# Histogram-Based Probability Density Function Estimation on FPGAs

Suhaib A. Fahmy

*School of Computer Engineering*
*Nanyang Technological University*
*Nanyang Avenue, Singapore*
sfahmy@ntu.edu.sg

***Abstract*—Probability density functions (PDFs) have a wide range of uses across an array of application domains. Since computing the PDF of real-time data is typically expensive, various estimations have been devised that attempt to approximate the real PDFs based on fitting data to an expected underlying distribution. As we move to more adaptive systems, real-time monitoring of signal statistics increases in importance. In this paper, we present a technique that leverages the heterogeneous resources on modern FPGAs to enable real time computation of PDFs of sampled data at speeds of over 200 Msamples per second. We detail a flexible architecture that can be used to extract statistical information in real time while consuming a moderate amount of area, allowing it to be incorporated into existing FPGA-based applications.**

## I. Introduction

Information on the probability density function (PDF) of sampled data has a wide range of uses across a variety of application domains. A variety of techniques have been proposed over the years [1], however computational complexity often means these cannot be implemented in real time. The preference is thus to rely on techniques that use a reduced data set, attempting to fit to a predefined parametric model. This can be inaccurate, especially if the PDF of the data is unknown or changing.

In most references to applications of PDF estimation in the literature, one-time statistics are computed on a block of data. In image processing applications, for example, the statistics are typically required for a single frame. In this paper, we are more interested in facilitating real-time monitoring of signals that may change over time. Specifically, adaptive systems may need to monitor changes in certain environmental factors, before making changes based on those statistics. An example might be a cognitive radio that reacts to channel occupancy statistics, or an adaptive networking node that may modify its routing behaviour based on network queue length statistics. In order for such applications to be feasible, we must be able to compute PDFs in real-time on streaming data.

In this paper, we present a PDF calculation architecture that can be integrated within FPGA-based applications, which is both flexible and moderate in terms of area usage. The aim is to allow designers of applications that can make use of PDF estimation to leverage this capability within their hardware designs. The primary focus is for adaptive applications, that we feel are becoming more practical on FPGAs.

In Section II, PDF estimation is discussed along with existing hardware approaches. Section III introduces the proposed architecture used for PDF calculations and how it has been tailored for FPGA implementation. Section V discusses some extensions to the architecture that allow it to compute more complex functions. In Section IV we present synthesis results and discuss accuracy issues. Finally, Section VI concludes by discussing further development of this architecture.

## II. Related Work

PDF estimation techniques fall into two categories: parametric and non-parametric. [1] Parametric techniques try to fit a known model to the data and deduce values for the model parameters based on the data. Non-parametric techniques use the samples themselves to construct the PDF. The most common non-parametric technique is the use of a histogram, which when normalised, gives the PDF. Given a good model, parametric techniques can give more accurate results with less data than a non-parametric model. However, the requirement for a good model means that where the PDF's nature is unknown or changing, parametric approaches can result in poor accuracy.

PDF estimation is of value in a number of different applications areas, including image processing [2], [3], machine learning [4], computer security [5], and medical imaging [6] among others.

Existing work on hardware architectures for PDF estimation is minimal. In [7] a histogram of an image is constructed for histogram equalisation. However, since this is done for one image at a time, they process accumulation at the end of histogram construction, during the blanking period between frames in a video, using a sequential approach. This would not be suitable for a constantly updating window. In [8], a novel technique for constructing histograms is shown, but with an application to median computation. In this paper, we use a similar approach to construct the histogram, but tailored to PDF calculation. In [3] a histogram is constructed within a hardware architecture but using a sequential approach, again on a per-image basis.

Elsewhere, in [9], a Parzen window PDF estimator is used as an example application for their performance migration tool. However it processes a single block of data loaded from a host PC, and the performance comparison is made between

**978-1-4244-8983-1/10/$26.00 ©2010 IEEE**

single- and dual-core acceleration of a mostly sequential implementation.

