FPGA PROTOTYPE QUEUING MODULE FOR HIGH PERFORMANCE ATM SWITCHING

H. Duan, J. W. Lockwood, and S. M. Kang
University of Illinois at Urbana-Champaign
NSF Engineering Research Center for Compound Semiconductor Microelectronics and
Beckman Institute for Advanced Science and Technology
405 N. Mathews Ave., Urbana, IL 61801
Tel: 1-217-244-1565 Fax: 1-217-244-8371 Email: ipoint@ipoint.vlsi.uiuc.edu

Abstract — FPGA technology has been used for the development and implementation of a prototype input queuing module of the Illinois Pulsar-based Optical INTconnect (iPOINT) Asynchronous Transfer Mode (ATM) testbed. Pipeline techniques were extensively used to solve timing problems and increase throughput. This prototype queuing module has been fully tested for bandwidth of *100 Mbps*.

Introduction

In this paper, we present the FPGA implementation of an input queuing module of iPOINT testbed [1]. This queuing module has been successfully prototyped on a XC4005-5 FPGA device and fully tested for bandwidth of $100 Mbps$. The CAD tools used are XACT from Xilinx Inc. and Design Architecture model of Mentor Graphics version 8.2.

The iPOINT testbed consists of UNIX workstations connected via optical fibers to a central Pulsar switch. The Pulsar switch uses input queuing and has a word-wide shift-register ring fabric [2]. The core of the switch is prototyped using a XC4013-5 device [3], as illustrated in Figure 1. Four ports operate at *100 Mbps* and the trunk port operates at *400 Mbps*, providing an aggregate throughput of *800 Mbps*. The queuing module is placed at each input port of the switch.

Function and Logic of the Queuing Module

Besides handling the interface logic for the Pulsar switch, the queuing module executes the CRC check on the header of each incoming ATM cell, performs VPI/VCI translation, directs the switch which port a particular ATM cell should be sent to, and buffers the input traffic in case of output conflict. Contending cells are dropped when the queue is full. Opto-electronic and electro-optic conversions and clock recovery are done by the AMD Taxi board [4] [5].

The queuing module is implemented as two components: a FIFO chip to store the contending cells and a XC4005-5 FPGA chip to handle the queuing module control logic. Figure 2 shows the logic diagram of this queuing module. Its operation is described by the following pseudo code:

```
read_machine()
{ while (power is on)
   /* --- procedure 1, parallel to procedure 2 --- */
   { if !(entire header of ATM cell in fifo)
        set Idle = 0; /* indicate idle state to switch */
     else  { read header from fifo;
             /* pipeline stages 1-5: */
             latch the header bytes into pipeline;
             enable header CRC check circuit;
             Update = 0; /* block lookup table update */
             generate address word for lookup table;
             VPI/VCI translation;
             header CRC check and find syndrome;
             if (syndrome==0) { set Correct_cell=1;
                                form new VPI/VCI;
                                generate STATUS WORD; }
             else drop the wrong cell;
             Update = 1; /* enable table update */ }
     wait for SEND_DATA = 0;
     /* indicate switch can transmit this cell */
     while ((SEND_DATA = 0) && (not (49th byte))
        read data from fifo;
        /* 1 byte/cycle, totally 48 */ }
  /* ---  procedure 2, parallel to procedure 1 --- */
  { if (Set_Lookup==1)
       /* indicate look table need to be updated */
       { start three-stage pipeline procedure;
         if (Update==1) { generate table address;
                          put data on Date_Bus;  }
                          /* 1st stage */
         if (Update==1) enable Table_Write signal;
                          /* 2nd stage */
         if (Update==1) disable Table_Write signal;
                        reset address and data buses;
                          /* 3rd stage */  } } }
}

write_machine()
{ while(power is on)
   { if ((fifo can store entire ATM cell)
         && (new cell arrives))
      { if (first byte of an atm cell) set CELL_FLAG = 0;
        else  set CELL_FLAG = 1;  /* mark the 1st byte */
        latch ATM byte into pipeline;
        generate fifo write control signals;
        write ATM byte into fifo; }
     else drop this cell;
     wait for next cell;  }
}
```

