Published in final edited form as:

IEEE Trans Nucl Sci. 2010 February ; 57(1): 71–77. doi:10.1109/TNS.2009.2030581.

Design of an FPGA-Based Algorithm for Real-Time Solutions of Statistics-Based Positioning

Don DeWitt,

Department of Electrical Engineering, University of Washington, Seattle, WA 98195 USA

Nathan G. Johnson-Williams[Student Member, IEEE],

Department of Electrical Engineering, University of Washington, Seattle, WA 98195 USA

Robert S. Miyaoka[Member, IEEE],

Department of Radiology, University of Washington, Seattle, WA 98195 USA

Xiaoli Li[Student Member, IEEE],

Department of Physics, University of Washington, Seattle, WA 98195 USA

Cate Lockhart,

Department of Pharmaceuticals, University of Washington, Seattle, WA 98195 USA

Tom K. Lewellen[Fellow, IEEE], and

Department of Radiology, University of Washington, Seattle, WA 98195 USA

Scott Hauck[Senior Member, IEEE]

Department of Electrical Engineering, University of Washington, Seattle, WA 98195 USA

Don DeWitt: dewitdq@earthlink.net; Nathan G. Johnson-Williams: nathanjw@u.washington.edu; Robert S. Miyaoka: rmiyaoka@u.washington.edu; Xiaoli Li: lixiaoli@u.washington.edu; Cate Lockhart: cmo4@u.washington.edu; Tom K. Lewellen: tkldog@u.washington.edu; Scott Hauck: hauck@ee.washington.edu

Abstract

We report on the implementation of an algorithm and hardware platform to allow real-time processing of the statistics-based positioning (SBP) method for continuous miniature crystal element (cMiCE) detectors. The SBP method allows an intrinsic spatial resolution of ~ 1.6 mm FWHM to be achieved using our cMiCE design. Previous SBP solutions have required a postprocessing procedure due to the computation and memory intensive nature of SBP. This new implementation takes advantage of a combination of algebraic simplifications, conversion to fixed-point math, and a hierarchical search technique to greatly accelerate the algorithm. For the presented seven stage, 127×127 bin LUT implementation, these algorithm improvements result in a reduction from $> 7 \times 10^6$ floating-point operations per event for an exhaustive search to $< 5 \times 10^3$ integer operations per event. Simulations show nearly identical FWHM positioning resolution for this accelerated SBP solution, and positioning differences of < 0.1 mm from the exhaustive search solution. A pipelined field programmable gate array (FPGA) implementation of this optimized algorithm is able to process events in excess of 250 K events per second, which is greater than the maximum expected coincidence rate for an individual detector. In contrast with all detectors being processed at a centralized host, as in the current system, a separate FPGA is available at each detector, thus dividing the computational load. These methods allow SBP results to be calculated in real-time and to be presented to the image generation components in real-time. A hardware implementation has been developed using a commercially available prototype board.

Index Terms

Continuous crystal; field programmable gate array (FPGA); PET detector

I. Introduction

Discrete crystal detector modules have traditionally been used to achieve high spatial resolution for small animal (preclinical) PET scanners. However, the cost goes up rapidly as crystal cross section gets smaller. We have previously presented a continuous miniature crystal element (cMiCE) detector as a lower cost alternative, with a statistics-based positioning (SBP) algorithm used for event positioning from list mode data [1]–[3]. We are currently developing a full preclinical scanner based on the cMiCE detector module. However, the processing and memory requirements of performing the event positioning from the list mode data results in an undesirable delay between the end of data collection and the positioning of the event data. Processor arrays are able to reduce this time; however, a goal of this work is to provide a real-time implementation in which processing occurs at the detector level. In this paper, several methods of algorithm optimizations, along with a pipelined FPGA implementation, results in local processing of our SBP algorithm at the detector level of over 250 K events per second.

