Report Number: WUCSE-2010-23

2010

Split and Merge Functions for Supporting Multiple Processing Pipelines in Mercury BLASTN

Authors: Jwalant Ahir, Jeremy Buhler, and Roger D. Chamberlain

Biosequence similarity search is an important application in computational biology. Mercury BLASTN, an FPGA-based implementation of BLAST for DNA, is one of the alternatives for fast DNA sequence comparison. The re-design of BLAST into a streaming application combined with a high-throughput hardware pipeline have enabled Mercury BLAST to emerge as one of the fastest implementations of bio-sequence similarity search. This performance can be further enhanced by exploiting the data-level parallelism present within the application. Here we present a multiple FPGA-based Mercury BLASTN design in order to double the speed and throughput of DNA sequence computation. This paper describes a dual Mercury BLASTN design, the detailed design of the split and merge functions, and simulation results.

2010-23

Split and Merge Functions for Supporting Multiple Processing Pipelines in Mercury BLASTN

Authors: Jwalant Ahir; Jeremy Buhler; Roger D. Chamberlain

Corresponding Author: roger@wustl.edu

Abstract: Biosequence similarity search is an important application in computational biology. Mercury BLASTN, an FPGA-based implementation of BLAST for DNA, is one of the alternatives for fast DNA sequence comparison. The re-design of BLAST into a streaming application combined with a high-throughput hardware pipeline have enabled Mercury BLAST to emerge as one of the fastest implementations of bio-sequence similarity search. This performance can be further enhanced by exploiting the data-level parallelism present within the application. Here we present a multiple FPGA-based Mercury BLASTN design in order to double the speed and throughput of DNA sequence computation. This paper describes a dual Mercury BLASTN design, the detailed design of the split and merge functions, and simulation results.

Type of Report: Other

Split and Merge Functions for Supporting Multiple Processing Pipelines in Mercury BLASTN

Jwalant Ahir[1], Jeremy Buhler[2], and Roger D. Chamberlain[1,2]
[1]BECS Technology, Inc., St. Louis, MO
[2]Dept. of Computer Science and Engineering, Washington University, St. Louis, MO

Abstract

Biosequence similarity search is an important application in computational biology. Mercury BLASTN, an FPGA-based implementation of BLAST for DNA, is one of the alternatives for fast DNA sequence comparison [3,4,5]. The re-design of BLAST into a streaming application combined with a high-throughput hardware pipeline have enabled Mercury BLAST to emerge as one of the fastest implementations of bio-sequence similarity search. This performance can be further enhanced by exploiting the data-level parallelism present within the application. Here we present a multiple FPGA-based Mercury BLASTN design in order to double the speed and throughput of DNA sequence computation. This paper describes a dual Mercury BLASTN design, the detailed design of the split and merge functions, and simulation results.

Introduction

Bio-sequence similarity search has emerged as one of the central problems in computational biology, and a significant amount of effort has been dedicated to increase its performance. The popular application BLAST (Basic Local Alignment Search Tool) used for bio-sequence similarity search employs a combination of heuristics and algorithmic improvements to accelerate the search by orders of magnitude [1,2]. However, the exponential growth in size of the input data sets makes the application compute-intensive, thus creating the need for further performance enhancements. Improvements to the architecture, algorithm, and data-path design, as well as increasing the efficiency of the underlying application topology, can therefore play a major role in saving computational time. Mercury BLASTN, the FPGA accelerator for the BLAST family of DNA comparison algorithms based on NCBI (National Center for Biological Information) BLASTN for traditional CPU based platforms, has shown remarkable increase in performance[ref]. These encouraging results have motivated us to speedup Mercury BLASTN by an additional factor of two by multiplexing two concurrent processing pipelines. This report describes the multiplexing design in order to achieve increased throughput with Mercury BLASTN.

