
Article:
Bogdan, I.A., Coca, D. and Beynon, R.J. (2009) Peptide mass fingerprinting using field-programmable gate arrays. IEEE Transactions on Biomedical Circuits and Systems , 3 (3). pp. 142-149. ISSN 1932-4545

https://doi.org/10.1109/TBCAS.2008.2010945

Peptide Mass Fingerprinting Using Field-Programmable Gate Arrays

István A. Bogdán, Daniel Coca, and Rob J. Beynon

***Abstract*—The reconfigurable computing paradigm, which exploits the flexibility and versatility of field-programmable gate arrays (FPGAs), has emerged as a powerful solution for speeding up time-critical algorithms. This paper describes a reconfigurable computing solution for processing raw mass spectrometric data generated by MALDI-TOF instruments. The hardware-implemented algorithms for denoising, baseline correction, peak identification, and deisotoping, running on a Xilinx Virtex-2 FPGA at 180 MHz, generate a mass fingerprint that is over 100 times faster than an equivalent algorithm written in C, running on a Dual 3-GHz Xeon server. The results obtained using the FPGA implementation are virtually identical to those generated by a commercial software package MassLynx.**

***Index Terms*—Biomedical computing, field-programmable gate arrays (FPGAs), mass spectrometry, optimization methods, proteins.**

I. Introduction

MASS spectrometry has evolved rapidly in the past decades, becoming one of the most reliable proteomics research tools. The amount of data that is currently generated by mass spectrometers around the world is growing at increasing rates. Processing and interpreting mass spectrometric data rely on computer algorithms and proteomic databases running on standard microprocessor systems.

Peptide mass fingerprinting is a protein identification technique in which mass spectrometry is used to determine the masses of peptide fragments generated by specific digestion. The proteins are identified by matching the measured molecular masses of peptide fragments against theoretical peptides generated from protein-sequence databases. Peptide mass fingerprinting involves two basic operations, namely, the processing of raw spectra to derive a mass fingerprint and using the mass fingerprint to search the protein database for a possible match. A correlation score is computed between the database entries and the unknown peptide fragment mass list. The matches with the highest score are the final protein list to be returned to the user.

While the processing time for performing protein identification is relatively low, it is still greater than the spectra acquisition time and is limited by the microprocessor clock frequency.

The most effective approach to speed up computations involves the development of dedicated hardware processors that are optimized to perform specific algorithms. The speed up in computation, compared with the standard sequential microprocessor, is achieved by concurrent implementation of different arithmetic and logic operations that make up a computational loop and by concurrent execution of several computation loops. A major drawback of this approach used to be the prohibitive costs associated with manufacturing a dedicated integrated circuit [application-specific integrated circuit (ASIC)].

The hardware implementation approach has become a cost-effective solution thanks to the availability of high-density field-programmable gate arrays (FPGAs) and of high-level system design and development tools, which make the implementation of very complex hardware designs possible with almost the same ease as the software implementation. An FPGA is a large-scale IC that can be programmed (and reprogrammed) after it has been manufactured.

Early attempts to use FPGA devices in biocomputation were made to accelerate gene-sequence analysis [7]. FPGAs, which are well suited for high-performance, high-bandwidth, and parallel-processing applications, have been successfully employed to speed up DNA sequencing algorithms [8]–[10], [6], [11], [13]. FPGAs were also used in the attempt to accelerate the search of substrings similar to a template in a proteome [12]. More recently, FPGAs have been used to accelerate sequence database searches with MS/MS-derived query peptides [4]. This hardware-based solution can reportedly locate a query within the human genome about 32 times faster than a software implementation running on a 2.4-GHz processor. A hardware-sequence alignment tool implemented in FPGA is also available [14].

This paper describes the design and hardware implementation of a raw spectra processor which performs all computational tasks involved in the generation of a mass signature from a raw spectrum, namely, smoothing, peak detection, and deisotoping. The processor, which is implemented on a Xilinx XC2V8000 FPGA and runs at 180 MHz, achieves more than 100 fold speedup compared with a C software implementation running on a dual 3-GHz Xeon Server with 4-GB of memory. In an earlier paper [15], we have successfully tested the implementation in terms of peak extraction and deconvolution accuracy against

Manuscript received February 06, 2008; revised June 02, 2008. Current version published May 22, 2009. This work was supported by the BBSRC under Grant BBS/B/16402 and sponsored by Xilinx, Inc., who donated the FPGA devices and design tools used in this work. This paper was recommended by Associate Editor R. Butera.

I. A. Bogdán and D. Coca are with the Automatic Control and Systems Engineering Department, Sheffield University, S1 3JD, U.K. (e-mail: i.bogdan@sheffield.ac.uk; D.Coca@Sheffield.ac.uk).

R. J. Beynon is with the Proteomics and Functional Genomics Group, Faculty of Veterinary Science, University of Liverpool, Liverpool, L69 7ZJ, U.K. (e-mail: r.beynon@liv.ac.uk).

Digital Object Identifier 10.1109/TBCAS.2008.2010945

commercial software implementations. This paper focuses in more detail on the actual processor design.

II. Algorithm Description

Following specific protein digestion, a MALDI-TOF mass spectrometer generates pairs of mass-to-charge (m/z) and abundance values symbolized with $d_k = (x_k, y_k), (k = 1, 2, \ldots, N)$. Typically, the number of points (N) in the spectrum ranges from a few thousand to a few hundred thousand. The determination of experimental peptide masses (the so-called peptide mass fingerprint) requires relatively complex processing of the raw mass spectrum in order to discriminate between spectral peaks that correspond to digested peptides and associated isotopes and the spurious peaks caused by noise and sample contamination.

The FPGA spectra processor was designed to implement, with some variations, an algorithm proposed in [1] which is used in a popular mass spectrometry software package. The current processor implements additional optional smoothing operation and uses a different algorithm to implement deisotoping [see Step 6)] . The algorithm was found to be computationally efficient and well suited for hardware implementation but, in principle, other peak extraction algorithms could be considered for hardware implementation.