The architecture presented in this paper allows for high-speed non-parametric PDF estimation based on the histogram method. It is targeted at applications with changing PDFs or where the model is not known in advance. The work in this paper extends beyond simply constructing a histogram to extracting a variety of statistical metrics in real time from the histogram. We also describe how the architecture can be adapted to kernel-based density estimation in Section V. Kernel-based estimation can give more accurate results from a reduced dataset.

## III. Parallel PDF Architecture

### A. Outline

The architecture presented here accepts sampled data at its input and computes the cumulative density function of that data, which is stored internally. By normalising by the window size, which can be changed to any power of 2, the resultant information contained in the circuit represents the probability density function. The desired output function of the PDF can then be chosen as required for a specific application. It is possible to compute $p(x = a)$ (where $x$ is a candidate sample value and $a$ is a given value), $p(x > a)$, $p(a < x < b)$ as well as percentile calculations. These do not impact the complexity of the architecture in any significant way, and the architecture is built to enable selection of the required function at runtime.

### B. Computing the PDF

In order to compute the PDF of sampled data, we are required to keep a tally of the number of occurrences of different input values. Doing this for all possible values would result in a histogram. Normalising a histogram, by dividing by the number of samples gives the PDF. In hardware, we are able to compile a histogram of input data using the technique previously presented in [8]. This involves keeping an occurrence counter for each possible input value and incrementing it when that value is seen at the input.

A hardware architecture containing a set of counters, one for each bin, and each enabled only when the corresponding value occurs at the input, would suffice for this. The number of bins impacts resource utilisation linearly in this scheme. For median calculation, we required a bin to represent each possible input value. For PDF estimation, the number of bins is typically less, since it is the shape that is of most interest, and a reduced number of bins leads to a "smoother" PDF. Analytical techniques described in [1] can be used to find the number of bins that would provide sufficient accuracy given some assumptions about the input data.

If we calculate that $2^l$ histogram bins is sufficient, we simply use the $l$ most significant bits of the input samples to address the histogram counters. This results in an architecture similar to the one shown in Fig. 1, where the *bin_en[]* signal determines whether a specific bin should be incremented. Since each bin can be addressed in parallel, this architecture could function on a continuous stream of data.

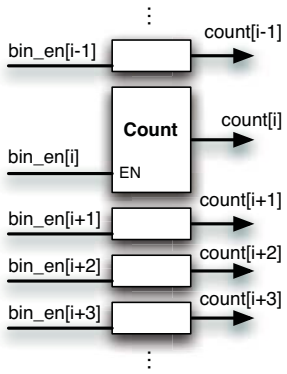


Fig. 1. Basic histogram bin structure.

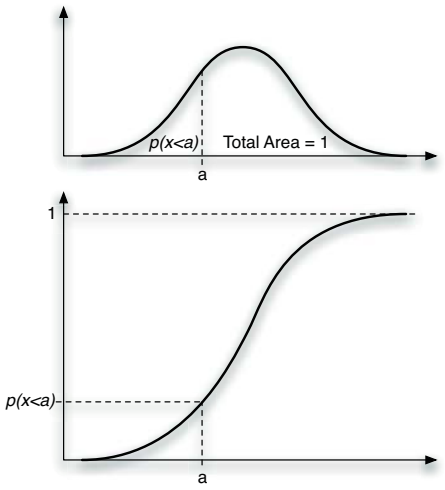


Fig. 2. Comparison of probability calculation from histogram and cumulative histogram.

While the circuit described above would give the PDF of the input samples, extracting meaningful statistics from this PDF involves further calculations. A $p(x = a)$ calculation would be straightforward, by looking up the count value at index $a$. However, more useful statistics can be gained by using the cumulative distribution function (CDF). The CDF of a set of data is simply an accumulation of the PDF values at each bin. Hence at index $a$, the count value would be equal to the sum of PDF bin values for all indexes from 0 to $a$. The CDF is, by definition, a monotonic function. Fig. 2 shows how using a PDF, anything other than the exact probability for a single index is represented by an area. Computing this in real time would require summation of all the necessary bins, which would be an expensive operation. With a cumulative histogram, a single lookup gives the probability for a range of values. To constrain the range on both sides, would only require one more lookup and the probability would be the difference between the two.

