FPGA Based Hardware Acceleration:
A Case Study in Protein Identification

Submitted by Aaron Richard Mandle
In partial fulfillment of the requirements of the degree of Bachelor of Science with Honors in the Division of Engineering at Brown University
4/22/08

Prepared under the direction of
Professor Sherief Reda, Advisor
Professor Jennifer Dworak, Reader
Professor Gabriel Taubin, Reader

Abstract

There exists a large and expanding base of existing software that has been developed and tested. Currently developing a dedicated hardware accelerator requires concurrently developing a software front-end specially designed to interact with the accelerator. This process is both time-consuming and limits the flexibility of both the software and the hardware.

To this end, we develop a methodology by which critical algorithms in existing software projects can be quickly identified and accelerated using FPGAs. Additionally, we develop software-hardware partitioning strategies and determine the best approach to minimize the communications overhead while allowing for the maximum speedup and scalability of the solution.

We demonstrate the feasibility of this process by developing a hardware accelerator for an open source protein identification software project. We implement a loosely coupled coprocessor interfaced across the PCI bus, which is capable of independently executing an algorithm critical to the results of the program. We achieve a speedup of 2.59 in the accelerated function. Additionally, opportunities for further refinement of the design are explored.

Table of Contents

Table of Figures

1 Section 1: Introduction to FPGAs and Hardware Acceleration

Data processing involving complex algorithms is increasingly common. Whether it is video editing, speech recognition, or analyzing experimental data, high volumes of data must be processed as quickly as possible. Computers are able to handle a wide variety of calculations but are general computing devices and thus not optimized for any single calculation. Application specific integrated circuits, or ASICs, are able to perform specific calculations much faster than a similar software routine. Unfortunately, ASICs are expensive to make and require flawless designs. Once they are fabricated, their logic cannot be altered. FPGAs offer a balance between the speed of ASICs and the flexibility of software.

1.1 Field-Programmable Gate Arrays

FPGAs are reconfigurable hardware chips that can be reprogrammed to implement varied combinational and sequential logic. FPGAs are low cost compared to ASICs and have the advantage of being quickly reusable. Their reprogram-ability offers great flexibility and the opportunity to quickly develop a prototype of a circuit. While not as fast as ASICs, FPGAs have an advantage in low volume prototyping and proof of concept applications. FPGAs allow hardware designs to the quickly and cheaply validated

An FPGA is made up of an array of programmable logic blocks. These logic blocks are connected by reconfigurable sets of wires which allow for signals to be routed according to the definition of the circuit. Custom circuits are defined by a user in a programming language, specifically intended to describe hardware.

Programming languages designed to describe hardware are known as hardware descriptive languages, or HDLs. The two most common HDLs are Verilog and VHDL. Both Verilog and VHDL offer a system by which code can be encapsulated and reused as well as a level of abstraction

for the programmer. Once written, HDL is compiled and translated into a configuration which is then transferred to the FPGA itself. This configuration fills in the logic blocks and correctly routes the signals within the FPGA such that the circuit described by the HDL is implemented in hardware.

1.1.1 Logic blocks

The most basic element an FPGA is a logic block. A logic block consists of a lookup table connected to a multiplexer, which chooses between the output of the lookup table and a flip-flop connected to the output of a lookup table intended to allow for storing values as a register. This design is able to implement arbitrary functions limited only by the size of the lookup table. These logic blocks are chained together to provide lookup tables in any size or configuration necessary. Logic blocks are connected together in local groups known as logic block clusters. Within a logic block cluster of logic blocks are fully connected to one another

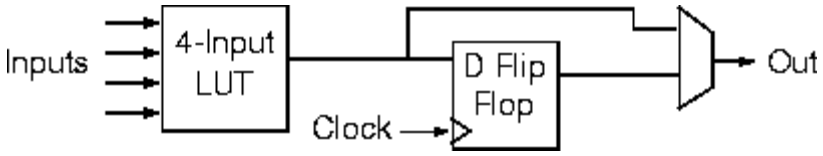

Figure 1: A basic logic block[i]

1.1.2 Routing

In order to connect the logic blocks in a meaningful manner, the outputs and inputs must be routed correctly. This requires the use of reconfigurable routing. Routing is organized into a grid of wires connected to each other via programmable switches. These switches are turned on and off based on the desired path of a signal. Ideally, the length of the connections should be kept as short as possible as longer wire segments result in larger delays.

Logic block clusters are interfaced with routing channels through connection blocks. These connection blocks allow a logic block cluster to be either disconnected or connected to the routing channel. This prevents multiple drivers on the same wire and avoids bus contention.

Due to the fact that there is not a wire connecting every output from a logic block cluster to every other input to a logic block cluster, signals must not only be routed from logic block to wire, but also from a wire to wire. This wire to wire routing is done using switch blocks. Switch blocks allow for the transfer of a signal on one wire to another wire. The topography of the wires and logic block clusters varies from design to design. There is a trade-off between delay and IC area, which must be addressed by the FPGA designer.

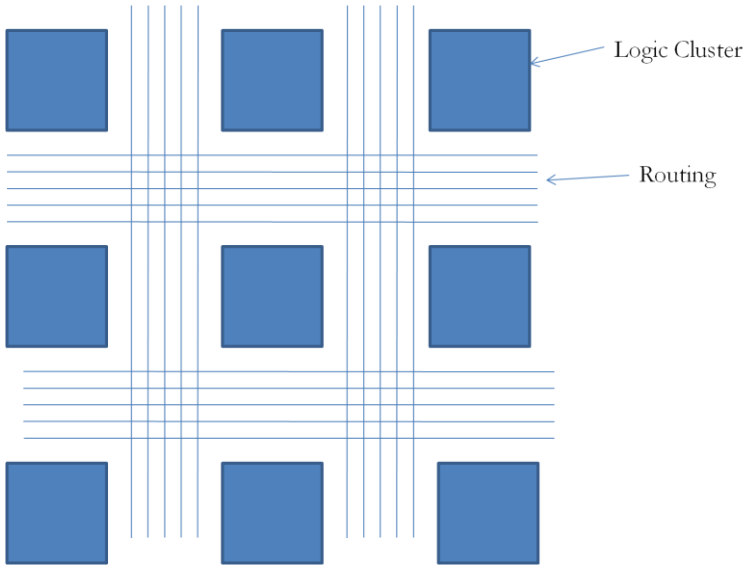

Figure 2: FPGA Routing

1.1.3 Embedded Elements

Many FPGAs also include commonly used elements as non-reconfigurable hardware in their designs. Commonly included hardware elements are embedded memory, multipliers, barrel-shifters, and fast adders. These included embedded elements do not need to be implemented in reconfigurable hardware, and thus free up valuable resources on the FPGA. Additionally, it is often

possible to implement a non-reconfigurable version of an element in considerably less space than the reconfigurable version would require.

1.1.4 Advantages

Unlike software, when an algorithm is implemented on an FPGA the process can be parallelized such that many independent calculations can be performed concurrently. It is this ability to parallelize an algorithm which makes a hardware accelerator such an enticing prospect. The ability to quickly and cheaply prototype hardware algorithms makes the use of FPGAs ideal for an endeavor such as this.

1.2 Hardware Acceleration Methods

The overarching goal of hardware acceleration is to increase the speed at which data can be processed by using custom hardware specifically designed to implement a specific routine. By doing so, the software can be sped up in two ways.

The first advantage is that the CPU is able to process other data, while the computation necessary for the accelerated routine is offloaded to the coprocessor. This makes the computation appear to be essentially free to the processor. The only time the processor must spend on the computation is the time that it takes to set up the coprocessor to begin its calculation and the time it takes to receive the results. As long as the overhead necessary to communicate with coprocessor is less costly than performing the actual computation, a speedup is realized.

The second potential gain is realized when the hardware accelerator is structured in such a way that it is able to calculate the result a faster than the software. In this case, the communications overhead, and the runtime of the hardware accelerator must be less than the time the software implementation of the same algorithm would take. If this condition is met the algorithm will be accelerated whether or not the processor is processing data in parallel with the coprocessor.

Ideally, both conditions would be met, where the processor is able to process data concurrently with the coprocessor and the custom hardware is faster than a software implementation. It is not however strictly necessary for the custom instruction to be faster than software implementation. It is only necessary that the overhead to begin the calculation is less costly than the calculation itself and that there is processing which can be done independent of the result of the accelerator.

1.2.1 Data Transfer

One of the main bottlenecks in hardware acceleration is data transfer. To ensure a fast data transfer, the hardware accelerator should be closely coupled to the main processor. An accelerator on the same die as the processor, for example, would be able to transfer data between itself and the processor faster than a coprocessor connected over USB. In general, it is desirable to have lower communications overhead when connecting the accelerator to processor.

Possible interfaces include USB, Ethernet, serial, and PCI. Any interface can be implemented if the hardware necessary for the controller can be fit onto the FPGA. When choosing an appropriate interface it is important to consider the latency and bandwidth. The bandwidth of an interface describes the rate of data transfer. As previously described, it is important that it does not take longer transfer the data to the coprocessor than it would to process the data on the processor.

The latency of an interface is also an important characteristic to consider. A high latency connection means that the data sent will arrive after a longer delay than a low latency connection. If latency and bandwidth seem like very similar concepts, it may be instructive to consider the following example. If a video signal is sent around the world, there will be a significant delay between the time it is sent and the time at which is received. This is due to the latency of the signal traveling around the world. At the same time, this signal arrives fully intact with all of the bits sent

eventually received. The number of bits received every second describes the bandwidth of the signal.

1.2.2 DMA

Direct Memory Access, or DMA, allows for memory access to be done independent of the central processing unit. This allows for higher transfer speeds for blocks of data. Instead of requiring the processor to copy each piece of data from the source to the destination, the use of DMA allows the processor to simply initiate a DMA transfer and then continue another task while the DMA controller handles the data transfer. This allows for significantly faster transfer speeds as well as higher utilization of the processor.