Step 1) Smoothing (Optional): Performing Savitzky–Golay smoothing over the raw input spectrum can reduce the effect of instrumentation noise. The algorithm is based on performing a least-squares linear regression fit of a polynomial of degree M over at least $M + 1$ data points around each point in the spectrum to smooth the data. The main advantage of this procedure is that it tends to preserve the shape of the signal peaks [3]. The smoothing operation is implemented as a standard finite-impulse-response (FIR) filter

$$ys_i = \sum_{j=1}^{F} b_j^M y_{i-j+1}$$

where F is the size of the smoothing window, $(F = 2q + 1), y$ is the input data stream, ys is the FIR output, and b_j^M are the time-varying filter coefficients. For a given filter of order M and (odd) frame size $F(F > M + 1)$, all of the coefficients needed to implement the smoothing operation to form a $F \times F$ matrix $[b_{i,j}^M]_{i,j=1}^F$. In a smoothing filter implementation, the last $(F-1)/2$ rows are used for the first F data points, the first $(F-1)/2$ rows are used with the last F data points, and the middle row is used with the rest of the spectrum. The Savitzky–Golay smoothing operation can be represented in matrix form as follows:

$$\begin{bmatrix} ys_1 \\ ys_2 \\ \cdots \\ ys_q \end{bmatrix}^T = \begin{bmatrix} b_{2q+1,1} & b_{2q+1,2} & \cdots & b_{2q+1,2q+1} \\ b_{2q,1} & b_{2q,2} & \cdots & b_{2q,2q+1} \\ \cdots & \cdots & \cdots & \cdots \\ b_{q+2,1} & b_{q+2,2} & \cdots & b_{q+2,2q+1} \end{bmatrix} \times \begin{bmatrix} y_{2q+1} \\ y_{2q} \\ \cdots \\ y_1 \end{bmatrix}$$

$$\begin{bmatrix} ys_{q+1} \\ ys_{q+2} \\ \cdots \\ ys_{N-q} \end{bmatrix}^T = \begin{bmatrix} b_{q+1,1} \\ b_{q+1,2} \\ \cdots \\ b_{q+1,2q+1} \end{bmatrix}^T \times \begin{bmatrix} y_{2q+1} & y_{2q+2} & \cdots & y_N \\ y_{2q} & y_{2q+1} & \cdots & y_{N-1} \\ \cdots & \cdots & \cdots & \cdots \\ y_1 & y_2 & \cdots & y_{N-2q} \end{bmatrix}$$

$$\begin{bmatrix} ys_{N-q+1} \\ ys_{N-q+2} \\ \cdots \\ ys_N \end{bmatrix}^T = \begin{bmatrix} b_{q,1} & b_{q,2} & \cdots & b_{q,2q+1} \\ b_{q-1,1} & b_{q-1,2} & \cdots & b_{q-1,2q+1} \\ \cdots & \cdots & \cdots & \cdots \\ b_{1,1} & b_{1,2} & \cdots & b_{1,2q+1} \end{bmatrix} \times \begin{bmatrix} y_N \\ y_{N-1} \\ \cdots \\ y_{N-2q} \end{bmatrix}.$$

More details about this procedure can be found in the original paper [3]. Typically, for processing a raw spectrum, filters of order $\mathrm{M} = 11$ and a frame length $F = 23$ were found to produce the best results for spectra having around 50 samples/(m/z). The hardware implementation allows the user to specify the size of the smoothing window $(F_{\max} = 43)$ and the corresponding filter parameters.

Step 2) Baseline and Noise Detection: The raw spectrum mass list $\{x_k | k = 1, 2, \ldots, N\}$ is divided into small intervals (m_i) of width ω, and local minimum (Z_i), maximum (Y_i), and abundances and their differences (W_i) are computed [1]. In the equation that will be shown, the (x, y) pairs correspond to data points

$$Z_i = \min(y | x \in m_i)$$
$$Y_i = \max(y | x \in m_i)$$
$$W_i = Y_i - Z_i, i = 1, \ldots, \lceil N/\omega \rceil$$

where $\lceil N/\omega \rceil$ denotes the smallest integer greater than or equal to N/ω. For each integer mass in the spectrum $(x_j, j = 1, 2, \ldots, k)$, a symmetric window (M_j) of width $\Omega\omega$ is placed around, and the baseline and noise levels are estimated as follows:

$$Y_{\mathrm{base}}(j) = \sum_{i=j(-)}^{j(+)} w_i Z_i$$
$$Y_{\mathrm{noise}}(j) = \sum_{i=j(-)}^{j(+)} w_i Y_i$$
$$w_i = \frac{\frac{1}{W_i^2}}{\sum_{p=j(-)}^{j(+)} \frac{1}{W_p^2}}$$

where $j(-)$ is the index of the leftmost subinterval (m_i) covered by M_j, and $j(+)$ is the index of the rightmost subinterval (m_i) covered by M_j [1]. Subsequently, the signal-to-noise ratio (SNR) (s_k) is computed for each spectral point $d_k, k = 1, 2, \ldots, N$

$$s_k = \frac{y_k - Y_{\mathrm{base}}(j_k)}{Y_{\mathrm{noise}}(j_k) - Y_{\mathrm{base}}(j_k)}, j_k = \lfloor m_k \rfloor$$

where $\lfloor m_k \rfloor$ denotes the largest integer smaller than or equal to m_k.

Step 3) Spectrum Segmentation: According to the signal-to-noise ratio (computed in the previous step) relative to a user adjustable threshold SN [1], the spectrum is segmented into three categories: noise ($d_k \in D_{\text{noise}}, s_k < 1$), support ($d_k \in D_{\text{support}}, 1 \leq s_k < \text{SN}$), and signal ($d_k \in D_{\text{signal}}, \text{SN} \leq s_k$).

Step 4) Peak Detection: Peaks are constructed from data points that are signal or support points and are bounded by noise. A collection of data points (P) used to construct a peak has the following criteria: $P = [d_j | x_j < x_{j+1}; j = 0, 1, \ldots, p, p+1]; d_0 \in D_{\text{noise}}; d_{\text{p+1}} \in D_{\text{noise}}; d_{\text{j}} \in D_{\text{support}}$ or $d_{\text{j}} \in D_{\text{support}}$ for $j = 1, \ldots, p$. The center of the mass (m_p) and relevant molecule abundance (a_p) are computed for each constructed peak [1] as follows:

$$m_p = \frac{\sum_{j=0}^{p+1} x_j y_j}{\sum_{j=0}^{p+1} y_j}$$
$$a_p = \max_P \{y_j\} - Y_{\text{base}}(j).$$

Step 5) Clustering: This involves grouping valid peaks into clusters. Two peaks P and Q are in the same cluster if $1 - \tau < |m_P - m_Q| < 1 + \tau$, where τ is a user-defined parameter (typically, $\tau = 0.2$) [1]. A valid cluster has at least one peak with a data point d_k in D_{signal}.