However, the CDF allows for more interesting statistics to be extracted in real time. For example $p(x < a)$ is now simply

the bin value at index $a$. Similarly $p(a < x < b)$ would be the bin value at index $b$ minus the bin value at index $a$. We can also extract percentiles, that is the value of $c$ such that $p(x < c)$ equals some given value, as will be discussed shortly.

To compute the cumulative histogram in real time, the architecture must be able to update all the necessary bins for each input sample in a single step. An input sample of value $x$ should increment the corresponding bin with index $x$ and all subsequent bins. This could be achieved by using a comparator as the enable signal for each bin counter. If the bin index is greater than or equal to the input sample, it should enable the incrementation, otherwise not. However, for an architecture with hundreds of bins, this would be costly in hardware terms.

Another approach previously exploited in [8] involves the use of a memory to enable the corresponding bins. This echoes somewhat the now defunct microprogrammed control digital design technique. As each input sample arrives, it is used to address a memory with pre-stored access patterns. The corresponding data is used to enable the necessary bins. For this case, the contents of the memory are very simple as shown in Table I. The embedded memory blocks on modern devices mean this technique is highly amenable to implementation on FPGAs. Furthermore, the sparse layout of these memories, and the fact that large memories are stitched together from smaller ones means routing is not hampered by this centralised control module.

TABLE I
ACCESS PATTERN MEMORY CONTENTS.

| Address | Contents[0:255] |
|---|---|
| 0 | 0xFFFF$\cdots$FFFF |
| 1 | 0x7FFF$\cdots$FFFF |
| 2 | 0x3FFF$\cdots$FFFF |
| 2 | 0x1FFF$\cdots$FFFF |
| 4 | 0x0FFF$\cdots$FFFF |
| 5 | 0x07FF$\cdots$FFFF |
| $\vdots$ | $\vdots$ |
| 253 | 0x0000$\cdots$0007 |
| 254 | 0x0000$\cdots$0003 |
| 255 | 0x0000$\cdots$0001 |

It would be fair to require the histogram to be calculated over a fixed-length sliding window. This would allow for continuous monitoring of the required data in real time. In order to allow this, a FIFO buffer of length equal to the window must be instantiated. This buffer keeps track of the samples in the window. As samples stream through, the oldest samples emerge at the other end of the FIFO and can be removed from the histogram. We only need to store the $l$ most significant bits of each input sample since that is the only portion used to address the bins.

To remove an old sample from the histogram, its corresponding bins must be decremented. By using the memory approach described above, this becomes straightforward. Embedded Block RAMs on Xilinx FPGAs have dual-port capability; this allows two addresses to be read in the same clock cycle. By extracting the access patterns for the new and oldest sample, it is possible to update the histogram in a single step by considering which bins need to be incremented, which should be decremented and which should be left as is. Further detail on this particular optimisation is given in [10]. The input to each bin is now not a count enable but rather a signal that either increments, decrements or maintains the count value.

In order to normalise the histogram, to compute the PDF, we need to divide the count values by the window length. Since division by arbitrary amounts is expensive in hardware, we restrict the window length to be a power of 2. Hence, we can accommodate window lengths such as $8K$, $16K$, $32K$, $64K$ and so on. Then normalisation is simply a case of shifting the binary count value to the right by the corresponding number of bits, $n$, where $2^n$ is the window size. So for a window size of $8K$ the count value of a particular bin is shifted right by 13 bits. This is equivalent to shifting the location of the binary point up by 13 bits, which is preferred since it maintains precision. Rather than lose any precision in the counter itself, all scaling is only done at the output of the circuit.

### C. Computing Centiles

Centiles are a useful tool in statistical analysis. For a dynamic system, centiles allow for quality of service oriented decisions. For example, in a networking context, a router might decide that 80% of packets should wait in a queue for less than a given threshold of time. In a cognitive radio, we might decide that a frequency channel is vacant if 95% of the activity in that channel falls below a certain noise margin. Given a cumulative histogram, we can compute these percentiles in real time using a novel technique.

