Design and Implementation of Open-Source SATA III Core for Stratix V FPGAs

Sumedh Guha[1], Wen Wang[1], Shafeeq Ibraheem[1], Mahesh Balakrishnan[2], and Jakub Szefer[1]
[1]Dept. of Electrical Engineering and [2]Dept. of Computer Science
Yale University
{sumedh.guha, wen.wang.ww349, shafeeq.ibraheem, mahesh.balakrishnan, jakub.szefer}@yale.edu

***Abstract*—SATA is the de-facto standard computer interface that connects a host, typically a computing device, to a persistent storage device, such as a hard drive or solid-state drive. In order for FPGA-based designs to be able to leverage the variety of persistent storage devices, a SATA core is needed. Over time, the SATA standard has been revised to provide greater bandwidth, with SATA III being the newest version of the standard. In this paper, we are the first to present a SATA III core designed for Altera Stratix V FPGAs. Our implementation is written using Verilog, and tested using an industry-standard SATA protocol analyzer. We evaluate the performance of our SATA core by measuring the throughput of random and sequential read and write operations using various hard drives and solid-state drives. In addition, we compare the complexity of our SATA III core implementation with those of the older SATA I and II open-source implementations, and show that SATA III is still feasible, using only about 11% of Stratix V FPGA resources.**

I. Introduction

Solid-state drives (SSDs) and hard drives (HDs) offer increasing amounts of cheap, fast, persistent storage with the capacity now routinely on the order of 1TB or even more. To efficiently access such amounts of storage, greater bandwidths are needed to connect a computing device to the storage device. The de-facto standard for connecting computers with storage devices is the SATA protocol. Widely deployed and supported, SATA is an ideal solution if a FPGA-based design needs to interconnect with a storage device. Over the years, SATA standards have been updated, and prior FPGA SATA core designs are not usable with the latest SATA III standard.

In this work, we build on an existing SATA I and II open-source design [1] developed for Xilinx devices, and present a new SATA III core design targeting Altera Stratix V FPGAs. The main advantage of the SATA III interface over prior versions is the theoretical maximum bandwidth of 600MB/s, which is greater than the theoretical maximum bandwidths of the SATA I (150 MB/s) and SATA II (300 MB/s) interfaces. The increased bandwidth is achieved with faster clock rates for the serial SATA link (3000MHz) to the disk and the internal SATA III core parallel data (150MHz). To support SATA III, Altera Native PHY is used along with our Physical, Link, and Command Layers, as described in detail in the later sections.

The SATA III core presented makes a number of contributions and improvements:

- The first open-source FPGA SATA core design targeting Altera devices.
- A new SATA physical layer designed to work with Altera Stratix V FPGAs at SATA III speeds using the Stratix Native PHY IP as the basic building block.
- The link and command layers of the SATA core written in Verilog.
- Better performance with HDs and SSDs, compared to existing SATA I and II designs.

978-1-5090-5602-6/16/$31.00 © 2016 IEEE

II. SATA III Background

Serial ATA (SATA) is a storage interface for connecting computing devices to peripheral devices (e.g. hard drives, solid-state drives, optical drives). SATA is an improvement upon Parallel ATA (PATA), an interface standard of the 1980s. SATA features several advantages over PATA, including improved bandwidth, hot-swapping, greater ease of integration, and lower cost. As a result, SATA has become the standard for storage interfacing since its introduction in 2003.

SATA III is one of the most recent revisions of the SATA standard, and runs at raw 6Gb/s line rate, or up to 600MB/s data rate when accounting for 8b/10b encoding. By use of SATA III core, today's HDs and SSDs are able to achieve rates well beyond 300 MB/s. All SATA interface standards, including SATA III, follow the same five-layer architecture described below.

A. Application Layer

At the Application Layer, the user specifies the operation (e.g. sequential or random access, read or write), the sector address, and the number of sectors involved with the operation. All operations are in terms of one or multiple 512-byte sectors. Based on the operation, read or write enable signals are sent to trigger the Command Layer. Upon completion of an operation, a status message is read along with data from the device.

B. Command Layer

The Command Layer receives the operation parameters and determines the appropriate sequence of Frame Information Structures (FIS) to be sent. A finite-state machine (FSM) accounts for the transitions between the different FIS transmissions. The FIS types include Register Transfer Host to Device, Register Transfer Device to Host, DMA Activate, and Data. Once it has decided on the FIS type to send, the Command Layer passes the FIS information to the Transport Layer.