After loading a query into the hardware pipeline, the current version of Mercury BLASTN consumes a stream of database symbols to identify and extend regions of similarity between the query and database. The idea explored here is to deliver a single copy of the database stream into two copies of the FPGA pipeline for computation. The resultant data streams output from the pipeline are multiplexed prior to delivery back to the software. An FPGA can support one Mercury BLASTN pipeline, and hence in our approach we require two FPGAs. An FPGA-to-FPGA Interface (FFI) is used to communicate between two FPGAs. The available hardware platform is a high-performance server motherboard with one or more AMD Opteron processors and two Xilinx Vertex-4 FPGAs on a PCI-X board.

This report describes the split and merge functions designed to support doubling the throughput of BLASTN. To simplify the exposition and initial simulation investigation, the split and merge functions are explored without having the actual BLASTN pipelines present.

Architecture

Figure 1 shows the high-level view of the FPGA design with two BLASTN pipelines. The IRD on the left provides the interface to the PCI-X bus (and subsequently to the processor). Incoming data to FPGA1 is split (replicated), and one path goes to the BLASTN pipeline on the same FPGA and the other path goes to a second BLASTN pipeline loaded onto FPGA2. The results out of the back end of each of these pipelines is merged for delivery back to software (via the IRD).

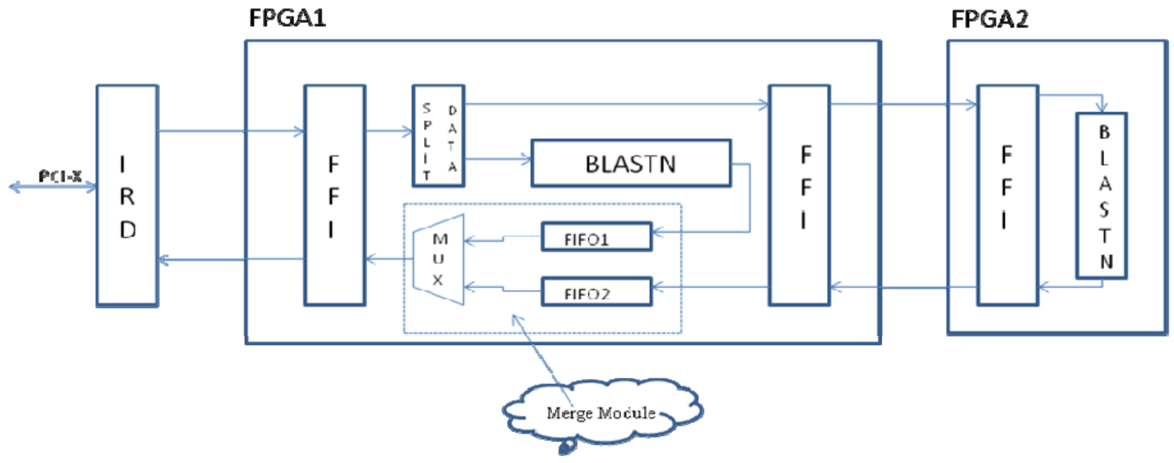

Figure 1: Dual FPGA approach to double the throughput of BLASTN

Operation

The objective is to compare a given query sequence against the database. Hence, the database sequences are sent along with commands and valid signals through the PCI-X bus to the split module via FPGA interface. The split module generates two identical copies of the input data and passes a copy to each BLASTN module in its respective FPGA. The data streams are processed by both BLASTN modules, and results are temporarily stored in their respective input buffers (FIFO BUF). Data from the input buffers is dequeued by control of the multiplexer's select lines. Here, we describe the internal design of the merge function.

Two counters – toggle counter (toggle_cnt) and VW counter (vw_cnt) – control the select lines of the multiplexer. Toggle counter is a down counter that determines when to switch between input buffers (FIFO BUF). VW counter is used for a special-purpose command called the VW (variable word) command. It is used to mark the beginning of a command block containing one or more 64-bit words transmitted on the command channel. Each BLASTN module preserves its output data in the respective input buffers. 64-bit control signal is processed when control valid (ctrl_vld) signal is asserted, whereas 64-bit data is processed when the data valid (data_vld) signal is asserted. When an input buffer gets full, the module generates a backpressure signal (wait_upstrm) signal indicating that it cannot accept additional input from the BLASTN pipeline. Another important signal is wait_dnstrm, generated when there is any back pressure from downstream in the pipeline.

