Efficient Hardware Generation of Random Variates with Arbitrary Distributions

David B. Thomas, Wayne Luk
Imperial College London
{dt10,wl}@doc.ic.ac.uk

Abstract

This paper presents a technique for efficiently generating random numbers from a given probability distribution. This is achieved by using a generic hardware architecture, which transforms uniform random numbers according to a distribution mapping stored in RAM, and a software approximation generator that creates distribution mappings for any given target distribution. This technique has many features not found in current non-uniform random number generators, such as the ability to adjust the target distribution while the generator is running, per-cycle switching between distributions, and the ability to generate distributions with discontinuities in the Probability Density Function.

1. Introduction

Many algorithms and applications require random variates taken from non-uniform probability distributions. Applications include: Ray Tracing, where reflectance distributions must be approximated for importance sampling [4, 7]; financial computing, such as evaluation of prices using the Brace, Gatarek and Musiela (BGM) interest model [21]; queuing simulations of communication networks and manufacturing processes [16, 3]; and Monte-Carlo evaluation of functions [15]. These applications rely on a fast, efficient, and accurate supply of random samples, so when implemented in hardware it is necessary to adapt existing variate generation algorithms from software where appropriate, and to develop new algorithms when they are not.

In this paper we describe a technique for the efficient hardware generation of random variates from any distribution, consisting of two parts: a hardware template for a simple yet powerful random variate sampling unit based on, a composite distribution approximation; and a process for generating a composite approximation for any target probability distribution. Unlike most other FPGA based methods for sampling probability distributions, this technique is completely general, and does not require any logic alterations to change the generated distribution. The method also has a number of advantages not found in other hardware generation methods, such as the ability to swap between multiple distributions per-cycle and the introduction of completely new distributions at run-time. Our chief contributions are:

- A probability distribution independent hardware algorithm for sampling non-uniform distributions. The algorithm specifically addresses the strengths and weaknesses of FPGAs to allow an area efficient, high-speed implementation.
- A process for creating composite distribution approximations to be used in the hardware algorithm. This process allows exact measurement of approximation error, and can approximate any target distribution, whether analytically or empirically derived.
- Demonstration of the speed, area efficiency and accuracy of the method, and discussion of the benefits and limitations.

This paper is organised as follows: Section 1 first presents previous work in non-uniform random number generation; Section 2 presents the algorithm and hardware architecture of the new random number generator; Section 3 explains how the data tables required to generate different distributions are pre-calculated in software; and Section 4 presents speed and quality results for the proposed approach, as well as comparing the performance to other methods.

2. Background

A continuous probability distribution is usually described in terms of the Cumulative Distribution Function (CDF), which is a function that determines the probability that a sampled value will be less than or equal to a given point; the first derivative of the CDF is the Probability Density Function (PDF).

$$CDF(D,x) = P(D \leq x) = \int_0^x PDF(D,y)dy \qquad (1)$$

CDFs may be defined analytically through a formula, or implicitly through empirically derived data. Well-known distributions such as the Gaussian and Poisson distribution are used in many applications, as most theoretical

14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'06)
0-7695-2661-6/06 $20.00 © 2006 IEEE

models are based around these distributions. However, in many simulations of real-world systems these distributions are used as approximations to empirically derived distributions, even when the measured distributions are known to be slightly different. For example, when simulating communication channels it is common to use a Gaussian noise model, fitted to the empirical noise distribution using the standard deviation. If the empirical PDF could be efficiently sampled instead, then this may result in more accurate simulations for many applications.

There are many different techniques which can be used for creating random samples taken from a specific non-uniform distribution. Some of these are distribution specific, for example the Box-Muller and Wallace method for sampling from the normal distribution [2, 17]. Other more general techniques allow potentially any distribution that satisfies certain constraints to be targeted, but must be manually customised to fit each specific distribution. These range from quite restrictive techniques such as the Ziggurat method [12], which can only approximate distributions with strictly decreasing densities, to general techniques such as CDF inversion, which can (in principle) be used to approximate any distribution.

There is usually a trade-off in quality, design time and performance, against the generality of a technique: creating a new distribution-specific approximation will usually require a large investment of development time, or acceptance of a poor quality or low performance approximation. If a technique applicable to a family of distributions (e.g. the Ziggurat method) can be used, the process can be semi-automated and will result in fairly predictable quality and performance, but only if the desired distribution belongs to that family. This paper concentrates on the most general purpose approximation methods, that can approximate any distribution with the minimum of design-time.

The most general purpose technique is CDF inversion, as this can be used to approximate any distribution where the CDF function can be inverted. Given CDF^{-1} and a random variate $u \sim U(0,1)$, then $CDF^{-1}(u)$ will provide random samples from the target distributions. For example, the Exponential distribution has a CDF of $1-e^{-x}$. This can be inverted to give an inverse CDF of $-\log(u)$, allowing sampling from the exponential distribution directly, although requiring the evaluation of log for every sample produced.

Even if the CDF cannot be analytically inverted, it is often possible to find some approximation with sufficient accuracy for a task, for example by using a rational polynomial approximation, or a piecewise approximation with polynomials. The Gaussian (or Normal) distribution has no closed-form solution for the inverse CDF, and so it is often approximated in this way. For example in [18] three high-degree rational polynomials are used to provide a transform accurate to 16 decimal digits. Approximations may also be used when an inverse function is available, but is prohibitively expensive to calculate: while the exponential inverse CDF can be evaluated directly, it may make more sense to use a less computationally expensive approximation, particularly if complete accuracy of the distribution is less important than speed.

