

# Architecture for Real-Time Nonparametric Probability Density Function Estimation

Suhaib A. Fahmy, *Member, IEEE*, and A. R. Mohan, *Student Member, IEEE*

**Abstract**—Adaptive systems are increasing in importance across a range of application domains. They rely on the ability to respond to environmental conditions, and hence real-time monitoring of statistics is a key enabler for such systems. Probability density function (PDF) estimation has been applied in numerous domains; computational limitations, however, have meant that proxies are often used. Parametric estimators attempt to approximate PDFs based on fitting data to an expected underlying distribution, but this is not always ideal. The density function can be estimated by rescaling a histogram of sampled data, but this requires many samples for a smooth curve. Kernel-based density estimation can provide a smoother curve from fewer data samples. We present a general architecture for nonparametric PDF estimation, using both histogram-based and kernel-based methods, which is designed for integration into streaming applications on field-programmable gate array (FPGAs). The architecture employs heterogeneous resources available on modern FPGAs within a highly parallelized and pipelined design, and is able to perform real-time computation on sampled data at speeds of over 250 million samples per second, while extracting a variety of statistical properties.

**Index Terms**—Accelerator architectures, adaptive systems, field-programmable gate arrays (FPGAs), histograms, probability density function (PDF).

## I. INTRODUCTION

INFORMATION on the probability density function (PDF) of sampled data has a wide range of uses across many application domains. A variety of techniques have been proposed over the years [1]; computational complexity, however, often means these cannot be implemented in real time within a system that processes continually streaming data. The preference, so far, has thus been to rely on techniques that fit a reduced data set to a predefined parametric model—what is termed *parametric* estimation. 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, statistics are typically required for a single image 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 adapting their behavior based on these statistics. The measurements are typically extracted from data that is processed within a streaming signal processing chain.

Real-time estimation of PDFs is required in applications like cognitive radios that react to channel occupancy statistics, or adaptive networking nodes that modify their routing behavior based on network queue length statistics. For such applications to be feasible, it must be possible to compute PDFs in real-time on streaming data, at rates that do not impact the high-performance requirements of the data processing.

In this paper, we present a PDF estimation architecture that can be integrated within field-programmable gate array (FPGA)-based applications, which is flexible and moderate in terms of area usage. The aim is to allow designers of applications that benefit from the PDF estimation, to leverage this capability within their hardware designs, without impacting the core processing stream. We target FPGAs because they are uniquely placed as an ideal platform on which to build adaptive applications, due to their reconfigurable nature.

In Section II, the fundamentals of the PDF estimation are presented along with a review of existing hardware approaches. Sections III and IV introduce the proposed architecture for histogram-based and kernel-based PDF estimation, respectively, and their FPGA-oriented design. Section V presents an empirical comparison of the two methods. In Section VI, we present hardware implementation results. Finally, Section VII concludes this paper.

## II. RELATED WORK

PDF estimation techniques fall into two categories: parametric and nonparametric [1]. The parametric techniques try to fit the data to a known model and deduce values for model parameters. These parameters can then be used to deduce statistics from the model's equations. The nonparametric techniques use the samples themselves to construct PDF. The most common nonparametric technique is constructing a histogram, which, when normalized, gives the instantaneous PDF. Given a good model, parametric techniques can give more accurate results with less data than a nonparametric model. The requirement for a good model, however, means that when the PDF's nature is unknown or changing, parametric approaches can result in poor accuracy. Furthermore, finding parameter values may not necessarily mean a wide variety of statistical properties can be easily calculated. It is also worth noting that when subtle changes in data distribution over time are required, parametric approaches can fail to provide sufficient discriminatory power.

Manuscript received December 19, 2011; revised May 7, 2012; accepted May 18, 2012. Date of publication June 29, 2012; date of current version April 22, 2013.

The authors are with the School of Computer Engineering, Nanyang Technological University, 639798 Singapore (e-mail: sfahmy@ntu.edu.sg).

Color versions of one or more of the figures in this paper are available online at <http://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/TVLSI.2012.2201187

PDF estimation is of value in a number of different applications areas, including image processing [2], [3], machine learning [4], computer security [5], medical imaging [6], and communications, among others. One example is multiuser direct-sequence code division multiple access which needs an estimation of the PDF for global noise [7]. This importance is certain to increase rapidly as we enter the age of adaptive computing. Adaptive systems rely on information about their environment and operating conditions to make intelligent decisions on adapting their operation. Previously a simple hard decision could be taken based on minimal data, whereas modern systems need an approach that takes into account more complex characteristics of the environment. For autonomous systems, having a “real” view of the situational statistics is essential in making informed decisions, and in aiding the designer to implement and adapt decision algorithms.

We first begin with a basic overview of the foundational theory behind the PDF estimation. As related analysis can be explored in much greater depth, the reader is invited to refer to the references for further discussion [1], [8], with just the basics presented here.

The cumulative distribution function (CDF) of a random variable  $X$  is defined by

$$F(x) = \Pr(X \leq x). \quad (1)$$

We can estimate this, from elementary probability theory, using the empirical CDF

$$F_n(x) = \frac{1}{N} \sum_{i=1}^n I_{(-\infty, x]}(x_i) \quad (2)$$

where  $x_1, x_2, \dots, x_n$  is a random sample from  $F$  and

$$I_A(x) = \begin{cases} 1, & \text{if } x \in A \\ 0, & \text{if } x \notin A. \end{cases} \quad (3)$$

Elementary probability tells us that PDF of a random variable is the derivative of the CDF,  $f(x) = F'(x)$ . Hence the empirical PDF can be stated as

$$f_n(x) = \frac{d}{dx} F_n(x) = \frac{1}{n} \sum_{i=1}^n \delta(x - x_i) \quad (4)$$

where  $\delta(t)$  is the Dirac delta function.

Architecturally, if we consider a bank of registers as being the contents of the histogram, then (2) represents a sample incrementing the value for all bins up to its own value. For (4), we can consider the sample to only increment its own bin. This forms the basis of our proposed architecture.

The fundamental idea of Kernel density estimation is to spread uncertainty in estimation through “spreading” the effect of each input sample by applying a kernel centered at its value, rather than a Dirac delta function, as in (4). Hence, a kernel estimator can be defined as

$$\hat{f}_n(x) = \frac{1}{nh} \sum_{i=1}^n K\left(\frac{x - x_i}{h}\right) \quad (5)$$

where  $K$  is the kernel and  $h$  is the bin width.

Existing work on hardware architectures for the PDF estimation is minimal. In [9] a histogram of an image is constructed for histogram equalization. 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 (sliding) window, for which the PDF values should be updated after each new sample is introduced. In [10], 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 tailor it to PDF calculation, with the addition of kernel-based estimation, and the extraction of statistics. In [3] a histogram is constructed within a hardware architecture but using a sequential approach, again on a per-image basis. Once more, this cannot be tailored for a streaming implementation.

Elsewhere, in [11] a Parzen window PDF estimator is used as an example application for a performance migration tool. However, it processes a single block of data loaded from a host PC, and the performance comparison is made between single- and dual-core acceleration of a mostly sequential implementation.