Since only one copy of BLASTN fits on a single FPGA, we use two FPGAs and therefore require FPGA-to-FPGA communication. to know about the FFI module. FFI (FPGA-to-FPGA Interface module) consists of two separate components: a controller that parses commands targeted to the FFI module, and an FPGA-to FPGA component that is responsible for sending and receiving data to and from an FPGA. FFIs are typically used in pairs and structurally are complementary to each other. The transfer of data from main memory to the FPGA is under the control of a firmware socket (DMA / IRD). In the current system, this firmware socket issues transactions on the PCI-X bus to perform DMA transfer to the main memory.

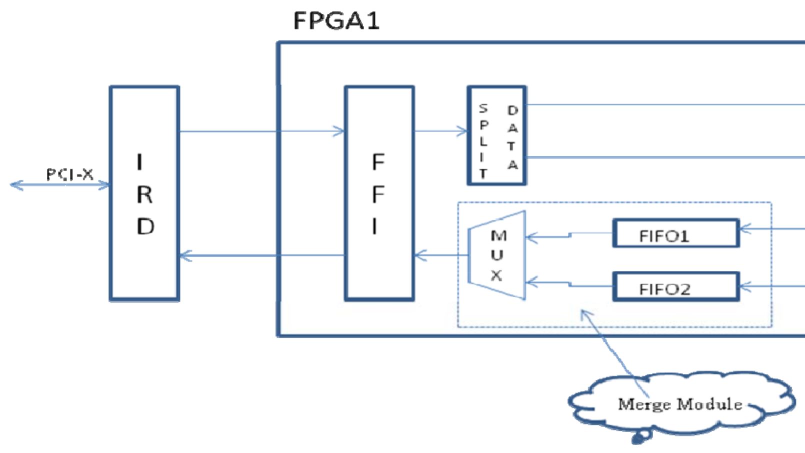

Figure 2: Experimental block diagram to check simulation results of a Split-Merge module

To test the system with minimum algorithmic overhead, we instantiated a design that invokes the Split and Merge modules but does not include any intervening processing. Input data is replicated and passed to the respective input buffers (FIFO BUF) as shown in Figure 2. The Mux select lines are selected by toggle_cnt and vw_cnt in order to read the input buffers alternately. The design was functionally verified successfully, and then we instantiated a single BLASTN module in FPGA1 is shown in Figure 3. Figures 4, 5, and 6 are portions of the entire simulation results that demonstrate the behavior of the system.

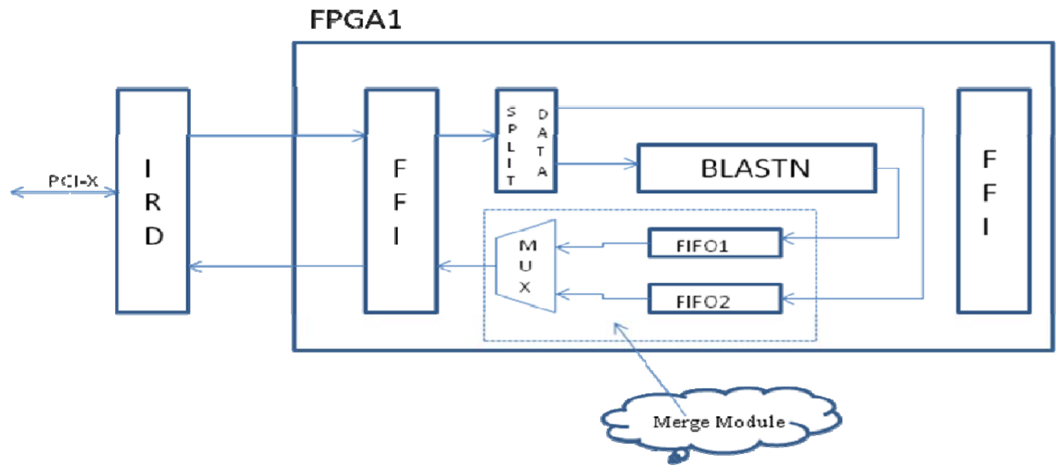

Figure 3: Experimental block diagram to check simulation results of a Split-Merge module with BLASTN

