

DEVELOPMENT OF IP CORE FOR SATA COMMUNICATION ON FPGA

**THIS THESIS IS SUBMITTED IN THE PARTIAL
FULLFILLMENT OF THE REQUIREMNT FOR THE DEGREE
OF BACHELOR OF TECHNOLOGY**

IN

ELECTRONICS AND INSTRUMENTATION ENGINEERING

BY

SAMPAD MOHAPATRA (Roll No. 109EI0049)

&

ROHIT NANDAN (Roll No. 109EI0331)



NATIONAL INSTITUTE OF TECHNOLOGY, ROURKELA



NATIONAL INSTITUTE OF TECHNOLOGY ROURKELA

CERTIFICATE

This is to certify that the thesis entitled “**DEVELOPMENT OF IP CORE FOR SATA COMMUNICATION ON FPGA**” submitted by Sampad Mohapatra (109EI0049) and Rohit Nandan (109EI0331) in partial fulfillment of the requirements for the award of BACHELOR OF TECHNOLOGY Degree in Electronics and Instrumentation Engineering at the National Institute of Technology, Rourkela (Deemed University) is the project work carried out by him under my supervision and guidance.

To the best of my knowledge, this thesis has not been submitted to any other University/ Institute for the award of any degree or diploma by Mr Mohapatra and Mr Nandan.

Date: 13th May, 2013

Prof. Debiprasad Priyabrata Acharaya
Department of Electronics and
Communication Engineering,
National Institute of Technology Rourkela,
Rourkela - 769008

ACKNOWLEDGEMENT

We avail this opportunity to express my indebtedness to our guide Prof. D. P. Acharaya. Department of Department of Electronics and Communication Engineering, National Institute of Technology, Rourkela, for his valuable guidance, constant encouragement and kind help at various stages for the execution of this dissertation work.

We also express our sincere gratitude to Prof. S. K. Meher, Head of the Department, Department of Electronics and Communication Engineering for allowing access to valuable facilities in the department.

We thank our parents for all the encouragement they gave us. And last but not the least; we thank all those friends who helped us in the course of this entire dissertation work.

Sampad Mohapatra Roll No: 109EI0049

Rohit Nandan Roll No: 109EI0331

Department of Electronics and Communication Engineering,

National Institute Of Technology, Rourkela

Rourkela-769008

Abstract

SATA is a fast evolving bus standard capable of transferring data to and from hard disk drives, SSDs and optical drives. It is at the heart of most computer systems today. SATA is a newer technology than ATA, while maintaining software compatibility with it. The SATA has a vast number of advantages over ATA. A detailed study on the working of SATA protocol has been done. Also parallel CRC and scrambler generator have been designed.

TABLE of CONTENTS

1. Introduction	1
1.1 Serial ATA	2
1.2 Intellectual Property (IP) core	3
2. Literature Review	4
2.1 ATA Standard	5
2.2 Motivation behind SATA	6
2.3 SATA architecture	8
2.4 Brief Description of SATA layers	9
2.4.1 Application Layer	9
2.4.2 Command Layer	11
2.4.3 Transport Layer	11
2.4.4 Link Layer	14
2.4.5 Physical Layer	15
2.5 Cyclic Redundancy Check (CRC)	17
2.6 Scrambler	18

3. Design	20
3.1 Steps taken to create a parallel CRC generator	21
3.2 Steps taken to create a parallel scrambler generator	23
4. Simulation and Result	25
4.1 CRC simulation	26
4.2 CRC result	26
4.3 Scrambler simulation	28
4.4 Scrambler result	28
5. Conclusion	
References	

TABLE of FIGURES

Fig. 2.1	SATA architecture blocks	8
Fig. 2.2	Application Layer provides SATA programming interface.	9
Fig 2.3	Transport layers create and decode each FIS	12
Fig. 2.4	Formats and Contents of the Register FIS-Host to Device	13
Fig. 2.5	Register FIS-Host to Device showing the field definition	13
Fig 2.6	Link layer manages link protocol via primitive generation or reception.	15
Fig 2.7	SOF primitive	15
Fig. 2.8	OOB signal	17
Fig 3.1	Parallel CRC block	22
Fig 3.2	Parallel Scrambler Block	24
Fig 4.1	Testbench waveform of CRC block	27
Fig 4.2	Testbench waveform of CRC block	29

