test results, whilst a larger sample size would allow the entire test suite to be run with a sufficient number of iterations. The speed at which the above data could be acquired was significantly limited by the low clock frequency used to clock the

[9]The Analog Discovery 2 can only record 10 million samples at a time, so such a measurement would require several samples to be taken and concatenated later for testing.

random bit generator — the iCE40's 48MHz high frequency oscillator could be used instead to overcome this limitation, though it is likely that this frequency would have to be divided down to a few MHz or less to minimise edge distortion due to low bandwidth in PCB traces and flying wires leading to a logic analyser, as well as to reduce cross talk between wires.

6.3 Scaling of Logic Implementations

6.3.1 Method

This section presents the results of investigating the effect of random number generator bit width (the number of bits in the output) on the size and complexity of the resulting FPGA implementation. The author used Yosys, a program from the Project IceStorm [16] toolchain, to synthesise Verilog RTL on a desktop computer[10]. The terminal commands used are contained within the Makefiles in the URNG/RNG repositories, allowing these measurments to be repeated by running `make synthurng`/`make synthrng` within the appropriate repository (see Section 5.1 for a list of repositories). At the end of the synthesis process, yosys prints statistics for the resulting logic circuit — this information is displayed in Section 6.3.2.

6.3.2 Results

The following tables are taken from Yosys' output. The table headings are described here [20]:

- Wires: The number of wires in the circuit where the multibit WIRE[X:0] counts as a single wire.
- Bits: The number of wire bits in the circuit where the multibit WIRE[X:0] consists of X+1 wire bits.
- CARRY: The number of logic cells used for their carry logic circuits (a combinational logic circuit useful for arithmetic operations).
- DFFE: The number of logic cells used as D flip-flops with clock enable input.
- DFFESR: The number of logic cells used as D flip-flops with clock enable and synchronous reset inputs.
- LUT4: The number of logic cells used as ROM 4 input look-up tables.
- U, S CPU Time: Time taken for synthesis to complete (user time, system time).

[10] AMD Ryzen 5 2600X Six-Core Processor 3.60GHz, 16GB RAM.

- Mem: Total RAM used by Yosys to synthesise design.

	No. of Wires		No. of Cells				Synthesis	
N	Wires	Bits	CARRY	DFFE	DFFESR	LUT4	U, S CPU Time/s	Mem/MB
1	*29*	*30*	*0*	*6*	*4*	*12*	*0.27, 0.113*	*28.93*
2	33	37	0	5	7	16	0.30, 0.05	29.00
3	*44*	*58*	*3*	*6*	*9*	*25*	*0.34, 0.08*	*29.01*
4	41	56	1	7	10	24	0.34, 0.08	29.02
5	*49*	*73*	*5*	*8*	*12*	*31*	*0.33, 0.06*	*29.04*
8	46	76	2	11	15	30	0.33, 0.17	35.29
16	60	117	3	19	24	45	0.52, 0.14	29.27
32	85	193	4	35	41	71	0.73, 0.19	29.55
64	119	326	5	67	74	106	1.14, 0.25	30.50

Table 3: Effect of varying uniform RNG bit width, BX, on logic circuit size. Odd numbered bit widths are printed in italics.

	No. of Wires		No. of Cells				Synthesis	
N	Wires	Bits	CARRY	DFFE	DFFESR	LUT4	U, S CPU Time/s	Mem/MB

Table 4: Effect of varying nonuniform RNG bit width, BY, on logic circuit size (for constant URNG bit width BX). Odd numbered bit widths are printed in italics.

	No. of Wires		No. of Cells				Synthesis	
N	Wires	Bits	CARRY	DFFE	DFFESR	LUT4	U, S CPU Time/s	Mem/MB

Table 5: Effect of varying URNG bit width, BX, on RNG logic circuit size (for constant RNG output bit width BY). Odd numbered bit widths are printed in italics.