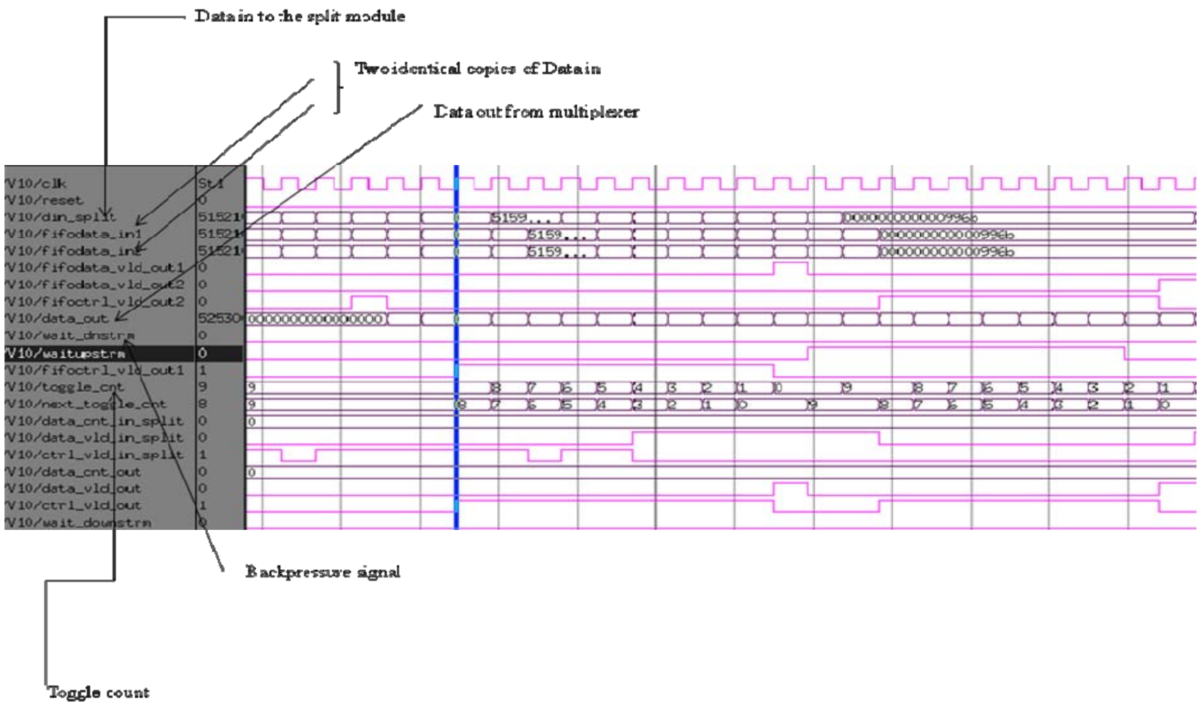
Data in to the split module
Two identical copies of Data in
Data out from multiplexer
Backpressure signal
Toggle count