Most of these CDF inversion approximation techniques either require extensive work for each new distribution or large amounts of RAM, but there are also automated techniques that attempt to produce an accurate inversion without large tables. In [6] a mechanical method is developed for the approximation of arbitrary inverse CDFs by adaptively splitting the domain into intervals and then approximating these intervals with Hermite polynomials. Another automatic method for sampling from arbitrary distributions is based on the ratio-of-uniforms method [10], which relies on sampling from a 2D plane in such a way that the ratio of the co-ordinates has the correct distribution. However, both these methods rely on floating-point operations, and cannot be easily applied to fixed-point calculations.

Previous work on FPGA based non-uniform random variate generation has concentrated on sampling the Gaussian distribution, due to it's importance in Bit Error-Rate testing of communications channels. Most of these use the Box-Muller transform and are concerned with creating efficient and accurate implementations of the $\sin(2\pi x)$ and $\sqrt{-2\log(x)}$ operations required [5, 8, 19]. While these methods introduce general purpose techniques, such as area efficient methods for function evaluation in hardware, applying these techniques to other distributions would require a substantial amount of manual work. The resulting fixed function evaluation pipelines also mean that the distribution produced by the generator is fixed, requiring a different random number generator for each distribution needed within a hardware design.

Others methods, such as the Wallace method [9], and Binomial approximations [1], have been also been used to generate the Gaussian distribution. The Wallace method could potentially be used to generate different distributions, but only the Exponential distribution has been documented. Providing support for other distributions would require theoretical work, as well as a different hardware architecture for each distribution.

In [20] the Ziggurat method is used to generate normal samples. The key strength of the Ziggurat method is that 99% of the time only two table lookups, a multiply, and a compare are needed per variate. In the remaining one percent of cases a more complex equation must be calculated, involving the exponential function. The hardware implementation takes advantage of the differences in throughput, using fast pipelined calculations for the common case, and an area efficient sequential calculation for the uncommon

case. Although the Ziggurat method is usually only used for the Gaussian and Exponential distributions, it can actually be applied to any symmetric distribution that decreases monotonically from a single peak. The main change is in the function evaluated in the uncommon case, which needs to be customised to each distribution, so different hardware designs would be needed for each distribution.

A distribution independent approach is taken in [14], where a table based approach is used to approximate the inverse CDF of the target distribution. The table contains regularly spaced samples from the inverse CDF to be sampled, which are used to provide a piecewise-linear approximation. Although this method is very general, it uses a large amount of storage (262KB) to achieve a good approximation. Even then, the regular sampling pattern will not allow a good approximation of steep gradients and discontinuities. More complex approximations, for example quadratic or cubic interpolation, would allow the table size to be decreased, but require more logic.

3. Generator Algorithm and Architecture

In this section the generator algorithm is explained, followed by an explanation of the hardware implementation of this algorithm and the available implementation options.

The proposed hardware distribution generator operates by taking multiple random variates with easy to generate distributions, and then combining them using a simple operation to provide a composite distribution. By selecting the component distributions and the combination method correctly, this composite distribution can be made to approximate a given target distribution. Exactly how good the approximation is depends on factors such as the smoothness of the target distribution, the number and shape of the component distributions, and the way that they are combined.

While the combination method could in principle be any operator, for efficient implementation only selection is considered in this paper, i.e. each value produced by the composite distribution will be an untransformed sample taken from one of the component distributions. To control the selection process an independent random variate is used, making the selection stateless, and independent of the component distributions.

Each composite generator requires a vector of n component random variates, $RC_1 ... RC_n$, with (as yet) unspecified distributions, and a selection random variate, RS, with a range of $[1,n]$. The value of the composite random variate C (i.e. the output from the random number generator) at step i, along with its overall PDF and CDF, can then be defined as:

$$C_i = RC_{k,i}, \text{ where } k = RS_i \quad (2)$$

$$P(C = x) = \sum_{k=1}^{n} P(RS = k) \times P(RC_k = x) \quad (3)$$

$$P(C < x) = \sum_{k=1}^{n} P(RS = k) \times P(RC_k < x) \quad (4)$$

Because selection is used to combine the distributions, only one component distribution needs to be sampled to produce each sample from the composite distribution. This means that the vector of component variates can be generated by using a single uniform random variate, UC, and selecting from a vector of transformation functions, $FC_1 ... FC_n$, chosen such that $RC_x \sim FC_x(UC)$. In the same way, the selection variate RS can be split into a uniform variate, US, and a transformation function FS. For the purposes of this paper it will be assumed that each component distribution is identically weighted, and FS is reduced to the identity function. The ith value generated by C is then:

$$C_i = FC_{us}(uc), \text{where } us = US_i, uc = UC_i \quad (5)$$

The reason for transforming the task of sampling one distribution into that of selection and sampling from amongst many component distributions is that this provides a very efficient hardware implementation. Specifically, the component distributions can be chosen such that they are implemented purely in terms of adds, bitwise logic, and the selection is reduced to memory lookups.

As input to the composite distribution generator, two uniform random sources US and UC are needed. An issue when generating uniform random numbers in hardware is that it is easy to implement pseudo-random number generators (PRNG) with a distribution $X \sim U[0,2^w)$ (In this paper we extend the standard $U(a,b)$ notation by allowing half-open ranges, so $X \sim U[a,b)$ means that $a \leq X < b$, or equivalently for integers $a \leq X \leq b-1$), but it is difficult to generate any other range without either introducing bias, or requiring large amounts of extra hardware.