1.2.3 Common Acceleration Strategies

The main strategy behind successful hardware acceleration is to convert temporal calculations into spatial calculation. This is accomplished by expanding the algorithm into multiple parallel calculations.

Often, arrays are used to calculate values. A systolic array, for example, consists of numerous data processing units connected by a mesh of wires. Data is streamed in one or both sides of the array and stored in a data processing, where an operation is performed on the values. Once the desired operation has been performed and results can be streamed out in a similar fashion. Such an array offers highly parallel computations as each data processing unit is able to operate independently and concurrently. Such arrays are commonly used for multiplication.

While the process of parallelizing an algorithm is often specific to the algorithm, the general approach is to determine which portions of the algorithm can be executed in parallel. A dedicated computational unit is then created for each part that can be parallelized. Once all the individual parts have been calculated, their results are combined for the final result. This is a common strategy

known as divide and conquer in which one task is divided up into smaller tasks, the results of which are recombined at completion.

One commonly used method for creating a hardware accelerator is to build a "system on a programmable chip" or SOPC. A SOPC consists of a soft-processor, which is loaded into the reconfigurable fabric of the FPGA.A custom instruction is then added to the soft-processor. This allows for easy integration of the hardware with a processor and allows for a high-degree of customization.

1.3 Hardware-Software Co-design

The goal of hardware-software partitioning is to determine which pieces of the project are best suited for hardware and which pieces of the project are best suited for hardware. This partitioning is driven by factors such as cost, efficiency, and speed.

When accelerating a routine for speed, it is desirable to minimize the bus traffic in hardware and software as well as to maximize the concurrency of the hardware and software process. In doing so, the total of runtime of the routine is minimized.

In order to determine the best place to separate the hardware from the software, the software must be profiled. The goal of profiling is to identify possible pieces of software that can be made into hardware and to determine which ones would most benefit from translation to hardware as well as where the greatest gains in performance can be found.

Profiling offers an overall view of the software from the project can be broken up into different tasks. These tasks can be represented as nodes on a graph where their edges represent the communications overhead between them. Depicted as such, the problem of partitioning the software and the hardware becomes a minimum-cut problem from graph theory in which the weight

of the edges crossing the boundary between software and hardware tasks is minimized. A diagram of such a representation can be seen below.

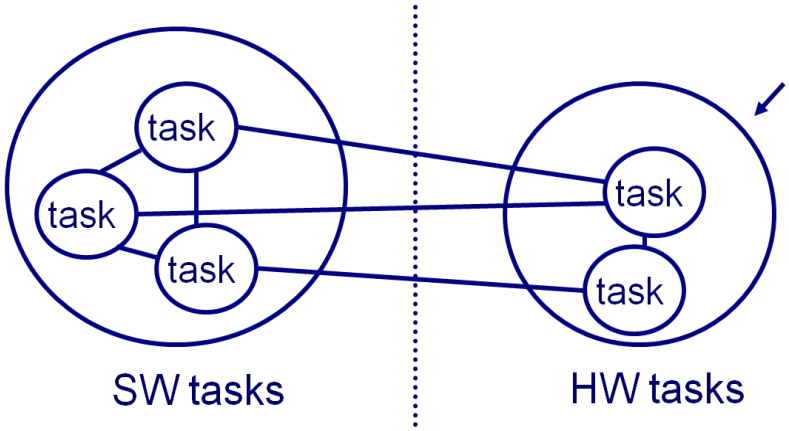

Figure 3: Hardware-Software Partitioning Graph[ii]

In the minimum-cut problem, the goal is to separate two groups which are connected by links while keeping the values of the links traversing between the two groups to a minimum. When the software and hardware are partitioned with the minimum weight of edges crossing the boundary, the minimum communications overhead costs will be incurred.

1.4 PCI Interface

The PCI bus provides a method for transferring data between the software side of the application and the hardware accelerator. It allows for full-featured programming on the host computer to take advantage of the hardware accelerator. The main problem with this approach is the complexity of the interface.

The actual specification is controlled by the PCI special-interest group which sells the specifications. There are four different modes in which the PCI bus can operate, Synchronous, Transaction/Burst, Bus mastering, plug-and-play. The clock runs at 33 MHz standard but some computers can also support 66 MHz .

1.4.1 Address Spaces

PCI has three distinct address spaces that can be written to: configuration space, memory space and I/O space. Both Memory and I/O are at dynamic addresses. The configuration space is the address starting at 0 for all PCI boards. This allows the host to read information about the capabilities and requirements of the board without initially knowing anything about it. PCI cards do not talk directly to the CPU but rather communicate through the PCI bridge which does translation for the CPU There is a 32-bit address and data bus on which the address requested is written and data is passed. These pins are bi-directional. Data is transferred between a bus master and a slave, or target.

1.4.2 Command/Byte Enable

The master drives the C/BE[3:0] signals during the address phase which indicates the type of memory transfer which is to occur. During the data phases the C/BE signals is used to indicate which of the four bytes is valid (Each byte is ¼ of the 32 bits of the address and data bus).

C/BE[3:0]	**Command Types**
0000	Interrupt Acknowledge
0001	Special Cycle
0010	I/O Read
0011	I/O Write
0100	Reserved
0101	Reserved
0110	Memory Read
0111	Memory Write
1000	Reserved
1001	Reserved
1010	Configuration Read
1011	Configuration Write
1100	Memory Read Multiple
1101	Dual Address Cycle
1110	Memory Read Line
1111	Memory Write and Invalidate

1.4.3 Bus Control

In order for either the target or the master to pause a transaction the IRDY and TRDY can be de-asserted. Valid data is transferred on each clock edge when both IRDY and TRDY are asserted. To request ownership of the bus the REQ signal is asserted and the bus arbiter will inform the requester that the request has been granted by asserting the GNT signal.

1.4.4 List of Required Pins

The pins that must be implemented to comply with the PCI standards are as follows:

Required Pins	Optional Pins:
AD[31:0]	AD[63:32]
C/BE[3:0]	C/BE[7:4]
PAR	PAR64
FRAME	REQ64
TRDY	ACK64
IRDY	LOCK
STOP	INTA
DEVSEL	INTB
IDSEL	INTC
PERR	INTD
SERR	SBO
REQ	SDONE
GNT	TDI
CLK	TDO
RST	TCK
	TMS
	TRST

1.5 Overview of Tools

1.5.1 GiDEL PROCSpark II Board

To implement the hardware, we made used of the GiDEL PROCSpark II board. This board implements the entire PCI interface and the provided software creates a driver for the board. This provides an abstraction layer and allows the user to interface with the board without managing the PCI interface directly.

The PROCSpark II consists of an Altera Cyclone II with "33,216 logic elements (LEs), 35 embedded multipliers, and over 483 Kbits of on-chip RAM"[iii] FPGA, external DRAM as well as 2 DMA's with an available 400 MB/s throughput.[iv]

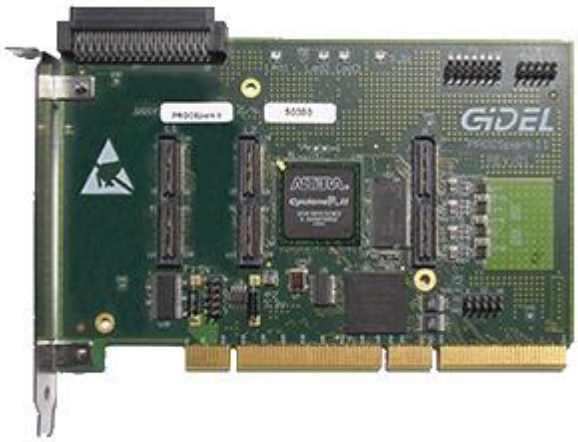

Figure 4: GiDEL PROCSpark II

1.5.2 GiDEL PROCWizard

The GiDEL PROCWizard software tool allows the user to define an interface on the FPGA. The user can choose from libraries of pre-existing interfaces. Registers, memory, and PCI interfaces can all be added from the program. Additionally, users can specify modules in which they will place their own custom HDL code.

Once the design has been fully specified, the user is able to generate HDL in their choice of languages, as well as generate a wrapper C class with which software can interface with the design. The system allows for rapid creation of a software-hardware interface and allows for easy partitioning of the custom logic, which is to be implemented on the PCI.

1.5.3 Altera Quartus II

The Altera Quartus II IDE is used in this project for its compiler, linker and fitter. While the PROCWizard generates the necessary interfaces and HDL code, it does not have the ability to

synthesize it. The Verilog files generated by the PROCWizard are synthesized using the Quartus tool. Additionally, the custom hardware is written and simulated within the Quartus environment.

2 Accelerating Protein Identification Case Study

Proteins are complex molecules which perform the functions necessary for life. Each protein consists of a different string of amino acids. This string of amino acids dictates the function of the protein.[v] To understand the workings of biological organisms it is critical to understand the working of proteins, and to do so, they must first be identified.

In order to study proteins, it is necessary to have a reliable and accurate method for identification. The two primary methods of protein identification are Edman degradation and mass spectrometry. Edman degradation works by separating off amino-acids one at a time from a peptide. These amino acids can then be identified using either chromatography or electrophoresis. Though Edman degradation is a slow, labor intensive process it has the advantage of requiring only a very small sample of the protein to be sequenced.