A value at the output of a particular bin counter in a cumulative histogram tells us that the corresponding number of occurrences have values that are less than or equal to that bin's index. Hence, if we have a value of 100 at bin index $b$, that means there have been 100 occurrences of samples with values up to $b$. Given a fixed window size of 1000, that would tell us that value $b$ is the 10th percentile. More generally, if we want the nth percentile, we need to find the index whose count value is equal to or exceeds $N/n$ where N is the PDF window size.

To do this in a hardware architecture, we can use a priority encoder combined with comparators. A priority encoder finds the highest index occurrence of a '1' in a binary word. If we know the value we are looking for, a bank of comparators, one for each bin counter, can be used to compare the bin counts to the required value. This results in a series of zeros for the initial bins, followed by ones for the bin corresponding to the first count to exceed this value, and all subsequent bins. The priority encoder gives the index of the first bin to output a one.

Constructing a priority encoder for hundreds of bits is not straightforward if speed is important. However, a technique in [11] shows how it is possible to pipeline many small priority encoders together into a larger one. Fig. 3 shows how 5 4-bit priority encoders can be combined into a 16-bit priority encoder. Similarly, 9 8-bit priority encoders can be combined

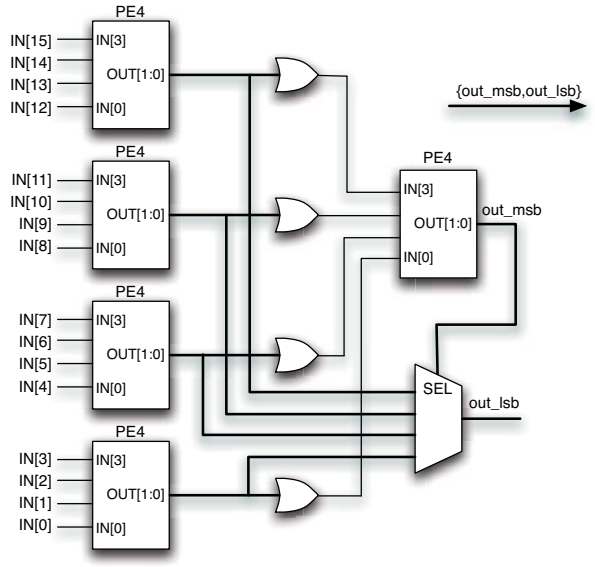


Fig. 3. A 16-bit priority encoder constructed from 5 4-bit priority encoders. The first stage computes the least significant bits of the output while the second computes the most significant bits and selects the required output from the first stage. The output is a concatenation of the two.

into a 64-bit priority encoder. By pipelining between each stage, only a few clock cycles of latency are added, while we are able to clock the circuit significantly faster.

### D. Architecture Design

Fig. 4 shows the overall architecture of the system. Samples enter the system and are fed into a FIFO, the output position of which can be set by the *fifo_length* signal. This offers fixed positions based on how the FIFO is constructed, but is always a power of 2. The output of the FIFO represents the oldest sample emerging from the sampling window. Together with the new sample, they are used to address the two ports of the dual port access pattern memory. The output of this memory tells each bin whether to increment, decrement or maintain the existing value. The outputs of all the bins (*bincount[0:N-1]*) are then passed through to the next stage, where statistics can be extracted.

Two types of statistic calculation are facilitated in this architecture and two of each unit are shown. The first gives a probability range output such as $p(x < a)$. The *prindex* signal is set to the required value of $a$ and the output represents $p(x < a)$. For $p(x > a)$, the value is simply subtracted from the window size. For $p(a < x < b)$, the two *pr_out* values are subtracted from each other. Recall that these counts must be normalised, and hence the position of the binary point is inferred to be $n$ from the left for a window size of $N = 2^n$.

The second type of calculation is the centile. We use the method described in Section III-C to calculate the input value at a given required centile. Since the counters store the histogram counts for all items in the window, the input to the centile calculation, *centval*, must be given as a proportion of the window size, rather than a raw percentile (in the range of 0 to 100). The Comp block compares the count values of all the bins, in parallel, to the required centile value returning a binary one for each bin where the count exceeds the required value. The priority encoder then determines the position of this crossing, giving the index for the required centile. The outputs can be used for monitoring the centiles themselves or inter-centile range, simply by tracking the difference between two centiles.