A consequence of this range limitation is that n (the number of component distributions) must be a binary power. Similarly all the functions $FC_1 ... FC_n$ must use a $U[0,2^w)$ variate as input. This is quite a restriction, but by embracing the limitations a very simple and area efficient generator can be created. Each FC function will apply a bitwise mask to UC, adjusting the variance, then add an offset, altering the mean. This allows any distribution of the form $U[a,a+2^{w-k})$ to be used for the component distributions, by applying the transformation $FC(x) = fc_{offset} + (x \otimes fc_{mask})$ (where $\otimes$ indicates bitwise-and). The (fc_{offset}, fc_{mask}) pairs can be stored in a ROM or RAM lookup table. The stored offset value will need to be the same width as the output width, but space can be saved in the table by storing just the width of the mask, then expanding it before use. The element width of the

table is then $rw + \lceil \ln_2 mw \rceil$, where rw is the width of the offset (and output value), which limits the range of values that the composite can produce; and mw is the width of the mask, which limits the maximum variance of component distributions (and determines the number of random bits that UC must supply).

Fig. 1 shows the hardware architecture needed to approximate a target distribution using a composite of rectangular component distributions. The six components required (identified with the same numbers in the figure) are:

1. A $U[0,n)$ PRNG used to select which of the n component distributions will be sampled in each cycle.
2. A lookup table containing the n $(fc_{offset}, fc_{width})$ pairs, which select the mean and variance of sampled rectangles.
3. An expander which takes fc_{width} and expands it to the right number of ones, producing fc_{mask}. This is implemented using a small LUT-based ROM.
4. A $U[0,2^{mw})$ PRNG used to provide uc, the variate used to generate a sample from within the chosen target distribution.
5. A bitwise-and between the expanded mask and uc, transforming the variate from $U[0,2^{mw})$ to $U[0,2^{fc_{width}})$
6. An adder, which adds the fc_{offset} to the adjusted variate, transforming it to a $U[fc_{offset}, fc_{offset} + 2^{fc_{width}})$ variate.

The sequence of values produced by this sampling process is distributed according to C (the PDF described at the start of this Section), and assuming that the selection of component distributions is correct, can approximate any target distribution. Few distributions can be *exactly* approximated with this method, as most interesting distributions cannot be tiled exactly with a finite number of rectangles. However, by using a large number of rectangles the distribution of the composite will start to approach that of the target distribution. The method used to select the mean and variance of the component distributions is described in Section 4.

Using rectangular distributions allows for a very space efficient implementation, but has the disadvantage that approximations to continuous PDFs will actually consist of many small up and down steps. With n components an approximation can have at most $2n$ steps in the PDF, so when the width of the output value is large relative to n it will be necessary to approximate gentle probability gradients with large areas of constant probability followed by abrupt jumps up and down. To solve this problem, the composite must be formed from component distributions with sloping sides, rather than abrupt steps.

One option is to use symmetric triangular distributions. These are easily formed by taking two uniform variates and

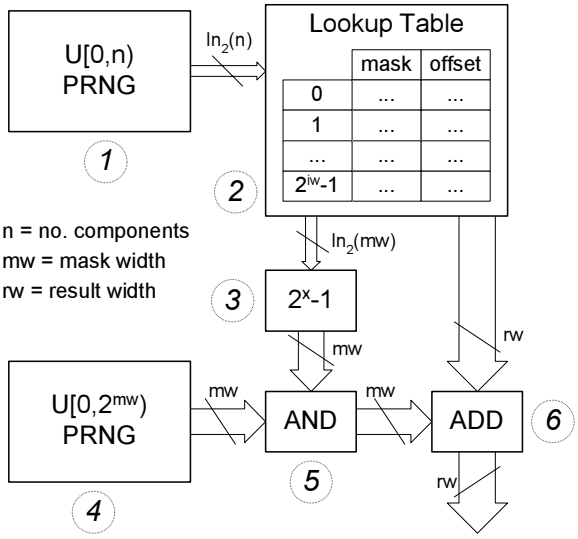

Figure 1. Architecture of hardware distribution generator

adding them together. Give two continuous uniform variates $X, Y \sim U(0,a)$ then the PDF of $T = X + Y$ will be:

$$P(T=x) = \begin{cases} x/a^2 & \text{if } 0 \le x \le a \\ (2a-x)/a^2 & \text{if } a < x < 2a \end{cases} \quad (6)$$

This means that if two a bit wide uniform variates are added together then the resulting distribution will be a discrete triangular distribution ranging from 0 to $2a-1$, with a single mode at a. The same technique of bit masking used to adjust the variance of the rectangular distribution can be used here, by masking both of the uniform variates. For w uniform random input bits a range of triangular distributions with widths from 1 up to $(w+1)/2$ can be generated. So using triangular distributions requires double the number of random bits and an adder, but has the benefit of being able to produce smooth slopes in the output PDF.

An advantage of triangular elements is that they scale up to higher output range resolutions. A gentle gradient that is approximated at a low output resolution will still provide a smooth slope if the output range and triangle resolutions are increased. This is not always the case, for example the peaks of PDFs will resolve into a row of tiny points as the range resolution is increased, but over most continuous distributions the curve will stay a curve.