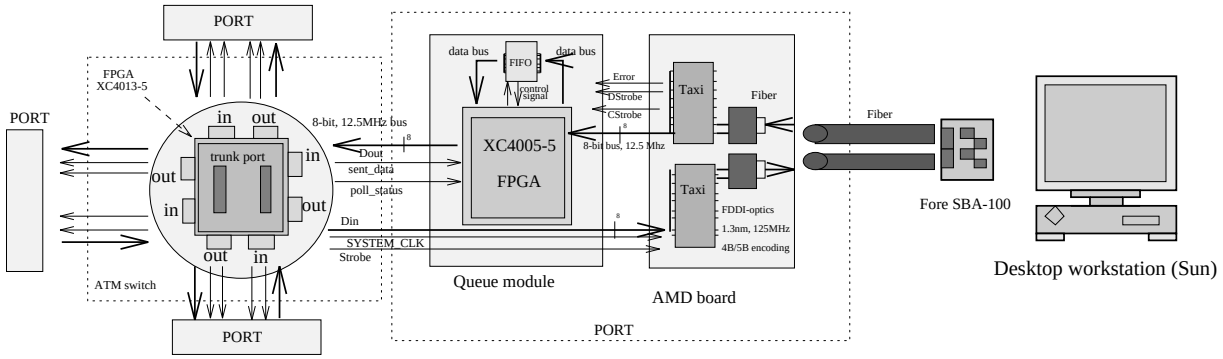


Figure 1: Queuing Module in the System

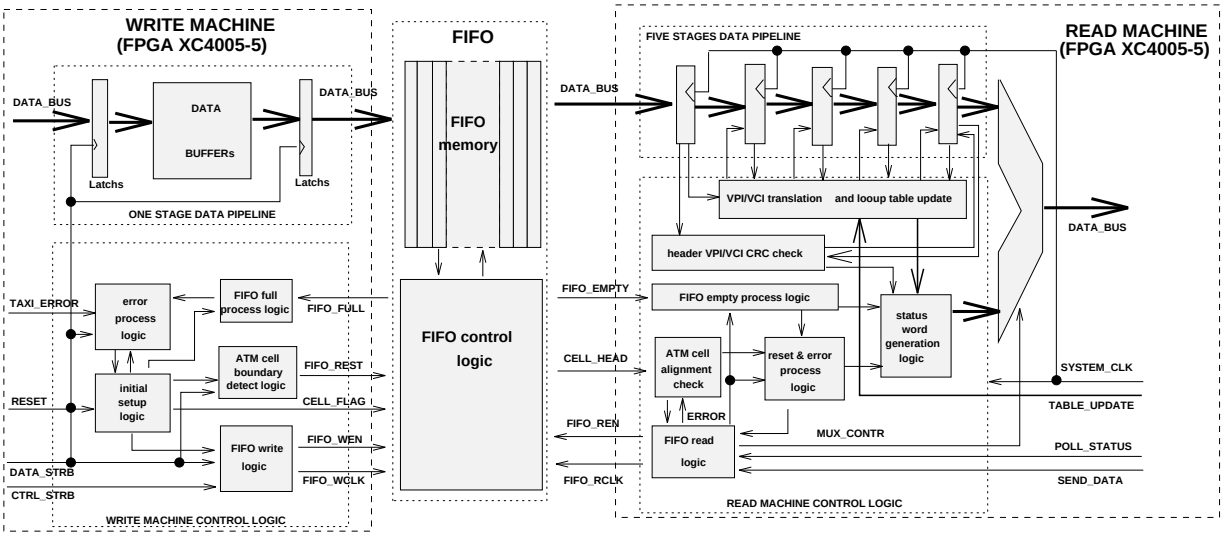


Figure 2: Queuing Module Logic Diagram

Pipelining for Timing Problems

The logic in the queuing module is inherently asynchronous, as the data transmit rate is controlled by a local oscillator on each source. The implementation of the queuing module requires a great deal of effort due to timing issues.

The characteristics of the FPGA device indicate that it is more suitable to prototype fully synchronous logic. The internal wiring structure of the FPGA device is such that a significant fraction of the delay is due to the number of PIP (Programmable Interconnect Point) through which a signal passes [6]. Therefore, the way to partition the design and route the connections determines the delay of signals and may introduce signal racing and skew. Because of the asynchronous clocks, the internal lookup table memory, and the external FIFO memory, the timing relations between the signals of the queue module are critical. Because of the incremental nature of this prototype design, it is important to generate a robust circuit, one that is insensitive to the randomizations of the placement and routing algorithms [7].

For prototyping the queuing module, rather than resorting to manual place and route, or excessively using the TIMESPEC attributes to guide the placement and routing procedure, we have solved the timing problems by extensive use of pipeline techniques.

Pipeline techniques are widely used in computer industry to make faster CPUs. They increase the CPU throughput by overlapping the execution of multiple instructions. These techniques can also be used in the FPGA applications. By breaking the complex combination logic blocks into several simple blocks and fitting them into a sequence of pipeline stages, the timing requirements are significantly simplified. Although the latencies of the signals are increased to some extent due to the overhead of each stage, the gain in the throughput and the resulting robustness of the circuit make pipeline techniques attractive.