Step 6) Peak Deisotoping: The major difference compared to [1] is the method used by the FPGA processor to implement aggregation of natural isotopomers (due primarily to the natural abundance of ^{13}C and ^{15}N). The algorithm implemented in FPGA uses Poisson distributions to approximate the isotopic patterns for every peptide [2]. The expected proportional abundance of the heavier isotopes $m_p\ (i), i = 1, 2, \ldots$, with respect to the monoisotopic peaks $m_p = m_p(0)$, are computed as follows:

$$E(i, m_p) = \frac{a_p F(m_p)^i}{i!}; i = 1, 2, \ldots; E(i, m_p) > 0;$$
$$F(m_p) = 0.000594 m_p - 0.03091 \quad (1)$$

where $\text{E}(1, m_p)$ is the theoretical abundance of the first isotope of $m_p, E(2, m_p)$ is the abundance of the second isotope, etc. The best-fit Poisson models of isotopic distributions are shown to match those of theoretical distributions [2].

The complete processing step for a cluster of R consecutive peaks can be summarized as follows.

- The leftmost peak in the cluster is always considered to be a monoisotope [2].
- Compute the expected abundance of the heavier isotopes $E(1, m_1), E(2, m_1), \ldots, E(R-1, m_1)$ [2].
- Subtract these higher contributions from the actual abundances of the next $R-1$ peaks $a_2 - E(1, m_1), a_3 - E(2, m_1), \ldots, a_R - E(R-1, m_1)$. Only the results higher than a threshold (ISOTHR) are retained for further processing [2].

These substeps are recursively repeated for the residual cluster until all of the residual peaks are less than the threshold [2]. For example, if $a_2 - E(1, m_1) > \text{ISOTHR}, (m_2, a_2 - E(1, m_1))$ becomes the next monoisotope and the steps are repeated.

Fig. 1. Deisotoping example. (a) Overlapped peaks. (b) Isotopes of peak 1. (c) Delsotoped residuals.

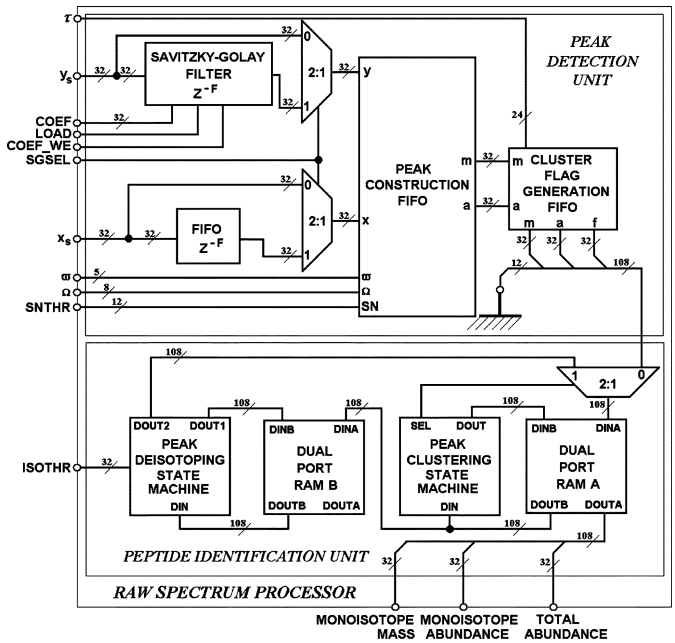

Fig. 2. Raw spectrum processor block diagram.

The procedure is illustrated on Fig. 1. Here, a small cluster of four peaks ($R = 4$) is depicted in Fig. 1(a). The first peak is considered monoisotope. Its isotopic distribution is shown in Fig. 1(b). Fig. 1(c) shows the resulting residual cluster after subtracting (b) from (a).

III. Hardware Implementation

The block diagram of the hardware processor is depicted on Fig. 2. The implementation has two major functional blocks: 1) a peak detection unit, which identifies all significant spectral peaks and 2) a peptide identification unit that generates the final list of peptide masses and associated abundances.

The peak detection unit implements smoothing, baseline, and noise-level estimation in order to discriminate between signal and noise peaks. The first block is a Savitzky–Golay smoothing filter [3] that implements the equations from the first algorithmic step. It has a user-defined window that can be chosen according to the instrument resolution setting (number of data points recorded per 1 m/z unit). The smoothing operation is optional; the user can specify if the data are preprocessed or not by the SGSEL input flag. A 43-tap FIR filter implements the Savitzky–Golay smoothing filter with coefficients that may be reloaded as user parameters in an LUT.

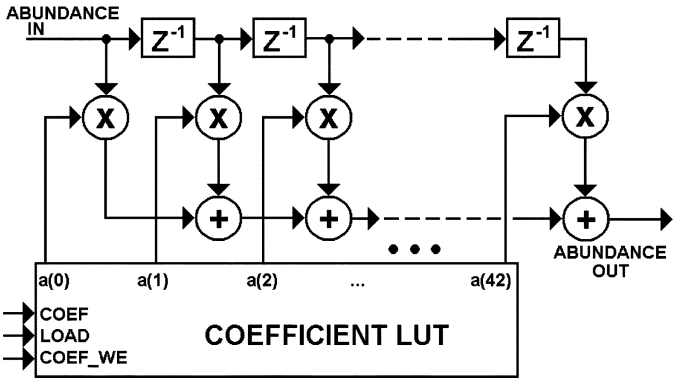

Fig. 3. Savitzky–Golay filter implementation.