Unfortunately the triangular distribution, while supporting slopes, cannot produce the vertical PDF steps available with rectangular components. An area efficient way of allowing both slopes and steps is to extend the technique used to generate triangular distributions, by adding two different uniform distributions together. Given two continuous uniform variates $X \sim U(0,a)$ and $Y \sim U(0,b)$ where $a \le b$

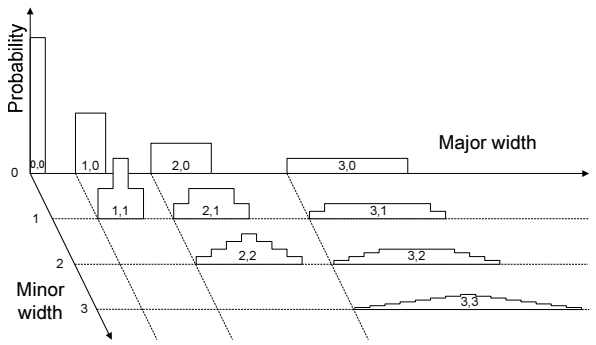

Figure 2. Trapezoid distributions with varying major and minor widths

then the PDF of $M = X + Y$ will be:

$$P(M = x) = \begin{cases} x/ba^2 & \text{if } 0 \le x < a \\ 1/b & \text{if } a \le x < b \\ \frac{(a-x)}{ba^2} & \text{if } b \le x < a+b \end{cases} \quad (7)$$

The two parameters a and b describe a family of trapezoids, or more correctly, truncated pyramids. This is shown in Fig. 2. The larger of the two parameters, here called the major width, controls the overall variance of the distribution, and indirectly the height. The smaller parameter, the minor width, controls the "pointiness" of the distribution, allowing both rectangular and triangular distributions to be used, as well as a range of intermediate shapes.

There is very little extra hardware cost for supporting trapezoid distributions over symmetric triangular distributions. The main change is that the lookup table containing the mask and offsets now needs to hold two masks, increasing the width of each element to $rw + 2\lceil \ln_2 mw \rceil$. This also means that two independent mask expanders (small LUT-based ROMs) are now needed. Apart from these two changes no extra hardware is required, and steps and slopes are both now available.

When element tables are stored in on-chip RAM blocks the dual access ports of most such RAMs allows certain advantages and improvements. The most immediate benefit is that each element lookup table can be shared between two distribution approximators, halving the number of block RAMs needed per unit. The two units can share the same table, generating the same distribution, or can operate out of different regions of the RAM to provide different distributions.

Another way that the proposed random number generator architecture can be used is to provide data dependent distributions. By maintaining distinct areas within the element lookup table, a different distribution can be sampled on each cycle. For example if a 2^k sized table is split into 4 sections then two of the table index bits can be used to select between distributions with no overhead. These distributions might be unrelated, or the index can be used as a runtime distribution parameter. For example, CCD pixel noise is partially dependent on light level, so when modelling the effects of this noise the value of incoming pixels can be used to affect the noise applied to them.

A key advantage of this generator architecture is that the distribution can be changed, simply by changing the constants stored in the table. This can be done even while the generator is running, either by using a spare port to change constants in a dual-ported table RAM, or by modifying the generator circuitry so that when a new constant is written into the RAM the generator will use that value for the current output cycle. As the table entries must be altered one-by-one, the distribution produced will morph from the original distribution to the new one, and by carefully selecting the order in which the elements are changed the intermediate distributions can be made to produce smooth transitions. This feature can be used to make time-varying distributions, for example by gradually replacing coefficients in the table from a larger external RAM bank or storage.

For applications where a one-cycle transition is needed between distributions, the element table can be double-buffered, with constants loaded serially into an alternate RAM. Once the constants are loaded the generator can flip between the RAMs, changing the distribution in a single cycle. Using dual-ported RAM this can be achieved even with a single RAM.

The simplicity of the algorithm and it's hardware implementation suggests many possibilities, but it is useless without some way of organising the component distributions so that they form a reasonable approximation to the target distribution. In the next section the method developed for creating the approximation tables is explained.

4. Approximation of Target Distributions

To generate samples from a given target distribution, the architecture outlined in the previous section requires data tables, which control the mean, variance, and (in the case of trapezoids) shape of the component distributions. The tables for each new target distribution must be precalculated in software, and are then available to be loaded into the hardware random number generators RAM at runtime. This section explains the process used to generate the data tables, and shows some features of typical approximated PDFs.

In the proposed hardware architecture, the components occur with fixed probability, and the number of components to be used is fixed. If a component's distribution has the right width but the wrong height, it cannot be stretched vertically to make it fit better: the only way of reducing the height of the component is to increase the width. The

area (or total probability), of each element must be preserved. Equally, if a good approximation has been reached using $2^n - 1$ elements, it is impossible to stop at that point, as there is no easy or efficient way of wrapping the index round at any number of elements other than 2^n. The only way of achieving a good fit is by having large amounts of overlap between elements, stacking them on top of each other where needed to achieve height. If n elements need to be positioned within an output range of bit-width r, then the number of possible configurations for all elements is $\approx r2^r n!$. With heuristics this number could be reduced, but it is still too large to brute force for useful values of r and n, particularly as the evaluation of each configuration is non-trivial.

A simple approximation strategy was developed, which produces usable results in a reasonable amount of CPU time (between seconds and hours), using a completely automated process. Rather than trying to incorporate a method for avoiding local minima into the search algorithm, the strategy is simply to select a single element within the composite, then try to reposition that element to provide a lower overall error (using one of the error metrics described below). If the error is lower, the new composite is selected, otherwise the change is discarded. So the search strategy is just to select any candidate configuration that has a lower error than the current configuration, regardless of whether this results in getting stuck in a local minima.