A field programmable gate array (FPGA) is a special purpose integrated circuit (IC) designed to perform complex interface and logic processing in a small footprint [4]. The FPGA is a re-configurable hardware platform consisting of a large number of simple logic elements with a matrix of interconnects that can be selectively enabled or disabled under the control of a downloadable configuration file. This allows the FPGA a flexibility that is not available with microprocessors or custom logic chips.

The flexibility of the FPGA allows many options to be considered in implementing the SBP method. The proposed system requires a method of generating solutions at a much higher rate than available with the postprocessing method currently used. Some of the hardware options to be explored include parallel processing and hardware acceleration of the individual computations. Software options include optimizing the algorithm performing the computations and conversion from floating-point to an integer or fixed-point implementation to reduce the computational complexity. These methods will be explored in the following sections.

II. Materials and Methods

A. cMiCE Detector Module

A cMiCE detector module is pictured in Fig. 1. It is composed of a 50 mm by 50 mm by 8 mm-thick slab of LYSO (Saint Gobain, Newbury, OH) coupled to a 64-channel multianode PMT (H8500, Hamamatsu Photonics K.K., Japan). One of the 50 mm by 50 mm surfaces was polished. All other surfaces were roughened. The polished side was coupled to the PMT using Bicon BC-630 optical grease. The face of the crystal opposite the PMT was painted white. The side surfaces of the crystal were painted black to reduce light reflections off the sides.

B. SBP

Suppose the distributions of the 64 PMT outputs, $M = m_1, m_2, \dots, m_{64}$ for scintillation position x are independent normal distributions with mean $\{\mu_i(x)\}$ and standard deviation $\{\sigma_i(x)\}$.

The likelihood function for making any single observation m_i from distribution M_i given x is

$$L[m_i|x] = \prod_{i=1}^n \frac{1}{\sigma_i(x) \sqrt{2\pi}} \exp\left(-\frac{(m_i - \mu_i(x))^2}{2\sigma_i^2(x)}\right). \quad (1)$$

The maximum likelihood estimator of the event position x is given by

$$\arg \min_x \left[\sum_i \frac{(m_i - \mu_i(x))^2}{2\sigma_i^2(x)} + \ln(\sigma_i(x)) \right]. \quad (2)$$

The SBP method requires that the light response function versus interaction location be characterized for the detector. Two SBP look-up tables (LUTs) corresponding to the mean and variance of the light probability density function (PDF) versus (x,y) position are created during the characterization process. The LUTs are 127×127 , corresponding to ~ 0.4 mm by ~ 0.4 mm pixels for binning. When a coincidence event is detected signals from all 64 channels are collected. A photopeak energy window (i.e., 511 keV $\pm 20\%$) is applied to the data and if the energy of the event falls within the window it is accepted as a good event and will be passed along for SBP positioning. For more detailed explanations about the SBP implementation refer to Joung and Ling [1], [2]. Please note that the Joung implementation was for a very thin 4 mm thick crystal and in that work it was determined that the $\ln(\sigma_i(x))$ term did not effect the positioning performance in the algorithm. However, as we have moved to a thicker crystal (i.e., 8 mm thick), it has been determined that the $\ln(\sigma_i(x))$ should be included in the calculation.

A sample of the positioning performance of a cMiCE detector module using our statistics-based positioning (SBP) method is illustrated in Fig. 2. Full width at half maximum (FWHM) contour plots of point fluxes perpendicular to the cMiCE detector are shown. The dots in the picture illustrate the locations of the point fluxes. The point fluxes are separated by 2.026 mm. The average intrinsic spatial resolution for the full detector (i.e., 44.6 by 44.6 mm² active area) is 1.6 mm FWHM. In comparison, using standard Anger positioning the useful imaging area is only ~ 34 mm by 34 mm. Further, the average spatial resolution for the Anger method is 2.2 mm and the positioning is biased [2].