C. Transport Layer

The Transport Layer constructs, and deconstructs, FISes sent from, and to, the Command Layer. The Transport Layer follows the ATA protocol for preparing the FIS payload for transmission and for extracting status information from received packets. If there are any errors, the Transport Layer requests a retransmission.

TABLE I
COMPARISON OF RESOURCE CONSUMPTION OF EXISTING FPGA IMPLEMENTATIONS OF SATA, ALONG WITH OUR NEW DESIGN.

Design	Open-Source	SATA II	SATA III	Brand	Model	Slices	BRAM	LUT	F/Fs
UMASS [1]	✓	✓		Xilinx	Virtex-4	5128	7	– [a]	– [a]
UNC [2]	✓	✓		Xilinx	Virtex-5	576	3	1282	986
Groundhog [3]	✓	✓		Xilinx	Virtex-5	652	0	1537	763
UNC [2]	✓	✓		Xilinx	Virtex-6	570	3	1334	894
Design Gateway [4]			✓	Xilinx	Virtex-7	476	2	1024	863
IntelliProp [5]			✓	Altera	Stratix V	– [a]	– [a]	2224	– [a]
Our	✓		✓	Altera	Stratix V	933[b]	8[b]	3266[b]	2073[b]

[a] This resource usage was not specified in the cited reference.
[b] For our design, resources used were calculated using conversion from Altera to Xilinx resources: 1 Slice = 2 ALM, 1 BRAM = 1 M20K, 1 LUT = 1 ALUT, and 1 F/F = 1 Reg.

D. Link Layer

The Link Layer takes care of framing and delivering each FIS. It uses primitives to mark the boundaries of the FIS. Primitives are also used for managing handshaking between the FPGA and the device. The Link Layer also computes a Cyclic Redundancy Check (CRC) using the data and appends the CRC to the end of the payload when sending FISes. Finally, the Link Layer scrambles the frame information by XORing the frame information with the output of a linear feedback shift register (LFSR). Scrambling is performed to prevent Electromagnetic Interference from corrupting the frames. The Link Layer uses the same CRC and scrambling mechanisms to check for errors and to descramble the incoming packets. If there are any errors, the Link Layer signals the Transport Layer to request for retransmission. The Link Layer also regulates frame transmission by checking buffer underflow and overflow for FISes.

E. Physical Layer

The Physical Layer serializes/deserializes frames received from the Link Layer and encodes/decodes them using the 8b/10b encoding/decoding scheme before transmission. The Physical Layer is also involved with Out-of-Band (OOB) signaling used to establish the physical link and to negotiate the data transmission speeds.

III. RELATED WORK

Researchers in the high-performance computing field have taken interest in combining FPGAs with nonvolatile storage devices. As an industry standard, SATA provides the perfect interface to use for accessing storage devices. Currently, there exist several implementations of SATA for FPGA devices.

One of the first open-source SATA implementations was Groundhog [3], a SATA host bus adapter (HBA) for Xilinx Virtex-5 FPGAs. Groundhog also supports native command queueing, an optimization for read and write command ordering introduced with SATA II.

Later, an open-source SATA II core was developed at the University of North Carolina at Charlotte (UNC) [2]. This SATA-core was designed for Virtex-6 devices and the ML605 board. The UNC group also added a DMA engine, bus interface, and a Linux block device driver to make the core available to the operating system.

A follow-on work by University of Massachusetts Amherst (UMASS) built off of the UNC design to create a SATA core for Virtex-4 devices, running at both SATA I and SATA II speeds. The UMASS core also features a replay buffer for retransmitting data FISes, a SATA Event Logger for transfering debugging information, and a new physical layer for handling the RocketIO MGT on the Virtex-4. [1].

Among commercial vendors, IntelliProp [5] and Design Gateway [4] are some vendors offering SATA designs. The commercial designs have resource usage slightly better than the open-source designs and are typically available for Xilinx and Altera Devices. Major downside of using the closed-source designs for research projects are the high license costs for the IP cores and no flexibility to adapt the code.

In our work, we design and implement the first open-source SATA III core compatible with Altera's Stratix V FPGAs, targeting the DE5-Net board. Table I presents details of the prior SATA cores and is used to compare them with our work. Details of our design and implementation are presented next.