Figure 4: Simulation waveform 1

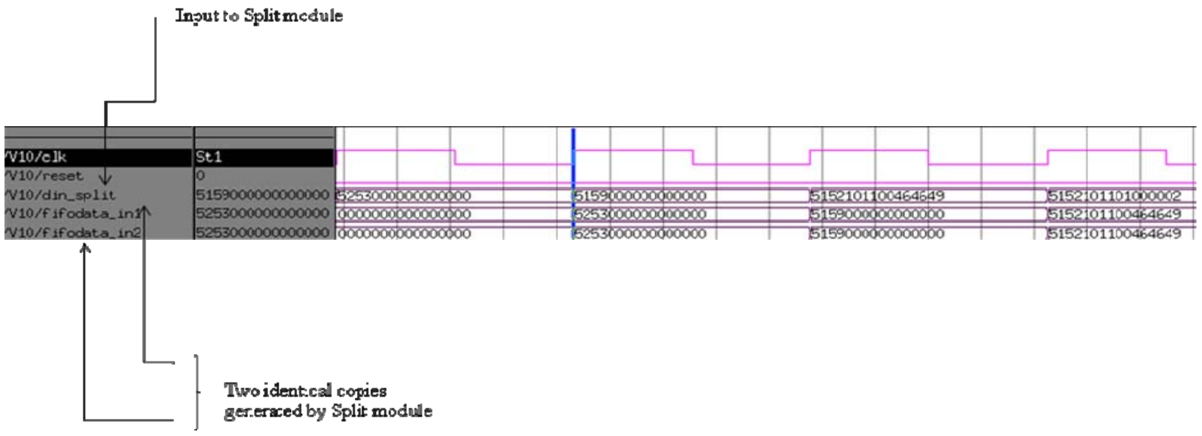

Figure 5: Simulation waveform 2

Figure 4 shows that input data (din_split) splits into two identical copies fifodata_in1 and fifodata_in2. Control valid (fifoctrl_vld_out), data valid (fifodata_vld) and output data (data_out) from mux are transferred at every clock cycle and FIFO is switched when toggle_cnt is reached to zero. Figure 5 is a zoom out of split copies of input data. Figure 6 shows the switching behavior of the FIFO; the 64-bit signal generated by Split-Merge module indicates which FIFO it switched from. Figure 7 is a Finite State Machine that demonstrates different states of operation of the Split-Merge design. A pseudo code followed by the FSM provides better idea of design operation.

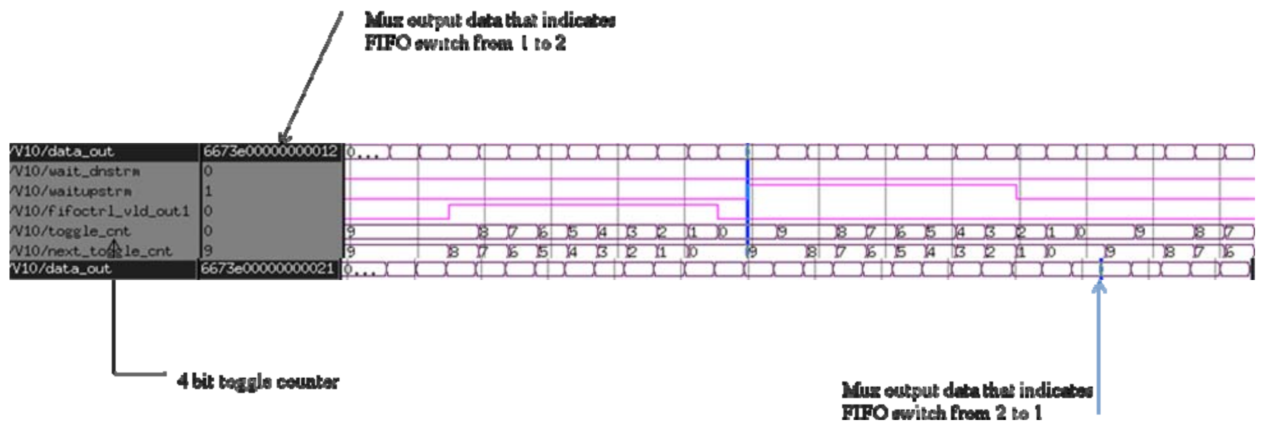
Mux output data that indicates FIFO switch from 1 to 2
/V10/data_out 6673e00000000012
/V10/wait_dnstrm 0
/V10/waitupstrm 1
/V10/fifoctrl_vld_out1 0
/V10/toggle_cnt 0
/V10/next_toggle_cnt 9
/V10/data_out 6673e00000000021
4 bit toggle counter
Mux output data that indicates FIFO switch from 2 to 1