6.3.3 Discussion

⟨ *TODO: evaluate e.g. number of LUTs etc.*⟩

⟨ *TODO: investigate how URNG and RNG logic implementations scale with various parameters (and how this compares to a naive implementation)*⟩

⟨ *TODO: investigate scaling of adder hardware (inferred from Verilog vs iCE40 hardware accumulators)*⟩

7 Conclusions

The work done over the course of this project is contained within many different GitHub repositories, which can make it difficult to see how various pieces of work fit together. For a relatively small project such as this, it may have been simpler to keep all work within a single repository, though this would make it more difficult to reuse and/or modify individual components in future research/projects.

7.1 Future Work

The most obvious path for future work is to continue with the implemenatation of the system outlined above, in order to reduce a working prototype. As a research project, the characterisation work described in Section 6 was prioritised over working towards a working prototype, for the purposes of inspiring future research.

One potential area for future research is the application of the Newton language to safeguarding against *transduction attacks* (attacks on a computer system performed by manipulating transducers i.e. sensors in the system). This idea was briefly investigated by the author in parallel to this project, resulting in the publication of a poster [5] presented by the author at EuroSys 2019 in Dresden. The overall idea behind the poster was that Newton could be used to describe plausible system behaviour (e.g. constraints imposed by physical laws governing the system), thereby allowing implausible behaviour to be recognised and interpreted as a possible transduction attack in progress. The poster is summarised in Appendix D.

References

[1] Jonathan Lim and Phillip Stanley-Marbell. "Newton: A Language for Describing Physics". In: *CoRR* abs/1811.04626 (2018). arXiv: 1811.04626. URL: http://arxiv.org/abs/1811.04626.

[2] Cynthia Dwork and Adam Smith. "Differential privacy for statistics: What we know and what we want to learn". In: *Journal of Privacy and Confidentiality* 1.2 (2010).

[3] Woo-Seok Choi et al. "Guaranteeing Local Differential Privacy on Ultra-Low-Power Systems". In: *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)* (2018), pp. 561–574.

[4] Peter Kairouz, Sewoong Oh and Pramod Viswanath. "Extremal Mechanisms for Local Differential Privacy". In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'14. Montreal, Canada: MIT Press, 2014, pp. 2879–2887. URL: http://dl.acm.org/citation.cfm?id=2969033.2969148.

[5] Gregory Brooks, Youchao Wang and Phillip Stanley-Marbell. *Safeguarding Sensor Device Drivers Using Physical Constraints*. Poster presented at EuroSys 2019, Dresden, Germany.

[6] *Physical Computation Laboratory*. http://physcomp.eng.cam.ac.uk/. Accessed 2019-05-18.

[7] Christian De Schryver et al. "A Hardware Efficient Random Number Generator for Nonuniform Distributions with Arbitrary Precision". In: *Int. J. Reconfig. Comput.* 2012 (Jan. 2012), 12:12–12:12. ISSN: 1687-7195. DOI: 10.1155/2012/675130. URL: http://dx.doi.org/10.1155/2012/675130.

[8] Cynthia Dwork et al. "Calibrating Noise to Sensitivity in Private Data Analysis". In: *Theory of Cryptography*. Ed. by Shai Halevi and Tal Rabin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 265–284. ISBN: 978-3-540-32732-5.

[9] *iCE40 Ultraplus MDP*. http://www.latticesemi.com/Products/DevelopmentBoardsAndKits/iCE40UltraPlusMobileDevPlatform. Accessed 2019-05-18.

[10] *opencv-contrib-python*. https://pypi.org/project/opencv-contrib-python/. Accessed 2019-05-19.

[11] Paul Viola and Michael Jones. *Rapid object detection using a boosted cascade of simple features*. 2001.

[12] *iCE40 Ultraplus*. http://www.latticesemi.com/Products/FPGAandCPLD/iCE40UltraPlus. Accessed 2018-12-26.

[13] Lattice Semiconductor. *Common Analog Functions Using an iCE40 FPGA*. Tech. rep. 2012.

[14] Paul Horowitz and Winfield Hill. *The Art of Electronics*. 3rd ed. Cambridge University Press, 2015. ISBN: 978-0-521-80926-9.

[15] V.G. Oklobdzija. "An implementation algorithm and design of a novel leading zero detector circuit". In: Nov. 1992, 391–395 vol.1. DOI: 10.1109/ACSSC.1992.269243.