1. INTRODUCTION

1.1 Serial ATA

Serial ATA (SATA) is a bus interface that bridges host bus adapters to storage devices like hard drives and optical drives. It is advancement to parallel ATA. SATA was designed for software compatibility with the ATA/ATAPI 6 specification. The advantages of migrating to SATA are as follows [1]:

- Lower pin count
- Higher performance (speed)
- Better cables and connectors
- Reliable performance
- Support for low voltage
- Reduced pin count leading to less reduction
- Lower voltages for smaller silicon sizes
- High transmission speeds between drive and host bus adapter
- Increased reliability
- Better cables and connectors

1.2 Intellectual Property (IP) core

It is a reusable piece of logic (hardware or software) that is the intellectual property of an individual, a group of individuals or a company. It can be licensed to others. IP cores can be used in ASICs or FPGA based designs.

2. Literature Review

2.1 ATA Standard

In Compaq computers, hard disk drives were initially connected to computers via a hard disk controller card plugged into PC expansion bus. This changed as the hard disk controller was placed on the drive itself and the necessity for an expansion card was eliminated. The PC-AT expansion bus was extended to the drive through a cable and an interface circuit called Host Bus Adapter. Other manufacturers began making drives similar to Compaq's design. These drives were called IDE drives. But due to lack of standard compatibility was a big issue.

A group was formed to develop a standard interface between IBM PC-AT and IDE drives. This specification was called ATA (Advanced Technology Attachment).

The original ATA bus extended onto the IBM PC-AT, exhibiting similar performance. But revisions of the specifications allowed better performance on the same ATA interface. They are as follows:

1. ATA: Supported two modes of operation. :
 - PIO (Programmed IO): CPU executes instructions which are responsible for data transfer between ATA device and memory.
 - DMA (Direct Memory Access)

The data transfer rates was at 5 MB/s

2. ATA-2/3

Bus transfer rate remained at 8 MHz but two bytes of data could be transferred in each clock cycle. This resulted in a data rate of 16 MB/s.

3. ATA-4

The transfer rate was increased to 16.5 MHz resulting in data rate of 33 MB/s.

4. ATA-5

Again the transfer rate was increased to 33 MHz resulting in 66 MB/s data rate.

For reliable operation of the no. of conductors were increased from 40 to 80.

5. ATA-6

ATA Host adapters and drives that supported the UltraDMA 100 multiplied the 33 MHz clock used in ATA-5 by a factor of 1.5 to get a transfer rate of 49.99 MHz achieving a throughput of approx. 100 MB/s.

6. ATA-7

UltraDMA 133 used DDR (Double Data Rate) to achieve an effective rate of 66 MHz with a maximum throughput of 133 MB/s.

2.2 Motivation behind SATA

The migration from Parallel ATA to Serial ATA was due the following compelling reasons:

- Lower pin count
- Higher Performance
- Better cables and connectors

- Greater reliability
- Low voltages

1. Low pin count: Parallel ATA has a large number of pins. But presently only 26 of them carry signals. They result in costly cables and chip. Also they take more space and cause heating issues
2. Higher performance: PATA provides for maximum transfer rates of 133 MB/s
But SATA operates at 1.5 Gbp/s with maximum transfer rate of 150 MB/s (Generation -1) and 300 MB/s (Generation -2).
3. Better cables and connectors: PATA cables have high pin count, limited length (18 inches), require more space. The pin connections can bend easily on mishandling. PATA doesn't support hot plugging. While SATA cables have only 7 conductors, maximum cable length of 1 m, smaller in size, cost is cheaper, have contacts rather than pins and allow hot plugging.
4. Greater Reliability: In PATA only UltraDMA operating at 33 MHz and higher support CRC (Cyclic Redundancy Check) check but only during transmission of DATA.

In contrast every packet of data sent across SATA bus has a CRC. Nearly all errors are detected by CRC.

5. Lower voltages: PATA operated at 5 V but SATA works at lower voltage of 0.7 V.

2.3 SATA architecture

SATA architecture consists of five layers:

- Application
- Command
- Transport
- Data Link
- Physical

These layers are shown in fig. 2.1. The host in this case is an FPGA and the Device is a hard drive.

