Bio-Sequence Analysis with Cradle's 3SoC™ Software Scalable System on Chip

Xiandong Meng
Department of Electrical and Computer Engineering
Wayne State University
5050 Anthony Wayne Dr.
MI, Detroit 48202
meng@ece.eng.wayne.edu

Vipin Chaudhary
Institute for Scientific Computing
Wayne State University
5143 Cass Avenue
Detroit, MI 48202
vipin@wayne.edu

ABSTRACT
With the dramatically increasing amounts of genomic sequence database, there is a need for faster and more sensitive searching for sequence similarity analysis. The Smith-Waterman algorithm, which utilizes dynamic programming, is a common method for performing exact local alignments between two protein or DNA sequences. The Smith-Waterman algorithm is exhaustive and generally considered to be the most sensitive, but long computation times limit the use of this algorithm. This paper presents a preliminary implementation of Smith-Waterman algorithm using a new chip multiprocessor architecture with multiple Digital Signal Processors (DSP) on a single chip leading to high performance at low cost.

Categories and Subject Descriptors
J.3 [Computer Applications]: LIFE AND MEDICAL SCIENCES - Biology and genetics

General Terms
Design, Performance

Keywords
Smith-Waterman algorithm, Digital Signal Processors, 3SoC chip

1. Introduction
Searching on DNA and protein databases using sequence comparison algorithm has become one of the most powerful technique to help determine the biological function of a gene or the protein it encodes. High sequencesimilarity implies significant functional similarity. Sequence comparison algorithms based on the dynamic programming method such as the Needleman-Wunsch [8] and Smith-Waterman [11] algorithms, provide optimal solutions. However, they are computationally expensive. For this reason, algorithm based on heuristics, such as FASTA, BLAST [1, 2], which are able to run faster but are less accurate, are widely used. Moreover, special purpose hardware [6] has been constructed to perform sequence searches at high speed, but these machines are quite expensive.

Smith-Waterman algorithm is the most sensitive algorithm to find the most similar subsequences of two sequences by dynamic programming. Several investigators [9, 12] have used SIMD based technology to speed up the Smith-Waterman algorithm. Other optimized algorithms [4, 7, 10] have been developed to improve the sequence searches. The algorithm has also been implemented on 144-PE FPGA [13] and special 512-PE board [5] to achieve high-speed homology search.

In this paper a method designed to exploit the advantages of Cradle's 3SoC™ [14] chip multiprocessor architecture to perform Smith-Waterman algorithm with both rapid and sensitive database searches is presented. The Cradle's 3SoC architecture is a new silicon architecture that has multiple RISC-like processors, DSP processors, DMAs, and programmable IOs that has tremendous heterogeneous compute and IO power to build a complete system. We investigate the potential of using this Software Scalable System-on-Chip to accelerate Bio-sequence analysis.

In the reminder of this paper, we first provide background on 3SoC chip architecture and sequence similarity search relevant details on Smith-Waterman algorithm in sections 2 and 3, respectively. Section 4 describes the issues with the implementation of the algorithm on the target architecture. Section 5 details the implementation of the algorithm on 3SoC. Section 6 deals with the performance of our preliminary implementation and conclusions derived. Finally, we end this paper with some pointers to our continuing work in section 7.

2. 3SoC™ Architecture Overview
Cradle's *Software Scalable System on Chip* (*3SoC*) architecture consists of dozens of high performance RISC-like and digital signal processors on a single chip with fully software programmable and dedicated input-output processors. The processors are organized into small groups, with eight digital signal processors and four RISC-like processors each sharing a block of local data and control memory, with all groups having access to global information via a unique on-chip bus—the Global Bus. It is because data, signal, and I/O processors are all available on a single chip, and that the chip is thereby capable of

SAC '04, March 14-17, 2004, Nicosia, Cyprus.

implementing entire systems [14]. The block diagram is shown as Figure 1.

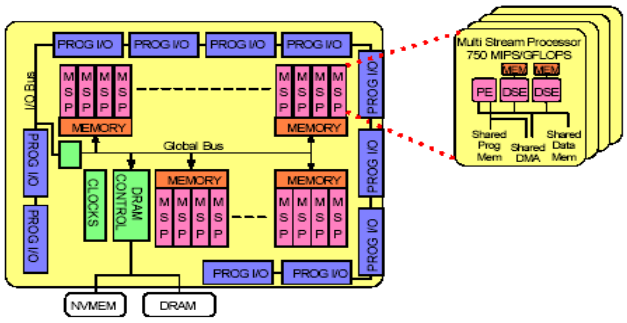

Figure 1: 3SoC Block diagram

The 3SoC is a shared memory MIMD (multiple instruction/multiple data) computer that uses a single 32- bit address space for all register and memory elements. Each register and memory element in the 3SoC has a unique address and is uniquely addressable.