[16] *Project IceStorm*. http://www.clifford.at/icestorm/. Accessed 2019-05-24.

[17] *sts*. https://github.com/arcetri/sts. Accessed 2019-05-25.

[18] Phillip Stanley-Marbell and Martin Rinard. "A Hardware Platform for Efficient Multi-Modal Sensing with Adaptive Approximation". In: *arXiv preprint arXiv:1804.09241* (2018).

[19] Lawrence E. Bassham III et al. *SP 800-22 Rev. 1a. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. Tech. rep. Gaithersburg, MD, United States, 2010.

[20] Lattice Semiconductor. *LATTICE ICE^{TM} Technology Library*. Tech. rep. Version 3.0. Aug. 2016.

[21] Kevin Fu and Wenyuan Xu. "Risks of Trusting the Physics of Sensors". In: *Commun. ACM* 61.2 (Jan. 2018), pp. 20–23. ISSN: 0001-0782. DOI: 10.1145/3176402. URL: http://doi.acm.org/10.1145/3176402.

[22] J. Liu, C. Yan and W. Xu. *Can you trust autonomous vehicles: Contactless attacks against sensors of selfdriving vehicles. In DEFCON24 (Aug. 2016)*. URL: http://bit.ly/2EQNOLs (visited on 11/03/2019).

[23] Analog Devices. *ANALOG DEVICES ADVISORY TO ICS ALERT-17-073-01*. Tech. rep. 14th Mar. 2017.

[24] Gregory Brooks. *issue-25, Physical Computation Measurement Data Repository*. URL: https://github.com/physical-computation/measurement-data/tree/issue-25 (visited on 11/03/2019).

A Risk Assessment Retrospective

The risk assessment submitted at the start of the project does not mention any specific hazards besides office (computer) work, since the project is predominately software/firmware based (all project hardware operated at low voltages i.e. 12V or less). No other hazards were encountered during the course of the project, since the random noise generator PCBs were manufactured by the Dyson Centre's Electronics Development Group. In retrospect, although not part of the initial project specification, the risk assessment could have anticipated the possibility of manufacturing PCBs for the project. The hazards associated with this activity include high temperatures (from a soldering iron/oven/hot air gun) as well as chemical hazards associated with solder, fume extraction etc.

B Derivation of an Upper Bound on *Indirect* Differential Privacy Loss

Let $\hat{X}$ be a random variable denoting a sensor measurement, including any measurement error.

Let $N^{\lambda}_{Laplace}$ be a random variable following a Laplace distribution with parameter λ. This represents random noise added to sensor measurements.

Let X be a random variable representing a *noised* sensor measurement i.e. the masked value that the differential privacy system provides to the outside world after applying Laplace distributed noise to a measurement:

$$X = \hat{X} + N^{\lambda}_{Laplace}. \tag{3}$$

A measurement event comprises the variable $\hat{X}$ taking on value $\hat{x}$. Similarly, a noising event can be defined by X taking value $x = \hat{x} + n$. We define an embedded system as having n sensors with measurements distributed as $\hat{X}_i$, $1 \leq i \leq n$.

Let Y denote a variable derived from one or more *noised measurements*, X, such as measurements from different sensors in an embedded system:

$$Y = f(X_1, X_2, ..., X_n) = f(\mathbf{X}),$$
$$y = f(x_1, x_2, ..., x_n) = f(\mathbf{x}).$$

This same mapping function $f(\mathbf{x})$ can take unnoised measurements as arguments:

$$\hat{Y} = f(\hat{X}_1, \hat{X}_2, ..., \hat{X}_n) = f(\hat{\mathbf{X}}),$$
$$\hat{y} = f(\hat{x}_1, \hat{x}_2, ..., \hat{x}_n) = f(\hat{\mathbf{x}}).$$

Using this function, an untrusted party can compute an instance value y by taking measurements x_1 to x_n — variable $\hat{Y}$ has experienced some privacy loss, $l_{\hat{Y}}$, since a noised instance value y has effectively been released to the outside world (i.e. untrusted parties), revealing some information about the private noise-free value $\hat{y}$ taken by variable $\hat{Y}$. The privacy loss can be quantified using a log-likelihood ratio [3] (Equation 5). This privacy loss function requires two parameters, $\hat{y}_a$ and $\hat{y}_b$, which are the possible *true* i.e. noise-free values for $\hat{Y}$ that maximise the privacy loss (Equation 4).