IV. DESIGN AND IMPLEMENTATION

The design of the SATA III core follows the layered design of SATA and the previous open-source project that it builds upon [1]. Figure 1 shows a high-level block diagram of the design. The right-hand side of the figure shows the SATA layers and how they correspond to our design. In particular, our Test Layer and Core Layer together correspond to the SATA Application Layer. Meanwhile, our Link Layer corresponds to the SATA Transport and Link Layers.

The top design unit is the Test Layer. It is used to trigger read and write tests, as well as to set options for the operations (sequential or random access). The Test Layer uses a random number generator based on a LFSR as a source of randomness for generating addresses for random accesses. The Test Layer communicates with the Core Layer, which is used to buffer data going to the disk. The core layer has `user_fifo`, a fifo wherein each sector of data to be written is stored. The data from `user_fifo` is passed to the Command Layer. The Command Layer waits for one sector of data (512 bytes) to be ready into `user_fifo` before issuing commands to the Link Layer. The Command Layer is responsible for generating FIS frame headers and data to be sent over to the Link Layer by buffering them in the `write_fifo`. The Link Layer in turn reads FIS headers and data from `write_fifo`, appends CRC data, and performs scrambling to create the FIS packet. The FIS is buffered in the transmitter fifo, `tx_fifo`, and read by the Physical Layer. Inside the Physical Layer, the FIS passes through an OOB submodule. OOB signaling is used during the beginning phase of transmission when the FPGA sends Out-of-Band signals to establish a link with the disk. After initialization, the OOB submodule passes the FIS to the

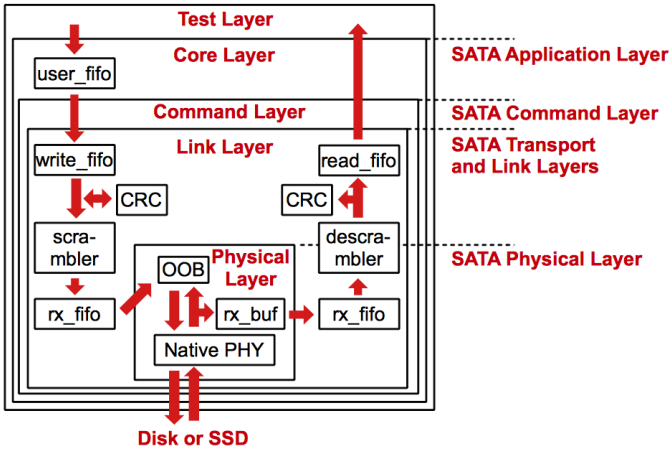

Fig. 1. Block diagram showing major components, buffers and FIFOs, and data flow between components. Control signals and state machines are not shown. The right-hand side shows the SATA layers corresponding to the modules in our design.

Fig. 2. Experimental setup with Sierra M6-1 [6] analyzer used to interpose on FPGA to disk SATA traffic for debugging and gathering measurements.

Native PHY. The Native PHY is responsible for serializing, encoding, and sending the data over the physical SATA wires to the HD or SSD.

In the reverse direction, the Native PHY deserializes the incoming FIS packet, decodes the FIS packet, and passes it to a receiver buffer, `rx_buf`. The receiver buffer is used to compensate for differences in the receive clock and the clock used for operation of the rest of the SATA III core. The `rx_buf` passes the FIS to a receiver FIFO, `rx_fifo`. The FIS from `rx_fifo` is descrambled and passed to a read fifo, `read_fifo`. Simultaneously, the FIS's CRC is processed to check for any receive errors. The `read_fifo` is directly accessible from the Test Layer for reading out the contents of the data received from the disk.

The IP (intellectual property) modules from Altera used in this design are the Native PHY as well as the buffers and FIFOs. They are configured and generated using Altera tools.

V. Evaluation

The design was tested using Altera DE5-Net FPGA with the help of a Sierra M6-1 [6] SATA analyzer. Figure 2 shows a diagram of the setup. The Sierra analyzer is on one side connected to the FPGA via the SATA cable, and on the other side it is connected to the HD or SSD via another SATA cable. The analyzer was used to confirm that the FPGA generated FISes follow the SATA III standard, to detect and to debug protocol errors, and to gather the measurements presented in this paper. All numbers presented are median values of multiple measurements. Due to space limitation, graphs are presented for only one solid-state drive and one hard drive from among multiple ones tested for this project.