2.1 Mass Spectrometry

Mass spectrometry, on the other hand, is fast and largely automated but requires a significantly larger sample of the protein and generates a huge amount of data which must be processed. Mass spectrometry works by first breaking a sample into ions at the ion source. These ions are then separated and sorted based on their masses. The amount of ions corresponding to each mass is then detected, and the resultant data is analyzed.[vi] This method is faster than Edman degradation, but its speed is currently limited by the time it takes to process the large amounts of data between sequencing each peptide.[vii]

2.1.1 Ionization

The first step in performing a mass spectrometry measure is to create the charged particles which are subsequently measured. The most common method of ionization consists of

bombarding a simple with high-energy electrons. This method is known as electron ionization, or EI. In order for this method worked a simple must be in a highly diluted gas phase. The impact of high energy electrons breaks the protein into ions.

2.1.2 Tandem Mass Spectrometry

Tandem mass spectrometry is a general term describing techniques whereby ions are run through mass spectrometric analysis twice. Fragmentation of the ions occurs between the stages of the tandem mass spectrometer.

2.1.3 Protein Identification

A protein identification study consists of the following steps: First the mass of the protein to be analyzed is measured with extremely high accuracy by mass spectrometry. Next, the protein is broken apart using an enzyme. Trypsin is a commonly used enzyme. This digested protein is run through a mass spectrometer. The results of this run are called a peptide map. For non de-novo protein identification, this is often enough information to determine the protein.

The data generated by the mass spectrometry process consists of spectra relating the relative intensity of each ion with the charge/mass ratio of the ion. An example of such a spectrum can be seen below:

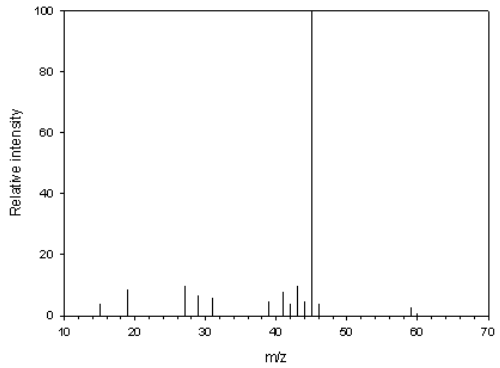

Figure 5: Example Spectra[viii]

A single run in a mass spectrometer can produce thousands of spectra. In the case of protein identification, these spectra are then compared to a pre-existing database of proteins. As the database increases in size, the number of included identifiable proteins also increases. This increases the number of comparisons that must be done and thereby increases the runtime

2.2 Prior Work

There are two main commercial software packages available for protein identification from mass spectrometry data, Sequest and Mascot. These programs are closed source and proprietary. As such they are of limited use for researching improvements to algorithms.

2.2.1 X!Tandem

Two open source protein sequencing software packages are also available. One, known as X!Tandem, is developed by the Global Proteome Machine Organization. The X!Tandem workflow is as follows:[ix]

1. read XML input parameter files;
2. read protein sequences from FASTA files;
3. read MS/MS spectra in common ASCII formats (DTA, PKL and Matrix Science);
4. condition MS/MS spectra to remove noise and common artifacts;
5. process peptide sequences with cleavage reagents, posttranslational and chemical modifications;
6. score peptide sequences; and
7. create an XML output file capturing the best scoring sequences and some statistical distributions relevant to the scoring process

The X!Tandem program has three main stages of searching for matching proteins. First, proteins are quickly identified from their tryptic peptides. A tryptic peptide is a peptide made up of a string of three amino acids. X!Tandem operates on the principle that, "For each identifiable protein, there is at least one detectable tryptic peptide."[xviii] This offers a fast way to identify proteins as only a small subsection needs to be matched to a database. Next, a database is created containing the proteins identified in the previous step. Finally, mutations of the database are created to include modified/non-enzymatic peptides. Unlike most programs however, X!Tandem only performs the mutations on the database of pre-identified proteins. This significantly reduces the number of proteins which must be modified and evaluated.[xviii]

The scoring of each protein is done by first computing the dot product of the measured spectra and the model spectra. Only the matching spectral peaks are considered in the score. The model spectrum consists simply of a list of the ions which should appear and does not take into consideration the relative quantities of the ions. An example of these spectra can be seen below.

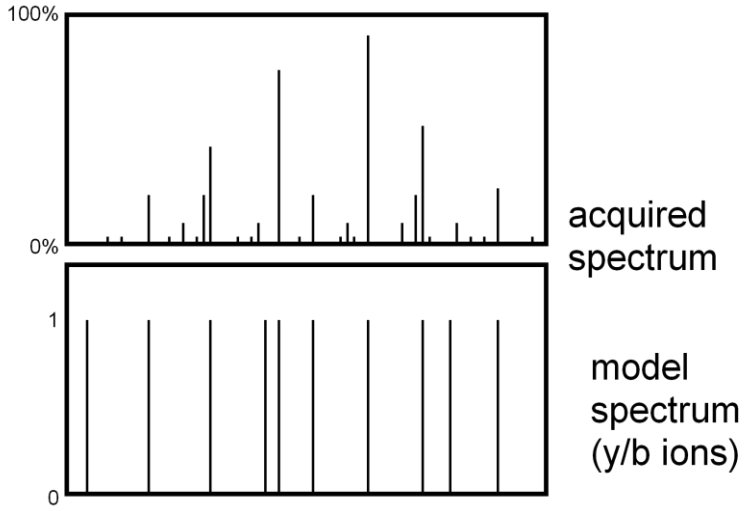

Figure 6: Model and acquired spectra[xviii]

The dot product of the measured spectra and the model spectra is known as the y/b score, or preliminary score, and is expressed by the following equation:

$$y/b\ Score = \left(\sum_{i=0}^{n} I_i \times P_i\right)$$

Where I is the spectrum intensities and P is a 1 or a 0 based on whether the peak was predicted by the model spectrum. Y ions are "peptide fragment ions appear to extend from the C-terminus" and b ions "extend from the amino terminus, or the front of the peptide"[x]

From the y/b score, the hyperscore is calculated by multiplying the y/b score by the factorial of the number of b and y ions assigned. This is calculation is based on the hypergeometric distribution. The hyperscore is defined as follows:

$$Hyper\ Score = \left(\sum_{i=0}^{n} I_i \times P_i\right) \times N_b! \times N_y!$$

X!Tandem then makes a histogram of all the hyperscores for all the possible matches in the database. The match with the highest hyperscore is assumed to be correct and all other matches are ignored.[xviii] This match is considered to be significant if the hyperscore is greater than the hyperscore corresponding to the x-intercept of all the other matches plotted on a log scale. This is best illustrated with the following two graphs:

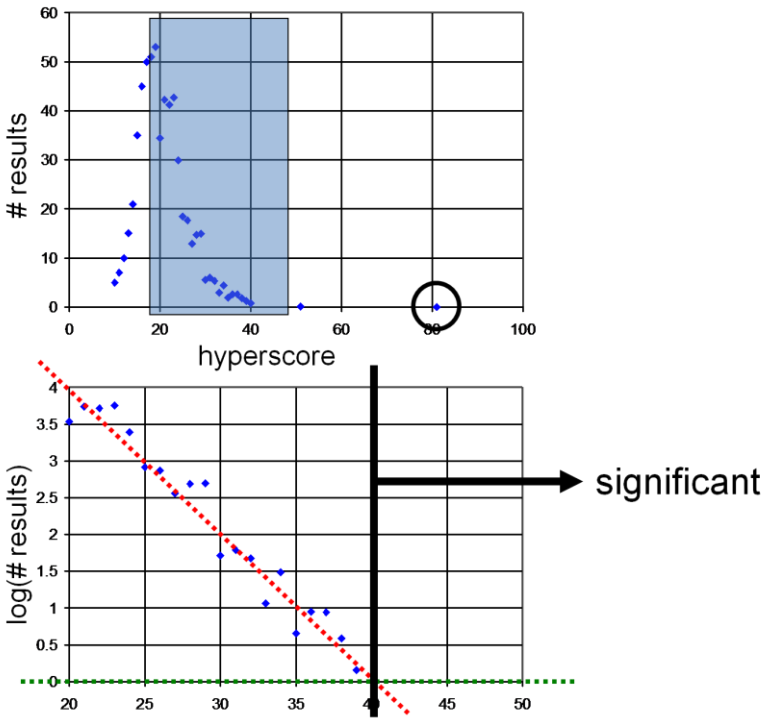

Figure 7: Determining match significance[xviii]

2.2.2 Inspect

The second open source sequencing algorithm is Inspect, created by the University of California, San Diego[xi]. Inspect uses de-novo sequencing to create a seed sequence with which the database is selectively filtered such that only the portions which are most likely to contain the sequence are searched. This significantly reduces the searching requirements of the software. A diagram of this flow can be seen below.