$$y_{obs} = f(\mathbf{x_{obs}}) = \text{value for } y \text{ calculated from noised measurements } \mathbf{x_i},$$

$$\hat{y}_a, \hat{y}_b = \underset{\hat{y}_a, \hat{y}_b}{\operatorname{argmax}}\, l_{\hat{Y}}(\hat{y}_a, \hat{y}_b), \tag{4}$$

$$l_{\hat{Y}}(\hat{y}_a, \hat{y}_b) = log\left(\frac{Pr\{Y = y_{obs}|\hat{Y} = \hat{y}_a\}}{Pr\{Y = y_{obs}|\hat{Y} = \hat{y}_b\}}\right). \tag{5}$$

Equation 5 can be rewritten as follows, where $\hat{\mathbf{x}}_\mathbf{a}$ and $\hat{\mathbf{x}}_\mathbf{b}$ are chosen to maximise $l_{\hat{Y}}$ as before, but subject to constraints $\hat{\mathbf{x}}_\mathbf{a} = f^{-1}(\hat{y}_a)$ and $\hat{\mathbf{x}}_\mathbf{b} = f^{-1}(\hat{y}_b)$:

$$l_{\hat{Y}}(\hat{\mathbf{x}}_\mathbf{a}, \hat{\mathbf{x}}_\mathbf{b}) = log\left(\frac{Pr\{\mathbf{X} = \mathbf{x_{obs}}|\hat{\mathbf{X}} = \hat{\mathbf{x}}_\mathbf{a}\}}{Pr\{\mathbf{X} = \mathbf{x_{obs}}|\hat{\mathbf{X}} = \hat{\mathbf{x}}_\mathbf{b}\}}\right). \tag{6}$$

This constrained optimisation problem could be solved to calculate an exact value for privacy loss. Alternatively, an upper bound on privacy loss can be determined using a far simpler calculation (the sum of privacy losses for each individual sensor):

$$\begin{aligned} l_{\hat{Y}}(\hat{\mathbf{x}}_\mathbf{a}, \hat{\mathbf{x}}_\mathbf{b}) &= log\left(\prod_{u=1}^{n} \frac{Pr\{X_u = x_{u,obs}|\hat{X}_u = \hat{x}_{u,a}\}}{Pr\{X_u = x_{u,obs}|\hat{X}_u = \hat{x}_{u,b}\}}\right), \\ &\leq log\left(\prod_{u=1}^{n} \frac{Pr\{X_u = x_{u,obs}|\hat{X}_u = \hat{x}_{u,c}\}}{Pr\{X_u = x_{u,obs}|\hat{X}_u = \hat{x}_{u,d}\}}\right), \end{aligned} \tag{7}$$

where:

$$\hat{x}_{u,c} = \underset{\hat{x}_{u,c}}{\operatorname{argmax}}\, Pr\{X_u = x_{u,obs}|\hat{X}_u = \hat{x}_{u,c}\},$$

$$\hat{x}_{u,d} = \underset{\hat{x}_{u,d}}{\operatorname{argmin}}\, Pr\{X_u = x_{u,obs}|\hat{X}_u = \hat{x}_{u,d}\},$$

i.e. the constraint $f^{-1}(\hat{y})$ has been removed. Equation 7 can be interpreted as the sum of the privacy losses incurred by the X variables i.e. $\sum_{u=1}^{n} l_{X_u}$. $Pr\{X_u = x_{u,obs} | \hat{X}_u = \hat{x}_u\}$ is simply the Laplace distribution of the noise applied to the measurement represented by $\hat{X}$, centred on value $\hat{x}$.