The architecture presented in this paper is the first to allow for real-time, high-throughput, nonparametric PDF estimation based on the histogram and kernel methods, extending the work in [12]. The architecture maintains the full state of the histogram in every clock cycle, allowing a variety of statistical properties to be computed. It is targeted at applications with changing PDFs or where an accurate model is not known in advance. We believe this architecture will be of great benefit in adaptive applications implemented on FPGAs, where this module can be included to provide real-time statistics for decision-making and adaptation, without requiring any data path modification—by virtue of its pipelined nature, it would not impact the host system’s performance.

### III. PARALLEL PDF ARCHITECTURE

#### A. Outline

The architecture presented here accepts sampled data at its input and computes the cumulative histogram of that data, which is stored internally. By normalizing, based on the window size, which can be changed to any power of 2, the resultant information contained in the circuit represents the cumulative density function. This enables us to extract various PDF statistics in a highly efficient manner. Two architecture variations are presented in this paper. The first computes the bare histogram, while the second uses a kernel-based approach that results in smoother histograms from less data. Maintaining the full histogram within the architecture has the benefit of allowing the designer to extract multiple statistics of interest, as required for a specific application.

The first half of the circuit computes the histogram. This is done by instantiating a bank of counters that keep a tally of the number of occurrences for each input value. This histogram unit is built such that it is updated as each new sample enters the system. The design is heavily pipelined to allow for

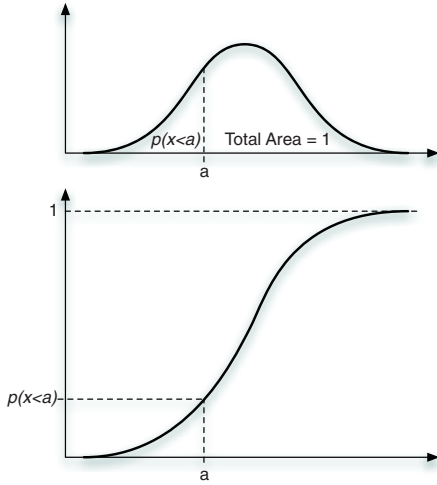


Fig. 1. Comparison of probability calculation from histogram and cumulative histogram.

maximum performance. The second half of the circuit consists of the statistical units that extract information of interest from the cumulative histogram, for use in the target application. We decouple these two parts so that the application designer can extract as many statistics as needed without impacting the functionality of the histogram component.

### B. Computing the CDF

A circuit to hold the PDF of sampled data is relatively straightforward. A bank of counters, one for each bin, can be arranged such that when a sample enters the system, the counter for the bin in which that sample falls is incremented, as in (4). Extracting meaningful statistics from such a PDF, however, involves further calculation. A  $p(x = a)$  calculation would be straightforward, by looking up the count value at index  $a$ . Anything more complex would require separate computation, which may take many cycles.

More useful statistics can be gained by using the 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. 1 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.

Hence, 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 centiles—that is the value of  $c$  such that  $p(x < c)$  equals some given value, as will be discussed shortly.

Hence, the first requirement of the histogram unit, is a bank of counters that can be individually enabled based on whether

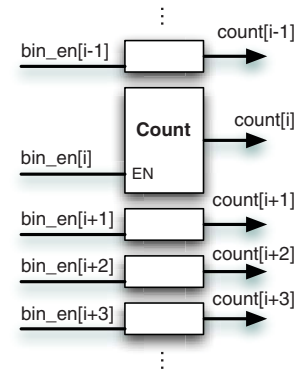


Fig. 2. Basic histogram bin structure.

TABLE I  
ACCESS PATTERN MEMORY CONTENTS FOR 64 BINS

Addr	Pattern[0:63]
0	11111111...11111111
1	01111111...11111111
2	00111111...11111111
2	00011111...11111111
4	00001111...11111111
5	00000111...11111111
⋮	⋮
61	00000000...00000111
62	00000000...00000011
63	00000000...00000001

they should be incremented by the current input sample. This is shown in Fig. 2, where each counter's  $bin\_en$  input indicates whether or not that bin should be incremented in a given clock cycle. The architecture requires a counter for each histogram bin, and hence the area of this part of the circuit varies linearly with the number of bins.

To compute the cumulative histogram in real time, the architecture must be able to update all necessary bins for each input sample in a single cycle. An input sample of value  $x$  should increment the bin corresponding to sample  $x$  and all subsequent bins. For an architecture with hundreds of bins using a comparator coupled with each bin to determine this is costly. In [10], we presented a method that uses a memory to enable the corresponding bins, in a method similar to microprogrammed control. As each input sample arrives, it is used to address a memory with pre-stored access patterns. The access patterns contain a series of bits that determine which bin should be enabled for each input value. For this case, the contents of the memory are simple, as shown in Table I (for a 64-bin design). The embedded memory blocks on modern devices mean this technique is highly amenable to implementation on FPGAs. Furthermore, the sparse layout of these memories on the FPGA fabric, and the fact that large memories are stitched together from smaller ones, means routing is not hampered by this centralized control module, and so high timing performance can still be achieved.

It is worth noting that the number of bins and the word length of the samples are two separate considerations. Various rules have been proposed to find the number of bins that would provide sufficient accuracy with some assumptions about the input data [1], [13]. This is typically less than the number of possible values that can be taken by input samples. Indeed having too many bins can result in a noisy density function, as we later illustrate in Section V. If we calculate that  $2^l$  histogram bins are sufficient, we simply use the  $l$  most significant bits of the input samples to address the access pattern memory. The size of this memory varies with the square of the number of bins, as it is always  $l \times l$ -bits.

Part of the motivation for this design is for the histogram to be calculated over a fixed-length sliding window. This would allow for continuous monitoring of the required data in real time, while allowing older values to fall out of the window, and hence, maintain “fresh” statistics. To allow this, a first-in first-out (FIFO) buffer of length equal to the window is instantiated. This buffer keeps track of the samples in the window. As samples stream through, the oldest sample emerges 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 memory. The resources required for the FIFO vary linearly with the length of the window, and logarithmically with the number of bins.

To remove an old sample from the histogram, its corresponding bins must be decremented. By using the pattern memory 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. The access patterns for the new and oldest sample are extracted in each cycle making it possible to update the histogram in a single step by considering which bins need to be updated. Fig. 3 shows an illustrative example on how this is determined. The control signals for each bin indicate whether to increment, decrement, or maintain the count value.

To normalize the cumulative histogram, to compute the CDF, we need to divide the count values by the window length. Since division by arbitrary integers is expensive in hardware, we have restricted the window length to be a power of 2, which means division is replaced by a shift. This makes sense, since when setting window size, it is the order of magnitude that will have an impact, rather than a small change in value. Hence, we can accommodate window lengths such as 8, 16, 32, 64 K, and so on. Normalization is simply a case of shifting the binary count value to the right by the corresponding number of bits,  $n$ , for a window size of  $2^n$ . So for a window size of 8 K the count value of a particular bin is shifted right by 13 bits. Rather than lose any precision in the counter itself, all scaling is only done at the output of the architecture.