As described in Tanner et. al, "InsPecT constructs database filters that proved to be very successful in genomics searches. Given an MS/MS spectrum S and a database D, a database filter selects a small fraction of database D that is guaranteed (with high probability) to contain a peptide that produced S. InsPecT uses peptide sequence tags as efficient filters that reduce the size of the database by a few orders of magnitude while retaining the correct peptide with very high probability."[xii]

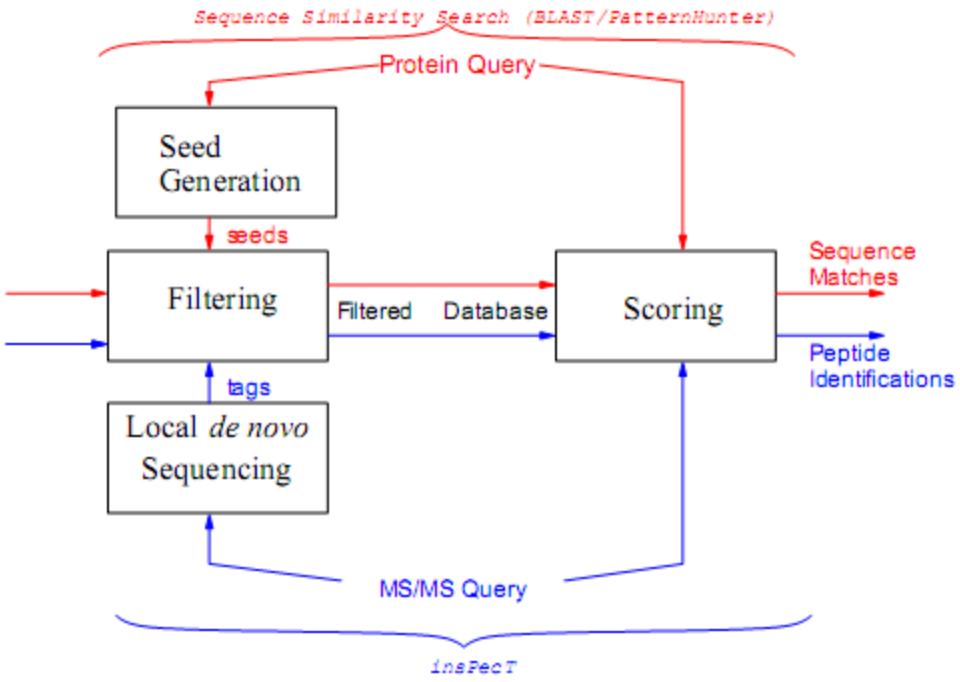

Figure 8: Inspect Program Flow

Due to the open source nature of these projects, they will serve as a starting point and baseline for hardware based improvements.

2.2.3 Hardware Based Approaches

Currently, the only currently available hardware product designed to process spectra and perform protein identification from mass spectrometry data is the Sage-N Sorcerer 2. It is designed to operate in conjunction with multiple software systems and is a very complex, specialized and costly product. The underlying architecture is unknown but it is advertised that it contains a 1.5 Terabyte redundant RAID array.[xiii] This level of hardware complexity will not be possible given the scale of this project.

In the paper, "Hardware-accelerated protein identification for mass spectrometry," Alex et. al develop a system design for accelerating common algorithms found in de-novo protein sequencing is proposed. This design makes use of an FPGA based accelerator and custom software on the host computer. The entire project was built from the ground up and is intended to accelerate de-novo sequencing and makes use of an accelerated database search to determine the individual pieces of the sequenced protein.[xiv]

2.3 Objective

The purpose of this research is to create a methodology for analyzing and evaluating software for acceleration as well as to create a scalable method for accelerating protein identification. To this end, we offer a hardware accelerator capable of performing calculations independent of the main software routines. Additionally, we develop software-hardware partitioning strategies and determine the best approach to minimize the communications overhead while allowing for the maximum speedup and scalability of the solution.

2.4 Analysis and Characterization

2.4.1 Benchmarks

In order to understand the speed of the current algorithms, both available applications have been benchmarked. The benchmarks were run on a 3.2 GHz hyper-threaded Pentium 4 with 706 spectra. The operating system kernel was the 2.6.20-16-generic Linux kernel. The times were measured using the time command. Both the cRAP and ups FASTA databases obtained from The Global Proteome Machine Organisation. The results from the benchmarking procedures can be seen in the table below:

Program Run Times (706 Spectra) No modifications	
Inspect (2007.09.05)	0m 4.252s
X!Tandem (linux-07-07-01-2)	0m 7.468s

Further data was collected on larger sized spectra, using a larger data base. For this benchmark, the same 3.2 GHz hyper-threaded Pentium 4 machine was used, this time running Windows XP. In addition to the cRAP and FASTA databases, the Homo_sapiens.NCBI36.47.pep.all database was used, as obtained from peptideatlas.org. The spectra which were analyzed were raw data from a plasma sample collected by the Human Proteome Organization Proteome Project, and downloaded from peptideatlas.org. First these spectra were run with no modifications:

Program Run Times (1733 Spectra) No modifications	
Inspect (2007.09.05)	37m 53.0781s
X!Tandem (win32-7-07-01-2)	0m 45.3s

The same spectra and database was run with the addition of a modification:

Program Run Times (1733 Spectra) Modifications 57.022 Daltons @ C

Inspect (2007.09.05)	38m 16.0312s
X!Tandem (win32-07-07-01-2)	0m 46.751s

Though the times differ by extremely large margins, neither program outputs an error, and both were run according to the documentation available. In order to further showcase the extravagant runtimes often encountered when processing ms/ms spectra, the following benchmark was run using the benchmarks described in Zosso et. Al[xv]. This consists of a test spectra of 17 proteins run against the SWISS-PROT database. The results are as follows:

Program Run Times (1333 Spectra) **Modifications +16@M, +1@[AG], +80@[ST]**	
Inspect (2007.09.05)	1h 54m 18.0217s
X!Tandem (win32-07-07-01-2)	11m 58.332s

Further testing was done to determine a typical run time. Spectra from the Sashimi data repository[xvi] were run against the same SWISS-PROT database[xvii]. These spectra are identified as "Raft Flowthrough." The results from this experiment show the lengthy amounts of time necessary to run the spectra from a real scientific experiment. The same modifications as in the previous experiment were used, but due to the increase in the number of spectra the run time rose dramatically.

Program Run Times (56771 Spectra)

Modifications +16@M, +1@[AG], +80@[ST]	
X!Tandem (win32-07-07-01-2)	> 16 hours

The analysis ran for over sixteen hours and still had not completed calculating the point mutations. This magnitude of computational time effectively demonstrates the need to accelerate the algorithm.

2.4.2 Code Profiling

Once both programs had been benchmarked, they were profiled using cachegrind. Cachegrind is an open source software tool which traces the execution of a program and returns a trace of all the function calls made. The results from the profiling highlight the most critical pieces of code. These critical pieces can then be translated into hardware and accelerated.

For X!Tandem, the most critical function was found to be mscore_tandem::dot. This function was called 2,763,582 times and accounts for 18.42% of the total run time. The callgraph for X!Tandem can be seen below. The function mscore_tandem::dot scores a peptide given a mass spectrum. This process is explained in the comments from the source code where it is noted, "Convolution scores are the sum of the products of the spectrum intensities and the scoring weight factors for a spectrum. Hyper scores are the product of the products of the spectrum intensities and the scoring weight factors for a spectrum."

In order to speed up this function, the main improvement will be to score as many of the peptides as possible in parallel. The actual function is not very computationally intensive, but the profiling shows that the function is called 2,763,582 times. The number of times this function has to be run will be inversely proportional to the number of parallel scoring functions can be fit on an

FPGA. The more parallelized this function is the faster the overall code will run up to a point; the entire program can only be sped up a maximum of 18.42% in accordance with Amdahl's Law.

Profiling Inspect provided surprising results. The second highest percentage of total run time was spent in the function `PrepareSpectrumForIonScoring`. According to the profiling data, 24.77% of the total run time was spent in this piece of the code, which was only called 1,412 times (twice for each spectra). This function primarily places each of the peaks into different "buckets" to allow for easier scoring.

It should be noted that the highest percentage of total runtime is taken up by the function `TagGraphGenerateTags` which runs for 27.35% of the program time, but over half of its run time is spent on the quicksort algorithm. Since it is infeasible to implement quicksort in reconfigurable hardware, it makes more sense to accelerate the `PrepareSpectrumForIonScoring` function. The body of this function can be seen in Appendix A.

2.4.3 Algorithm

Due to the greater use of the software package in the scientific community as well as the accessibility of the code base, acceleration will be focused on the X!Tandem project. As previously stated, the most time is spent in the function `mscore_tandem::dot`.

As explained by Brian Searle in his presentation, "X!Tandem works by matching the acquired MS/MS spectra to a model spectrum based on peptides in a protein database. The model spectrum is very simple, based on the presence or absence of y and b ions. Only matching spectral peaks — the ones marked in the figure — are considered. Any peaks that don't match, in either the model or acquired spectra, are not used. X!Tandem's preliminary score is a dot product of the acquired and model spectra. Because only similar peaks are considered, this is the sum of the intensities of the matched y and b ions. "[xviii]

Within `mscore_tandem::dot`, there are four key variables which are utilized in the core of the algorithm. The variables and their assigned meanings are as follows:

m_lM	m+H positive error
m_plSeq	residue mass as an integer
m_fI	intensity of the peak
m_pfSeq	scoring weight factors

The calculation done by the dot function is to compare all the m+H positive errors to the residue masses, and when there is a match, the intensity of the peak is multiplied with the residue mass and the product is added to the running score. At the end of the function both the total score and the number of matches are returned. This operation can be seen in further detail below:

```
if(itType->m_lM == m_plSeq[a]){
     fValue0 = itType->m_fI * m_pfSeq[a];
          if(fValue0 > 0.0){
               lCount++;
                    fScore += fValue0;
               }
          }
     }
}
```

This operation is executed in a loop for all the items in the `m_vmiType` vector as well as the `m_plSeq` array. Both of these data structures have variable sizes, which makes the acceleration of the algorithm more complex. The general approach to accelerating this algorithm will be to unroll the for loops which compare every element of the `m_vmiType` vector with every element of the

`m_plSeq` array and perform all of the comparisons, multiplications and additions in parallel. This is possible because there are no dependencies between iterations of the loops.

Loop unrolling is a method of optimization whereby the instructions called within a loop are duplicated to form a longer sequence. This reduces the number of times the loop must be executed and decreases the amount of overhead involved in the programs execution.

One complication with the algorithm is that the arrays do not have fixed lengths. This requires the hardware accelerator to dynamically change its behavior based on the length of the input arrays. Thus, while the loop can be unrolled partially, it cannot be unrolled fully as there is no defined upper bound on the length of the arrays.