Note that this privacy loss is incurred when all values $x_{1,obs}$ to $x_{n,obs}$ are provided to the outside world (i.e. the event where an attacker transitions from having no information to the system to being provided with $x_{1,obs}$ to $x_{n,obs}$); for each subsequent individual query response $x_{u,obs}$, the resulting privacy loss is only:

$$l_{\hat{Y}}(\hat{\mathbf{x}}_\mathbf{a}, \hat{\mathbf{x}}_\mathbf{b}) = log\left(\frac{Pr\{X_u = x_{u,obs} | \hat{X}_u = \hat{x}_{u,a}\}}{Pr\{X_u = x_{u,obs} | \hat{X}_u = \hat{x}_{u,b}\}}\right), \tag{8}$$

where values for $\hat{\mathbf{x}}_\mathbf{a}$ and $\hat{\mathbf{x}}_\mathbf{b}$ are the same as those used in Equation 6 (if calculating exact privacy loss) or Equation 7 (if calculating an upper bound).

Since the Newton language recently gained a mutual information operator, the author attempted to factor mutual information into this calculation. Unfortunately, it appears that mutual information alone does not provide enough information to calculate indirect privacy loss — the exact nature of the correlation between two (or more) random variables is required i.e. the conditional distribution for the unknown variable given known ones:

Define $\hat{X}$ and X as before and let $\hat{Y}$ be a random variable correlated with $\hat{X}$. Both $\hat{X}$ and $\hat{Y}$ can take values within some range (e.g. due to finite sensor precision):

$$\hat{X} \in R_{\hat{X}},\ \hat{Y} \in R_{\hat{Y}},$$

The mutual information $I(\hat{X}; \hat{Y})$ is defined as:

$$I(\hat{X}; \hat{Y}) = \int_{R_{\hat{Y}}} \int_{R_{\hat{X}}} p(\hat{X}, \hat{Y}) log\left(\frac{p(\hat{X}, \hat{Y})}{p(\hat{X})p(\hat{Y})}\right) d\hat{X} d\hat{Y}, \tag{9}$$

where $p()$ denotes the probability density function for a random variable. The author has been unable to insert this value into the privacy loss equation, however a value for privacy loss can be obtained if the conditional distribution of $\hat{X}$ given $\hat{Y}$ is known, as this allows the conditional distribution of X given $\hat{Y}$ to be calculated:

$$Pr\{X = x | \hat{Y} = \hat{y}\} = \int_{R_{\hat{X}}} Pr\{X = x | \hat{X} = \hat{x}\} Pr\{\hat{X} = \hat{x} | \hat{Y} = \hat{y}\} d\hat{x}. \tag{10}$$