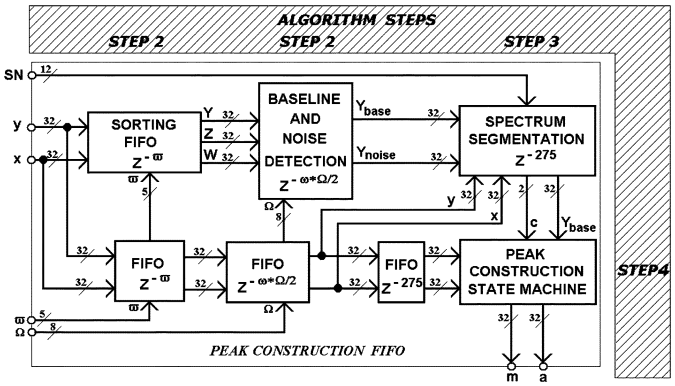

Fig. 4. Peak construction first-in first-out (FIFO) block diagram.

The coefficients are loaded into the FIR on its dedicated inputs (COEF, LOAD, COEF_WE). The first F spectral abundances are loaded as coefficients and the last $(F-1)/2$ rows of the filter coefficient matrix are loaded on the FIR data input. Then, the middle row of the filter coefficients is loaded as coefficients and the spectrum is loaded at the FIR data input. Finally, the last F data points are loaded as FIR coefficients, the last $(F-1)/2$ rows of the coefficient matrix are loaded as data input. The block diagram of the FIR filter, depicted on Fig. 3, is implemented as a single channel highly parallel filter by using a Xilinx Logicore block [5]. While the abundances are smoothed, the mass list is delayed by the FIR latency. Processing time for this step is given by

$$t_1 = \left\{ 3\left(\left\lfloor \frac{F+3}{4} \right\rfloor 64 + 18 + F \right) + (F-1)^2 + N \right\} T_{\mathrm{CLK}}$$

where N is the spectrum length, T_{CLK} is the clock period, and all constants inside the first round bracket are specified in the Xilinx LogiCore FIR implementation datasheet [5].

The peak construction pipeline ($311+\omega+\omega\Omega/2$ stages long) is depicted in Fig. 4. It implements the algorithmic steps described in the previous section (Steps 2–4). The sorting FIFO detects the minimum (Z) and maximum (Y) abundances and computes their differences (W) over a sliding window of length ω. It is implemented by using a filter with a structure similar to a median filter that sorts in ascending order its input data stream over its filter length. Instead of computing the median, the maximum and minimum values are found.

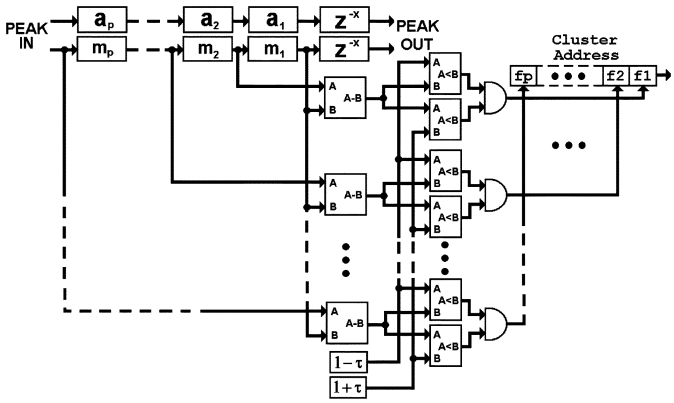

Fig. 5. Cluster flag generation FIFO.

Baseline and noise are computed over a bigger spectral interval of Ω small windows of length ω Baseline and noise detection are implemented using 32-b pipeline dividers and accumulators. Spectrum segmentation is performed by computing an SNR for each delayed spectrum data point (x, y) according to Step 3). The result is then compared with the user-selectable threshold (SNTHR), and a classification flag c coded on two bits is assigned for each data point. The flag c can be 1, 2, or 3 depending on whether the respective data point is classified as noise, support, or signal, respectively. The original spectral points (x, y), their associated classification flag c, and the baseline (Y_{base}) are aligned and fed into the peak construction state machine. Here, the centered mass m_k and baseline-subtracted abundance a_k of spectral peaks are computed according to algorithm Step 4).

Steps 1–4 are implemented in a pipeline manner and the total processing time is given by

$$t_{1-4} = \left[3\left(\left\lfloor \frac{F+3}{4} \right\rfloor + 18 + F \right) + (F-1)^2 + \omega + \omega\Omega/2 + 311 + N \right] T_{\mathrm{CLK}}.$$

The last functional block of the peak detection unit—the cluster flag generator FIFO is depicted in Fig. 5. Its role is to detect possible peak candidates that are isotopes of one or more singly charged chemical compounds, separated by the mass of a neutron. This group of peaks is called a cluster. Clustering involves grouping together peaks so that the m/z (mass to charge ratio) distance between two successive peaks is between $1-\tau$ and $1+\tau$, where τ is a user-selectable value, typically set to 0.20. The circuit is a delay line for the input data peaks with a maximum length of p.

It is assumed that the spectrum is sorted by increasing mass. The distance between the masses of all consecutive signal peaks starting with the lowest mass value m_1 is computed. To speed up computations, there are p circuits that compute mass differences $m_1-m_2, \ldots, m_{p+1}-m_1$ between m_1 and the following p consecutive mass values $m_2 < m_3 < \cdots < m_p < m_{p+1}$ in parallel. In our design, p is an adjustable parameter, which is selected according to mass spectrometer resolution, to be larger than the maximum number of signal peaks that are registered

within a window of $1 + \tau$ m/z. Typically, about 50–100 samples/(m/z) are taken, so the FIFO length $p = (60-120)/3 =$ 20-40 or less. The output of the circuit is a cluster flag f of p bits, which is generated for each peak (m_k, a_k). If the distance between m_1 and m_k is within the range of $1 \pm \tau$ m/z, the kth bit in f_1 is set to 1, indicating that m_k is a potential isotope of m_1. If all of the bits in the flag f_1 are zero, this indicates that m_1 has no isotopes.

The peptide identification unit consists of two dual-port random-access-memory (RAM) devices A and B (embedded RAM blocks), and two state machines: 1) a clustering and 2) a deisotoping machine. The output of the peak detection unit (peak mass m_k, peak abundance a_k, and cluster flag f_k)) is stored in RAM (A) at consecutive addresses, starting from zero. After all peaks are stored, the clustering state machine switches the input multiplexer and clustering starts.