Fig. 4 shows the overall architecture of the system. Samples enter the system and are fed into the FIFO, the output position of which can be set by the *fifo\_length* signal, allowing the window size to be varied at run time. The FIFO is constructed, such that it can be set to any useful power of 2. We use embedded Block RAMs to implement the FIFO, thus

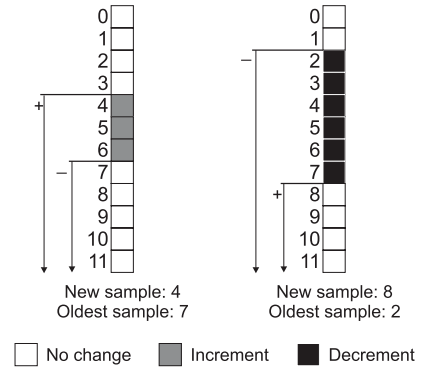


Fig. 3. Determining which bins are to be incremented, decremented, and unchanged, based on the current new and old samples. + indicates the new sample, while - indicates the old sample.

reducing logic usage and increasing performance. 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, to update the histogram values. The count outputs of all the bins (*bincount*[0:N-1]) are then passed to the next stage, where statistics can be extracted, as discussed in the next section.

### C. Extracting Statistics

Now the CDF data is present within the circuit, and we can add units to extract statistics. We previously showed that probability ranges are straightforward. For fixed  $p(x < a)$  calculations, we simply extract the value in the  $a$ th counter. If we desire the value of  $a$  to be adjustable, a multiplexer allows this to be done. Similarly, for an intermediate probability range, we use two multiplexers to extract the cumulative count for two bins, and subtract them. 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  $p(a < x < b)$ , two *pr\_out* values, one corresponding to each of  $a$  and  $b$  are subtracted from each other.

Centiles are a useful tool in statistical analysis. One of the unique features of this architecture is that it allows these statistics to be extracted with minimal overhead. 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 an opportunistic cognitive radio node, 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 centiles 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 centile, since 100 of

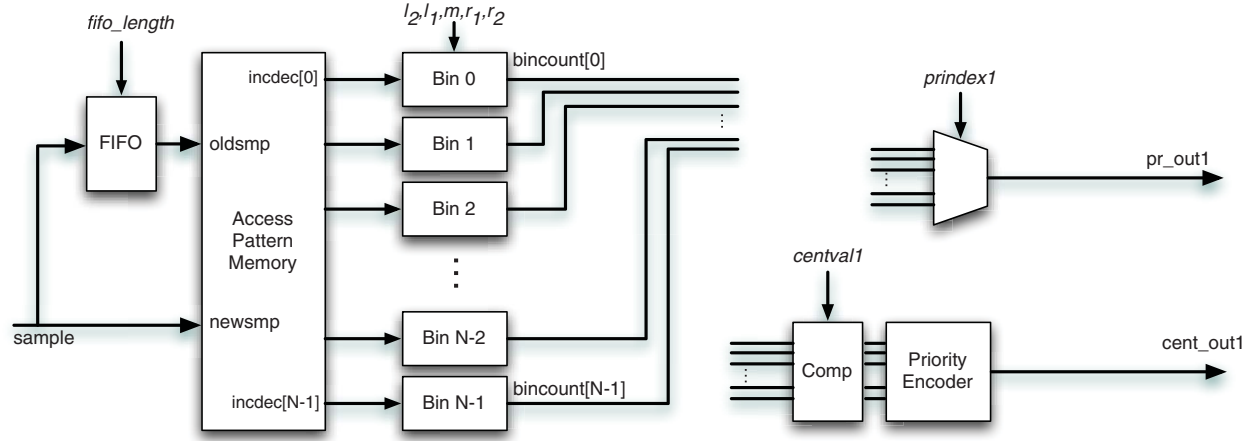


Fig. 4. Complete PDF estimator architecture, showing the window FIFO, access control memory, and bank of bin counters, along with statistics units.

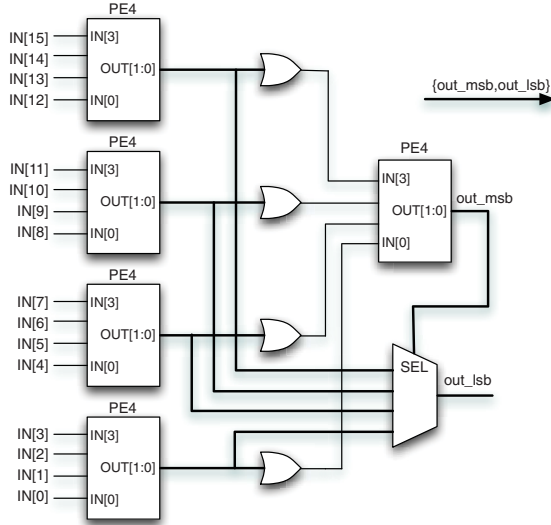


Fig. 5. 16-bit priority encoder constructed from five 4-bit priority encoders. The first stage computes the least significant bits of the output while the second computes the most significant bits and selects the required output from the first stage. The output is a concatenation of the two.

the 1000 samples (i.e., 10%) fall at that value or below. More generally, if we want the  $n$ th centile, 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 determines 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 [14] shows how it is possible to pipeline many small

priority encoders together into a larger one. Fig. 5 shows how five 4-bit priority encoders can be combined into a 16-bit priority encoder. Similarly, nine 8-bit priority encoders can be combined into a 64-bit priority encoder, and so on. By pipelining between each stage, only a few clock cycles of latency are added, while we are able to clock the circuit significantly faster.

The centile value, *centval*, is given as a proportion of the window size, rather than a raw percentage value. The *Comp* block compares the count values of all the bins, in parallel, with 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 the first bin with a high output, giving the index for the required centile. The outputs can be used for monitoring the centiles themselves or to calculate inter-centile range, simply by tracking the difference between two centiles.

As many of these statistical calculation components can be added, as needed. Further compositions are possible since the histogram data is all stored within the bin counters.

#### IV. KERNEL-BASED ARCHITECTURE

The architecture presented in the previous section computes the PDF-based statistics from a raw histogram representation of the input data. The primary drawback of that approach is that it requires a large number of samples to give an accurate estimate. Due to the discontinuous nature of the bins, a small number of samples leads to errors in the estimated PDF. Section V discusses this in more detail.

Kernel density estimation allows us to model the uncertainty in input accuracy by spreading the effect of an input sample into adjacent bins. The result is increased accuracy in the resulting PDF compared with the histogram-based approach, for the same number of samples. Alternatively, we can match the accuracy of the histogram-based approach with fewer samples. The result at the system level is that extracted statistics are more accurate, and as they rely on fewer samples, changes in the PDF are apparent sooner.