The element to be repositioned is selected using one of three processes: **Worst Element**: for each element the approximation error is calculated if that element is removed. The element that causes the largest reduction in error due to it's absence is repositioned to minimise overall error. If no better configuration is found, then the next worst element is considered, and so on. This is particularly effective during the initial stages of curve fitting. **Worst Point**: The contribution of each point in the range to the error metric is calculated, then the points are sorted by decreasing error. All the elements that contribute some probability to the worst point are then repositioned from largest contributor to smallest contributor. **Random Element**: An element is selected at random, then repositioned to minimise overall error. This is a good polishing technique once the approximation is already close to a final solution.

The automated process uses each of these strategies in turn, switching to the next strategy when the number of iterations without improvement exceeds a heuristically derived threshold. The solution found through this fully automated strategy will often be good enough by itself, but if the approximation is not deemed good enough then the operator can perturb a small set of elements, or change the search strategy, forcing the algorithm to start re-optimising the curve.

The application supports a large number of error metrics, which can be changed at runtime to alter the search process. The two most effective have been found to be sum of relative error and sum of cubed absolute error, respectively:

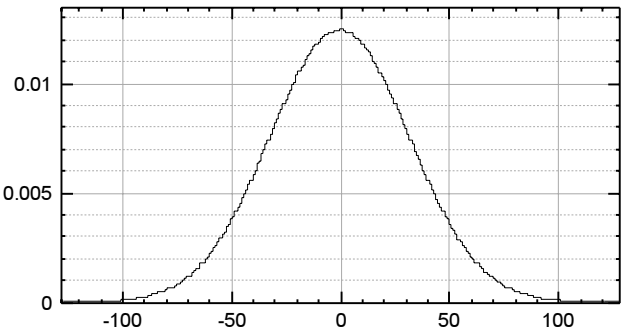

Figure 3. PDF of approximation to a Gaussian distribution over $\pm 4\sigma$ using 1024 rectangles with an 8 bit range.

$$\sum_i \frac{|o_i - e_i|}{e_i} \qquad \sum_i |o_i - e_i|^3 \tag{8}$$

where o is the array of approximated PDF probabilities, while e is the equivalent array of probabilities from the target PDF. The sum of relative error has been found to be very effective at approximating the tails of distributions, but if very few elements are used will often neglect the peaks. The sum of cubed error behaves in the opposite way, and will concentrate on the largest absolute errors, often found near the distribution peaks, but will ignore the tails once they fall below some probability threshold.

Fig. 3 shows an approximation to the Gaussian distribution, using 1024 rectangles over an 8 bit range. For programming convenience the Gaussian variance is scaled so that all the discrete values in the range correspond to an integer, so 32 units on the scale represents one standard-deviation (σ). Once in hardware the integers will be interpreted as fixed point numbers, and the stepped line shows the exact probability of each output value.

Fig. 4 zooms in on the peak of the curve, and also shows the curve of the target PDF (the target distribution is also quantised into the same output values as the approximation, but it is shown as a continuous PDF to make the two easy to distinguish). The slight irregularities in the approximation are visible as uneven steps. Also note that the distribution is not perfectly symmetric, which could be critical for some applications. Constraints such as symmetry or perfect mean/variance can be applied in the search process if these properties are needed (for example, elements can be displaced as symmetric pairs).

Fig. 5 shows a close-up of the distribution's tail. This demonstrates one of the problems with this technique when

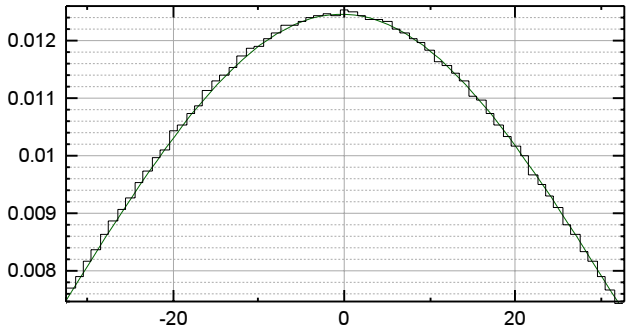

Figure 4. Zoomed section of peak of PDF from Fig. 3 over $\pm 1\sigma$.

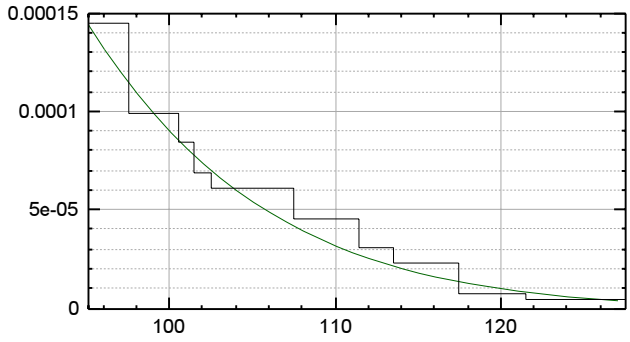

Figure 5. Zoomed section of tail of PDF from Fig. 3 from 3σ to 4σ.

attempting to approximate low probability values, as at some point the element shape with minimum height is reached. At this point the relative error of the approximation starts to increase sharply, as the approximation error approaches the target value. The approximated tail can be extended out further by increasing the number of elements, which reduces the height of all elements, allowing thinner shapes to be used. Unfortunately doubling the number of elements only halves the minimum probability that can be produced. Between 4σ and 5σ the Gaussian probability drops by two orders of magnitude, so it would require roughly four times as much RAM to extend the tail by one σ. One solution is to use non-uniform probabilities on elements to allow the creation of elements with a range of heights, but that possibility is not explored in this paper.