Clustering is simply a sorting process in which peaks belonging to the same cluster are grouped together and stored at consecutive memory locations. In addition, clusters are also indexed so that consecutive clusters are stored consecutively in the memory. If the pth bit of a cluster flag f_k at address k is one, the peak at address $k + p$ is in the same cluster as the original peak form address k. The process continues until the flag associated with a signal peak in the cluster only has zero entries. Clustering is implemented as a state machine that sequentially reads the first dual-port RAM (A). Each location stores the peak information as a $32 + 32 + 32 + 12 = 108$ 108-b word: 32 b for mass, 32 b for abundance, 32 b for the cluster flag, and 12 b for the cluster index. The cluster flag is used to calculate the memory location of the peaks that are part of the same cluster while the cluster index is an integer that uniquely identifies clusters.

The clustering process is illustrated in Fig. 6. Here, the triplets $(m_1, a_1, f_1), (m_2, a_2, f_2), \ldots (m_N, a_N, f_N)$ in RAM (A) are generated by the cluster flag generator FIFO, explained earlier, and stored at consecutive addresses from 0 to $N-1$. All of these peaks are analyzed starting from address 0. A 12-b counter that stores the maximum cluster index during operations is reset to 1. In this example, [Fig. 6(a)], the $p-1$ bit of the cluster flag f_1 is one. This indicates that the peak (m_p, a_p) stored at address $p-1$ is a heavier isotope of the first peak (m_1, a_1). In the first step, the mass and abundance of the first peak from location 0 in RAM (A) is written to location 0 from RAM (B), and the cluster index of the peak at the first location of RAM (B) is assigned to the clustering index counter value (1—in our case).

The cluster flag field, where f_1 was stored, is overwritten with the address of the next memory location in RAM (B), where the next peak from the cluster will be stored. The process is repeated for the next peak from cluster 1 which is stored at address $p-1$ in RAM (A). Since, in this example, the cluster flag f_p of the peak (m_p, a_p) is null, this means that there are no more peaks to be added to the current cluster (i.e., cluster 1 will consist of only two peaks), and the cluster index counter is incremented to 2. Clustering continues with the remaining peaks from RAM (A) that were not yet visited. In this example, the next peak that will be considered is address 1. Here, the cluster flag is 0, so there are no other peaks to be added to this cluster. The process continues until all of the peaks from RAM (A) are visited. As a result, RAM (B) will hold the same peak list, this time, it is rearranged

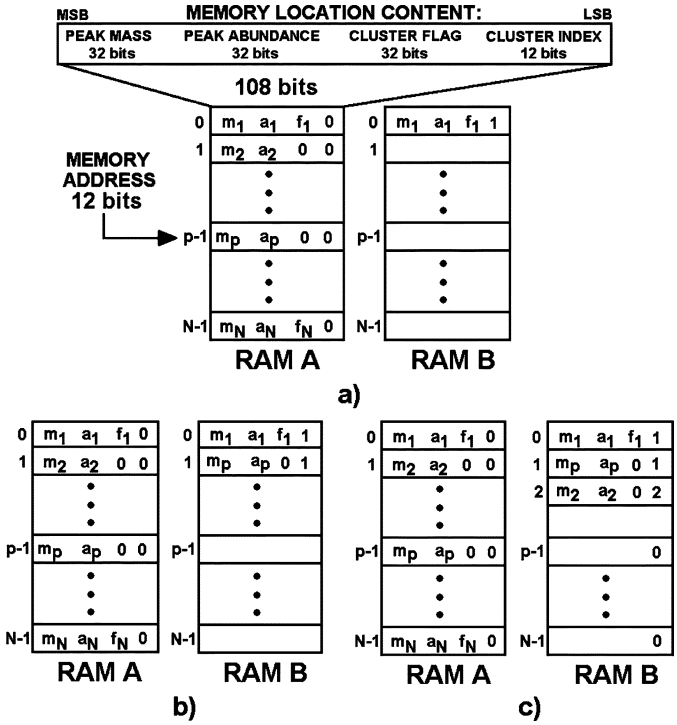

Fig. 6. Memory operations during clustering. (a) STEP 1. (b) STEP 2. (c) STEP 3.

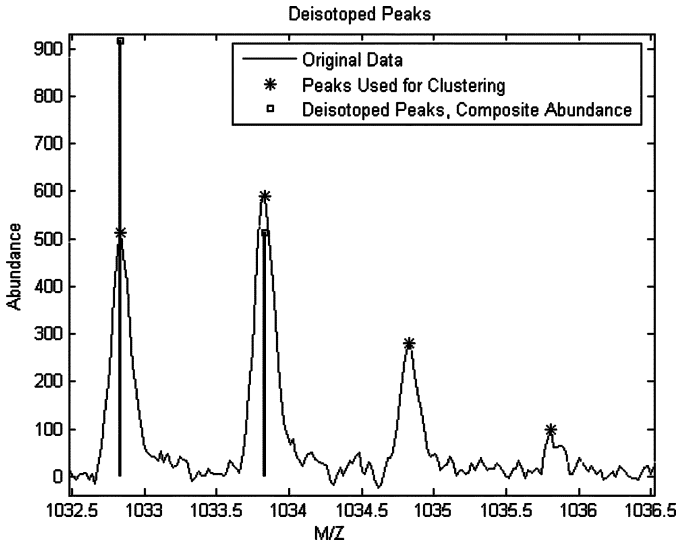

Fig. 7. Fragment of a processed spectrum.

in ascending order according to cluster index and spectral mass. The time required to perform clustering depends on the number N_P of detected peaks in the spectrum, the number of identified clusters, and the number of peaks in a cluster. In the worst-case scenario, the processing time is given by

$$t_{\mathrm{CLUST}} = 10(32N_P - 528)T_{\mathrm{CLK}}$$

where the multiplicative factor of 10 is given by the longest cycle of the clustering state machine.