Figure 6: Simulation waveform 3

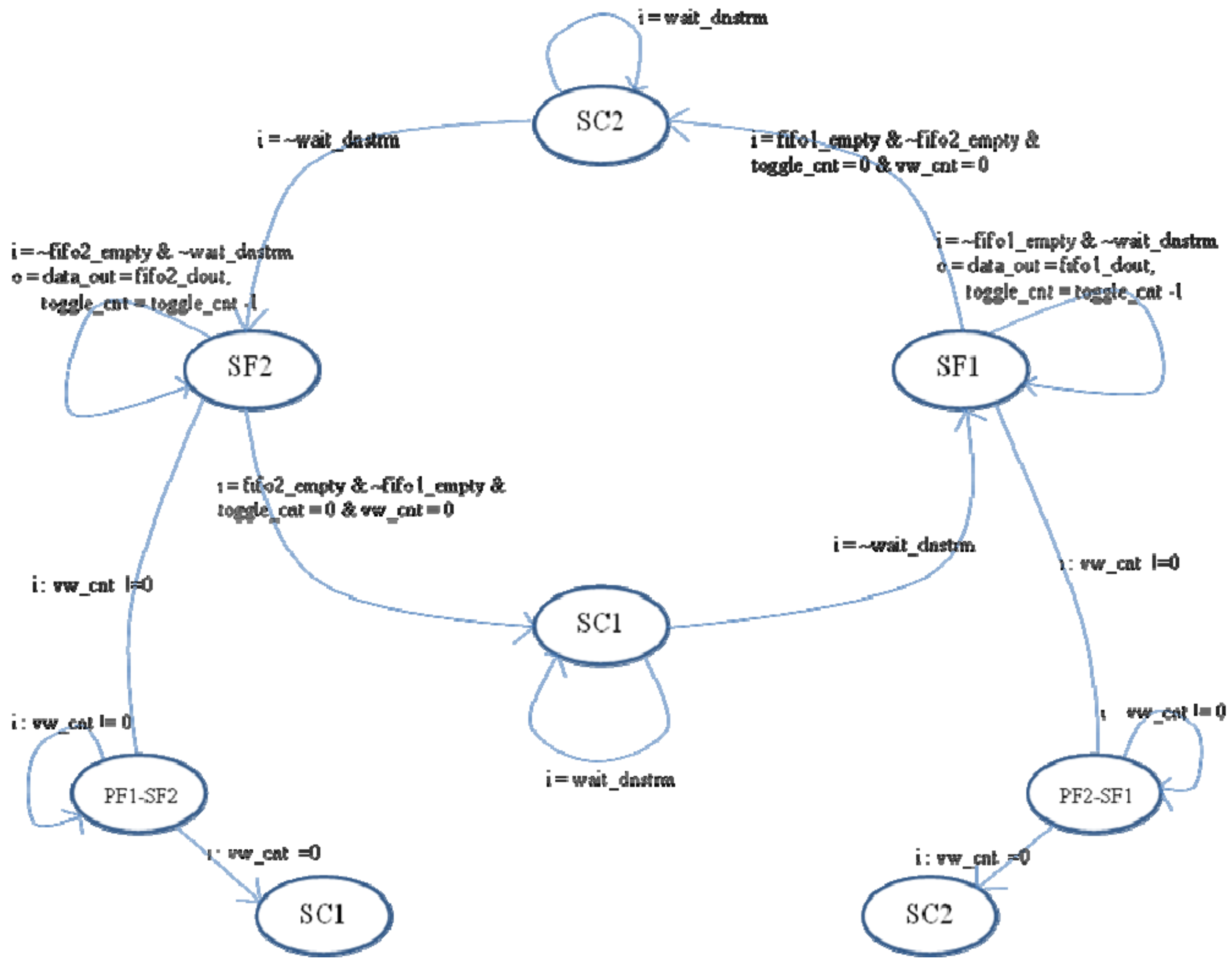
i = wait_dnstrm
SC2
i = ~wait_dnstrm
i = fifo1_empty & ~fifo2_empty & toggle_cnt = 0 & vw_cnt = 0
i = ~fifo2_empty & ~wait_dnstrm o = data_out = fifo2_dout, toggle_cnt = toggle_cnt -1
i = ~fifo1_empty & ~wait_dnstrm o = data_out = fifo1_dout, toggle_cnt = toggle_cnt -1
SF2
SF1
i = fifo2_empty & ~fifo1_empty & toggle_cnt = 0 & vw_cnt = 0
i = ~wait_dnstrm
i: vw_cnt != 0
i: vw_cnt != 0
SC1
i = wait_dnstrm
i: vw_cnt != 0
i: vw_cnt != 0
PF1-SF2
PF2-SF1
i: vw_cnt = 0
i: vw_cnt = 0
SC1
SC2