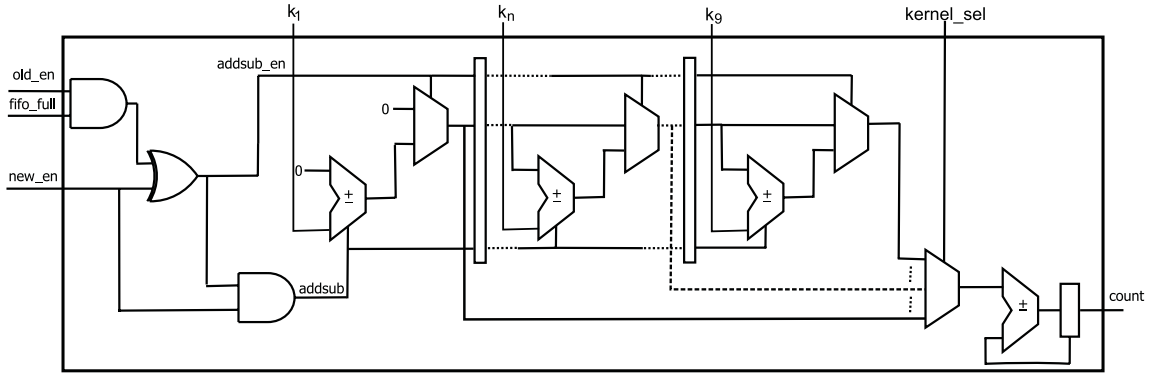


Fig. 6. Bin architecture for kernel density estimation.

TABLE II  
KERNEL-BASED CUMULATIVE HISTOGRAM

Sample	Kernel Values Added	Access Pattern Bits				
		$l_2$	$l_1$	$m$	$r_1$	$r_2$
→	$l_2$	1	0	0	0	0
	$l_1 + l_2$	1	1	0	0	0
	$m + l_1 + l_2$	1	1	1	0	0
	$r_1 + m + l_1 + l_2$	1	1	1	1	0
	$r_2 + r_1 + m + l_1 + l_2$	1	1	1	1	1
	$r_2 + r_1 + m + l_1 + l_2$	1	1	1	1	1
	$r_2 + r_1 + m + l_1 + l_2$	1	1	1	1	1
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	$r_2 + r_1 + m + l_1 + l_2$	1	1	1	1	1
	$r_2 + r_1 + m + l_1$	0	1	1	1	1
	$r_2 + r_1 + m$	0	0	1	1	1

#### A. Computing the Kernel Density Estimate

To enable this, each sample must be able to address the set of bins required for its kernel. Each sample affects its corresponding bin as well as neighboring bins. The number of neighboring bins affected depends upon the width of the kernel. Each bin counter should be incremented by the sum total of all the kernel values affecting it. The different kernel values that overlap when an input sample arrives are illustrated in Table II. The kernel values in the table are represented as  $\{l_2, l_1, m, r_1, r_2\}$ , with  $m$  being the middle value. A kernel width of 5 is used for illustration. The kernel is a symmetric function that integrates to 1.

In this example, a sample arrives centered at the third bin. Since the kernel width is 5, this sample affects the two bins prior (0 and 1), and the two following (3 and 4). All subsequent bins are also incremented to produce the cumulative histogram.

Hence, for the kernel-based design, adders are required to update the bins, as opposed to the basic histogram design where bins were either incremented or decremented. The width of these adders depends on the width of the kernel coefficients.

The other challenge is how to implement the control for the bins now that a single sample addresses multiple bins. It would be possible to use a separate access memory for each kernel coefficient. That way, when a sample arrives, each coefficient

enables the corresponding bins, and the coefficient values are added within the bin. But this approach would consume a large number of memory blocks and would make the circuit more complex. Furthermore, the width of the kernel would then impact area significantly.

Looking at Table II once more, we can see that when an input sample arrives, its corresponding bin is incremented by  $m$ , the bins immediately above are incremented by  $l_2$  and  $l_1 + l_2$ . Similarly the bins immediately below are incremented by  $r_1 + m + l_1 + l_2$  and  $r_2 + r_1 + m + l_1 + l_2 (= 1)$ , respectively. It is clear from the table that the access pattern for the middle value  $m$  of the kernel for a given incoming sample is the same as in Table I for the standard histogram. The access patterns for other kernel values are simply obtained by shifting the access pattern memory left or right and appropriately zero-padding them. Therefore, the access patterns for all the kernel values can be obtained from a single access pattern memory by appropriately connecting them to address the bins. We can wire up the single access pattern memory in such a way that the control signals for all the kernel coefficients are extracted at the same time. To maintain the PDF contents with each input sample, these additions must be completed in a single clock cycle.

To remove an old sample from the histogram, the kernel values for its corresponding bins must be subtracted. We further extend the technique described in Section III-B, using the same access pattern memory, to decide how to modify the count value of each bin. Each bin now has an input signal for each kernel coefficient that indicates whether the coefficient value should be added or subtracted, or left unchanged.

Kernel values are represented using fixed point numbers of  $k$  bits with the binary point to the left, as the values are all less than 1. More accuracy and precision can be obtained by using more bits. Since the kernel values always integrate to 1, the maximum number of bits required to store the sum of all the kernel values will be  $k + 1$  with the binary point to the right of the most significant bit. The counters in each bin should be wide enough to accommodate the maximum value, which depends upon the window size. For a window size of  $N = 2^n$ , the width of the count register will be  $n + k + 1$ . Normalization can be applied by shifting the binary point to the left by  $n$  bits and truncating the fractional part.



### B. Architecture Design

The overall PDF estimation architecture shown in Fig. 4 is modified to allow the kernel values to be input and stored inside a register array. Using a *kernel\_sel* signal, the width of the kernel can be chosen at run-time.

The only significant modification is to the architecture of the bin nodes, as shown in Fig. 6 for a maximum kernel width of nine coefficients. Each bin receives 9-bit signals *new\_en* and *old\_en*, which indicate whether the bin is accessed by the new and oldest samples, respectively, for each kernel coefficient. XOR-ing *old\_en* and *new\_en* gives a signal that indicates whether the bin value should change for each kernel coefficient. In the first stage of the pipeline, the left most value of the kernel is added/subtracted depending upon the corresponding bit of the *addsub* signal. Its output is gated by *addsub\_en*, that is, the output of the adder/subtractor is not propagated through the pipeline if *addsub\_en* is low for that particular kernel value. The pipeline register is used to latch the intermediate counter value, *addsub*, and *addsub\_en* signals for subsequent stages. As the pipeline is filled, the kernel values are added/subtracted and the accumulated value reaches the last stage at which it is added to the bin counter register. After a latency equivalent to the width of the kernel in bits, the count register is updated for every cycle as new samples arrive and old samples emerge from the FIFO. The number of adder/subtractors required is equal to the maximum width of the kernel, that is, 9 in this case. By pipelining the adders, drastic performance improvement is achieved for an increase of only a few cycles in latency.

The resulting CDF data can now be processed to extract statistics in the same way as the histogram design. The architecture processes a new input sample in every cycle, and maintains the correct CDF with each new sample that enters.