The final processing step required to generate the peptide fingerprint is deisotoping. In practice, deisotoping is required be-

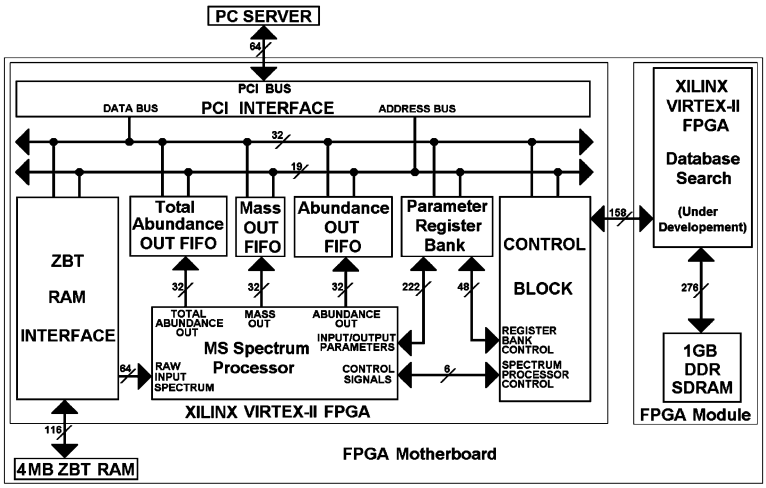

Fig. 8. Block diagram of the complete protein identification solution.

cause one cluster may contain more than one peptide. In other words, the peaks in a cluster can be viewed as a superposition of isotopic distributions of two or more peptides. The deisotoping unit identifies all peptides (the mass of the monoisotope) within a cluster and calculates the total abundance of the peptide corresponding to each monoisotope. The hardware implementation of deisotoping is based on an approximation of isotopic patterns by Poisson distribution as presented in Step 6) of the algorithm in the previous section [2]. Starting with the first peak in a cluster (assumed to be a monoisotope), the algorithm generates the theoretical isotopic distribution based on peak height (abundance) and mass value. The computed abundance values are then subtracted from the original peaks at the corresponding m/z values. Following subtraction, any negative abundance value is set to zero. The procedure was illustrated in Fig. 1. The step is then repeated, with the remaining (height-adjusted) peaks. At each step, the monoisotopic mass value and original and total abundance (the sum of the monoisotopic peak and its theoretical isotopic abundances) are recorded in the final peak list.

The deisotoping unit processes previously computed clusters from the dual-port RAM (B), writes back partial results in RAM (B), and the final peak list in RAM (A). When deisotoping starts, the first location from RAM (B) is read in the first step. The first peak of each cluster is considered monoisotope and is written to RAM (A), where the final peaks are stored. The theoretical abundance $E(1, m_1)$ of the first heavier isotope is computed by using (1) and is used to update the total abundance field (used previously to store the cluster flag) as $a_1 + E(1, m_1)$. Then, the next peak from the cluster is read (m_p, a_p) from RAM (B) and the difference $a_{\rm p} - E(1, m_1)$ between the abundance of this peak a_p and the theoretical abundance contribution $E(1, m_1)$ of the previous monoisotope is computed. If the residual abundance $a_{\rm p} - E(1, m_1)$ is less than a user-defined threshold (ISOTHR), the peak (m_p, a_p) is considered to be a higher isotope of the detected monoisotope and is no longer stored or processed.

However, if $a_p - E(1, m_1) > \text{ISOTHR}$, the residual peak $(m_p, a_p - E(1, m_1))$ is stored back to RAM (B) to be analyzed as the second monoisotope (peptide) in the cluster. The identification of the isotopes of the first peak in the cluster continues until all of the peaks in the cluster are visited.

TABLE I
PEAK PROCESSOR USER-DEFINED PARAMETERS

Name	Description
ω	Small window length; valid choice 3, 5, 7, ..., 15.
Ω	Large window length; valid choice: 2, 4, 6, ... 30.
τ	Isotope distance tolerance, used in clustering; possible range is (0, 0.5].
ISOTHR	Threshold used in deisotoping; below it peaks are not analyzed.
SNTHR	Used to classify peaks in noise, support and signal.

The total abundance of the peptide is updated step by step until the last peak in the cluster is processed. Deisotoping of the first cluster continues with the second monistotope residual peak) identified in the cluster and so on.

This operation is performed for all clusters stored in RAM (B). When processing ends, the harvested peak list in RAM (A) is ready to be used to search the protein database.

If the number of detected peaks is N_P and the number of clusters is N_C, the minimal processing time for deisotoping is N_P when each cluster has one peak ($N_P = N_C$). If each peak in each cluster is a monoisotope, each cluster is processed in $1 + 2 + \cdots + S$ cycles, where S is the number of peaks in the cluster. If $S = [N_P/N_C]$ is the average number of peaks in a cluster, the processing time for deisotoping is

$$t_{\rm DEISO} = 28N_P([N_P/N_C] + 1)T_{\rm CLK}/2, N_C > 1$$

where the multiplicative constant 28 is the length of the longest cycle in the deisotoping state machine.

An example of the spectrum fragment from the range of 1032.5–1036.5 m/z is depicted in Fig. 7. The figure shows all peaks in the cluster, the identified peptide masses, and the abundance associated with each peptide.

The implemented peak processor has a number of user-adjustable parameters, which are given in Table I. A block diagram of the complete FPGA solution, including the database search system, is shown in Fig. 8.

TABLE II
IMAPCT OF SPECTRUM LENGTH ON PERFORMANCE

Spectrum size	Processing time [ms], dual Xeon 3GHz processors	Processing time [ms], Virtex-II FPGA, 180MHz	Speed Gain (*St.Dev.*)
25488	20.27	0.1632	124.20 (*43.86*)
50448	31.23	0.3105	100.56 (*1.38*)
75168	47.33	0.4557	103.86 (*6.13*)
101040	62.50	0.5607	111.46 (*0.9*)
125184	79.17	0.7557	104.76 (*5.17*)
150144	114.33	0.8547	133.76 (*8.97*)
175104	130.20	1.0024	129.88 (*7.46*)
200976	188.63	1.1219	168.13 (*15.55*)

The implementation uses fixed-point arithmetic on 32 b with the fractional part on 12 b.

IV. RESULTS

The spectrum processor was implemented on an FPGA motherboard equipped with a Xilinx Virtex-II XC2V8000 FPGA (46 592 individual slices/8 million equivalent gates), 4-MB ZBT RAM communicating with the host PC server via a PCI interface (32 b, 33 MHz). The server is a Dual 3.06 GHz Xeon processor machine with 4-GB RAM.

The initial design was developed by using Xilinx's System Generator (6.3) for Matlab (7.0). The resulting VHDL code was refined and optimized by using Xilinx ISE Foundation (7.1) and Modelsim (SE 5.7d). The mass/abundance data were represented as unsigned integer numbers (32 b with 12 b after the radix point).