C. Equation Simplification

The proposed method to implement SBP within an FPGA relies upon multiple levels of optimization. The first is reducing the number of computations required for each iteration of the likelihood equation, the central component of the SBP method. The log likelihood function for each signal channel can be represented as (3)

$$\frac{(m_i - \mu_i(x))^2}{2\sigma_i^2(x)} + \ln(\sigma_i(x)). \quad (3)$$

The objective is to manipulate this function to minimize those numerical operations which are expensive to perform. With the computational resources available in this project the following guidelines are employed:

- Adds and subtracts are efficient.
- Multiply is relatively efficient, but should be kept to a minimum.

- Squares are special cases of multiply, but should also be minimized.
- Division is very inefficient because it requires an iterative approach.
- Transcendental functions should be avoided.

Each iteration of (3) consists of seven operations:

1. Subtract $(m_i - \mu)$.
2. Square the result.
3. Square σ_i .
4. Multiply the result by 2.
5. Divide the result of step 2 by the result of step 4.
6. Find the natural log of σ_i .
7. Add the result of step 6 to the result of step 5.

The transcendental function of step 6 increases the complexity of this method well beyond the seven operations, and needs to be eliminated. Also, the division should be replaced with a multiplication by a precomputed inverse if possible. Following the guidelines given above leads to the simplification shown in (4). If we let $A = \mu_i(x)$, $B = 1/(\sqrt{2} * \sigma_i(x))$ and $C = \ln(\sigma_i(x))$, (3) can be rewritten as (4).

$$((m_i - A) * B)^2 + C. \quad (4)$$

In this simplification, the three variables A, B, and C replace the original μ and σ in the data tables. Variable A is simply equal to μ . Variables B and C can be precalculated by the host before being stored in the data tables used for this calculation, saving the time and resources of both the arithmetic operations and the transcendental function. With these simplifications, only four operations are required:

1. Subtract $(m_i - A)$.
2. Multiply the results by B.
3. Square the results.
4. Add C.

This greatly reduces the computational complexity of each iteration of the likelihood equation and will be substituted for the original equation in all of the following uses.

D. 21 Channel Solution

The original SBP solution summed the log likelihood equation for each of the 64 channels of the 8×8 sensor. However, it was found that using channels far from the peak reduced positioning accuracy. Using the 21 channels surrounding the peak energy, as shown in Fig. 3, led to better positioning performance. In addition to improving positioning, this results in a ~3:1 reduction in the number of operations needed for each solution.

E. Conversion to Fixed-Point

Floating-point calculations are computationally expensive in an FPGA implementation. Converting to integer representations, while retaining the required resolution, necessitates multiplying by a constant before the conversion to move fractional parts of the value into the

resulting integer. Testing has shown that the values used in the algorithm can be reduced to a 2-byte fixed point number while maintaining the range of values and accuracy of the original algorithm [5]. The fixed point numbers can be precomputed and stored on the hardware for use with the FPGA, allowing all calculations to be made using fixed point numbers. This conversion significantly reduces memory requirements and computational effort.

F. Hierarchal Search

An exhaustive search method has the advantage of being easy to implement and is guaranteed to always find the absolute minimum solution. It has the disadvantage of being very inefficient, both in terms of the number of iterations, and the number of memory accesses to find a solution. The solution set, Fig. 4, illustrates the algorithm's output for all possible positions given an event. The vertex of the solution set is the location at which the event is most likely to have occurred. Observation of the properties of the solution set made it apparent that a structured search could be used to converge to a solution with many fewer iterations.

Of the structured searches explored [5], it was found that a hierarchal search provided the best combination of reduced iterations and memory usage, allowing memory to be partitioned in a way that was conducive to a pipelined implementation. This search functions by finding the solutions at a number of evenly distributed samples points. The minimum result from these samples identifies the region holding the ultimate solution and a new, smaller solution space is sampled, centered on the previous best solution. This process continues iteratively until the solution space is a single pixel. This allows separate tables for each stage of the search consisting of only the values that could be used in that stage. The table size increases in each stage, with the last stage containing data that covers the entire range of positions.