5. Results

In this section, we examine the performance and resource usage of the approximation unit in hardware, and assess the quality of the resulting distribution approximations.

The hardware architecture was implemented using Handel-C, synthesised using Synplify, and then placed and routed using ISE 6.1 to produce Virtex-2 configurations. Table 1 shows the hardware speed and resource usage for different combinations of parameters. Note that the results include the area required for all uniform random numbers required by each unit. These are provided by a Xorshift based generator [11] with a period of 2^{128}, which takes approximately 113 slices in isolation.

Elements	Bits	Slices	MHz	LUTs	FFs
Rect×16	10	121	220	85	217
Tri×16	10	136	210	89	244
Trap×16	10	137	211	90	244
Rect×1024 - B	14	144	215	105	237
Tri×1024 - B	14	160	212	104	271
Trap×1024 - B	10	188	208	120	310
Rect×512 - B	24	267	205	156	452
Tri×512 - B	24	253	198	157	449
Trap×512 - B	24	270	195	220	526

Table 1. Performance and resource usage of approximation units in the Virtex-II architecture, using different numbers and types of component distributions. Rows marked B use a single block ram, otherwise LUT ram is used; Bits is the generator output width.

The first group of generators uses small LUT based RAMs to hold the element tables rather than block RAMs, and represent the case where only a very crude approximation is required. The second group uses a single block RAM, allowing more elements for a better approximation, while the third group increases the generator output width. The speed remains around 200MHz for all configurations, without any specific optimisations for speed, meaning that each generator will be able to provide 200M samples/s.

We now present results on the quality of the samples that composite generators can provide. The results presented here should be seen as benchmarks of the current approximation system, rather than of composite generators themselves, as improved methods for creating approximations may improve quality, without requiring any changes to the underlying generator architecture used in hardware.

Unless otherwise specified the approximations used in this section were generated automatically without any human interaction. An arbitrary time limit was also imposed on the approximation process, dependant on the number of elements being used. For an approximation with k elements, the approximation system was allowed $\sqrt{k}$ minutes of CPU time on a 2.4GHz Athlon with 2GB of RAM. This means that, for example, approximations with 8 elements have about 3 minutes CPU time, while those with 2048 elements get about 45 minutes. The χ^2 tests used here were performed by bucketing n samples at the range resolution,

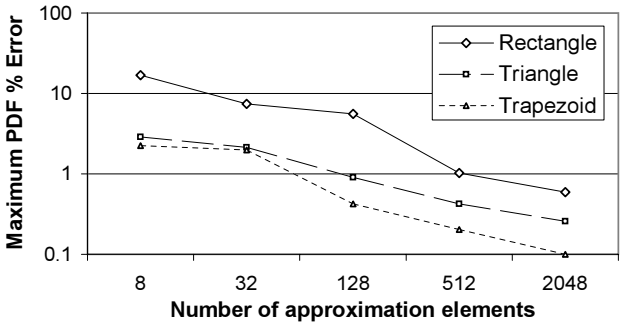

Figure 6. Approximation accuracy when using different component types to approximate the Weibull distribution.

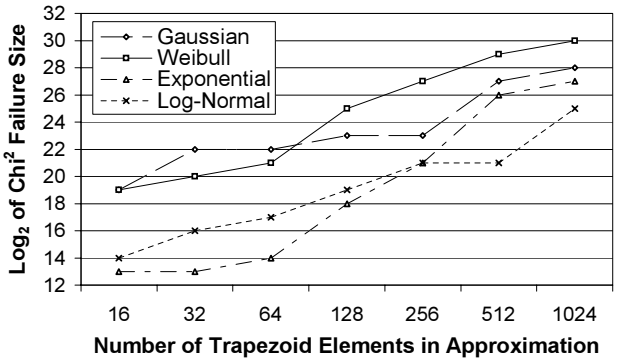

Figure 7. Number of samples that can be taken from trapezoid approximations without failure of the χ^2 test (at the 5% significance level).

then coalescing adjacent buckets until the expected count per bucket exceeds $\frac{n}{n^{0.4}}$.

Fig. 6 shows the maximum PDF error found while approximating a Weibull distribution for different component shapes and numbers of components The error shown is the maximum absolute error, expressed as a percentage of the largest probability in the target PDF. The rectangular approximation is at a disadvantage with any smooth continuous distribution like the Weibull, and this shows as a very poor error for 8 blocks, but gradually improving for larger numbers of blocks. The triangular and trapezoid approximations start off at a lower error, due to the ability to match the coarse features of the distribution even with a low number of components. Both the rectangular and trapezoid approximations roughly half the maximum PDF error for each quadrupling of the component count, while the triangular approximation does slightly worse, with the error dropping by about a third each time.