Another optimization technique, common sub-expression elimination, or CSE must be kept in mind when translating the software to hardware. CSE makes use of already calculated values and propagates them rather than calculating them two separate times. This is especially important in software where the values cannot be computed in parallel. In hardware, this technique can be used to reduce the size of the accelerator, but it is crucial to reuse values when doing so does not affect the delay of the circuit.

2.5 Accelerator Design

2.5.1 Architecture Design

The design of the accelerator is driven by delay, size constraints and operation overhead. In order to maximize the speedup, we wish to implement as many accelerators on the available hardware as possible. Additionally, we wish to minimize the delay for the obvious effect of speeding up the calculations.

2.5.2 Register Based Approach

The first method attempted was to use the FPGA to implement a custom instruction, but to continue to control the execution step by step from the software. This was achieved by implementing the inner the conditional multiplication and addition from the innermost loop of the dot function. A block diagram of this implementation can be seen below.

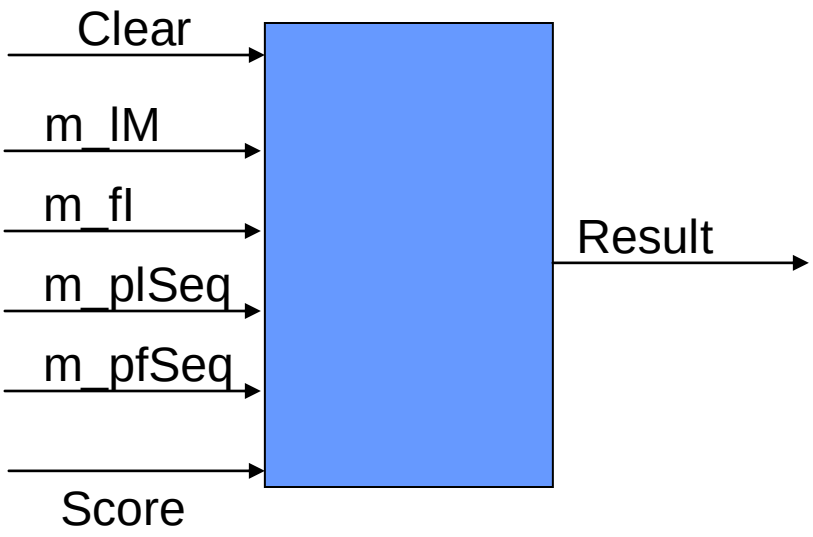

Figure 9: Register Based Dot Function

This method required constant attendance from the software and had considerable overhead. Each and every value had to be transferred to the board individually. This substantial overhead completely outweighed any performance increase the combinational conditional multiplication and addition had to offer. As such this method performed worse than the software version of the dot product.

When implemented, this design took only 302 logic elements, roughly 1% of the available area on the FPGA. The generated layout of this design can be seen in figure 17 in the appendix.

While the area was miniscule, to complete an entire array 1024 values long took the hardware version 500 microseconds while it only took the software 1.47 microseconds. Due to the overhead, the hardware version was 340 times slower. Additionally, the software had to constantly

monitor the hardware and feed data to it while it was running. While an unsuccessful accelerator, this initial attempt underscores the importance of minimizing the communications overhead.

2.5.3 DMA Transfer, Array Depth of One

This design utilizes the PROCSpark II board's DMA capabilities. The software opens a DMA channel to the board and transfers all of the data at once. This allows the hardware accelerator to calculate the result while the software performs other tasks.

As a result of the nested for loops in the original algorithm, the hardware must have the capability to revisit values. The dynamic size of the input arrays makes it impossible to statically unroll the loop. As such, the hardware must be able to handle the loop itself. This is accomplished by making use of FIFO buffers and looping the outputs back into the input. A block diagram of this design can be seen below

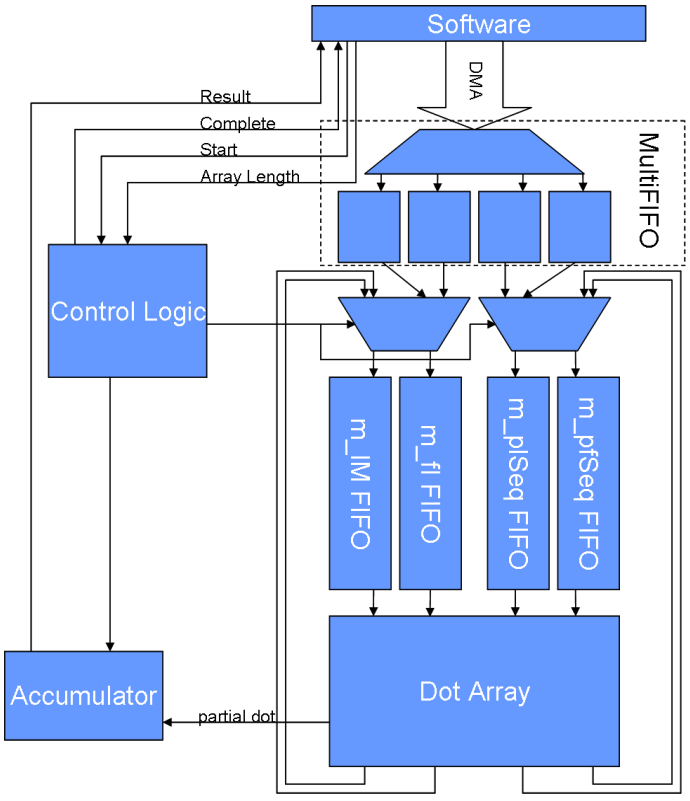

Figure 10: DMA dot-array depth of one

The actual dot array is implemented by the logic shown in the appendix. It consists of the same logic as is in the inner loop of the mscore_tandem::dot function. Two values are compared, and the multiplication of the other two values is returned, dependent on the first comparison.

The following is the data flow for the calculation of a dot using the DMA accelerator with a dot array depth of one:

1. All four arrays are transferred from the computer to the MultiFIFO buffers.
2. The array length, `LENGTH`, is reported to the accelerator
3. The start signal is given
4. The write-back multiplexers are set to route the data from the MultiFIFO to the simple FIFOs and the data is moved to the simple FIFOs
5. The write-back multiplexers are set to route the data from the end of the Dot Array back into the simple FIFOs.
6. The registers are primed with the first four values
7. For `LENGTH` values the m_lM and m_fI data are clocked out of their FIFOs and into the dot array.
    a. For `LENGTH` values the m_plSeq and m_pfSeq data are clocked out of their FIFOs and into the dot array.
        i. Each time a new value is read into the dot array, the result is added to the accumulator.
8. The Result is written in the result register
9. The complete signal is sent back to the computer

When implemented, this function takes 7,040 logic cells (21 percent of the Cyclone II). The layout generated by the Quratus II tool can be seen in figure 18 in the Appendix. The calculation is ready for the host after a certain time which varies according to the length of the arrays as follows:

$$T_{calculation} = Length_1 * Length_2 * 1/F_{clock}$$

In general, the software will be delayed for only the time it takes to transfer the data to the board. The host must initiate the transfer of data. This consumes a third of the time that computing the full dot product in software would have taken. For example, to transfer 4096 32-bit integers to the board takes 3.2 ms while for the software dot product it would take 9.4 ms to compute the result will become available after an amount of time determined by the following formula:

$$T_{Total} = T_{Transfer} + T_{calculation.}$$

2.5.4 DMA Transfer, Multiple

This design also utilizes the DMA capabilities of the board, and in addition, offers a speed advantage over the single-depth accelerator described previously. This increase in speed, however, comes at the price of area. In this design the values are clocked into the registers a certain depth deep. This allows depth^2 conditional multiplication and adds to be performed and reduces the number of times the second array data must be cycled through the simple FIFOs.

The following is the data flow for the calculation of a dot using the DMA accelerator with a dot array depth, `ARRAY_DEPTH`, of greater than one:

1. All four arrays are transferred from the computer to the MultiFIFO buffers.
2. The array length, `LENGTH`, is reported to the accelerator
3. The start signal is given
4. The write-back multiplexers are set to route the data from the MultiFIFO to the simple

FIFOs and the data is moved to the simple FIFOs

5. The write-back multiplexers are set to route the data from the end of the Dot Array back into the simple FIFOs.
6. The registers are primed with `ARRAY_DEPTH` values from each of the four simple FIFOs
7. The m_lM and m_fI arrays are filled with `ARRAY_DEPTH` data until all `LENGTH` data have been evaluated. data are clocked out of their FIFOs and into the dot array.
    a. For each set of `ARRAY_DEPTH` data the m_plSeq and m_pfSeq data are clocked out of their FIFOs and into the dot array.
        i. Each time the dot array is filled with `ARRAY_DEPTH` data, the results of the comparisons and multiplies are added to the accumulator.
8. The Result is written in the result register
9. The complete signal is sent back to the computer