This algorithm yields a large reduction in the number of points that need to be sampled. Variations in the number of sampled points at each iteration were simulated, ranging from a 2×2 array of samples at each iteration of the search, resulting in an 8 iteration search in a 127×127 solution space to 5×5 samples resulting in a three iteration search. These require a total of $2 \times 2 \times 8 = 32$ sampled points for the 2×2 search to $5 \times 5 \times 3 = 75$ points for a 5×5 search.

The optimal solution was found to be a hierarchal search with 3×3 samples at each iteration, dividing the solution space to approximately one-quarter of its previous size. An example of this search on a 63×63 solution space is shown in Fig. 5. This example shows the five iterations necessary to fully search the 63×63 space. The solution space of each successive stage is shown by the increasingly smaller box. Within each solution space, the nine X-Y points to be tested are shown, centered on the best solution from the preceding stage. The search concludes with a three by three search at a spacing of one pixel.

The regularity of this search allows each level of the search to consist of an almost identical process, where each iteration starts with a center point for the search and the spacing between points to be tested at this iteration. For the 3×3 search shown, 9 sets of characterization data table values are needed for the second stage, one for each position to be tested (see Fig. 6). The next iteration will consist of potential sample positions at one-half the spacing, with sets of points centered on each of the previous stage's points, with the first stage defaulting to being centered on the center of the entire data table. This results in each stage's data table having approximately four times the number of testable locations as the previous stage, but still only a fraction of the overall table, until the last stage is reached. The last stage is the only stage that needs access to the entire table. Each iteration will begin

with the best solution from the previous stage as the center point of its search and will test all points adjacent to it at the current resolution. As each stage passes its best result to the next stage, eventually the last stage is reached where the spacing between adjacent pixels is one. The results of this stage represent the best solution available for the data table used. Using this hierarchal search method always results in solution.

Structuring the search in this way allows points sampled during each iteration to be independent of calculations currently underway in other iterations. This allows calculations for different events to be pipelined, allowing multiple stages to be processed simultaneously.

A second advantage of this method is that the storage requirement for each stage's data table is reduced for each iteration prior to the last. Ideally, for n samples, the new solution space would contain n times as many possible solutions as the previous iteration. For our example with nine samples, we would hope for nine times as many potential sample points at the next iteration. However, experiments determined that when the final solution was located approximately midway between two sampled points, this solution was not always nearest to the best sampled point. The cause of this was found to be that the solution set is not completely symmetric and is subject to some solution noise. To solve this problem, each successive iteration must operate on a subset that is significantly larger than $1/n$ times the size of the previous iteration.

Our approach is to have the solution space dimension at each stage to be an integer power of 2, minus 1 ($2^n - 1$). Constraining each level to have an odd number of values allows each solution space to have a pixel in the exact center, with other X-Y locations spaced at a power of two. A representation of a simple four stage system is shown in Fig. 6. The top row of Fig. 6 shows the spacing of the locations that can be tested at each stage. The bottom row of Fig. 6 shows how these values are stored using a compressed addressing scheme where the unused values are not included in the table for that stage. This allows the data table at all stages except for the final stage to be stored in a much smaller memory space. The storage of the odd number of values is padded with blank values to allow separate X and Y addresses to access the desired coordinate without requiring an address calculation. The proper value is addressed by simply appending the binary representation of the X location to the Y location. For example, if the desired X coordinate is 11_b and the Y coordinate is 01_b , the address of the desired value is at 1101_b . As the solution moves from one stage to the next, address translations to match the next stage are accomplished by multiplying the X and Y coordinates by two. This is accomplished by appending a 1 to the binary address values. For the example above, $11_b \rightarrow 110_b$ and $01_b \rightarrow 010_b$, so the resulting address of the same coordinate in the next stages addressing scheme would be 110010_b . This allows the hardware system to do in-stage addressing and next-stage address translations in zero time.