A. Throughput, Excluding Drive Latency

Figures 3 and 4 show the throughput for sequential and random reads and writes for the HD and SSD. The is raw throughput excluding drive latency, i.e. only consider the time for actual transfer of the data on the wire. As a reference, the red horizontal line at the top of each graph shows the theoretical maximum throughput for SATA III, which is 600MB/s. For sequential reads and writes, the throughput increases as the number of sectors per transaction increases to 16. This conforms to the fact that a Data FIS can contain at most 16 sectors of data, thus incurring minimal overhead and maximizing performance at that point.

For random reads and writes, the throughput stays about constant, at the same level as the sequential read or write for 1 sector. This is consistent with the fact that every Data FIS can specify one starting address and the size of the request. Thus, for a request that reads or writes N random sectors, the request has to be broken down into N separate 1 sector Data FISes, each with its separate (random) address.

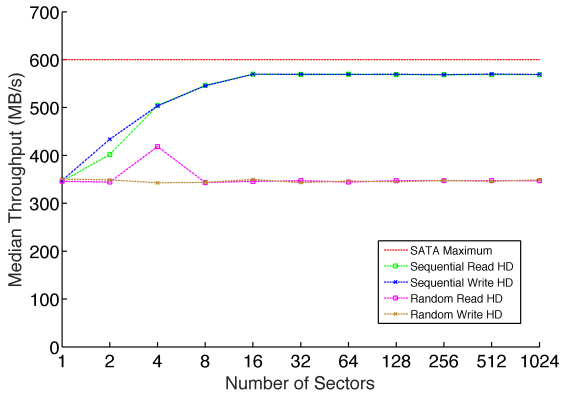

Fig. 3. SATA III core sequential and random throughput, excluding drive latency, for HD with 512-byte sectors.

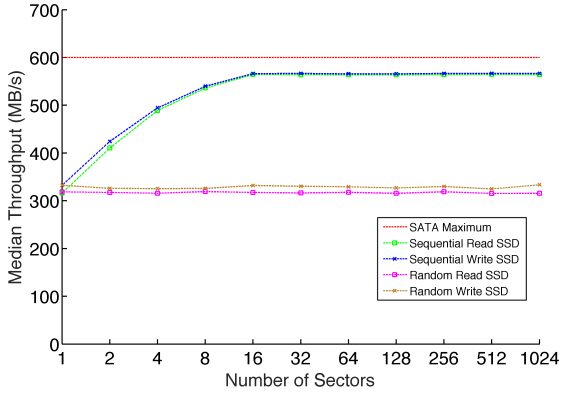

Fig. 4. SATA III core sequential and random throughput, excluding drive latency, for SSD with 512-byte sectors.

B. Throughput, Including Drive Latency

Throughput, excluding drive latency, was used to show that the FPGA operates correctly and can almost reach the theoretical maximum throughput. This, however, will not be realistic performance in real applications. Consequently, Figures 5 and 6 show the throughput including drive latency.

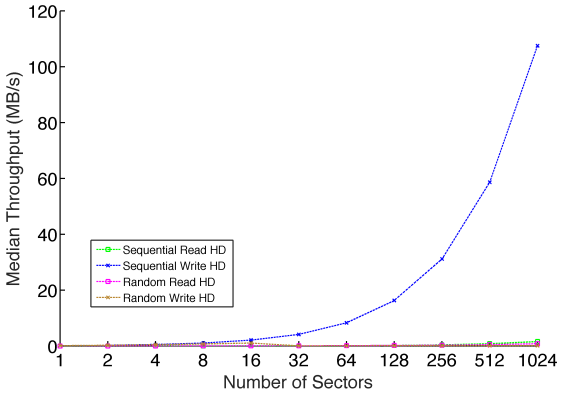

Fig. 5. SATA III core sequential and random throughput, including drive latency, for HD with 512-byte sectors.

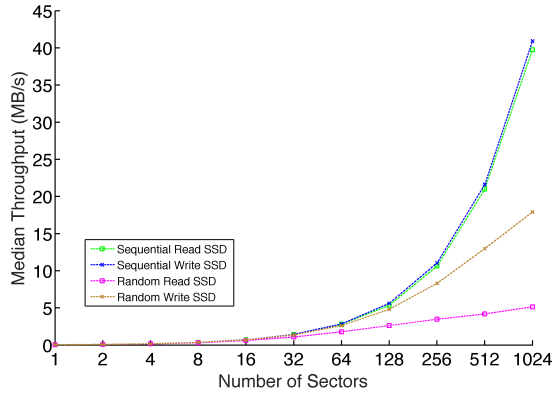

Fig. 6. SATA III core sequential and random throughput, including drive latency, for SSD with 512-byte sectors.