## V. ESTIMATION ACCURACY

As a designer using the above architecture within a larger system, it is worth investigating the impact of design parameter choices on the resulting resource usage. To recap, the first choice is whether to use the histogram-based approach or kernel-based estimation. Then the number of bins must be chosen. This impacts area linearly for both schemes. The length of the window of interest must also be chosen; this impacts the length of the FIFO, as well as the width of the counters in the bins. For the kernel-based approach, the designer can also choose the width of the kernel, that is, the number of bins across which an input sample's effect is spread. This impacts the number of adders needed in the bins, and the width of the bin registers. The architecture has been designed such that a maximum range for each of these parameters can be set at synthesis, but each can be changed at run-time using configuration inputs.

A series of simulations were performed to study the accuracy of histogram-based PDF estimation and kernel density estimation. A large sample of 1024K random numbers was generated from predefined distributions (normal, Rayleigh, etc.). Smaller subsets of this large sample set were used to estimate the PDF using both histogram-based and kernel

density estimation methods, averaging over a few hundred windows. The lengths of the subsets were chosen to match the settable FIFO lengths. PDFs were estimated by varying the FIFO length and number of bins for each combination. Mean-squared error (MSE) was estimated by comparing the PDF estimated with the original distribution. It is worth noting that this analysis is empirical and data-dependent. Although there is significant analytical work that investigates the impact of such choices, it is well beyond the scope of this paper, and the reader is referred to [1] and [8] for such discussion.

### A. Impact of Window Size

Fig. 7 shows the MSE with varying FIFO lengths for a different numbers of bins. Each plot shows the MSE for the histogram-based approach (hist), and for three to nine kernel coefficients (k3–k9). It is clear that as the number of samples considered (i.e., the FIFO length) increases, the resulting PDF becomes more accurate in all cases. In fact, we can see that for 64 bins, a nine-coefficient kernel converges to a higher error rate than the raw histogram, likely because the spreading effect of the kernel is too pronounced. A smaller window results in linearly increased error (the x-axis is exponential, as per the possible window sizes).

It is also clear that irrespective of the bin size, kernel density estimation consistently outperforms the histogram-based method, reducing error by a half from a raw histogram to a kernel with three elements. This improvement is especially apparent with small window sizes. As the FIFO length increases and approaches several thousand samples, the MSE for all cases converge. Adding further coefficients to the kernel-based approach gives increased accuracy, but this diminishes as the kernel is widened. The conclusion is that a kernel-based design with three coefficients matches the performance of a histogram-based design with a window that is twice as large. Increasing to seven kernel coefficients can halve the window size again for comparable performance.

For real-time systems, being able to estimate a PDF more accurately with fewer samples would be beneficial, as it would result in a circuit that uses less area. Furthermore, a smaller window means a system is able to monitor rapid changes in statistics. As this effect is more pronounced in the smaller size range, this is of greater significance, as we are keen to enable compact implementations, and hence, a reduced number of bins is likely.

### B. Impact of Number of Bins

Fig. 8 shows the MSE for varying bin sizes for different window sizes. The general trend across the graphs shows that as we increase window size, MSE decreases, as in the previous figure. However, now we can see that the number of bins also has an impact on accuracy. With a smaller window size, and hence fewer samples to consider, a large number of bins leads to a sparsely populated histogram, and hence the resulting shape is spiky. This means that the instantaneous PDF obtained for any window is likely significantly different from the original PDF we are trying to estimate. As the window size (FIFO length) increases, this trend becomes less apparent,

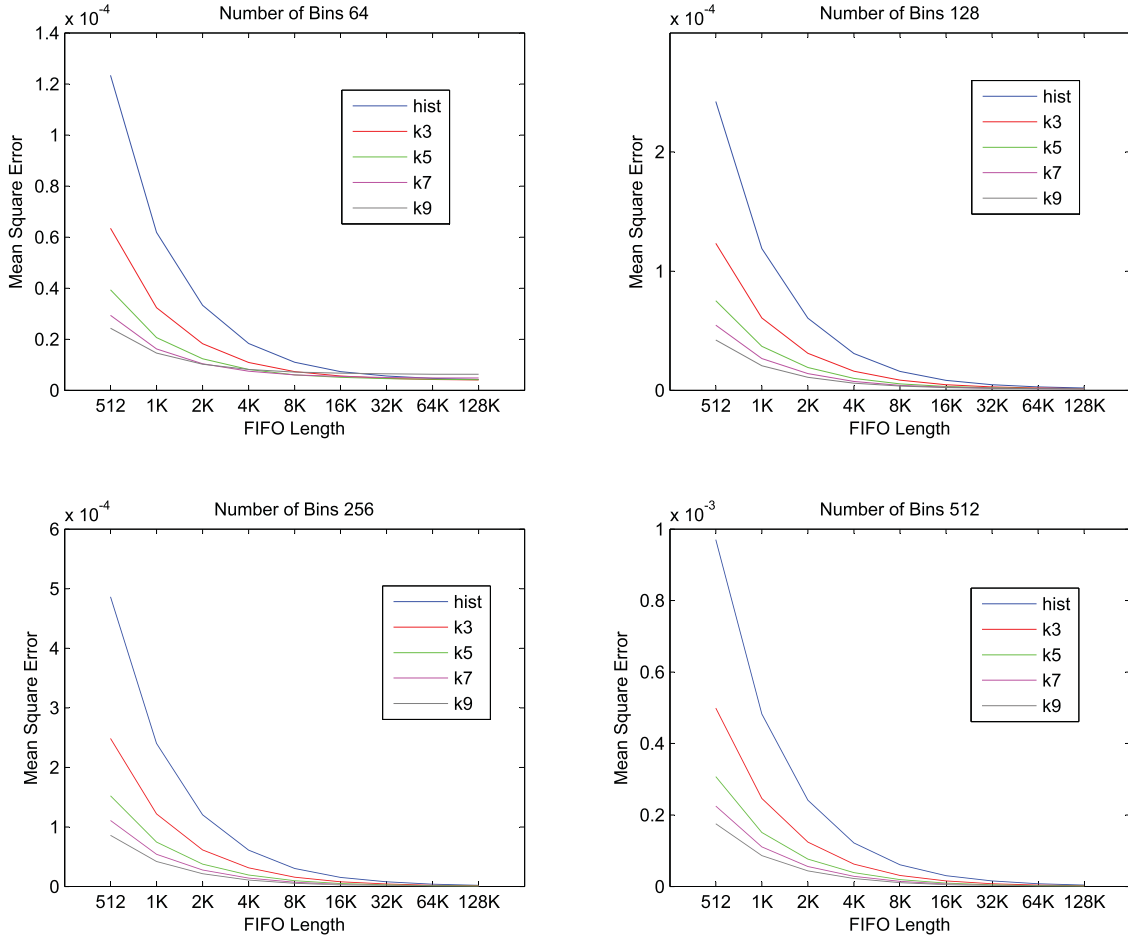


Fig. 7. MSE for varying window (FIFO) lengths for histogram (hist) and kernel-based calculation, with three, five, seven, and nine kernel coefficients (k3, k5, k7, and k9, respectively).

and we find that a higher number of bins can bring some increased fidelity. It is also worth noting that as we add more kernel coefficients, we suffer less with a larger number of bins. The reason here is that the kernels spread samples across multiple bins, hence smoothing out the spikiness. This effect is illustrated for an example distribution in Fig. 9, where we see that an increased number of bins results in an instantaneous estimate that is very noisy, though the kernel-based approach does mitigate this somewhat.