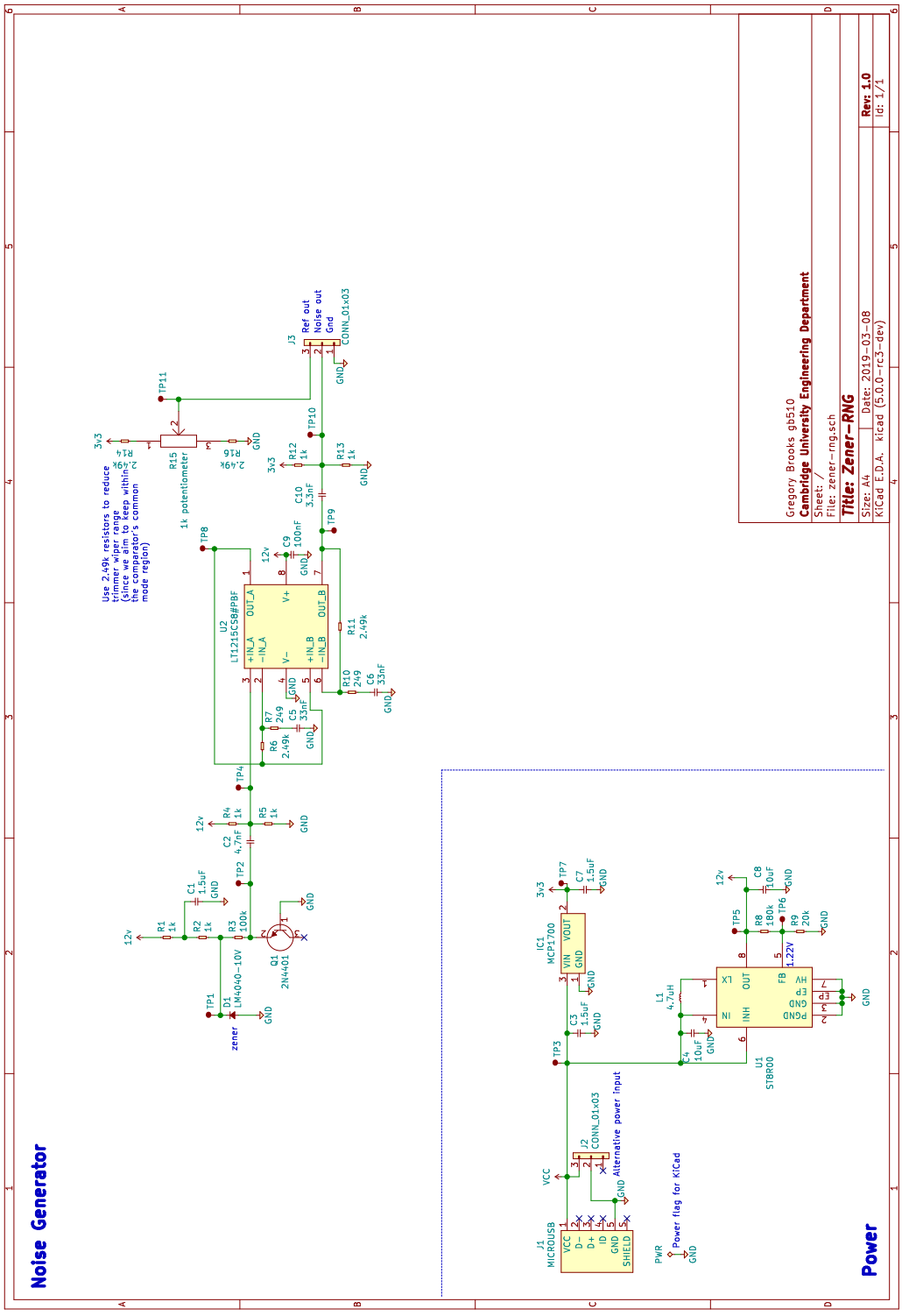
Equation 10 can be substituted into Equation 11, the formula for privacy loss incurred by $\hat{Y}$ as a result of observing a value x_{obs} for X. This results in Equation 12:

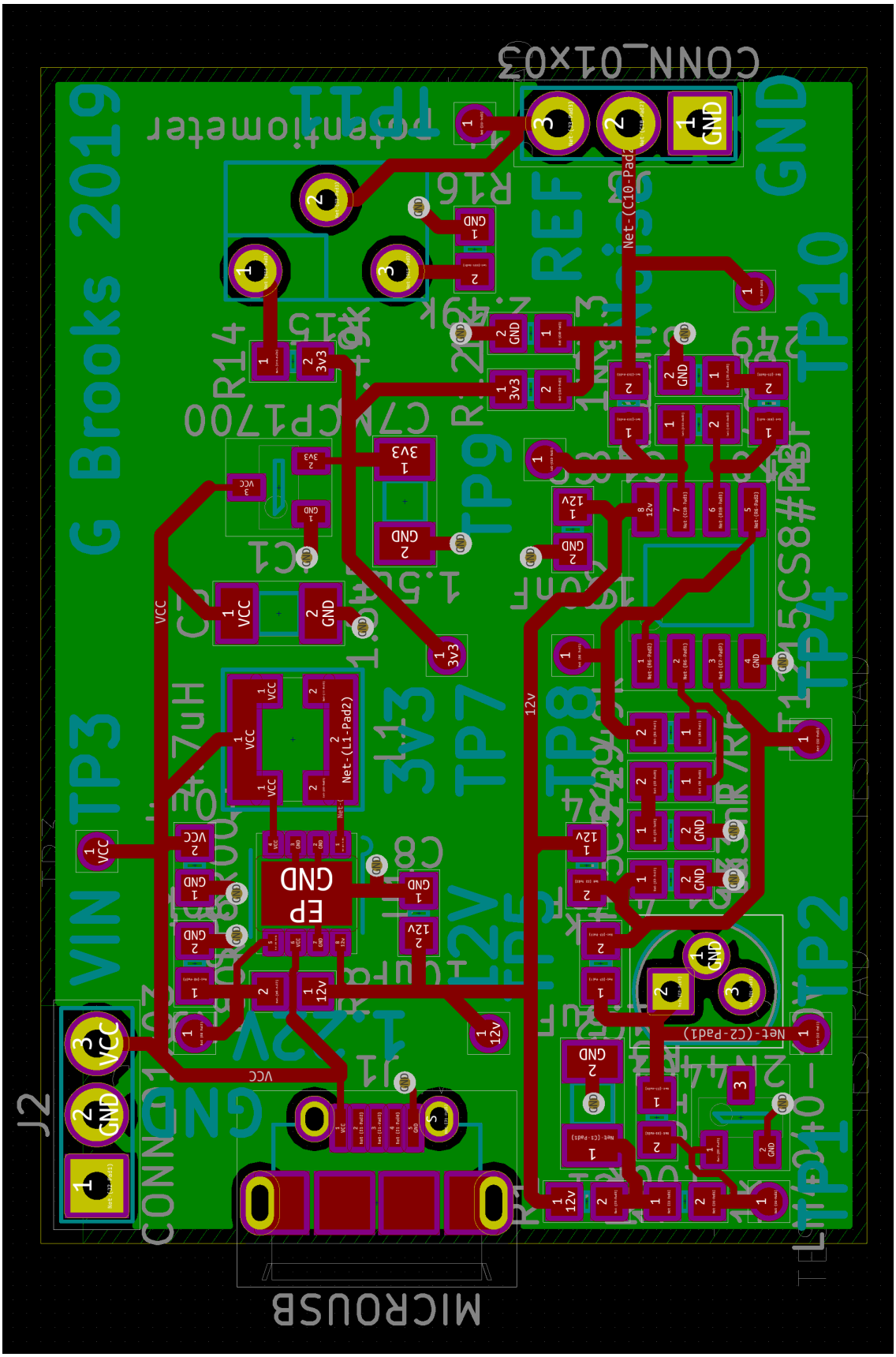
$$l_{\hat{Y}}(\hat{y}_a, \hat{y}_b) = log\left(\frac{Pr\{X = x_{obs}|\hat{Y} = \hat{y}_a\}}{Pr\{X = x_{obs}|\hat{Y} = \hat{y}_b\}}\right) \tag{11}$$

$$= log\left(\frac{\int_{R_{\hat{X}}} Pr\{X = x_{obs}|\hat{X} = \hat{x}\}Pr\{\hat{X} = \hat{x}|\hat{Y} = \hat{y}_a\}d\hat{x}}{\int_{R_{\hat{X}}} Pr\{X = x_{obs}|\hat{X} = \hat{x}\}Pr\{\hat{X} = \hat{x}|\hat{Y} = \hat{y}_b\}d\hat{x}}\right), \tag{12}$$

for $\hat{y}_a, \hat{y}_b = \text{argmax}_{\hat{y}_a,\hat{y}_b}(l_{\hat{Y}})$ as before.

C Hardware Entropy Source Schematic and PCB




G Brooks 2019
VIN
TP3
TP11
TP9
TP7
TP8
TP4
TP10
TP2
GND
3V3
12V
VCC
J2
CONN_01X03
MICROUSB
EP
GND
Net-(L1-Pad2)
Net-(C2-Pad1)
Net-(C10-Pad2)

D EuroSys 2019 Poster: Safeguarding Sensor Device Drivers Using Physical Constraints

This poster [5] was submitted to EuroSys 2019, to communicate the idea of using information about a physical system to condition electronic sensor measurements. The example discussed in the poster is the detection and safeguarding against transduction attacks [21], where an attacker manipulates a sensor's output to gain control over a system. For example, research by Liu, Yan and Xu [22] has shown that the proximity sensors on a Tesla automobile can be fooled into providing erroneous (or even no) data using ultrasonic interference produced by a device built using off-the-shelf electronic components. An attacker is able to influence the system's behaviour without needing to directly manipulate the execution of the vehicle's firmware.