In Fig. 7, the χ^2 test is used to determine how many samples can be take from four different distributions before failure. All the distributions were approximated using

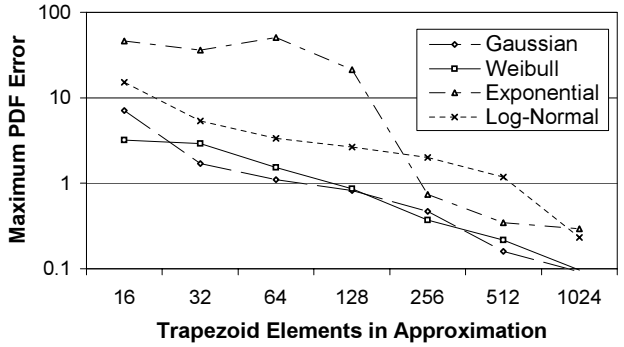

Figure 8. Accuracy of trapezoid approximations with different numbers of elements for four different target distributions.

trapezoid components. The approximation to the Weibull distribution appears to be much more effective than all the others, probably because the mode is more extended and smoother than the other distributions. The Exponential distribution causes difficulty for the approximation system, as the system could not place triangles at the mode to approximate the steep slopes without the triangles extending into negative numbers (where the Exponential PDF is zero), which leads to a poor approximation in the part of the graph with highest probability. The maximum PDF error as percent of maximum PDF for the same set of generators is also shown in 8, where it is clear up until 256 components are used the approximation error is very large.

To explore how effective the method can be if accuracy concerns override speed and area, a more complex approximation to the Gaussian distribution is developed. This uses 2048 trapezoids, with a 16 bit output range, requiring 3 block RAMs to implement the 2048x24 bit table. The approximation process was also allowed enough CPU time to reach a steady state (about four hours), and where necessary the search process was manually tweaked. This results in a distribution that extends to about $\pm 5.1\sigma$, beyond which it is impossible to approximate small enough probabilities with equal probability trapezoids. The maximum absolute PDF error is 3.4×10^{-07}, or 0.34% of the peak probability. This generator is able to to provide 2^{27} samples before failing the χ^2 test.

Using the same approximation data, two separate generators are then executed using a shared table, and the two output samples are combined, taking advantage of the Central-Limit-Theorem to improve quality. Rather than combination through addition, the samples are combined using subtraction, as this removes any asymmetries of the single distribution. The combined PDF is non-zero within $\pm 7.2\sigma$), and is much more accurate, with the generator able to pass the χ^2 test using 2^{30} samples.

Table 2 shows a comparison between methods for creating Gaussian samples, including previous hardware implementations, component generator based hardware (referred to as "Trapezoids" for the rest of this section), and software implementations. The existing hardware and software methods are all specialised for the Gaussian distribution, while (except for the CLT component generator), the component generators are generic and can produce any distribution. All the generators are implemented in the Virtex-II family, except for the Ziggurat, which uses Virtex-II Pro. The "Dual" Trapezoid generator is the case where two generators are sharing a dual-ported RAM. The reported resource usage, speeds, and number of samples before χ^2 were extracted from the papers where they were presented, except for the χ^2 failure point for the Box-Muller, which was reported in [9]. The software speeds were measured on a 2.1GHz Athlon with 2GB of RAM.

The Trapezoids all use less logic than the other hardware methods, although the CLT based generator is only just smaller than the Box-Muller. However, the Box-Muller uses 5 block multipliers, and all three use at least two block multipliers. Similarly, all the Trapezoids use less block RAMs than the Gaussian generators. When slices, multipliers and block RAMs are all considered together it is clear that the Trapezoids use much less area, and will put less pressure on resource usage in any design that uses random number generators.

The speed of the Trapezoids also compares very well with the other hardware methods. Although the Box-Muller is actually 50MHz faster, it uses explicit placement and device specific features to achieve the best speed. A better comparison is with the Wallace and Ziggurat designs, which were implemented using Handel-C and VHDL respectively, and so use a very similar tool chain to that used to create the composite generators. The Wallace is approximately 40MHz slower than the composite, or about 80% of the Trapezoid speed. Although the Ziggurat appears to be a little faster than the Wallace, it should be noted that this was implemented using a Virtex-II Pro, which provides a small intrinsic speed advantage over the Virtex-II, and so on equal terms is probably about the same speed. Most applications that consume non-uniform random samples will proceed to apply numerical operations to the samples, so it is unlikely that 200MHz will be a bottleneck, but the "Dual" generator, where the RAM table is shared between two generators, can provide 392 MSamples/s.

Comparing the generators on quality is very difficult, as methodologies vary between papers. The testing performed on the Wallace generator is by far the most extensive, and it clearly passes the χ^2 test convincingly for more than 2^{32} samples, producing extremely high quality samples. In the same paper, independent testing is also performed on the Box-Muller generator (the original documentation does not include any results), and it is concluded that the generator has serious deficiencies, failing at just 2^{17} samples. This mainly seems to be cause by a large spike in the PDF at the origin, and is likely a symptom of a bug rather than any flaw in the architecture or methodology.

The quality of the Trapezoid generators is their weakest point: even with the Central-Limit extension only 2^{30} samples can be generated before failure. At 200MHz this represents only about 5.5 seconds worth of output. Although the quality is better than that of the Gaussian specific Box-Muller generator, a bug-free Box-Muller would be expected to provide samples of at least or higher quality. However, for many applications the provided quality may be good enough, and future improvements to the approximation process can be expected to improve the quality.

When compared with software generators the main advantage of the Trapezoid generators is that they provide over twice the sample generation rate. However, for other distributions the difference may be much greater, as not all distributions have received the attention and optimisation of the Gaussian. Analytically defined distributions specific to a single task or empirically defined distributions will be much slower in hardware, having to rely on one of the arbitrary distribution generation methods available in software.

6. Summary

This paper has demonstrated the viability of a method for hardware approximations of arbitrary probability distributions in hardware. The hardware architecture has been shown to be competitive with traditional methods for non-uniform random number generation in hardware, with lower area and higher speed than most Gaussian specific random number generators. A technique for creating approximations to arbitrary probability distributions has also been presented, and statistical quality for sample sizes up to 1 billion variates has been shown. The key advantage of this technique over other hardware non-uniform random number generation techniques is that the hardware design is independent of the distribution to be produced, and the distribution can be changed at runtime.