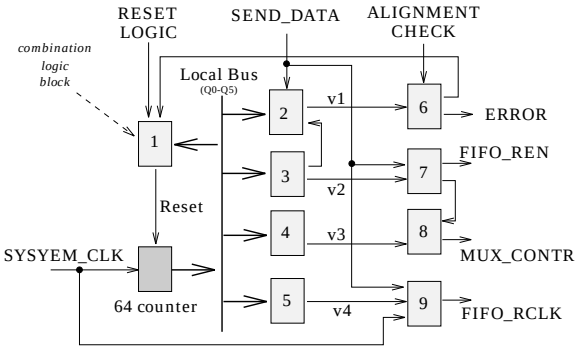


Figure 3: Example: Original Design

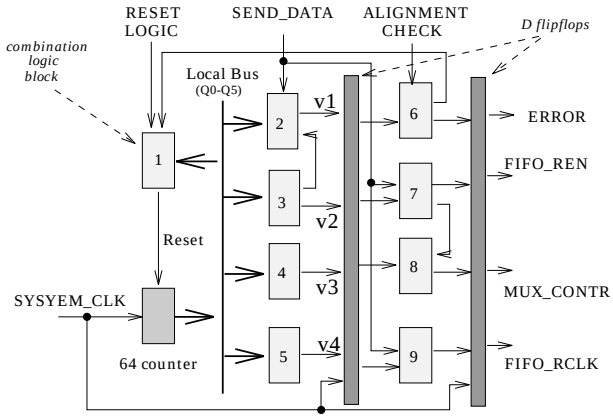


Figure 4: Example: Improved Design

As an example, consider the "FIFO read logic" in READ MACHINE (Figure 2), which generates read signals to the external FIFO memory. The correct operation of the circuit depends on the timing relations between the output signals: FIFO_RCLK, FIFO_REN, and MUX_CONTR. The initial design is shown in Figure 3. Although the Xilinx timing analysis predicted correct operation, the actual circuit failed to function properly.

Error tracing indicated that the malfunction originated from the considerable signal skew of $Q0$–$Q5$, which introduced glitches on V_1–V_4, then propagated to the output control signals. The timing relations between output signals varied due to the randomization of the placement and routing algorithms. The first attempt to solve the problem was to guide the placement and routing procedure using the methods listed in the Xilinx CAD tools [8]. Occasionally, we managed to get good routing result, and the queuing module worked properly. But most of time, the FPGA implementation failed.

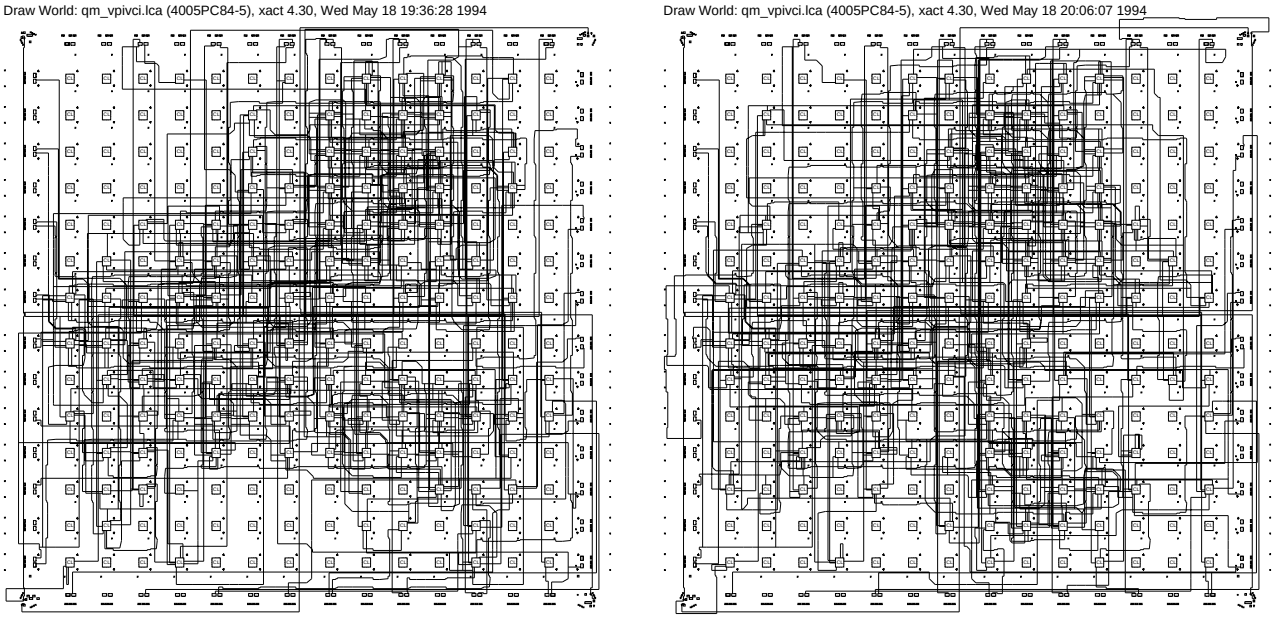
Figure 4 shows the improved design with pipelining. The combinational logic blocks are grouped into a two-stage pipeline. The timing relations between signals are guaranteed, and the glitches on V_1–V_4 will not influence the operation of the next stage. Additionally, the overlapping of the generation of output signals for two consecutive bytes increases the throughput of the circuit.