The architecture shown includes two of each calculation type, however this can be modified based on the application requirements. Further compositions are possible since the histogram data is all stored within the bin counters.

## IV. Results

The above architecture was implemented using VHDL on a Xilinx Virtex-4 XC4VLX80. We selected a maximum configurable window size of $128K$ samples and set the maximum number of bins to 256. We used only the standard bin access pattern and allow for two direct probability and two percentile calculations in parallel, as in Fig 4.

For the FIFO, we used Xilinx embedded memory cores. Since we only need access to specific points within the FIFO queue, FIFOs of length 8K, 8K, 16K, 32K, and 64K were chained together, allowing for window sizes of 8K, 16K, 32K, 64K and 128K, while maintaining maximum speed. For the priority encoder, the technique detailed in Section III-C was used to build a 256-bit module using four 64-bit and one 4-bit priority encoders. The 64-bit priority encoders are created using nine 8-bit priority encoders. For the access pattern memory, a standard Block RAM module in CoreGen was used. It stitches together 8 Block RAMs to create the $256 \times 256$-bit memory.

For the parameters given above, we obtained the area results shown in Table II. The architecture comfortably achieved a 200MHz clock speed.

TABLE II
FPGA resource utilisation.

| Module | LUT/FF Pairs | BlockRAMs |
|---|---|---|
| Access Memory | 100 | 8 |
| Window FIFO | 350 | 64 |
| Centile Unit (ea.) | 620 | 0 |
| Full System | 11,350 | 76 |
| Available | 35,840 | 200 |

The area usage is dominated by the histogram circuitry, which in this case consists of 256 14-bit counters and associated control circuitry for histogram updating. The BlockRAM usage is primarily due to the FIFO. If smaller windows are required, this usage is reduced. We believe the area consumption, while not completely insignificant is sufficiently compact, at around 30% of the chosen device, for use within larger systems. Furthermore, optimisations can be made to reduce area usage, by reducing the number of bins for the histogram, reducing percentile calculations, or reducing the reconfigurability of the window size.

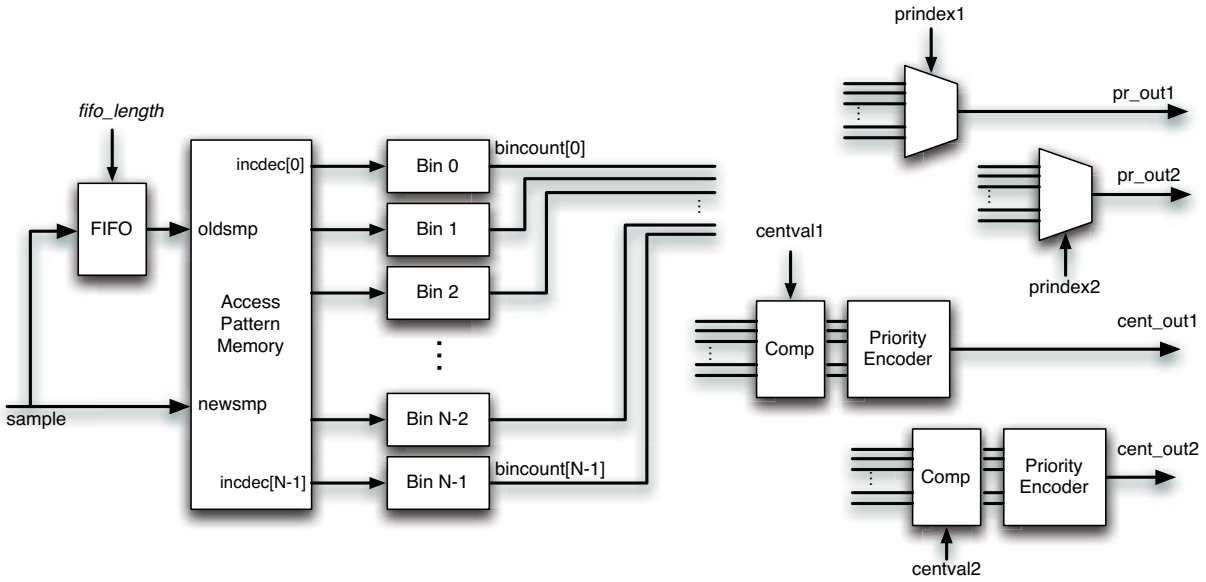


Fig. 4. Complete PDF estimator architecture.