The user FPGA from the motherboard is used to implement the spectrum processor (Virtex-II XC2V8000). The actual design occupies 73% of total FPGA resources (34 139 slices) and runs at 180 MHz.

The 4 MB of ZBT RAM is enough to store 512 K samples of mass-abundance pairs of 32 b each.

The motherboard has sockets that can host additional FPGA modules hosting a Virtex-II XC2V8000 FPGA device and 1 GB of DDR SDRAM. These modules are used to host the database search engine [18].

In the first instance, the hardware implementation was validated by comparing the results of processing mass spectrometric data generated by the FPGA implementation and the equivalent C software implementation of the algorithms. The reference C program was run on a dual 3.06-GHz Xeon processor server. In all tests, both implementations produced identical results.

The impact of the spectral length on processing time was measured by using spectra with various lengths but constant isotopic composition and noise levels. The results are summarized in Table II.

Each C simulation was repeated 30 times and the average processing time is used here. The standard deviation for the speed gain is shown in brackets. The average speed gain of the FPGA implementation is 122.07. The processing times are shown in Fig. 9. The FPGA processing time is unscaled, but the processing time of the C implementation is scaled down by a factor of 100.

It should be noted that the time elapsed for initializations of memory locations before the effective processing of data and disk-access time are not included in the software processing time. Only the processing of effective algorithmic steps is measured. Initializations add, on average, 30 ms to the C processing time, resulting in an increased average speed gain of 169. The maximum speed gain, however, could be as high as 200.

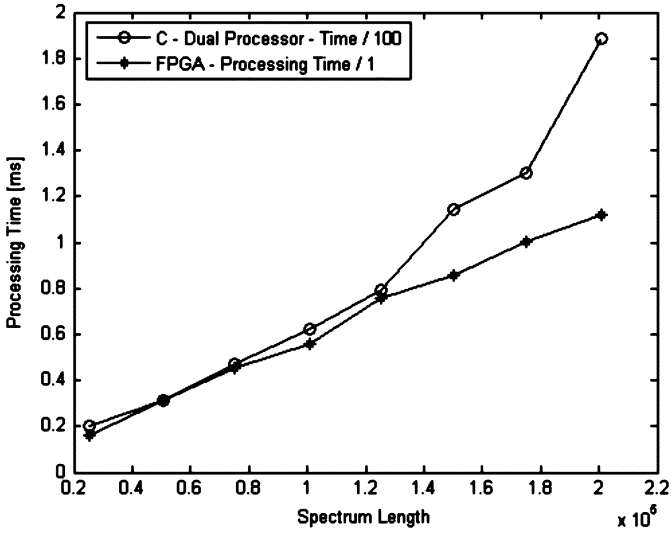

Fig. 9. Impact of spectrum length on processing time.

TABLE III
IMPACT OF SPECTRAL COMPLEXITY ON PERFORMANCE

Relevant peaks	Processing time [ms], dual Xeon 3GHz processors	Processing time [ms], Virtex-II FPGA, 180MHz	Speed Gain (*St.Dev.*)
50	142.03	1.0108	140.51 (*7.8*)
101	141.10	1.0127	139.32 (*3.9*)
144	140.75	1.0133	138.90 (*5.3*)
200	142.63	1.0152	140.48 (*6.6*)
240	140.50	1.0158	138.31 (*6.9*)
284	142.17	1.0170	139.78 (*3.9*)
327	144.25	1.0204	141.36 (*5.9*)
382	145.23	1.0415	139.43 (*5.24*)

Another simulation shows the effect of the spectral complexity over performance. The spectral length was kept constant at 175 000 data points, while the isotopic complexity (the number of relevant peaks) was modified. A different noise profile was used here, compared to the previous simulations. The performance results are given in Table III. The results are also shown in Fig. 10.

In this case, speed of the hardware processor is almost linear with insignificant fluctuation. On average, the speed gain is influenced by the spectral length and less influenced by the spectral profile. The peak detection unit is a pipeline so the performance of this block is linear of the spectral length and it does not depend on the spectral composition.

The performance of the peptide identification unit significantly depends on the spectral profile (noise, number of detected peaks, and their abundances). However, its processing time is insignificant—compared to total processing of the spectra by the peak detection unit.

The FPGA spectrum processor was compared to a commercial product MassLynx (Waters Corp.). A recombinant protein

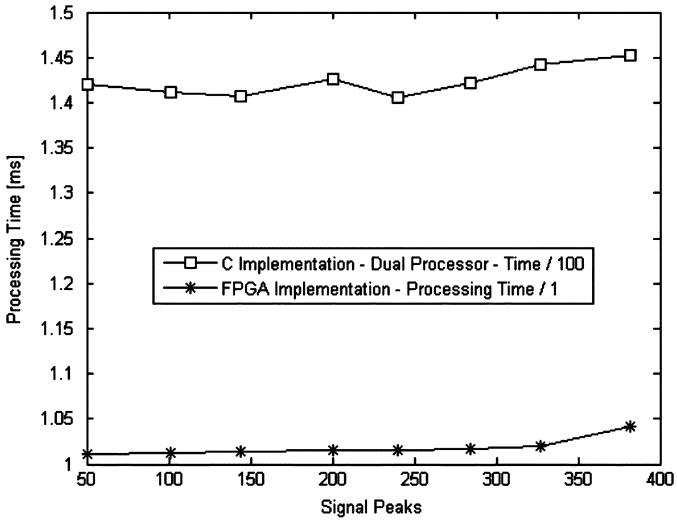

Fig. 10. Impact of spectrum complexity on processing time.

designed as an internal standard for multiplexed absolute protein quantification [16], [17] was digested with trypsin to release 20-limit peptides of known identity. The digested material was analyzed by MALDI-ToF MS and the raw data were processed separately by using the MassLynx package and the FPGA processor.

The FPGA implementation and MassLynx correctly identified all of the peaks [15]. The intensities of the different peaks correlated well, irrespective of the method used to process the spectrum (sample correlation coefficient of 0.9874).

V. Conclusion

This paper presented a hardware solution, based on FPGA devices, to accelerate algorithms used in peptide mass fingerprinting. Results show that a significant increase in processing speed is achieved, compared to software implementations running on conventional microprocessor-based systems. Once the processing speeds are achieved, it is possible to implement real-time processing of spectra during DAQ.