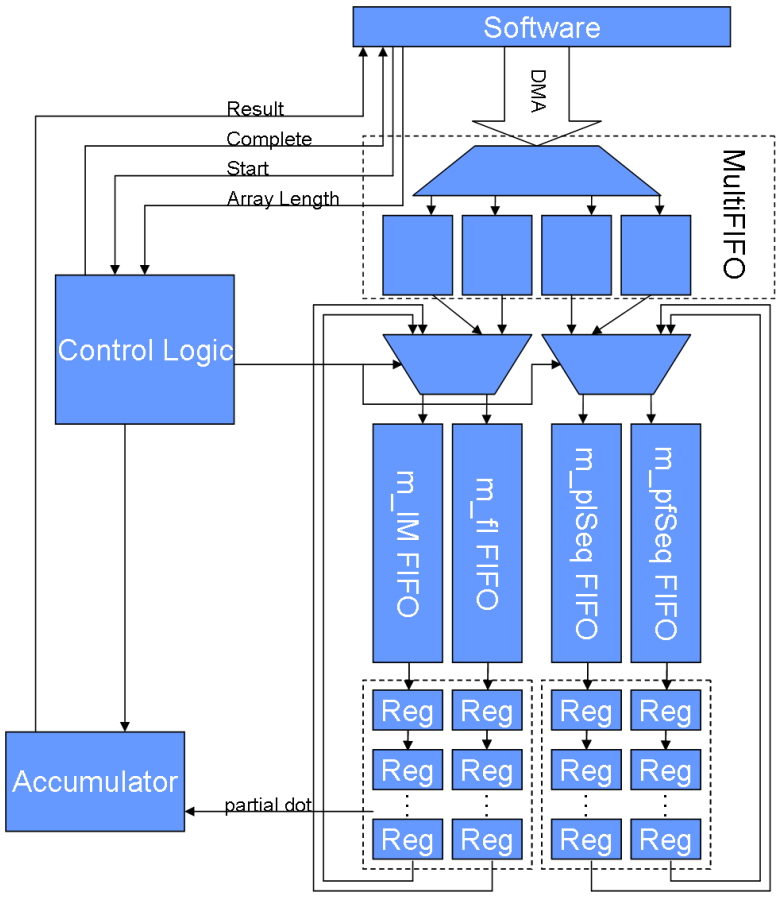

Figure 11: DMA dot-array depth greather than one

As with the previous DMA implementation, the software will be delayed for only the time it takes to transfer the data to the board. The host must initiate the transfer of data. This consumes half the amount of time that computing the full dot product in software would have taken. The difference between this design and the single-depth version is the delay before the result become available. For this version, the result will become available after an amount of time described by the following formulae:

$$T_{Total} = T_{Transfer} + T_{calculation}$$

$$T_{calculation} = (Length_1/Array\_Depth) * Length_2 * 1/F_{clock}$$

Although the speed increases linearly with respect to Array_Depth, the dot-array size (number of multiply/adds) increases with the square of Array_depth. As such it is necessary to make a tradeoff between

2.6 Host Software

In order to test the accelerators and verify their functionality test bench software was written to interface the host computer with the hardware accelerator. The test bench software fills buffers with data and transfers the data to the dot accelerator. The accelerator is also told the length of the arrays to process and given the signal to start. While the board processes the data, the test bench waits for the "complete" signal.

The timing is measured using the C library, timer.h . Due to the fact that the resolution on this timer is 1 ms, the operations were repeated multiple times in a loop, and the loop was timed to gain a more accurate measurement.

3 Results

Transfer time took 78.8% of the total run time of the accelerated function, and currently presents itself as the largest bottleneck to achieving higher performance. One possible solution to this problem, batch transfers, is discussed below in the future work section.

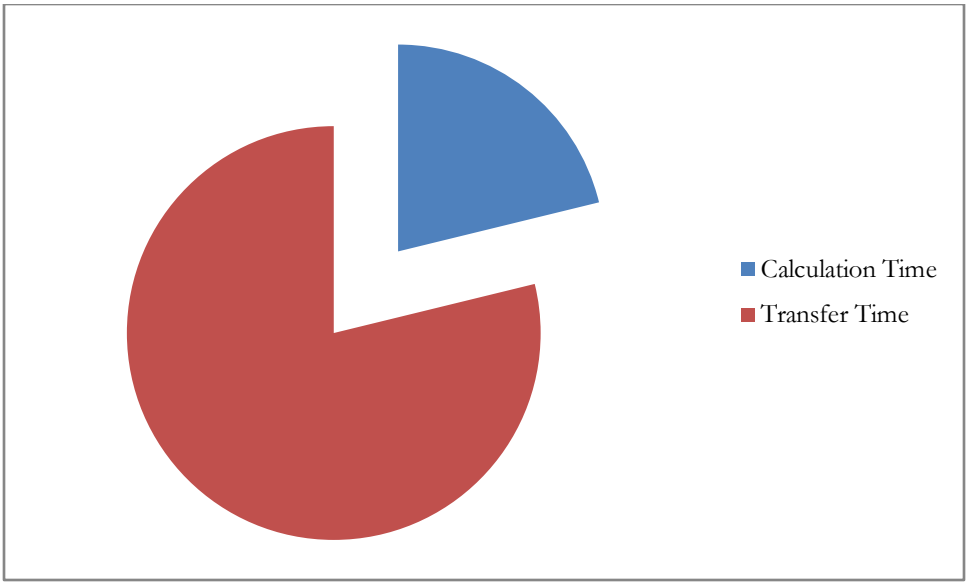

Figure 12: Division of accelerator run-time

3.1 Future Work

3.1.1 Software

In order to utilize the hardware accelerator, it is necessary to interface the hardware with the existing X!Tandem project. Analysis of the software shows that the `mprocess::score` is the controlling method for scoring. This function is where the all the possible cleavage peptides are created given a sequence of peptides and each peptide is tested and scored. This method will be altered to use accelerated dot function to score the peptides.

3.1.2 Batch Transfers

As the DMA overhead penalty is minimized for large data transfers, it would be more efficient to transfer multiple spectra to be analyzed at the same time. In order to accomplish this, the spectra to be analyzed must be accumulated and then transferred to the board as one DMA transfer. This improvement has the potential to reduce the amount of overhead and CPU time per dot.

Data transfer times were measured for arrange of array sizes. The results of these measurements show that data transfer rates scale favorably with respect to the array size. These results can be seen in the figures below. The time saved per integer increases until 15,000 integers are transferred, at which point the time savings remain constant. This suggests that it would make sense to break the data transfers into groups of 15,000 integers each and process those as one batch, thus reducing the overhead required.

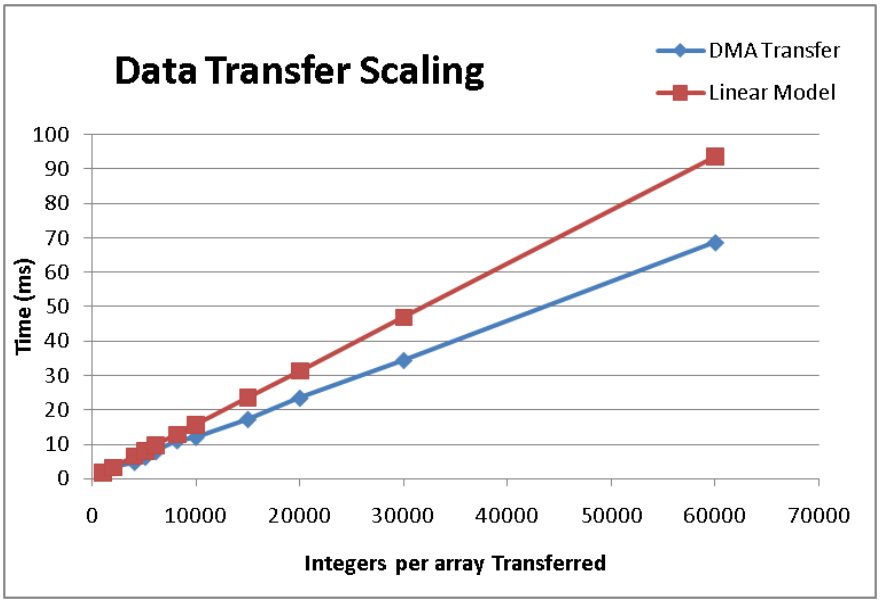

Figure 13: Data transfer scaling

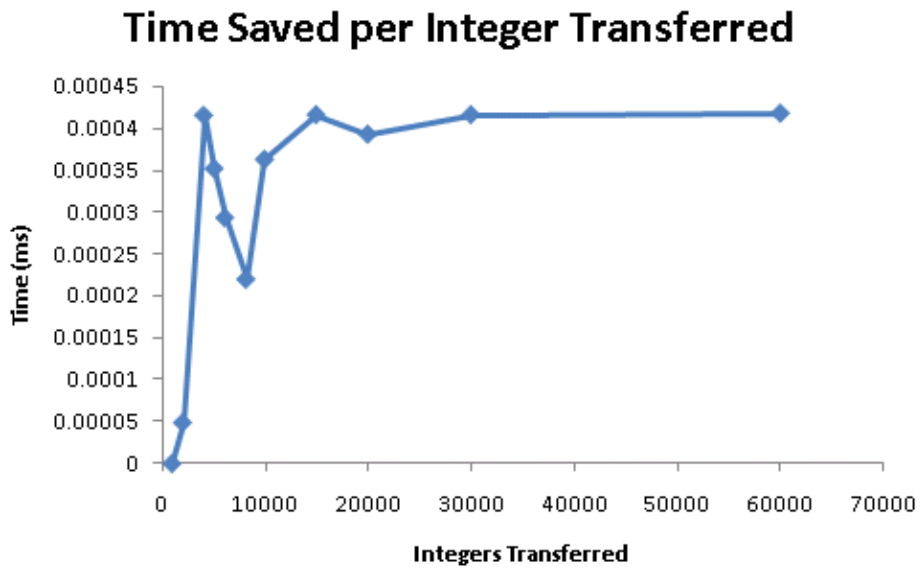

Figure 14: Time saved per integer tranferred

3.1.3 Soft-processor Co-processor

A possibility for reducing the amount of effort involved in translating a software routine into a hardware routine is to include a full soft-processor as the accelerator. This soft-processor could interface with the PCI bus and achieve the same data transfer rate as the custom accelerator using DMA. The main advantage of using this setup would be that the processing could be done with software routine loaded on the soft-processor. Additionally, existing tools such as Altera's C2H compiler could be leveraged to translate software routines directly into hardware. This work flow would allow for even more rapid prototyping of hardware accelerators than what is possible when writing a custom accelerator in Verilog.