2.1. Quads

The Quad is the primary unit of replication for 3SoC. A 3SoC chip has one or more Quads, with each Quad consisting of four PEs, eight DSEs, and one Memory Transfer Engine (MTE) with four Memory Transfer Controllers (MTCs). In addition, PEs share 32KB of instruction cache and Quads share 64KB of data memory, 32K of which can be optionally configured as cache. Thirty-two semaphore registers within each quad provide the synchronization mechanism between processors. Figure 2 shows a Quad block diagram. Note that the Media Stream Processor (MSP) is a logical unit consisting of one PE and two DSEs.

Processing Element--The PE is a 32-bit processor with 16-bit instructions and thirty-two 32-bit registers. The PE has a RISC-like instruction set consisting of both integer and IEEE 754 floating point instructions. The instructions have a variety of addressing modes for efficient use of memory. The PE is rated at approximately 90 MIPS.

Digital Signal Engine--The DSE is a 32-bit processor with 128 registers and local program memory of 512 20-bit instructions optimized for high-speed fixed and floating point processing. It uses MTCs in the background to transfer data between the DRAM and the local memory. The DSE is the primary compute engine and is rated at approximately 350 MIPS for integer or floating-point performance.

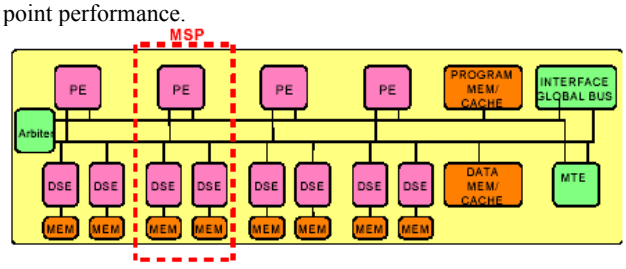

Figure 2: Quad Block diagram

2.2. Communication and Synchronization

Communication--Each Quad has two 64-bit local buses: an instruction bus and a data bus. The instruction bus connects the PEs and MTE to the instruction cache. The data bus connects the PEs, DSEs, and MTE to the local data memory. Both buses consist of a 32-bit address bus, a 64-bit write data bus, and a 64-bit read data bus. This corresponds to a sustained bandwidth of 2.8 Gbytes/s per bus.

The MTE is a multithreaded DMA engine with four MTCs. An MTC moves a block of data from a source address to a destination address. The MTE is a modified version of the DSE with four program counters (instead of one) as well as 128 registers and 2K of instruction memory. MTCs also have special functional units for BitBLT, Reed Solomon, and CRC operations.

Synchronization--Each Quad has 32 globally accessible semaphore registers that are allocated either statically or dynamically. The semaphore registers associated with a PE, when set, can also generate interrupts to the PE.

2.3. Software Architecture and Tools

The *3SoC* chip can be programmed using standard ANSI C or a C-like assembly language ("CLASM") or a combination thereof. The chip is supplied with GNU-based optimizing C-compilers, assemblers, linkers, debuggers, a functional and performance accurate simulator, and advanced code profilers and performance analysis tools. Please refer to *3SoC* programmer's guide [15].

Cradle's 3SoC is new hybrid parallel computer architecture for high performance computing. Hybrid architectures are the combination of the SIMD (single instruction/multiple data) and MIMD (multiple instruction/multiple data) paradigm within a parallel architecture in order to accelerate compute intensive tasks.

The 3SoC chip board can be easily connected to a host computer which runs WINDOWS by configuring an EPP compatible parallel port in system BIOS.

3. The Smith-Waterman algorithm

The Smith-Waterman algorithm [11] is perhaps the most widely used local similarity algorithm for biological sequence comparison. In Smith-Waterman database searches, the dynamic programming method is used to compare every database sequence to the query sequence and assign a score to each result. The dynamic programming method checks every possible alignment between two given sequences. The two sequences define a matrix in which every cell represents the alignment of two specific positions in the two sequences. The value of each cell depends on the residues located in these positions.