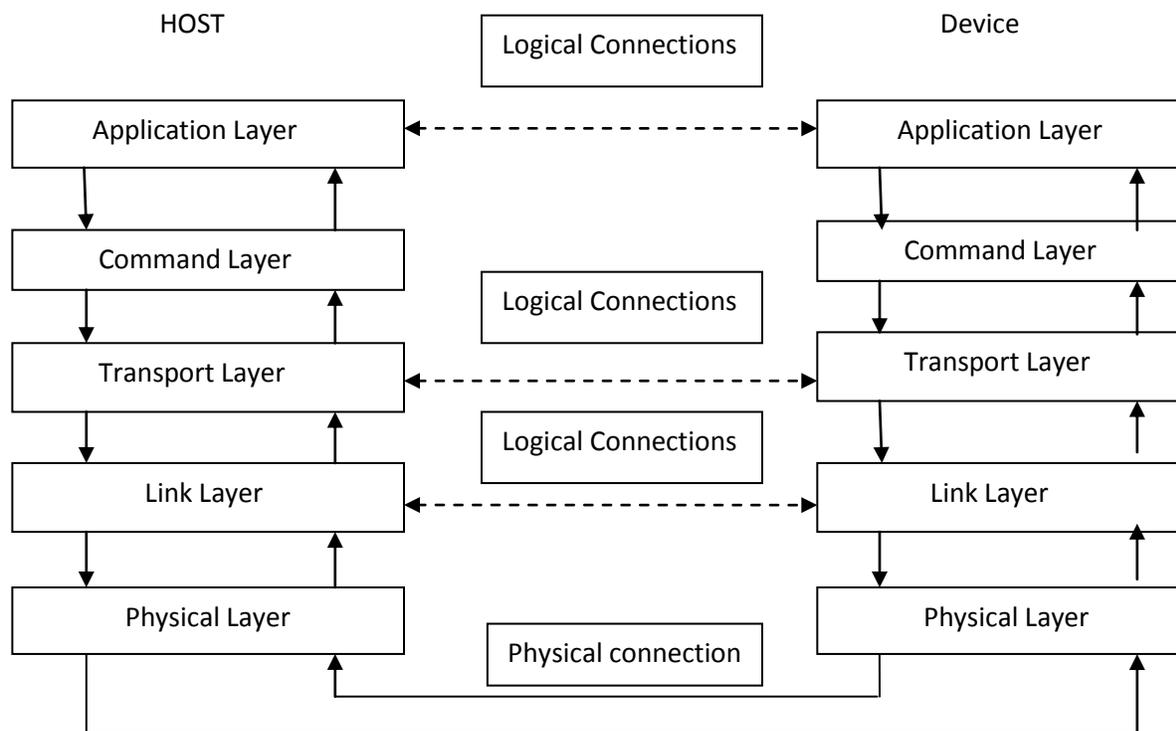


Fig. 2.1 SATA architecture blocks

2.4 Brief Description of SATA layers

2.4.1 Application Layer

Communications across the SATA interface consist primarily of transferring FISes (Frame Information Structures). A FIS can be delivered from HBA to the drive or from drive to HBA, and it may be delivered from the HBA to the drive, and it may contain shadow or ATA register contents, data, control information etc. In general, a FIS originates at the Application layer and is received by the Application layer on the opposite end of the link. Fig. 2.2 shows Application layer at the top of the hierarchy for both the HBA and SATA drive.