The results for 32 bins in Fig. 8 seem to defy the general trend. On closer examination, we discover that the error is in fact constant regardless of the window size. This suggests that in this case, the limiting factor in the resulting estimation's accuracy, is the number of bins rather than the window size. Generally for these tests, we found that the best performance was obtained with bin sizes of 64 and 128 for the chosen distributions.

Determining the optimal number of bins clearly has a significant impact, and as discussed in Section III-B, there has been a significant amount of analytical work in this area. In [15], kernel-based PDF estimation is shown to perform far better than the histogram-based method and a mathematical model to estimate the optimal kernel width and bin size is presented. From the experiments presented here, it can be concluded that kernel density estimation provides smoother

results using fewer samples and with a smaller number of bins.

## VI. IMPLEMENTATION RESULTS

The above architectures were implemented and simulated in Verilog then synthesized and targeted to a Xilinx Virtex-5 XC5VFX70T, as found on the Xilinx ML507 development board. We selected a maximum configurable window size of 128K samples and set the maximum number of bins to 128. We used the standard bin access pattern and allow for two direct probability and two percentile calculations in parallel.

For the FIFO, we used Xilinx embedded memory cores. Since we only need access to specific points within the FIFO queue, FIFOs of length 8, 16, 32, and 64K were chained together, allowing for window sizes of 8, 16, 32, 64, and 128K, while maintaining maximum speed. For the priority encoder, the technique detailed in Section III-C was used to build a 128-bit module using two 64-bit priority encoders. The 64-bit priority encoders were created using nine 8-bit priority encoders. For the access pattern memory, a standard Block RAM module in CoreGen was used. It stitches together four-Block RAMs to create the 128 × 128-bit memory. Unfortunately, due to the structure of the primitives, only a quarter of the addressable locations are used. This is because each 18 kB memory can only be configured from 16 K × 1



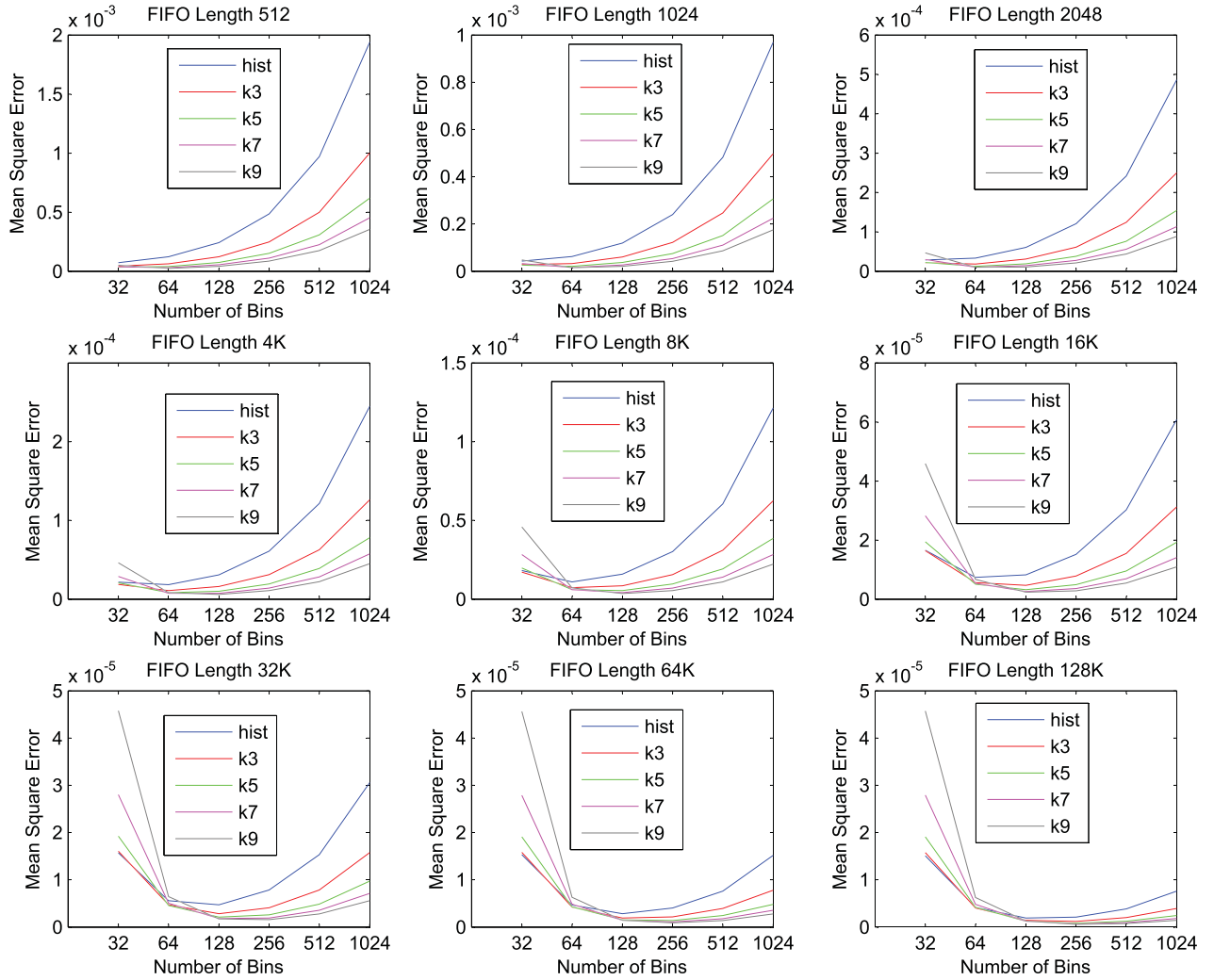


Fig. 8. MSE for varying number of bins for histogram (hist) and kernel-based calculation, with three, five, seven, and nine kernel coefficients (k3, k5, k7, and k9, respectively).

TABLE III  
FPGA RESOURCE UTILIZATION

Module	PDF		KDE	
	Slices	BRAMs	Slices	BRAMs
Access Memory	0	4	0	4
Window FIFO	462	32	462	32
PDF Unit	324	0	568	0
Centile Unit (ea.)	430	0	870	0
Full System	7640	36	22360	36
Available	44800	148	44800	148

to  $512 \times 32$ . Four  $512 \times 32$  memories are arranged side-by-side for this memory.

Using the Xilinx cores for the FIFO was also found to speed up synthesis and reduce logic usage significantly, as the tools are unable to automatically infer large primitives efficiently.