As the solution progresses, the only information passed from one stage to the next is the center point on which to base that stage's search. Each stage has access to data table information at twice the resolution as the previous stage, but will only be accessing values directly adjacent to the center value, as shown in Fig. 7. This is done by sequentially adding then subtracting one from the values of the center X and Y positions passed from the previous stage.

In this example, it can be seen that the spacing between tested locations reduces from four in the 15×15 space, to two in the 7×7 space, to one pixel as the last 3×3 stage is reached. The one pixel spacing of the last stage results in an exhaustive search of this reduced space.

G. Pipelined Architecture

To attain the desired processing rates for this project, multiple events must be simultaneously processed. Although parallel processing paths could easily be formed in the FPGA, multiple copies of the complete data tables would be required to support this architecture. Using the search described above, a pipelined architecture is possible. In this design, each stage of the hierarchical search is implemented with dedicated hardware that has exclusive access to a data table appropriately sized for the stage. As each event's processing at one stage is completed, the intermediate results are passed as a starting point to the next stage, allowing the next event to immediately begin processing. This results in multiple events simultaneously being at various stages of their processing without requiring multiple complete copies of the data tables.

H. Memory Division

The characterization tables for each sensor must be available for solving the likelihood function. The most direct approach would be to store the tables in the FPGA's on-chip SRAM memory. Processing the events would be simplified compared to accessing off-chip synchronous dynamic (SD) or double data rate (DDR) memory. The difficulty with this method is the large size of the tables. Each of the three tables requires 16-bit values for each of the 8×8 channels of sensed data, for 127 possible positions in each of the X and Y axes. This equates to $[3 \text{ Tables}] \times [2 \text{ byte values}] \times [8 \times 8 \text{ channels}] \times [127 \times 127 \text{ possible X - Y locations}]$ or >6 Mbytes. This is well above the approximately 300 kB to 1.1 MB available in members of the family of FPGA devices chosen for this project (i.e., Altera Stratix II or III devices, Altera, San Jose, CA).

The prototype design employs a mix of FPGA internal memory and high-speed external memory. This allows all the small data tables required by the early stages of the process to have individual data and address busses, allowing parallel access by the different stages. The largest data tables are kept in external memories where adequate storage is available. Because most of the processing is done in the multiple earlier stages, memory bandwidth requirements are significantly reduced to the external memories.

I. Input Buffering

Although the average event rate is expected to be 250 K events per second or less, because of the random nature of positron decay instantaneous peak rates as high as 2 times this value are expected. An input FIFO buffer allows data collection at up to the peak rate, and delivery to the SBP algorithm at no more than the designed rate.

J. Prototype Implementation

A block-diagram of the system design is shown in Fig. 8. The numbered components of this system are:

1. An interface to the incoming data.
2. A FIFO buffer. This allows the randomly occurring samples to be collected at up to the peak rate, but processed at their average rate. This is necessary in the target system because the events occur randomly and may momentarily occur at faster rates than the SBP algorithm can process them. This buffer allows them to be stored and dispensed to the algorithm at a lower average rate.
3. This block determines the row and column of the sensor with the peak energy. This is made available to the SBP algorithm to allow the 21-cell version of the SBP algorithm to calculate the proper channels to process for the corresponding event.