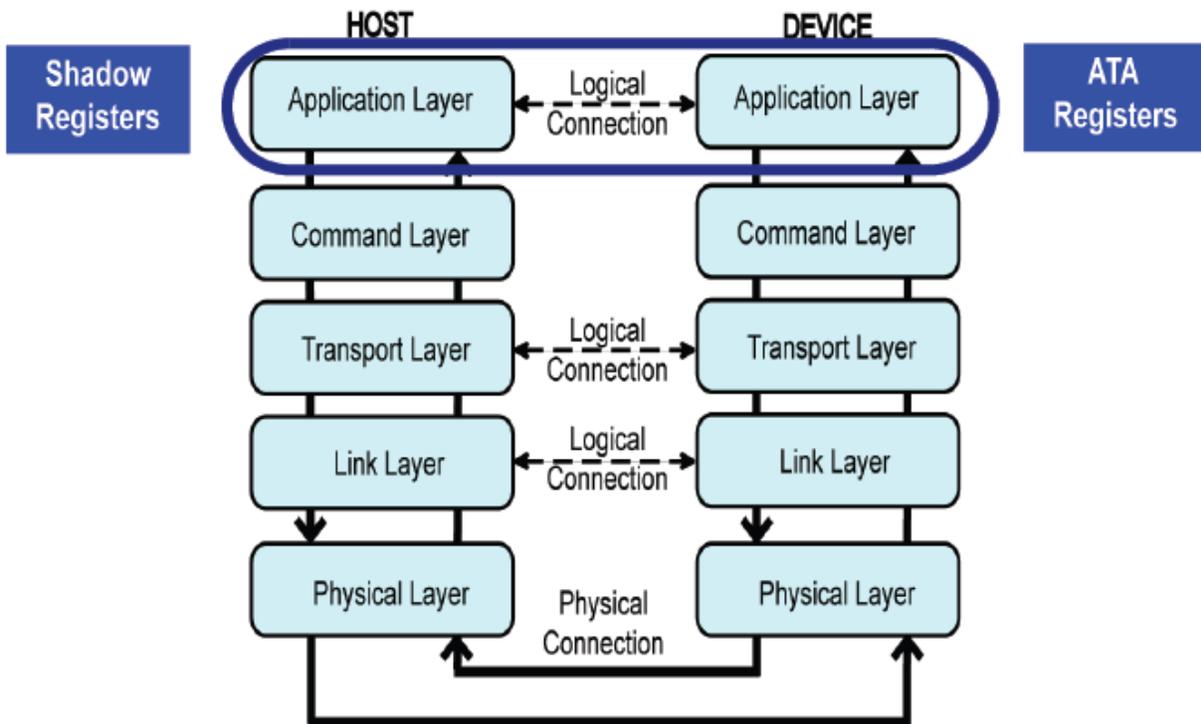


Fig. 2.2 Application Layer provides SATA programming interface.

Registers within SATA host adapter must be initialized by host software when a command is to be performed. Depending on the command being issued by software, two groups of registers may need to be configured:

- DMA Registers (when a DMA command being is being issued)
- Legacy ATA Registers

In SATA a copy of the legacy ATA registers, called “shadow registers” is kept in the Host Adapter. Host software initializes the shadow registers as require for the particular command to be issued. Software then issues the command by writing its corresponding value to the command register, which is also part of the shadow registers. This causes the SAT Application layer to forward the shadow register contents to the Transport layer. This information must be forwarded to the drive where the actual ATA registers reside within the drive’s Application layer. The logical communication shown between the application layers of the host and device promotes the concept of information being passed between the shadow registers and the ATA registers.

When the SATA Drive receives FIS containing a new command, it processes the command, and once the command has completed, it forwards the contents of its ATA registers to the Shadow registers within the HBA’s Application layer to report final status.

During command processing the HBA and the drive may exchange a number of FISs. When a FIS is received the contents of one or more registers within the Application layer are generally changed. In this way, the HBA and drive communicate and exchange information as a command executed.

2.4.2 Command Layer

When commands are delivered and executed in a SATA implementation, two or more FISs are typically exchanged between host and drive. The type and sequence of FISs exchanged is a function of the command being performed. The command layer is responsible for managing the high-level protocol, by verifying receipt of the expected FIS and by issuing any subsequent FIS to be returned in response. The command layer state machine defines the appropriate action closely coupled and one could consider the Command layer as being part of the Application layer functionality.

2.4.3 Transport Layer

Requests to transmit a FIS may come from the Application layer or from the command layer. In either case, the Transport layer is responsible for creating a compliant FIS. Fig. 2.3 depicts the Application layer sending shadow register contents to the Transport layer to initiate a command. The FIS created by the Transport layer is called a Register FIS- Host to Device because it contains the contents of the shadow registers that must be forwarded to the drive.

FISs are stored in small buffers within the transport layer. This buffering supports retries of most FISs in the event of transmission errors. Note the logical connection between the

Transport layers in fig. 2.3. This connection represents the movement of a FIS from a transmit buffer in the transaction layer at the opposite side. Note also that the Transport layers use a flow control mechanism to ensure these buffers do not overflow or underflow.