Pipeline techniques are extensively used in the queuing module implementation. VPI/VCI translation and ATM cell header CRC check are implemented using a five-stage pipeline (the header of ATM cell is five bytes long). The new VPI/VCI value is stored in a lookup table addressed by the current VPI/VCI. The lookup table is built by memory components within the FPGA device. The content of the table can be updated by an external switch controller. Memory access operations require that timing relations between signals are satisfied. For read access, the address bus must be stable before the read enable signal is issued and must be kept stable until a certain amount of time after the read enable is turned off. For VPI/VCI translation, signals on the address bus are generated by combining the outputs of the first four stages. The timing constrains can be easily satisfied because the signals on the address bus only change at a clock edge. This pipeline processes five bytes at different stages simultaneously, therefore the throughput can be increased by a factor of five.

A three-stage pipeline is used to handle lookup table update operation, which is a memory write access. For successful operation, both the address bus and the data bus must be stable before the write enable signal is valid and are turned off after the write signal is turned off. The timing requirements are satisfied by executing the operation in three clock cycles. Valid data and address are put onto and taken away from the corresponding buses at the first and the third clock cycle, respectively, and the write enable signal is only set to be valid during the second clock period. Two update operations instead of three can be overlapped within this pipeline due to the operation dependency of the updating logic. Therefore, the throughput can be doubled.

A pre-pipeline stage is added to this three-stage pipeline for the cooperation of the above two pipelines so that they will not conflict when accessing the lookup table.

In general, however, pipeline techniques can not solve the timing problems fully by themselves. Within each pipeline stage, it is important to use CAD tools judiciously to ensure that the signals can propagate within the pipeline period. For example, HMAP and FMAP macros can be used to reduce the delay by guiding the best merge for a given design, and TIMESPEC attributes can be used to specify timing constrains of groups of signals.



(a) routing result I

(b) routing result II

Figure 5: Working Routings

Extensive circuit tests show that the functionality of the improved circuit is no longer sensitive to the variation in placement and routing. Figure 5 illustrates two different working routings of identical pipelined queuing module circuit.

Conclusion

In conclusion, the maturity of FPGA industry has made it possible to significantly speed up the hardware design and prototyping of digital systems by programming the FPGA chips. However, due to the structure of its internal wiring, the FPGA device is more suitable for prototyping synchronous circuits. Signal timing becomes a major issue for asynchronous design and pipeline techniques are highly recommended. The asynchronous queuing module of iPOINT testbed has been successfully prototyped based on the method of pipelining.

References

[1] J. W. Lockwood, C. Cheong, S. Ho, B. Cox, S. M. Kang, S. G. Bishop, and R. H. Campbell, "The iPOINT Testbed for Optoelectronic ATM Networking," in Conference on Lasers and Electro–Optics, Baltimore, MD, pp. 370–371, 1993.

[2] J. Murakami, "Non-Blocking Packet Switching With Shift-Register Rings." Ph. D. Dissertation, University of Illinois at Urbana–Champaign, 1991.

[3] J. W. Lockwood, H. Duan, J. J. Morikuni, S. M. Kang, S. Akkineni, and R. H. Campbell, "Scalable Optoelectronic ATM Networks: The iPOINT Fully Functional Testbed," Unpublished.

[4] Advanced Micro Devices, Inc., TAXIchip Integrated Circuits, rev. 1.3 ed., Apr. 1991.

[5] Advanced Micro Devices, Inc., TAXIchip (DC–CAB) Data Checker Board User's Manual, 1991.

[6] S. Trimberger, "A Reprogrammable Gate Array and Applications," Proceedings of the IEEE, pp. 1030–1041, July 1993.

[7] R. Goering, "Users Seek Better FPGA Layout," Electronic Engineering Times, pp. 1, 35–36, Jan. 1993.

[8] Xilinx, Inc., XACT Mentor Version 8 Interface User Guide, May 1993.