Future work includes:

- Improvements to the approximation search process, as well as an attempt to determine how close to the optimal solution the approximations actually are.
- Investigation of non-uniform probability weighting of elements to improve approximation accuracy.
- Evaluation of accuracy-resource tradeoffs and overall quality for real-world distributions and applications.

Class	Previous Hardware Methods			Trapezoid Composition				Software	
Method	Box-Muller	Wallace	Ziggurat	Single	Single	Dual	CLT	Ziggurat	Wallace
Reference	[19]	[9]	[20]	-	-	-	-	[13]	[17]
Elements	-	-	-	512	2048	2048	2048	-	-
Width	24	24	35	24	16	16	17	-	-
Speed (MHz)	245	155	168	195	193	196	194	2.1GHz	2.1GHz
Slices	455	895	891	270	256	411	451	-	-
RAMs	5	7	4	1	3	3	3	-	-
DSPs	5	4	2	0	0	0	0	-	-
Rate (MS/s)	245	155	168	195	193	392	194	37	81
χ^2 Fail	2^{17}	$> 2^{32}$	$> 2^{30}$	2^{26}	2^{27}	2^{27}	2^{30}	$> 2^{32}$	$> 2^{32}$

Table 2. Comparison of different methods for generation Gaussian random numbers.

References

[1] R. Andraka and R. Phelps. An FPGA based processor yields a real time high fidelity radar environment simulator. In *Conference on Military and Aerospace Applications of Programmable Devices and Technologies*, 1998.

[2] G. E. P. Box and M. E. Muller. A note on the generation of random normal deviates. *Annals of Mathematical Statistics*, 29(2):610–611, 1958.

[3] J. A. Buzacott and J. G. Shanthikumar. *Stochastic Models of Manufacturing Systems*. Prentice Hall Professional Technical Reference, 1992.

[4] R. L. Cook. Stochastic sampling in computer graphics. *ACM Transactions on Graphics*, 5(1):51–72, 1986.

[5] J.-L. Danger, A. Ghazel, E. Boutillon, and H. Laamari. Efficient FPGA implementation of gaussian noise generator for communication channel emulation. In *IEEE International Conference on Electronic Circuits and Systems*, pages 366–368. IEEE Computer Society Press, 2000.

[6] W. Hörmann and J. Leydold. Continuous random variate generation by fast numerical inversion. *ACM Transactions on Modeling and Computer Simulation*, 13(4):347–362, 2003.

[7] J. Lawrence, S. Rusinkiewicz, and R. Ramamoorthi. Efficient BRDF importance sampling using a factored representation. *ACM Transactions on Graphics*, 23(3):496–505, 2004.

[8] D.-U. Lee, W. Luk, J. D. Villasenor, and P. Y. Cheung. A gaussian noise generator for hardware-based simulations. *IEEE Transactions On Computers*, 53(12):1523–1534, december 2004.

[9] D.-U. Lee, W. Luk, J. D. Villasenor, G. Zhang, and P. H. Leong. A hardware gaussian noise generator using the wallace method. *IEEE Transactions on VLSI Systems*, 13(8):911–920, 2005.

[10] J. Leydold. Short universal generators via generalized ratio-of-uniforms method. *Mathematics of Computation*, 72(243):1453–1471, 2003.

[11] G. Marsaglia. Xorshift RNGs. *Journal of Statistical Software*, 8(14):1–6, 2003.

[12] G. Marsaglia and W. W. Tsang. A fast, easily implemented method for sampling from decreasing or symmetric unimodal density functions. *SIAM Journal on Scientific and Statistical Compututing*, 5(2):349–359, June 1984.

[13] G. Marsaglia and W. W. Tsang. The ziggurat method for generating random variables. *Journal of Statistical Software*, 5(8):1–7, 2000.

[14] J. M. McCollum, J. M. Lancaster, D. W. Bouldin, and G. D. Peterson. Hardware acceleration of pseudo-random number generation for simulation applications. In *IEEE Southeastern Symposium on System Theory*, pages 299–303, 2003.

[15] N. Metropolis and S. Ulam. The monte carlo method. *Journal of the American Statistical Association*, 44:335–341, 1949.

[16] M. N. O. Sadiku and M. Ilyas. *Simulation of Local Area Networks*. CRC Press, 1994.

[17] C. S. Wallace. Fast pseudorandom generators for normal and exponential variates. *ACM Transactions on Mathematical Software*, 22(1):119–127, 1996.

[18] M. J. Wichura. Algorithm AS 241: The percentage points of the normal distribution. *Applied Statistics*, 37(3):477–484, 1988.

[19] I. Xilinx. Additive white gaussian noise (AWGN) core. CoreGen documentation file, 2002.

[20] G. L. Zhang, P. H. Leong, D.-U. Lee, J. D. Villasenor, R. C. Cheung, and W. Luk. Ziggurat-based hardware gaussian random number generator. In *International Conference on Field Programmable Logic and Applications*, pages 275–280. IEEE Computer Society Press, 2005.

[21] G. L. Zhang, P. H. W. Leong, C. H. Ho, K. H. Tsoi, D.-U. Lee, R. C. C. Cheung, and W. Luk. Reconfigurable acceleration for monte carlo based financial simulation. In *International Conference on Field-Programmable Technology*, pages 215–224. IEEE Computer Society Press, 2005.