Figure 7: Finite State Machine to demonstrate inner operation of Split-Merge module

Pseudo code

```
case (state)
            SF1: // Sending FIFO1 data
                 begin
                     if (toggle_cnt != 0)
                        data_out = fifodata_out1;
                        if (toggle_cnt = 0 & VW_count != 0)
                            begin
                                data_out = fifodata_out1;
                                next_state = PF2-SF1;
                            end
                 end

            PF2-SF1: // Pending FIFO2_Sending FIFO1 data
                  begin
                      if (VW_count != 0)
                         data_out = fifodata_out1;
                      if (VW_count = 0  & ~fifo2_empty)
                         next_state = SC2;
                  end

            SC2: //Switch FIFO1 to FIFO2
                  begin
                      data_out = 0;
                      next_state = SF2;
                  end

            SF2:  //Sending FIFO2 data
                  begin
                      if (toggle_cnt != 0)
                         data_out = fifodata_out2;
                      if (toggle_cnt = 0 & VW_count != 0)
                         begin
                             data_out = fifodata_out2;
                             next_state = PF1-SF2;
                         end
                  end

            PF1-SF2:  // Pending FIFO1_Sending FIFO2 data
                  begin
                      if (VW_count != 0)
                         data_out = fifodata_out2;
                      if (VW_count = 0  & ~fifo1_empty)
                         next_state = SC1;
                  end

            SC1:  //Switch FIFO2 to FIFO1
                  begin
                         data_out = 0;
                         next_state = SF1;
                  end
endcase
```

Conclusion

This paper presents the design of a Split-Merge module that supports Mercury BLASTN on two FPGAs multiplexed in such a way that achieves double the speed and throughput of biosequence search application compared to single Mercury BLASTN system. So far, we have successfully prototyped the Split-Merge module with one BLASTN module on a single FPGA. The design is being extended to the two FPGA system that is shown in Figure 1. Simulation results are very encouraging. The comparison of the final results produced by the implementation will be performed shortly to observe the improvement in performance.

References

[1] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, et al., Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–10, 1990.

[2] S. F. Altschul, T. L. Madden, A. A. Sch¨affer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman, Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. *Nucleic Acids Research*, 25:3389–402, 1997.

[3] Jeremy Buhler, Joseph Lancaster, Arpith Jacob, and Roger Chamberlain, "Mercury BLASTN: Faster DNA Sequence Comparison using a Streaming Hardware Architecture," in *Proceedings of Reconfigurable Systems Summer Institute*, July 2007.

[4] Praveen Krishnamurthy, Jeremy Buhler, Roger Chamberlain, Mark Franklin, Kwame Gyang, Arpith Jacob, and Joseph Lancaster, "Biosequence Similarity Search on the Mercury System," *Journal of VLSI Signal Processing,* 49(1):101-121, October 2007.

[5] Joseph Lancaster, Jeremy Buhler, and Roger D. Chamberlain, "Acceleration of Ungapped Extension in Mercury BLAST," *Microprocessors and Microsystems*, 33(4):281-289, June 2009.