For the parameters given above, we obtained the area results shown in Table III. The architectures were both able to achieve over 250 MHz clock frequency.

The area usage in both cases is dominated by the histogram circuitry. For histogram-based estimation it consists of

128 16-bit counters and associated control circuitry. For kernel density estimation, it consists of 128 25-bit registers with adder-subtractors and associated control circuitry. The centile unit in the kernel-based architecture occupies a larger area due to the wider registers required by the comparators. This could be mitigated by considering a subset of the bits starting at the most significant, if area optimization is essential. BlockRAM usage is primarily due to the FIFO and hence unchanged. Reducing window size would result in a reduction in BlockRAM requirements, while reducing the number of bins for the histogram, would reduce the logic requirements significantly.

It is clear that for an identically parameterized design, the kernel-based approach consumes almost three times the logic of the histogram-based approach. Since kernel-based PDF estimation can obtain a better result from fewer samples, it is possible to decrease the FIFO length without impacting estimation accuracy. In the next section, we explore some variations.

Rather than compare identical design parameters, we would like to take advantage of the accuracy opportunity afforded by the kernel-based approach to reduce its area overhead. A number of parameter variations were considered to

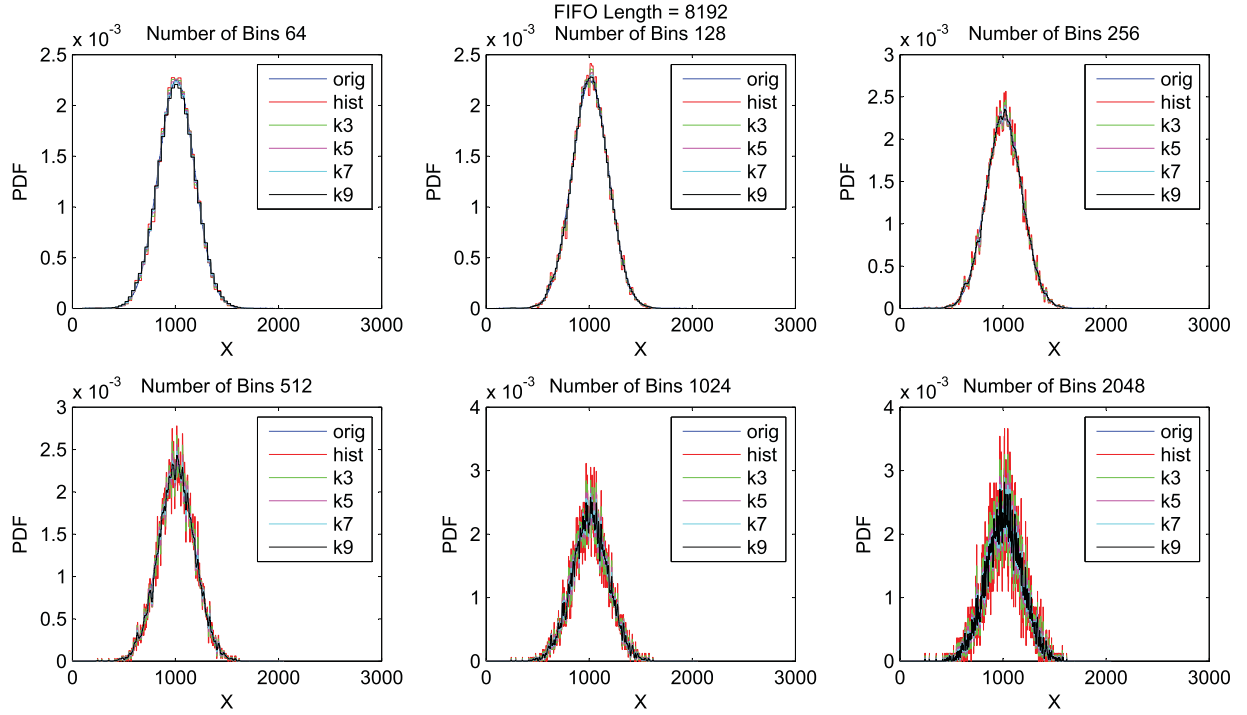


Fig. 9. Effect of increasing number of bins on an approximated PDF for a fixed FIFO length of 8192 samples, showing the increased noise with a high number of bins. Approximations shown for histogram (hist) and kernel-based calculation, with three, five, seven, and nine kernel coefficients (k3, k5, k7, and k9, respectively), overlaid on original distribution (orig).

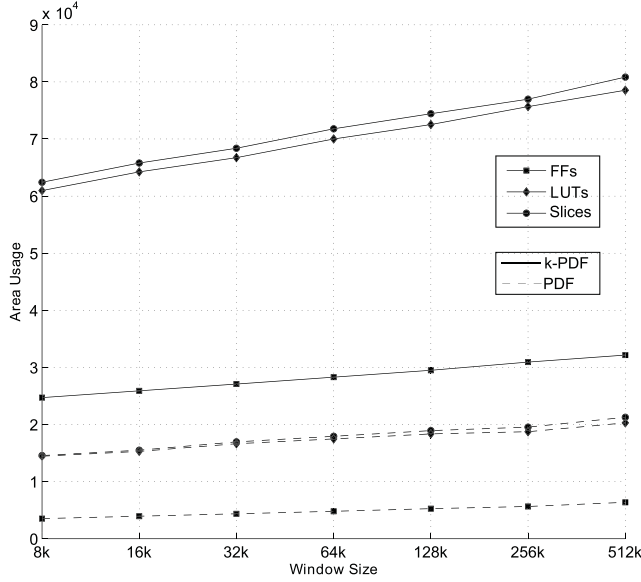


Fig. 10. Resource usage for varying window sizes.

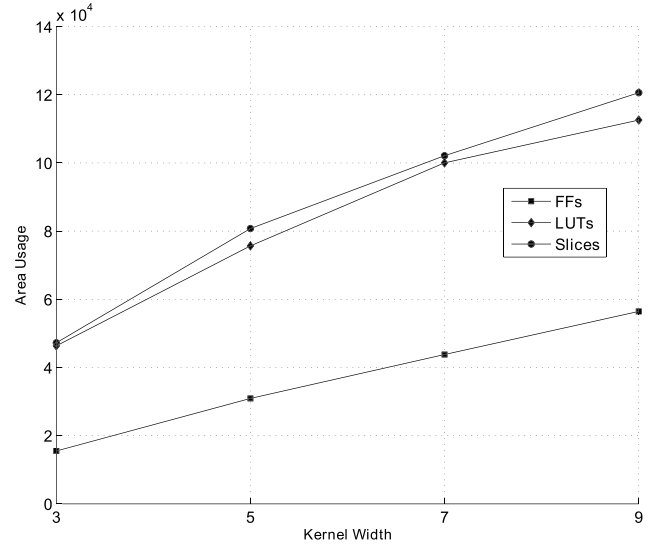


Fig. 11. Resource usage for varying number of kernel coefficients.

thoroughly investigate the proposed architecture. The results were obtained by synthesizing the architecture for a Xilinx Virtex-5 XC5VLX330 to accommodate a large number of bins.