The poster illustrates how the Newton language can be used to describe a physical system, in this case an accelerometer mounted to a PCB without vibration isolation. In this scenario, vibrations of the PCB (e.g. due to a nearby loudspeaker or even, in the case of a smartphone, due to loudspeakers mounted to the PCB) are measured by the accelerometer, obscuring a *true* acceleration measurement i.e. acceleration of the entire system due to gravity. This effect is particularly pronounced if the board is driven at its resonant frequency, since the resulting oscillation will have a greater amplitude [23]. This phenomenon could, in theory, be used as part of a transduction attack e.g. where a smartphone's loudspeaker is used to interfere with a software application's estimate of the device's physical orientation.

To test whether this phenomenon can be distinguished from regular measurement noise, the author performed an experiment [24] using an MMA8451Q accelerometer mounted on an FRDM-KL03Z board. With the sensor resting stationary on a horizontal surface, five minutes worth of accelerometer samples were recorded at 10Hz in order to obtain a probability distribution for the accelerometer's measurement noise (along the z axis, aligned with the vertical); this data was empirically observed to fit a Laplace distribution. This measurement was then repeated with the board resting on top of a smartphone playing a 440Hz audio tone — in theory, this data (random samples from a sinusoid) would fit a bimodal beta distribution (see section D.1 for derivation) allowing a log likelihood ratio to be computed:

$$LLR = -2\sum_{i=1}^{N} \frac{\frac{1}{\pi}\left|\frac{1}{\sqrt{A^2\omega^4 - \ddot{x}_i^2}}\right|}{\frac{1}{2b}exp\left(-\frac{|\ddot{x}_i - \mu|}{b}\right)} \tag{13}$$

The value of this log-likelihood ratio indicates whether the audio tone based transduction attack described here is likely to be occuring. For the data collected in the experiment, the log-likelihood ratio was found to be around -24600.08 whilst the tone was playing, and +3731 when it was not; the negative value indicates that the measurements taken whilst the tone was playing were indeed more accurately described by the bimodal beta distribution, compared to the Laplace distribution of sensor noise, and vice versa for the positive value.

D.1 Derivation of Bimodal Beta Distribution

Let T be a uniformly distributed random variable representing the time at which acceleration is sampled:

$$T \sim U\left(-\frac{\pi}{\omega}, \frac{\pi}{\omega}\right),$$

$$f_T(t) = \begin{cases} \frac{\omega}{2\pi} & -\frac{\pi}{\omega} \leq t \leq \frac{\pi}{\omega} \\ 0 & \text{otherwise} \end{cases}, \tag{14}$$

$$F_T(t) = \begin{cases} 0 & t < -\frac{\pi}{\omega} \\ \frac{\omega(t+\pi/\omega)}{2\pi} & -\frac{\pi}{\omega} \leq t \leq \frac{\pi}{\omega} \\ 1 & t > \frac{\pi}{\omega}. \end{cases}. \tag{15}$$

Then let X = g(T) represent the acceleration value at time T. By considering the dynamics of the system, we know that:

$$g(t) = A\omega^2 sin(\omega t), \tag{16}$$

where A is a frequency-dependent constant equal to the product of quality factor and board displacement at resonance. We can also define a monotonically increasing function $h(x)$ as the inverse of $g(t)$:

$$h(x) = g^{-1}(x) = \frac{1}{\omega} arcsin\left(\frac{x}{A\omega^2}\right). \tag{17}$$

The cumulative distribution function for X can therefore be written in terms of the cumulative

distribution function for T, accounting for the fact that g(t) is not monotonic:

$$F_X(x) = F_T(h(x)) + \Big(1 - F_T(\pi/\omega - h(x))\Big). \tag{18}$$

Taking the derivative results in the probability density function for X:

$$f_X(x) = \Big(f_T(h(x) - f_T(-h(x)))\Big)\frac{d(h(x))}{dx} \tag{19}$$

$$= \begin{cases} \frac{1}{\pi\sqrt{A^2\omega^4 - x^2}} & |x| \leq A\omega^2 \\ 0 & \text{otherwise} \end{cases}. \tag{20}$$