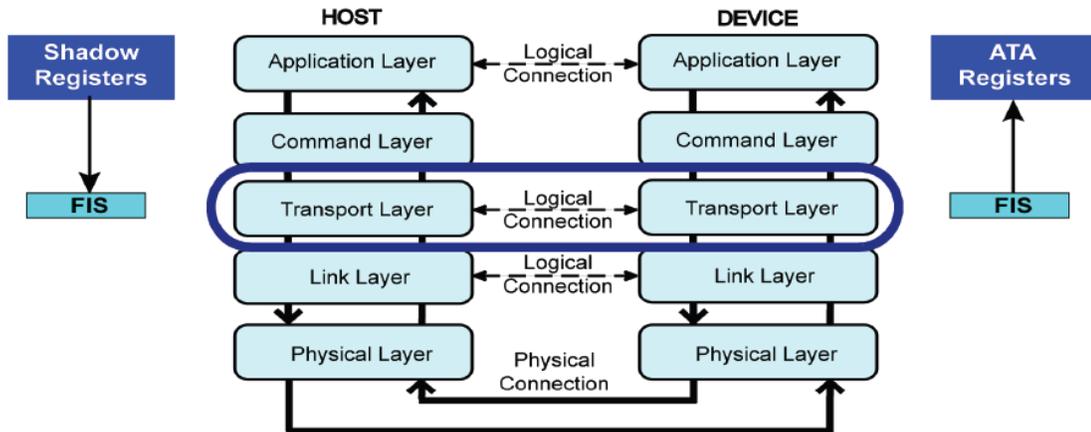


Fig 2.3 Transport layers create and decode each FIS

The formats and contents of the 20 byte Register FIS is illustrated in Fig 2.4. The FIS ID is 27h, several byte and bit fields are reserved and the other fields contain the contents of the ATA command and Control register fields. The fields in Fig 2.4 contain the legacy register names for backward references as done in the 1.0a specification. Fig. 2.5 shows the contents of the same Register FIS with the LBA labels, which are used in the later versions of the SATA specifications.

	+3	+2	+1	+0
	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
DW 0	Features	Command	C R R Reserved	FIS Type (27h)
DW 1	Dev/Head	Cyl High	Cyl Low	Sector Number
DW 2	Features (exp)	Cyl High (exp)	Cyl Low (exp)	Sec Num (exp)
DW 3	Control	Reserved (0)	Sec Count (exp)	Sector Count
DW 4	Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)

Fig. 2.4 Formats and Contents of the Register FIS-Host to Device

	+3	+2	+1	+0
	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
DW 0	Features	Command	C R R Reserved	FIS Type (27h)
DW 1	Device	LBA High	LBA Middle	LBA Low
DW 2	Features (exp)	LBA High (exp)	LBA Mid (exp)	LBA Low (exp)
DW 3	Control	Reserved (0)	Sec Count (exp)	Sector Count
DW 4	Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)

Fig. 2.5 Register FIS-Host to Device showing the field definition

2.4.4 Link Layer

This layer manages the FIS transmission protocol. A major part of this protocol is generating and decoding small packets called primitives. These primitives are employed in a variety of ways to facilitate link transmission protocol, including:

- Indicate end and beginning of each FIS
- Bus arbitration
- FIS transmission status
- Flow control
- Link power management
- Clock compensation

Fig 2.6 depicts the transmission of a packet from host to drive. The FIS shown adjacent to the link layers includes Start and End primitives that inform the receiver of packet boundaries. Each primitive consists of a 4 byte packet of information that is generated within Link layer and is decoded by the receiver's Link layer.

The detector detects these primitives and removes them from the FIS at the Link layer.

Fig 2.7 denotes the presents the format of a SOF primitive that is placed at the beginning of an FIS. The values shown are 10 bit encoded values.