4 Conclusions

A method for analyzing and characterizing an existing software project to find routines suitable for acceleration has been demonstrated. A speedup of 2.59 was demonstrated. The

designated function was accelerated in a manner which cut the processing time for the main CPU to approximately one-third of its original time for the specified function. The hardware accelerator was interfaced with software on the host computer over a PCI bus. The use of DMA transfer over the PCI bus coupled with a sequential hardware accelerator has been shown to provide an effective and efficient way to accelerate a specific software routine to accelerate protein identification.

5 Appendix

```
module compareAndMult(m_lM, m_fI, m_plSeq, m_pfSeq, out);

input          [31:0]    m_lM;
input          [31:0]    m_fI;
input          [31:0]    m_plSeq;
input          [31:0]    m_pfSeq;
output reg     [31:0]    out;
reg            [31:0]    mult;

always@(m_lM or m_fI or m_plSeq or m_pfSeq) begin
     mult = m_fI*m_pfSeq;
     if(m_lM == m_plSeq && m_fI*m_pfSeq > 0) begin
          out = mult;
     end
     else begin
          out = 0;
     end
end

endmodule
```

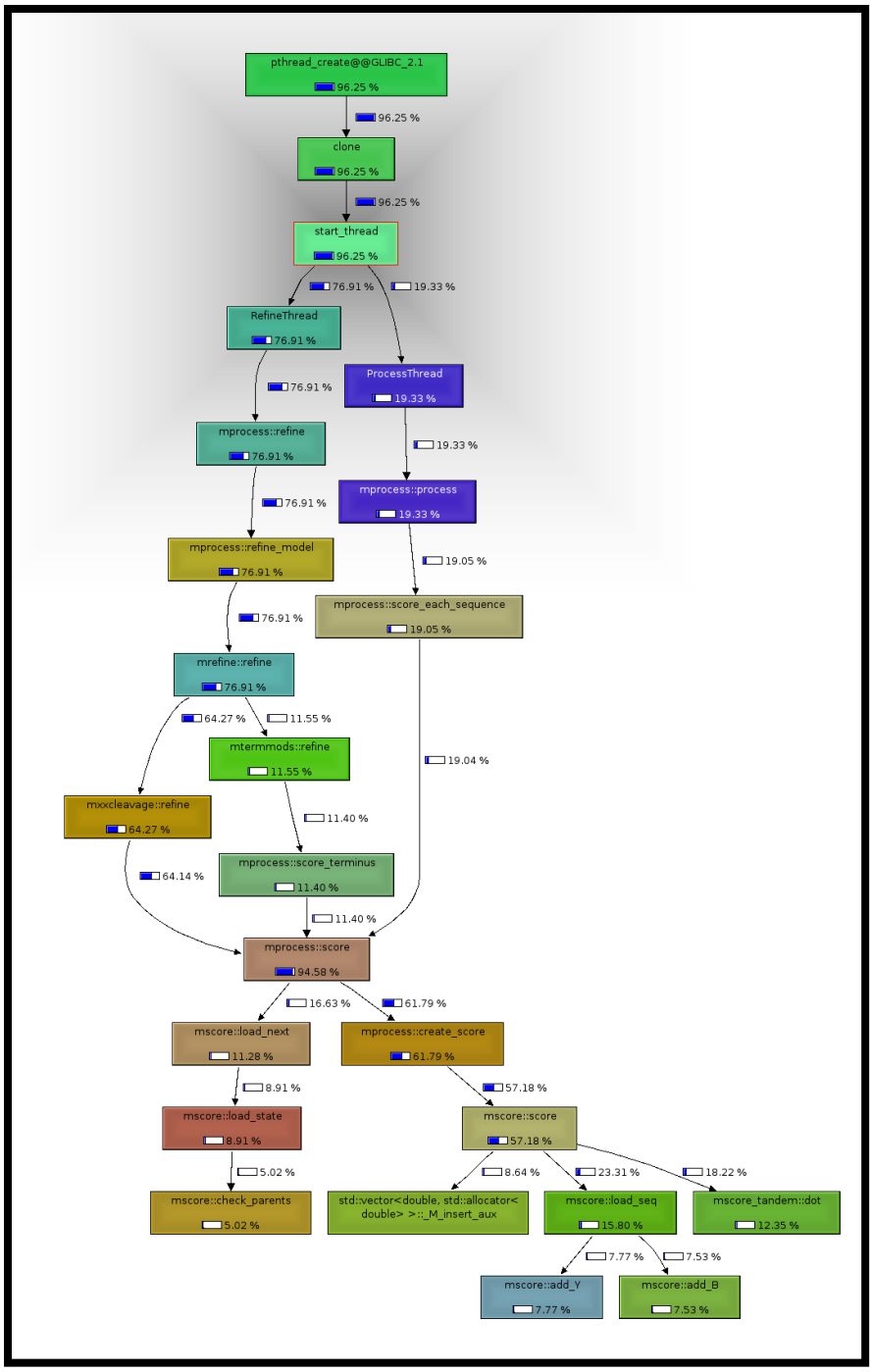

Figure 15: X!Tandem callgraph

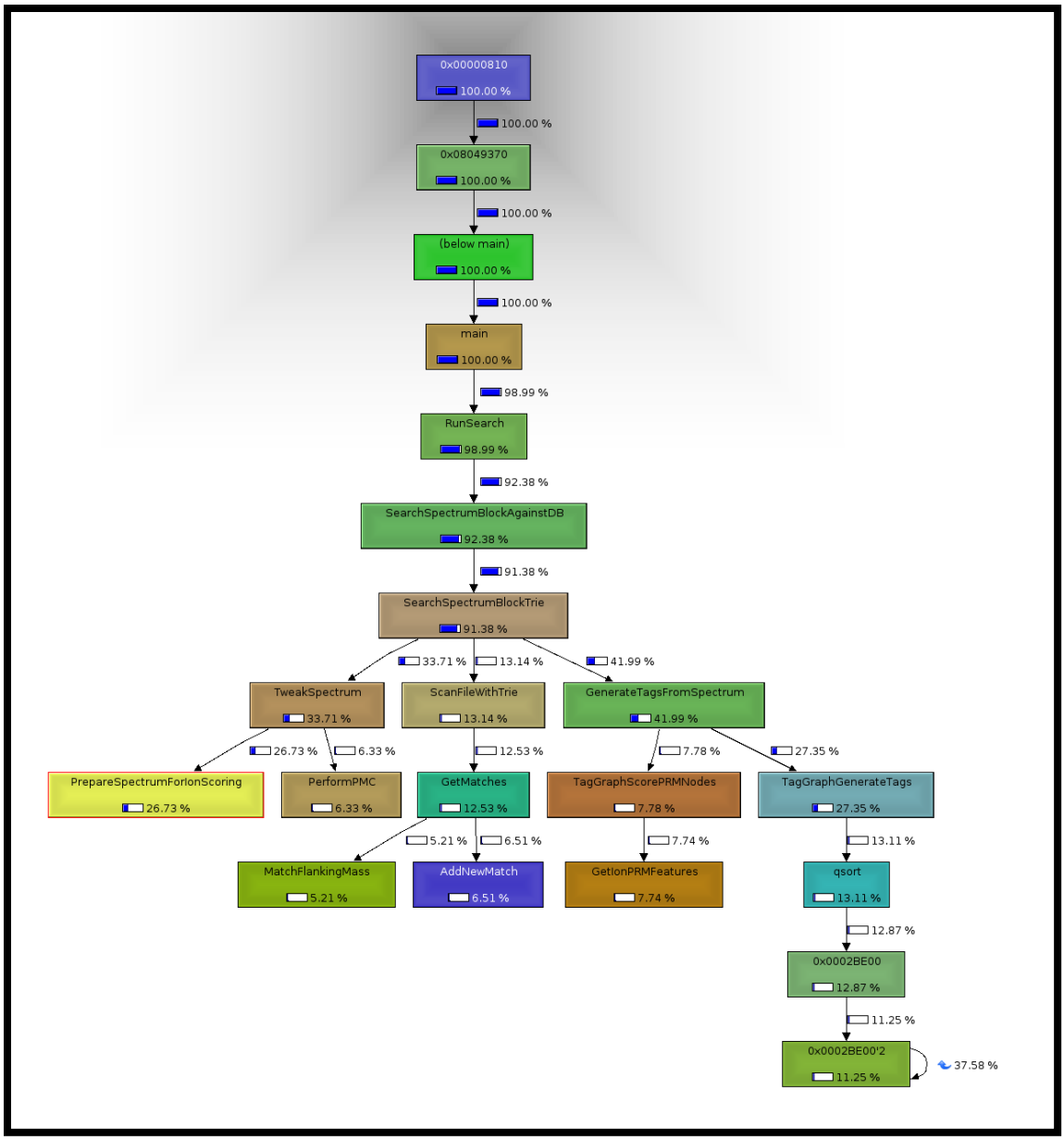

Figure 16: Inspect callgraph

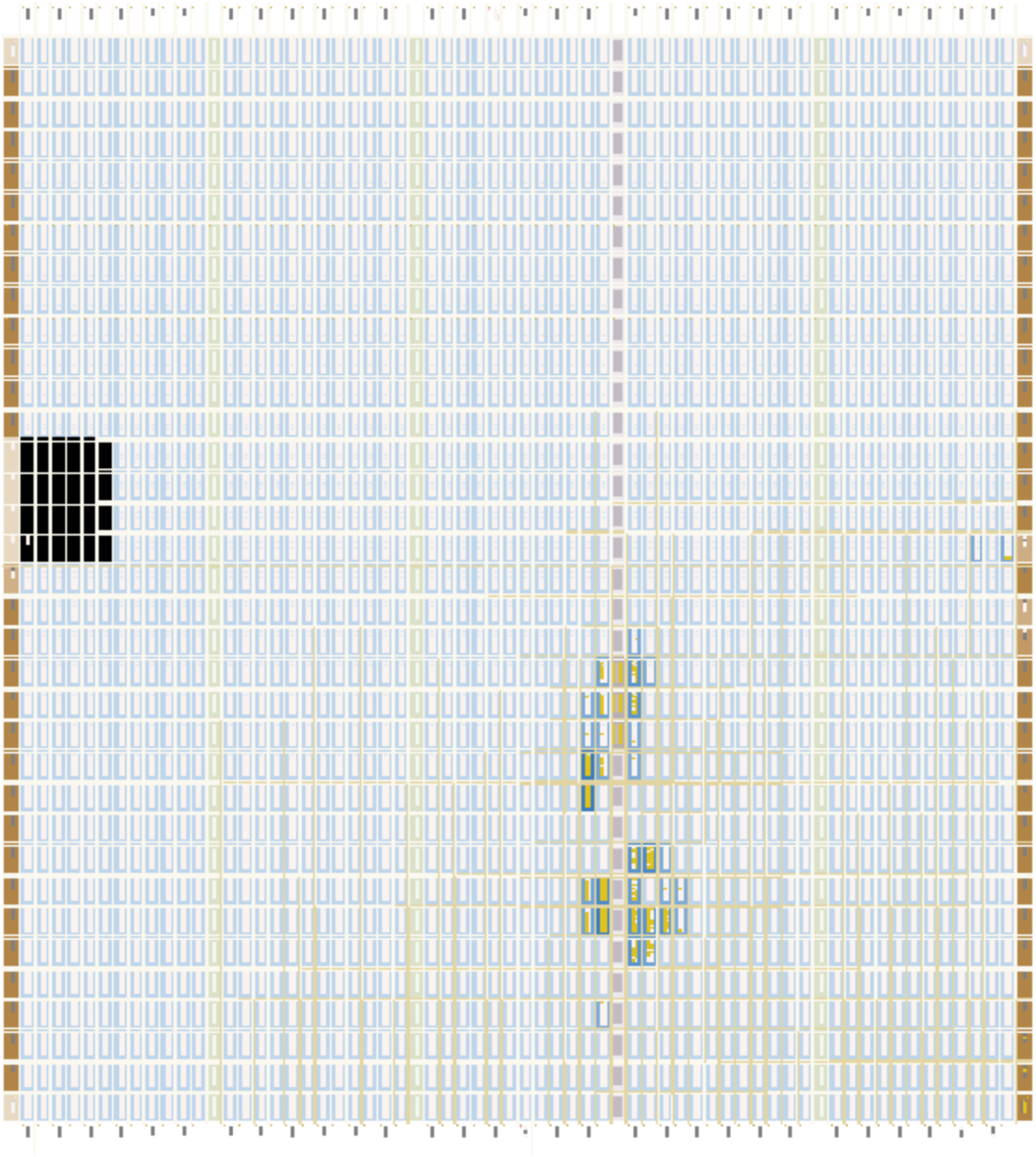

Figure 17: Register-based design floor plan

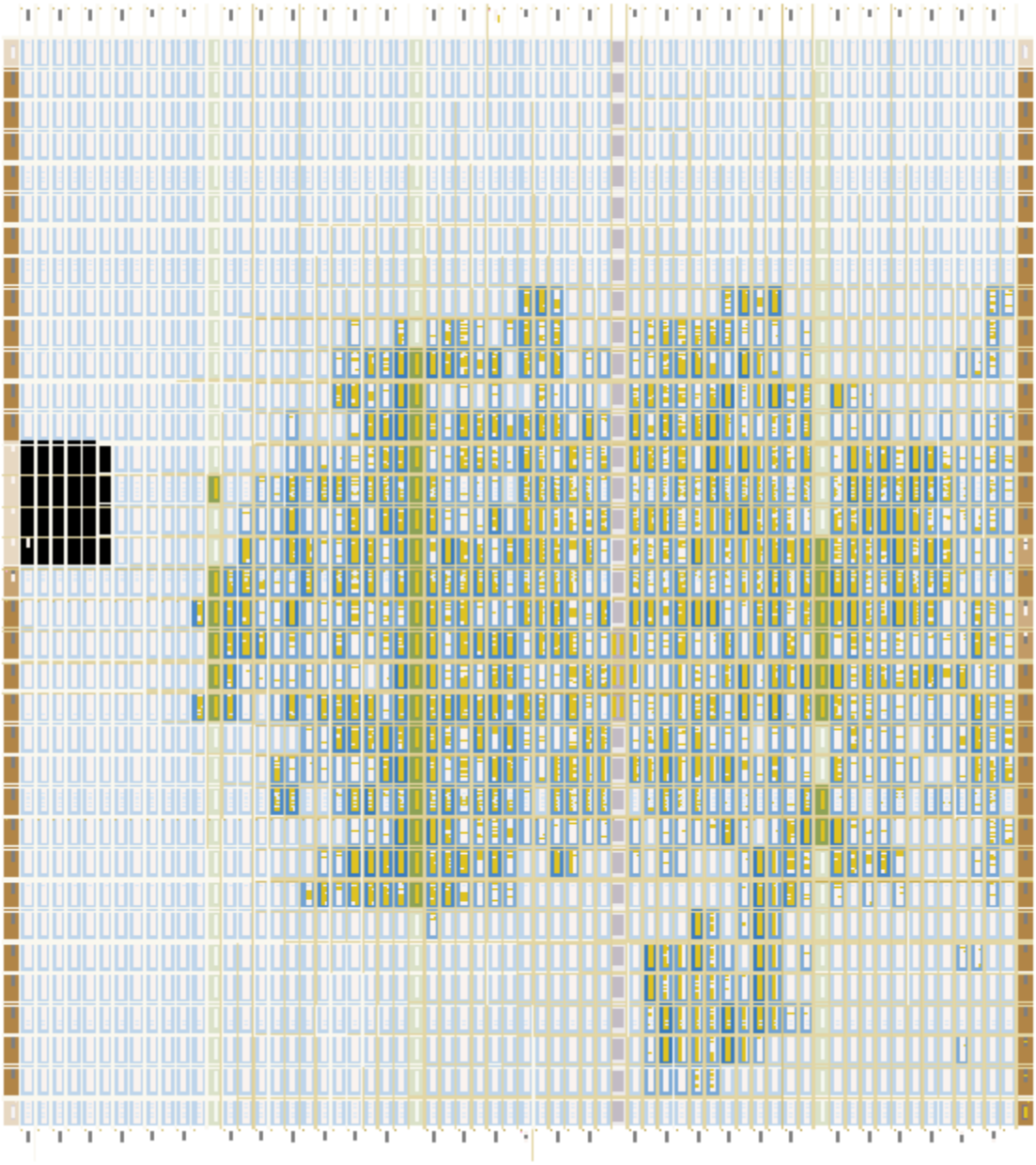

Figure 18: DMA based floor plan

6 References

[i] Betz, Vaughn. "FPGA Architecture." University of Toronto. <http://www.eecg.toronto.edu/~vaughn/challenge/fpga_arch.html>.

[ii] Reda, Sherief. "RC Principles: Software." Brown University, Providence. 4 Oct. 2007. Na <http://ic.engin.brown.edu/classes/EN2911XF07/lecture08.ppt>.