4. The values of an event are stored locally at each stage so that they can be accessed for the calculations required at that stage. As processing of a particular stage completes, the values are passed along to the next stage. Double-buffering is used to allow each stage to transfer the values to the next stage before either stage has completed its current calculations. This allows the transfer of data to overlap the calculations.
5. Row center (R_c), Column center (C_c). The values of the row and column centers for the 21-cell solution are also stored locally at each stage.
6. The characterization tables for each stage holds the data required for the SBP algorithm.
7. The SBP calculations to determine the likelihood values are performed here. As each X-Y position is tested, if the sum is less than previously tested positions the X-Y value is saved. When the last X-Y position is tested, the X-Y position with the lowest sum is made available to the next stage as the center X-Y position ($X_c - Y_c$) of the positions to be tested.
8. Increment dX , and dY . This sequences each step of the algorithm through the proper X and Y locations for each X and Y position to be tested. The dX (delta X) is added to the X_c (X center) to determine proper position on which to perform the likelihood calculation. The Y values are computed in the same way. Additional timing signals are generated here to signify occurrences of the first X-Y and last X-Y tested to coordinate resetting values at the beginning of calculations and transferring values at the end of calculations.
9. Increment $dRow$ and $dCol$. This sequences the likelihood calculation through all the rows and columns of the 21-cell solution. Similar to the X-Y locations, the delta values for the row and column are added to the center location of the 21 cells to be included in the calculation. Additional timing signals are generated here to signify occurrences of the first row/column and last row/column to coordinate resetting values at the beginning of each iteration of the likelihood function.

When the computations at each stage are complete the event data, the row and column centers, and the newly computed X and Y centers (X_c , Y_c) are passed to the next stage. Each stage operates identically to the previous stage except that the data tables are larger at each successive stage. The number of potential X-Y positions that will be tested for each event at each stage are the same, but they can have a larger number of potential X_c and Y_c positions on which to center this stage of the search. This is accommodated by appending a single bit to the least significant end of the X_c and Y_c values as they are transferred to the next stage. This also explains the apparent lack of a first stage. The first stage data table would consist of only one location, so the results of this computation are known without requiring any computations. The address of this value (0,0) is passed to stage two and used as the starting point for that stage. The result at each stage is the X-Y location of the best fit to the characterization data at that stage. The result of the final stage is the value that represents the highest precision available in this implementation.

III. Results

To test the accuracy of our design, the results of a number of sampled locations were compared to the results of the exhaustive search on an event by event basis. These comparison tests were run on experimental data. The results show that over 90% of the events show no error and over 99% of the positioned events are within 1.4 pixels (0.54 mm), representing all the pixels immediately adjacent to the reference pixel. Note that in this comparison, diagonal pixels are represented by the Euclidean distance.

The accuracy of the simplified algorithm can be confirmed by running tests with both versions of the algorithm using the same data. A sample test point for both the integer and floating point methods, Fig. 9, illustrates no noticeable difference between the two methods. The average FWHM intrinsic spatial resolution for a range of point positions starting in the corner of the crystal and moving toward the center of the crystal along a diagonal (Table I) is almost identical for the two methods. The results in Table I are consistent with our measurement of an average intrinsic spatial resolution of ~ 1.6 mm FWHM across the useful imaging area of the detector as shown in Fig. 2. The largest difference between the two methods was for the data point near the corner of the crystal. We know that positioning near the edges and corners of a monolithic crystal detector are the most challenging. On the other hand, we have shown that using LUTs with some depth of interaction information can improve the decoding performance of a monolithic crystal detector, especially near the corners of the crystal [3]. Therefore, we are currently working on an FPGA implementation that will allow for depth of interaction processing. This implementation is beyond the scope of this paper.

The throughput of the system is defined as the rate at which new solutions are completed. In this pipelined design, it is equal to the time for one stage to complete. For the current design, each stage requires nine summations of the 21-channel solution, each being able to complete in one clock cycle. Additionally, three clock cycles are required for transferring this value to the next stage. This results in a new event starting, and one completing, every 192 clock cycles. At the designed clock rate of 70 MHz, this results in a throughput of >60 k events per second.

The latency is defined as the time from the acceptance of the data to the presentation of the processed SBP position to the host. Disregarding the time required to pass the detector data through the FIFO and the time to pass the results to the host, this is equivalent to the combined time of all the stages. For a complete seven stage system, this would be seven times 192 clock cycles at 70 MHz, or approximately $20 \mu\text{s}$.

IV. Conclusion

This study showed that optimizations to the SBP algorithm to allow real-time implementation on an FPGA can be made without adversely affecting the accuracy of the results. These optimizations included algebraic manipulation of the SBP equation to allow higher speed computations, reduction of the number of cells required for each solution, conversion to fixed-point math, and the use of a structured search algorithm.