References

[1] J. Samuelsson, D. Dalevi, F. Levander, and T. Rögnvaldsson, "Modular, scriptable and automated analysis tools for high-throughput peptide mass fingerprinting," *Bioinformatics*, vol. 20, pp. 3628–3635, 2004.

[2] E. J. Breen, F. G. Hopwood, K. L. Williams, and M. R. Wilkins, "Automatic poisson peak harvesting for high throughput protein identification," *Electrophoresis*, vol. 21, pp. 2243–2251, 2000.

[3] A. Savitzky and M. J. E. Golay, "Smoothing and differentiation of data by simplified least squares procedures," *Anal. Chem.*, vol. 36, pp. 1627–1639, 1964.

[4] T. A. Anish, M. Dumontier, J. S. Rose, and C. W. V. Hogue, "Hardware-accelerated protein identification for mass spectrometry," *Rapid Communi. Mass Spectrom.*, vol. 19, pp. 833–837, 2005.

[5] Datasheet DS240 May 2004, Distributed Arithmetic FIR Filter V9.0, Xilinx Inc..

[6] D. Lavenier, "Speeding up genome computations with systolic accelerator," *SIAM News*, vol. 31, no. 8, pp. 1–8, 1998.

[7] B. Fagin, J. G. Watt, and R. Gross, "A special-purpose processor for gene sequence analysis," *Comput. Appl. BioSci.*, vol. 9, pp. 221–226, 1993.

[8] R. Hughey, "Parallel hardware for sequence comparison and alignment," *Comput. Appl. BioSci.*, vol. 12, pp. 473–479, 1996.

[9] P. Guerdoux-Jamet and D. Lavenier, "SAMBA: Hardware accelerator for biological sequence comparison," *Comput. Appl. BioSci.*, vol. 13, pp. 609–615, 1997.

[10] A. Wozniak, "Using video-oriented instructions to speed up sequence comparison," *Comput. Appl. BioSci.*, vol. 13, pp. 145–150, 1997.

[11] A. S. Guccione and E. Keller, "Gene matching using Jbits," in *Proc. Reconfigurable Computing is Going Mainstream, 12th Int. Conf. Field-Programmable Logic and Applications*, 2002, pp. 1168–1171.

[12] A. Marongiu, P. Palazzari, and V. Rosato, "Designing hardware for protein sequence analysis," *Bioinformatics*, vol. 19, pp. 1739–1740, 2003.

[13] H. Simmler, H. Singpiel, and R. Männer, "Real-time primer design for DNA chips," *Intersci. Concurrency Comput.: Practice Experience*, vol. 16, pp. 855–872, 2004.

[14] T. Oliver, B. Smidth, D. Nathan, R. Clemens, and D. Maskell, "Using reconfigurable hardware to accelerate multiple sequence alignment with ClustalW," *Bioinformatics*, vol. 21, pp. 3431–3432, 2005.

[15] I. Bogdán, D. Coca, J. Rivers, and R. J. Beynon, "Hardware acceleration for processing mass spectrometric data for proteomics," *Bioinformatics*, vol. 23, pp. 724–731, 2007.

[16] R. J. Beynon, M. K. Doherty, J. M. Pratt, and S. J. Gaskell, "Multiplexed absolute quantification in proteomics using artificial QCAT proteins of concatenated signature peptides rates," *Nature Meth.*, vol. 2, pp. 587–589, 2005.

[17] J. M. Pratt, D. M. Simpson, M. K. Doherty, J. Rivers, S. J. Gaskell, and R. J. Beynon, "Multiplexed absolute quantification for proteomics using concatenated signature peptides encoded by QconCAT genes," *Nature Prot.*, vol. 1, pp. 1029–1043, 2006.

[18] I. Bogdán, J. Rivers, J. R. Beynon, and D. Coca, "High-performance hardware implementation of a parallel database search engine for real-time peptide mass fingerprinting," *Bioinformatics*, vol. 24, pp. 1498–1502.

István A. Bogdán received the B.S. degree in applied electronics from "Transilvania" University, Brasov, Romania, in 1995, and the M.Phil. degree in electronic engineering from Sheffield University, Sheffield, U.K., in 2006.

From 1998 to 2001, he was a Research Assistant in the Electronic and Electrical Engineering Department, Sheffield University. In 2004, he joined the Automatic Control and Systems Engineering Department, Sheffield University, as a Research Assistant. His interests are parallel computing, hardware-accelerated algorithms, embedded systems, field-programmable gate-array systems and very-large scale integrated chip design.

Daniel Coca received the M.Eng. degree in electrical engineering from the "Transilvania" University, Brasov, Romania, in 1993 and the Ph.D. degree in control systems engineering from Sheffield University, Sheffield, U.K., in 1997.

Since 1997, he has been a Research Associate in the Automatic Control and Systems Engineering Department at the University of Sheffield. From 2002 to 2004, he was a Lecturer in the Department of Electrical Engineering and Electronics at the University of Liverpool, Liverpool, U.K. Since 2004, he has been with the Department of Automatic Control and Systems Engineering, Sheffield University, where he is currently Senior Lecturer. His research interests are modelling, identification and control of complex systems, and bioimaging and biological data analysis using reconfigurable computers.

Dr. Coca is a Chartered Engineer (CEng) and member of the Institution of Engineering and Technology (IET).

Rob J. Beynon received the B.Sc. degree in biochemistry and the Ph.D. degree in enzymology from the University of Wales, Wales, U.K., in 1974 and 1978, respectively.

He began his career at the University of Liverpool, Liverpool, U.K., and in 1993, became Chair of Biochemistry at the University of Manchester Institute of Science and Technology, Manchester, U.K. His main areas of research are protein chemistry, proteomics, proteolysis, and proteolytic enzymes. He is the author of many papers. He assumed Chair of Veterinary Basic Sciences, University of Liverpool, Liverpool, U.K., in 1999, was the Chair of the Engineering and Biological Systems Committee of the Biotechnology and Biological Sciences Research Council from 2006 to 2009, and was Associate Dean for Research from 2002 to 2008.

Prof. Beynon was the winner of the Glaxo Partnership Award 1994 and of the Marbocyl Prize for Research Excellence in 2006. He was a member of the Research Assessment Exercise Subpanel 16 (2005 to 2008)