Scores in the first row and column are defined as zero. Entries L(i, j)in all other cells of the matrix are defined as the score of the best alignment ending in the position matching x_i and y_j, and are calculated using the following recurrences:

$$L(i, j) = \max\{E(i,j), L(i-1, j-1)+s(x_i, y_j), F(i, j), 0\};$$

where

$$E(i, j)=\max\{ L(i, j-1)+a, E(i, j-1)+b\}$$
$$F(i, j)=\max\{ L(i-1, j)+a, F(i-1, j)+b\}$$

where $s(x_i, y_j)$ is the score of a match or mismatch between x_i and y_j. In the above equations, a is the opened gap penalty and b is the extended gap penalty.

All the searching process can be divided into two phases. In the fist phase, all the elements of two sequences have to be compared and form a scoring matrix. Following this recurrence equation in Figure 2, the matrix is filled from top left to bottom right with each entry $L_{i, j}$ requiring the entries $L_{i-1, j}$, $L_{i, j-1}$ and $L_{i-1, j-1}$ with gap penalty a=b at each step.

Once scores in all cells are calculated, the second phase of the algorithm identifies the best local alignments. Since they might be biological relevant, alignments with score value above a given threshold are reported. Thus, for each element of the matrix a backtracking procedure is applied to find out the best local alignment. Our performance evaluation is based on the first phase because the workload of comparing two given sequences is fixed, but the trace-back of an alignment segment is dynamic. The final alignment may be longer or shorter highly depending on similarity.

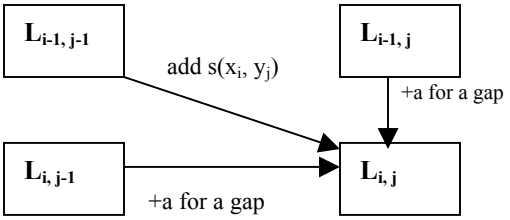

Figure 3: Dynamic programming illustration

In all, for a pair of sequence lengths m and n, the problem can be solved readily in O(m×n) time and O(m×n) space. Moreover, the order of computation of the values in the alignment matrix is strict because the value of any cell cannot be computed before the value of all cells to the left and above it have been computed.

4. Parallel Smith-Waterman algorithm Design

4.1 Background Analysis

The Smith-Waterman algorithm produces the most sensitive local pairwise alignments. However, Smith-Waterman has a high complexity, O(m×n), where m and n are lengths of the sequences being compared. The following is a breakdown by core component of the relative time taken in the unoptimized Smith-Waterman [12], as indicated by the Vtune Performance Analyzer.

Code portion	% of total time
Upper element plus gap penalization	25%
Left element plus gap penalization	15%
Upper-left element plus gap penalization	25%
Value choice	15%
Recording result	10%
Other	10%

Figure 4. Time distribution of Smith-Waterman algorithm

As should be evident from this table, the most significant time-sinks are the upper element and upper-left element calculation, but over 90% work is worthy of mapping to DSE processor.

Figure 5 shows the data dependencies in Smith-Waterman algorithm. As mentioned in the previous section, there are three possible alignments to choose from when calculating one element: alignment of the symbol in the row considered with gap -- horizontal arrow, alignment between the symbols in the row and column considered with match or mismatch – diagonal arrow, and

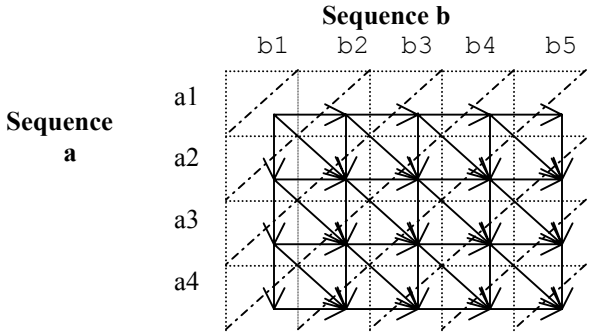

Figure 5: Data dependency in Smith-Waterman alignment matrix

alignment of the symbol in the column considered with a gap – vertical arrow. This means that rows or columns can't be computed in parallel. The only elements on each successive anti-diagonal (labeled dashed line in figure 5) are processed in parallel. These data dependencies present a serious challenge for sufficient parallel execution on a general-purpose parallel processor. Although it exposes parallelism, it brings some additional problems. If one element is processed by one processor at one time, the computation requires a large quantity of processors and synchronization overhead will be high. An additional problem is unbalanced workload. For example, if the computation would start from the upper left corner of an n*n matrix, the efficiency of each processor is only 50% on average. Based on the above analysis, for achieving the maximum performance, we let each DSE compare one database sequence with the query sequence individually in parallel.