The hierarchal search method developed as part of this study greatly increases the search speed over current implementations and allows the memory requirements of the SBP method to be distributed across multiple stages and both internal and external memories. This allows a multistage pipeline solution to be used. This was a significant factor in increasing the speed of the SBP solution.

Collectively, these optimizations allow an accelerated SBP solution to be found at a rate that exceeds the maximum expected average event rate, to an accuracy that compares satisfactorily with existing methods. This project has shown that it is possible to use an FPGA implementation of SBP to facilitate real-time processing of event data for a PET system. This is a major step forward in the development of a research preclinical PET system being built at the University of Washington.

Acknowledgments

This work was supported in part by an NIH-NIBIB Grant R21/R33 EB001563 and by Altera Corporation.

References

1. Joung J, Miyaoka RS, Lewellen TK. CMiCE: A high resolution animal PET using continuous LSO with a statistics based positioning scheme. Nucl Instrum Meth Phys Res A Aug;2002 489(1–3):584–589.
2. Ling T, Lee K, Miyaoka RS. Performance comparisons of continuous miniature crystal element (cMiCE) detectors. IEEE Trans Nucl Sci 2006;53:2513–2518.
3. Ling T, Lewellen TK, Miyaoka RS. Depth of interaction decoding for a continuous crystal detector. Phys Med Biol Apr;2007 52:2213–2228. [PubMed: 17404465]
4. Hauck, S.; DeHon, A., editors. Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation. New York: Morgan Kaufmann/Elsevier; 2008.
5. DeWitt, D. Master's thesis. Dep. Elect. Eng., Univ. Wash; 2008. An FPGA implementation of statistical based positioning for positron emission tomography.

Fig. 1.
Picture of cMiCE detector with 8 mm thick LYSO crystal and white paint.

Fig. 2.

Contour plot illustrating the FWHM for point fluxes perpendicular to the detector with 2.026 mm spacing. Fluxes go to within 2.7 mm of the edge of the crystal. The dots represent the point flux locations on the detector surface. The average FWHM intrinsic spatial resolution of the detector is 1.6 mm FWHM.

Fig. 3.
21 channel masking, centered on the peak energy channel.

Fig. 4. The complete solution set of all possible positions given a sample event. The minimum point of the solution set is selected as the most likely position.

Fig. 5.
Five iterations of a 3×3 search on a 63×63 solution space.

Fig. 6.
Compressed storage for data tables.

Fig. 7.
Diagram of the last three iterations of a hierarchal search.

Fig. 8.
Block diagram of FPGA implementation of SBP. The numbered components are defined in Section II-J.

Fig. 9.
Sample point source image for floating point (left) and fixed-point (right) calculations. Pixel dimensions are 0.385 mm by 0.385 mm.

Comparison of Reference (Original SBP Implementation) and Optimized Calculation (FPGA Implementation) FWHM Values

TABLE I

Position from edge of crystal (mm)		Exhaustive search FWHM (mm)		Simplified algorithm FWHM (mm)	
X	Y	X	Y	X	Y
2.70	2.73	1.95	2.50	1.41	2.69
4.73	4.73	1.99	1.18	2.05	1.07
6.75	6.75	1.89	1.69	2.06	1.85
8.78	8.78	1.60	1.44	1.68	1.52
10.80	10.80	1.43	1.42	1.35	1.42
12.83	12.83	1.27	1.38	1.21	1.23
14.86	14.86	1.45	1.50	1.40	1.41
16.88	16.88	1.36	1.14	1.41	1.30
18.91	18.91	1.26	1.24	1.23	1.23
20.93	20.93	1.40	1.33	1.32	1.38
22.96	22.96	1.39	1.21	1.35	1.15
24.99	24.99	1.74	1.41	1.61	1.58
Average FWHM		1.51	1.49	1.56	1.45