Area usage for varying window sizes was studied for both the architectures keeping the number of bins to 256 and a fixed kernel width of five 12-bit coefficients. The results for both architectures are shown in Fig. 10. The bin counter width was accordingly increased in each case. The general trend for area requirements can be described as being of the form  $A + \log_2 B$ ,

where  $B$  is the window size and  $A$  is the fixed area required by the rest of the design regardless of window size. Window size impacts Block RAM usage more significantly.

The area usage for varying numbers of kernel coefficients was studied for the kernel density estimation architecture keeping the number of bins to 256 and a maximum configurable window size of 128K. The results are shown in Fig. 11. An increase in the number of kernel coefficients increases the number of adder-subtractors, associated control circuitry, and pipeline registers.

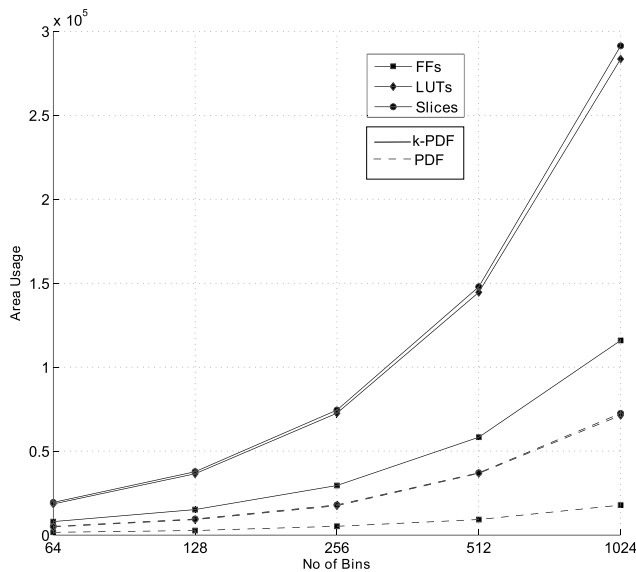


Fig. 12. Resource usage for varying number of bins.

The next variation involves varying the number of bins. The kernel architecture was fixed to five coefficients and a maximum configurable window size of 128K was used. The results for this set of implementations are shown in Fig. 12. Clearly, the number of bins impacts logic usage significantly, as each bin includes the increment/addition circuitry. We can also see that by using kernel density estimation, we are able to reduce the window size, and hence Block RAM usage, but we pay a price in increased logic usage.

The resultant usage in a real system thus depends significantly on how the unit is configured, but it is clear that by design, the proposed architecture is scalable without adversely impacting throughput. By thoroughly pipelining the design, it is able to process a new sample in every clock cycle, at a throughput of over 250 million samples per second.

## VII. CONCLUSION

We showed a novel architecture for real-time computation of PDF estimates based on the histogram and kernel density estimation methods. It makes extensive use of FPGA resources to parallelize and accelerate 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 showed an extended architecture for kernel-based PDF estimation capable of changing kernel widths at run-time, without loss in performance. The architecture can process data streams at 250 million samples per second. We also presented simulation results that help illustrate the trade-off in selecting between raw histogram-based PDF estimation and kernel-based estimation.

In future work, we plan to explore the use of this architecture within adaptive systems, to better understand the impact of design decisions, and to compare the two estimation

approaches. We feel that this architecture can enable a range of new applications incorporating intelligent adaptation.

## REFERENCES

- [1] D. Scott, *Multivariate Density Estimation. Theory, Practice and Visualisation*. New York: Wiley, 1992.
- [2] S. Phung, A. Bouzerdoum, and D. Chai, "Skin segmentation using color pixel classification: Analysis and comparison," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 1, pp. 148–154, Jan. 2005.
- [3] B. Wang, S. Qian, and H. Zhou, "A real-time contrast enhancement algorithm for infrared images based on plateau histogram," *Infr. Phys. Technol.*, vol. 48, no. 1, pp. 77–82, 2006.
- [4] C. Zhang and T. Chen, "An active learning framework for content-based information retrieval," *IEEE Trans. Multimedia*, vol. 4, no. 2, pp. 260–268, Jun. 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 *Proc. Int. Workshop Construct. Side-Channel Anal. Secure Design*, 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," *Electron. Lett. Comput. Vis. Image Anal.*, vol. 5, no. 4, pp. 1–11, 2005.
- [7] Y. Li and K. H. Li, "Iterative PDF estimation and turbo-decoding scheme for DS-CDMA systems with non-Gaussian global noise," in *Proc. IEEE Global Telecommun. Conf.*, Nov. 2001, pp. 3262–3266.
- [8] B. Silverman, *Density Estimation for Statistics and Data Analysis*. London, U.K.: Chapman & Hall, 1986.
- [9] X. Li, G. Ni, Y. Cui, T. Pu, and Y. Zhong, "Real-time image histogram equalization using FPGA," *Proc. SPIE*, vol. 3561, pp. 293–299, Aug. 1998.
- [10] S. Fahmy, P. Cheung, and W. Luk, "High-throughput one-dimensional median and weighted median filters on FPGA," *IET Comput. Digital Tech.*, vol. 3, no. 4, pp. 384–394, Jul. 2009.
- [11] B. Holland, K. Nagarajan, and A. George, "RAT: RC amenability test for rapid performance prediction," *ACM Trans. Reconfig. Technol. Syst.*, vol. 1, no. 4, pp. 22–31, 2009.
- [12] S. A. Fahmy, "Histogram-based probability density function estimation on FPGAs," in *Proc. Int. Conf. Field Program. Technol.*, 2010, pp. 449–453.
- [13] D. Scott, "On optimal and data-based histograms," *Biometrika*, vol. 66, no. 3, pp. 605–610, 1979.
- [14] H. Sasama and M. Yoneda, "Priority encoder applicable to large capacity content addressable memory," U.S. Patent 5 555 397, Sep. 10, 1996.
- [15] H. Shimazaki and S. Shinomoto, "Kernel bandwidth optimization in spike rate estimation," *J. Comput. Neurosci.*, vol. 29, nos. 1–2, pp. 171–182, 2010.



**Suhaib A. Fahmy** (M'01) received the M.Eng. degree in information systems engineering and the Ph.D. degree in electrical and electronic engineering from Imperial College London, U.K., in 2003 and 2007, respectively.

He was a Research Fellow with the University of Dublin, Trinity College, and a Visiting Research Engineer with Xilinx Research Labs, Dublin, from 2007 to 2009. He joined the the School of Computer Engineering, Nanyang Technological University, Singapore, as an Assistant Professor in 2009. His current research interests include reconfigurable computing, high-level system design, and computational acceleration of complex algorithms.

**A. R. Mohan** (S'09) received the B.E. degree in electronics and communication engineering from Anna University, Chennai, India, and the Ph.D. degree from the School of Computer Engineering, Nanyang Technological University, Singapore, in 2005 and 2011, respectively.