4.2 General Strategy

The structure of the sequence similarity problem and dynamic programming lead to many opportunities for parallel computation within the 3SoC architecture. At the lowest level, micro-parallelism techniques have been used to take advantage of specific features of 3SoC parallel architecture. At a higher level, the searching of large databases leads itself to an implementation with multiples searches distributed across separate processors.

An individual search job specifies one query sequence to be searched against one or more sequence databases. In order to achieve parallel speed-up, database is split into multiple small subsequences, and 3SoC chip processes these subsequences in parallel, and integrates the results into a unified output. The PE coordinates the following activities on multiple DSE processors:

- Initialization of sequence alignment tasks, each of which requires the comparison of a query sequence against a block of databases sized so that the entire task can be distributed within the all-available DSE processors.
- Creation of the standard Smith-Waterman algorithm on all DSEs.
- Integration of the task results into a unified output.

MSP is a logical processing unit in 3SoC chip. There are one PE and two DSEs in each MSP. Therefore, 12 MSPs consisting of 24 DSEs can process 24 database sequences simultaneously in a three quad 3SoC. Here PE is a master or coordinator, which controls two corresponding DSEs to do the computation and one MTE to transfer data between on-chip memory and DRAM. This approach guarantees to generate the correct results as the serial version of Smith-Waterman since each DSE uses exactly same executable to perform the search computation.

4.3 Task Scheduling Strategy

In order to obtain an efficient parallel implementation, it is fundamental to achieve a good distribution for both data and computation. Our strategy is to fully take advantage of the hardware architecture in keeping processors busy, separation of tasks and so on. The task scheduling strategy is based on a master/slave approach. In essence, the PE acts as a master, scheduling and dispatching blocks of database sequence to the slaves, DSE processors, which perform the algorithm calculations. When the DSEs report results for one block, the PE sends a new block of sequence. Additional improvements are obtained by applying double buffering strategies (pre-fetching data) that reduce or eliminate DSE processor inactivity while waiting for a new block of sequences. The DSE has one new block ready for computation as soon as the other block is completed.

High performance is achieved in two ways. First, a large database is partitioned evenly so that all the DSE processors have enough work to do and the dynamic scheduling enables load balancing. Second, the DSE processors are kept busy by using double buffering strategy.

5. Implementation Details

The implementation of Smith-Waterman algorithm on 3SoC chip, composed of a master and a number of workers (slaves), is a parallel application.

Master: The Master process, executing on a PE processor, accepts an initial search task and sets it up for processing by the workers. The Master process manages tasks execution, database access, data transfer, worker coordination, and outputs unified results. Another important part of the Master is the data provider, which manages the genomic databases used for Smith-Waterman tasks. It delivers the block of databases to the workers. Essentially, the workers may think of as having data in their own local memories, i.e., they search their local cached copies only.

The Master is also charged with moving the result of the computation matrix for backtracking. In the previous section we mentioned that for each element, the program has to send 3 calculated values to compare. One solution to this problem is to divide the computation matrix into blocks, where PE0 controls two buffers of DSE0 for block assignment. The maximum block size is fixed in order to fit into the quad local memory. So, for short sequences, the computation sub-matrix may fit completely into local memory, and the adjacent data can be always kept in double buffers. In this way, DSE does all local reads without doing the expensive DRAM reading. And for long sequences, the sub-matrix has to be divided into several segments to move multiple times.

Workers: Worker processes execute on DSE processors that performs the actual alignment search by running a local copy of the standard Smith-Waterman algorithm. In the case of Smith-Waterman searching, each DSE makes use of double buffering scheduling algorithm to decide what part of database search to perform. DSE does the computation on a large number of independent data and must be processed efficiently in parallel.