Here, the throughput is calculated by dividing the data size by total time of request, from when initial command is sent to drive, to when all of data is returned. This can be more easily compared to end-to-end throughput numbers reported for storage systems. As the number of sectors per request increases, the throughput increases. Notably, the HD write throughput is quite high, as data gets placed in the drive's buffer (and only later it is pushed to the slow physically rotating disk platers). For SSD, the sequential performance is better than random accessed, again, conforming to expectation.

C. Resource Usage

Table I showed existing FPGA-based SATA implementations and the resource consumption for each, along with our design. Direct comparison is not possible here as most designs use different FPGA chips from different vendors. However, with our calculated conversion from Altera to Xilinx resources, some comparison is possible and the complexity of the designs can be approximately compared. Our design consumes only about 11% of the Stratix V resources, even though it supports the latest SATA III protocol. The design leaves most of the FPGA chip free for other logic or for instantiating multiple SATA III cores.

D. Code Complexity

Table II shows the size of the codebase developed for this design. Altera IP code is omitted. Overall, over 2500 lines of Verilog were written for this project.

TABLE II
LINES OF CODE REQUIRED FOR EACH MODULE, COMPUTED USING THE `cloc` [7] PROGRAM.

Module	Implementation	Lines of Code
Test Layer	Verilog	498
Core Layer	Verilog	264
Command Layer	Verilog	470
Link Layer	Verilog	990
Physical Layer	BDF	–[a]
OOB	Verilog	336
CRC	Verilog	103
(de)scrambler	Verilog	103

[a] BDF (Block Design File) in Altera Quartus is a graphical format for schematic specification of a design, thus lines of code is not specified.

VI. CONCLUSIONS

This paper presented the design and implementation of the first open-source SATA III core on Stratix V FPGAs. Building on existing open-source code for SATA I and II, the new core is able to support the newest hard drives and solid-state drives with the SATA III protocol. Our implementation was written using Verilog and tested using an industry-standard Sierra M6-1 SATA protocol analyzer. The analyzer showed correct operation of the SATA III protocol using our design, and was used to gather performance results showing read and write data rates up to 600MB/s. The final design used only about 11% of Stratix V FPGA resources, leaving most of the chip for other logic so that the SATA III core can serve as a building block for other, bigger designs that integrate FPGA and persistent storage.

A. Code Availability

The source code for the presented design will be made available at http://caslab.eng.yale.edu/code/sata.

ACKNOWLEDGMENT

We would like to thank Altera for the donation of the DE5-Net FPGA boards used in the experiments and the Quartus software licenses. We are also thankful to Altera support for their assistance in understanding the Native PHY, as well as Prof. Fengnian Xia and his group for use of their high-speed oscilloscope in debugging our design.

REFERENCES

[1] C. Gorman, P. Siqueira, and R. Tessier, "An Open-Source SATA Core for Virtex-4 FPGAs," in *International Conference on Field-Programmable Technology (FPT)*, 2013, pp. 454–457.
[2] A. A. Mendon, B. Huang, and R. Sass, "A High Performance, Open Source SATA2 Core," in *International Conference on Field Programmable Logic and Applications (FPL)*, 2012, pp. 421–428.
[3] L. Woods and K. Eguro, "Groundhog - A Serial ATA Host Bus Adapter (HBA) for FPGAs," in *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2012, pp. 220–223.
[4] "Design Gateway, SATA IP Transport & Link Layer Core," http://www.dgway.com/products/IP/SATA-IP/dg_sata_ip_data_sheet_7series_en.pdf, accessed July 5, 2016.
[5] "IntelliProp, PC-SA101A-HI SATA Host App Core," http://intelliprop.com/cores-bridge-board-datasheets.htm, accessed July 5, 2016.
[6] "Teledyne LeCroy, Sierra M6-1 SATA Protocol Test System," http://teledynelecroy.com/protocolanalyzer/protocoloverview.aspx?seriesid=279, accessed July 5, 2016.
[7] A. Danial, "Count Lines of Code," https://github.com/AlDanial/cloc, accessed July 5, 2016.