## V. Architecture Extensions

The architecture presented thus far computes the histogram and maintains the real-time statistics over the sliding window. The structure of the circuit enables some interesting modifications that we shall briefly mention here, and which require further investigation.

Using a memory to control the access pattern to the bins for updating the histogram serves as a level of indirection between the samples and bin indexes. Hence, alternative patterns can change the way the input sample values are mapped to the histogram bins, allowing us to analyse data in a non-uniform manner.

It is also possible to model uncertainty in input accuracy by spreading the effect of an input sample into adjacent bins. This mirrors Kernel-based PDF estimation. To enable this, each sample must be able to address the set of bins required for its kernel. Instead of binary increment/decrement of the corresponding bins, the circuit could be modified to allow to addition/subtraction using kernel-based values stored in a larger memory.

More interestingly, in an adaptive system, it would be possible to change the histogram access pattern at runtime based on existing statistics. For example, interquartile range could determine that the centre of the PDF needs to be extended, and an appropriate access pattern could be loaded at runtime.

## VI. Conclusion

We have shown a novel architecture for real time computation of PDF estimates based on the histogram method. It makes extensive use of FPGA resources to parallelise the algorithm. We showed how a cumulative histogram can be constructed in parallel, how statistical properties can be extracted in real-time, and how priority encoders can be used to extract further statistics. We also detailed a technique for building wide priority encoders with high performance. We showed how this architecture could be extended to more advanced PDF calculation methods, and presented results for a single basic configuration.

Future work involves implementing the extensions proposed in this paper and more architectural exploration as well as an analysis of the PDF parameters for various applications.

## References


[1] D. Scott, *Multivariate Density Estimation. Theory, Practice and Visualization*. Wiley, 1992.

[2] S. Phung, A. Bouzerdoum, and D. Chai, "Skin segmentation using color pixel classification: Analysis and comparison," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 1, pp. 148–154, 2005.

[3] B. Wang, S. Qian, and H. Zhou, "A real-time contrast enhancement algorithm for infrared images based on plateau histogram," *Infrared Physics and Technology*, vol. 48, no. 1, pp. 77–82, 2006.

[4] C. Zhang and T. Chen, "An active learning framework for content-based information retrieval," *IEEE Transactions on Multimedia*, vol. 4, no. 2, 2002.

[5] F. Flament, S. Guilley, J.-L. Danger, M. Elaabid, H. Maghrebi, and L. Sauvage, "About probability density function estimation for side channel analysis," in *Proceedings of International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE)*, 2010, pp. 15–23.

[6] A. Moussaoui, K. Benmahammed, N. Ferahta, and V. Chen, "A new MR brain image segmentation using an optimal semi-supervised fuzzy c-means and PDF estimation," *Electronic Letters on Computer Vision and Image Analysis*, vol. 5, no. 4, pp. 1–11, 2005.

[7] X. Li, G. Ni, Y. Cui, T. Pu, and Y. Zhong, "Real-time image histogram equalization using FPGA," in *Proceedings of SPIE*, 1998, pp. 293–299.

[8] S. Fahmy, P. Cheung, and W. Luk, "Novel FPGA-based implementation of median and weighted median filters for image processing," in *Proceedings of International Conference on Field Programmable Logic and Applications (FPL)*, 2005, pp. 142–147.

[9] B. Holland, K. Nagarajan, and A. George, "RAT: RC amenability test for rapid performance prediction," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 1, no. 4, pp. 22:1–31, 2009.

[10] S. Fahmy, P. Cheung, and W. Luk, "High-throughput one-dimensional median and weighted median filters on FPGA," *IET Computers and Digital Techniques (IET-CDT)*, vol. 3, no. 4, pp. 384–394, July 2009.

[11] H. Sasama and M. Yoneda, "Priority encoder applicable to large capacity content addressable memory," *United States Patent*, no. 5,555,397, 1996.