[iii] Fienberg, Bruce. "GiDEL Ships PROCSpark II Development Board Featuring Altera'S Low-Cost FPGAs." 25 May 2005. Altera. 18 Apr. 2008 <http://www.altera.com/corporate/news_room/releases/releases_archive/2005/products/nr-cyclone2_procspark2.html>.

[iv] "PROCSpark II: Cyclone II - Based PCI Prototyping Board for Frame Grabbing, DSP, Imaging, Vision, Rapid Prototyping." GiDEL. 14 Apr. 2008 <http://gidel.com/procSpark%20II.htm>.

[v] "U.S. Department of Energy Research News." 9 Apr. 2008 <http://www.eurekalert.org/features/doe/2001-06/drnl-pib061902.php>.

[vi] Gross, Jurgen H. Mass Spectrometry. Heidelberg: Springer, 2004.

[vii] Protein Identification Tutorial." IonSource 26 Oct 2007 <http://www.ionsource.com/tutorial/protID/idtoc.htm>.

[viii] Cook, Steven. "Spectroscopy." 10 Apr. 2008 <http://www.steve.gb.com/science/spectroscopy.html>.

[ix] Robertson Craig and Ronald C. Beavis, Bioinformatics, 2004, 20, 1466-7.

[x] "De Novo Peptide Sequencing." Ion Source. 19 Apr. 2008

<http://www.ionsource.com/tutorial/DeNovo/b_and_y.htm>.

[xi] "CSE Bioninformatics Group." November 20, 2007. <http://proteomics.bioprojects.org/Software/Inspect.html>.

[xii] Tanner, Stephe, Hongjun Shu, Ari Frank, Ling-Chi Wang, Ebrahim Zandi, Marc Mumby, Pavel A. Pevzner, and Vineet Bafna. "InsPecT: Fast and Accurate Identification of Post-Translationally Modified Peptides From Tandem Mass Spectra."

[xiii] "Sage-N Research Products." October 24, 2007. <http://www.sagenresearch.com/products.html>.

[xiv] Alex, Anish. "Hardware Accelerated Protein Identification."

[xv] Zosso et. al. "Tandem Mass Spectrometry Protein Identification on a PC Grid"

[xvi] "Sashimi Data Repository." 17 Nov. 2007 <http://sashimi.sourceforge.net/repository.html>.

[xvii] "UniProtKB/Swiss-Prot." 17 Nov. 2007 <http://www.ebi.ac.uk/swissprot/>.

[xviii] Searle, Brian C. "X!Tandem Explained." Proteome Softwar Inc. Portland. Na.