5.1 Double Buffering Optimization

In each quad of 3SoC processor, there is a special Memory Transfer Engine (MTE) processor providing each quad with four Memory Transfer Controllers (MTCs) transferring data between the local memory and the DRAM in the background. Note that each Quad has 64KB of data and cache, and each quad has 8 DSE processors. Therefore, each DSE has up to two 4KB local memory buffers for data transfer. MTE has ability to move block of data from DRAM to the local memory of PE/DSE with high speed. Once the MTE has finished transferring the data, it interrupts the PE, signaling the completion of the data transfer. The PE then lets DSE start its computations. Hardware semaphores are used for interrupts.

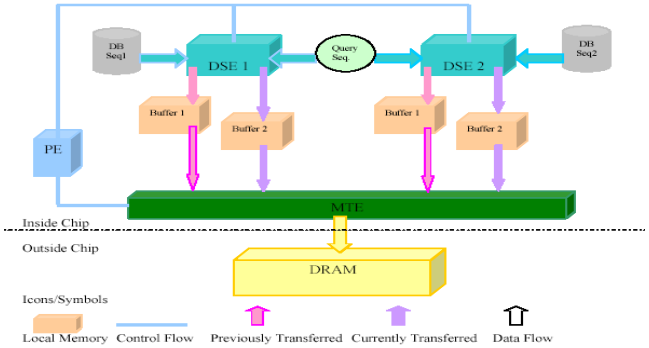

Figure 6: 3SoC Double Buffering Architecture

Double buffering strategy is used to transfer the database sequence to local memory from DRAM and move the computation matrix block back to DRAM as shown in Figure 6. This keeps the DSEs working uninterrupted at high speed. PE and DSE communicate using shared flags.

5.2 SIMD Optimization

SIMD instructions enable a DSE processor to exploit parallelism by dividing wide registers into smaller units and applying an instruction on each unit simultaneously. The DSE consists of the following: 128 word dual port data memory, ALU, multiplier, external memory read and write FIFOs, field access unit and a Program Control Unit (PCU).

Each character occupies one byte memory. When DSE starts to read the block sequence in its local memory, the database sequence and query sequence can be stored in the DSE FIFO memory. Each DSE read is one word (four bytes). The read FIFO is 128 characters deep to insure that the DSE has enough data to work with while the read logic is out getting more.

A set of Field Access registers, which was originally deigned for graphics, is used here. When the DSE applies the Smith-Waterman algorithm, it has to break one word into four parts using FD, FAC and FA registers, i.e., it retrieves four bytes

(characters) from one register to do the sequence similarity alignment one by one, then write the corresponding results to computation matrix. For example, in Figure 7 (left), FA is used as a data source register in an instruction, the contents of FD will be rotated right by 8 bits at the end of the instruction, i.e., the register contents will be rotated after the field extraction.

Figure 7 (right) is an example DSE CLASM code to show how the FA retrieve four characters in seven instructions. In this example, a 32-bit number contains ATGC format is disassembled into the A, T, G and C fields. The FA values show the 32-bit value of the extracted fields. Setting correct value to FAC control register can achieve the above result.

FD Register

A	T	G	C
C	A	T	G
G	C	A	T
T	G	C	A

FA Register

0	0	0	C
0	0	0	G
0	0	0	T
0	0	0	A

```
//DSE CLASM code
// 4 buffers get A, T, G and C respectively

//Set the FAC register to 0x00208208;
fac = nfac;

//FIFO READ () reads 4 bytes at one time
READ (seqa_ptr, 1);

Fd = FIFO;          // Data put into FD

Buffer1  =fa;       // char C in buffer1
Buffer2  =fa;       // char G in buffer2
Buffer3  =fa;       // char T in buffer3
Buffer4  =fa;       // char A in buffer4
```

Figure 7: FD & FA register usage and CLASM code example

6. Performance and Conclusions

The proposed strategy has been implemented and tested on the 3SoC chip taking advantage of its unique features, using 24 DSEs. The results here compare a single 100 MHz 3SoC chip with dual 350 MHz Pentium II processors, 400 MHz Sun Sparc and a 1.7 GHz Pentium IV processor. Figure 10 reports times for searching a 21M elements protein database, for various query lengths with Smith-Waterman algorithm as the first phase search. 3SoC chip is 2.5 times faster than a dual 350 MHz Pentium II processor, 3.5 times faster than a 400 MHz Sun Sparc, and a little bit faster than 1.7 GHz Pentium IV processor.

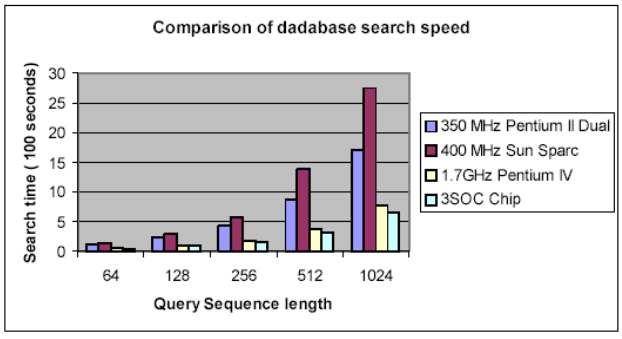

Figure 8: Comparison of database search speed on different Processors

In our preliminary implementation, 3SoC has proved to be highly performance efficient and scalable for sequence alignment. Even though the current performance is lower than some special designed hardware, 512-PE board [5] or 1536-PE Fuzion 150[10], this is a valuable baseline to do the biosequence alignment on a general-purpose DSP processor. Other factors that make 3SoC very attractive are its cost. A three quad 3SoC based PCI card would cost a fraction of the cost of the competition. Moreover, the low power of the processor enables design of a system with multiple 3SoC chips on a single board leading to the creation of a powerful system in small space that can be plugged into a desktop or server.

7. Future Work

We are currently working to optimize our current implementation by utilizing other available resources on 3SoC. A system is being designed that will enable multiple 3SoC chips to be integrated on a single PCI card thereby enabling much faster sequence alignment. We are also looking into the redesign of our strategy for implementation on a new version of 3SoC, which has significantly more SIMD MAC instructions that could speedup our current implementation tremendously.

8. References

[1] Altschul, S.F., Gish, W., Miller, W., Myers, E.W. and Lipman, D.J. Basic local alignment search tool. J. Mol. Biol.215, (1990), 403-410

[2] Bjornson, R.D., Sherman, A.H., Weston, S.B., Willard, N. and Wing, J. TurboBLAST: A Parallel Implementation of BLAST Built on the TurboHub, International Parallel and Distributed Processing Symposium: IPDPS Workshops (2002), p.0183

[3] Eyre, J. and Bier, J DSP Processors Hit the Mainstream. IEEE Computer Magazine (1998) Vol. 31, No. 8. pp. 51-59

[4] Gotoh, O. An improved algorithm for matching biological sequences J. Mol.Biol.(1982) 162, 705-708

[5] Grate, L., Diekhan, M., Dahle, D. and Hughey, H. Sequence Analysis With the Kestrel SIMD parallel Processor.Pacific Symposium on Biocomputing 2001 pp.263-74

[6] Hughey, R. Parallel hardware for sequence comparison and alignment. (1996) Comput. Appl. Biosci. 12, 473-479

[7] Lau, F. An Integrated Approach to Fast, Sensitive, and Cost-Effective Smith-Waterman Multiple Sequence Alignment. Bioinformatics Module, (2000) MB&B 452

[8] Needleman, S. and Wunsch, C. A general method applicable to the search for similarities in the amino acid sequence of two sequences. . J. Mol. Biol., 48(3), (1970), 443-453

[9] Rogens, T. and Seeberg. E. Six-fold speed-up of Smith-Waterman sequence database searches using parallel processing on common microprocessors. Bioinformatics, 16, (2000), 699-706.

[10] Schmidt, B., Schroder, H. and Schimmler, M. Massively Parallel Solutions for Molecular Sequence Analysis, International Parallel and Distributed Processing Symposium: IPDPS Workshops (2002), p. 0186

[11] Smith, T.F. and Waterman, M.S. Identification of common molecular subsequences. J. Mol. Biol., 147, (1981), 195-197

[12] Taylor, Stewart Applying MMX Technology to the Smith-Waterman Algorithm. Department of Electrical Engineering, Stanford University, www.stanford.edu/~sntaylor/mis299/report.htm

[13] Yamagucchi Y., Maruyama, T. High Speed Homology Search with FPGA. Pacific Symposium on Biocomputing 2002

[14] *3SoC* Documentation--*3SoC* 2003 Hardware Architecture, *Cradle Technologies, Inc*. Mar 2002.

[15] *3SoC* Programmer's Guide, *Cradle Technologies, Inc*., http://www.cradle.com, Mar 2002.