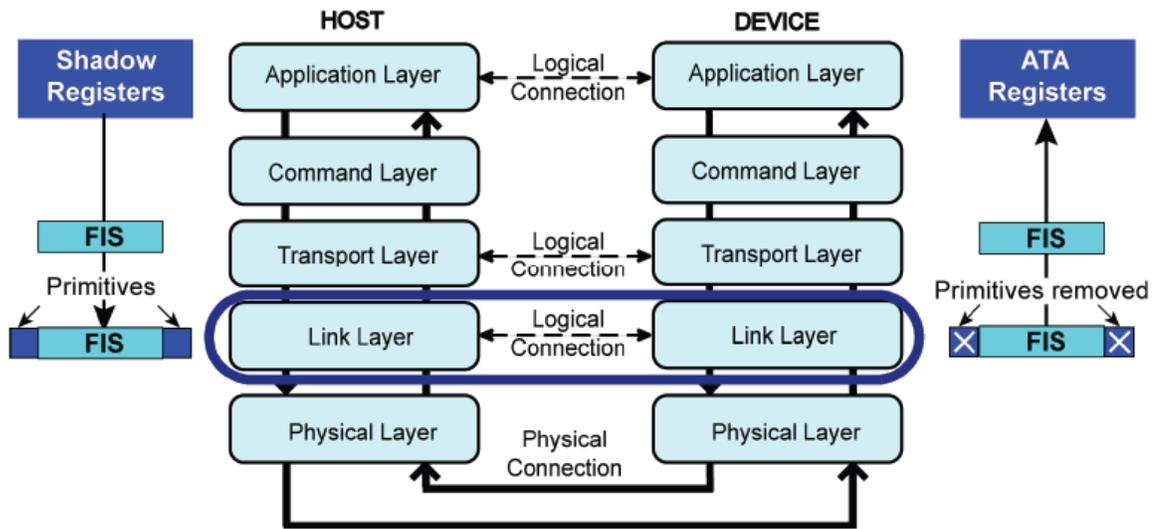


Fig 2.6 Link layer manages link protocol via primitive generation or reception.

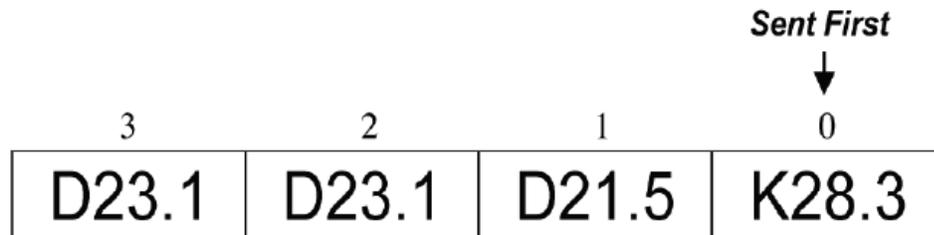


Fig 2.7 SOF primitive

2.4.5 Physical Layer

This layer resides hierarchically at the bottom of the SATA layer architecture and performs two general roles:

- Establishing link communication following reset
- Transmit and receive FIS and primitive traffic

After reset, the hardware in devices on both ends of a link automatically begins the process of detecting whether another device is present and at what speed the interface can run. The initialization takes place using OOB (Out Of Band) signaling even before the devices are able to recognize dwords as information from the transmitter. OOB uses the same signal path as every other bit from the transmitter, meaning they are not side-band signals, but occurs before the link is trained. Following OOB signal the link is trained or synchronized to the incoming bit streams. When link initialization is completed the target device delivers a FIS to the host to identify itself.

Fig 2.7 illustrates the OOB burst consisting of six bursts of differential signaling followed by specified periods of electrical idle prior to the next sequence of six bursts. Because this signaling occurs just after hardware reset, the receivers at both ends of the link will not yet be synchronized to the incoming data stream. This means that the bursts will not be detected by the normal receive logic and are therefore Out Of Band relative to normal in-band signaling. Instead the OOB bursts and idle periods are detected by squelch detection logic. The OOB sequence is as follows:

1. HBA signals COMRESET and the drive detects it.
2. Drive detects completion of COMRESET.
3. Drive signals COMINIT and it is detected by the HBA.
4. HBA detects completion of COMINIT.
5. HBA signals COMWAKE and it is detected by the device.
6. Drive detects completion of COMWAKE.

7. Drive signals COMWAKE and the HBA detects it.
8. HBA detects completion of COMWAKE and enters Link initialization.
9. Drive enters link initialization in response.

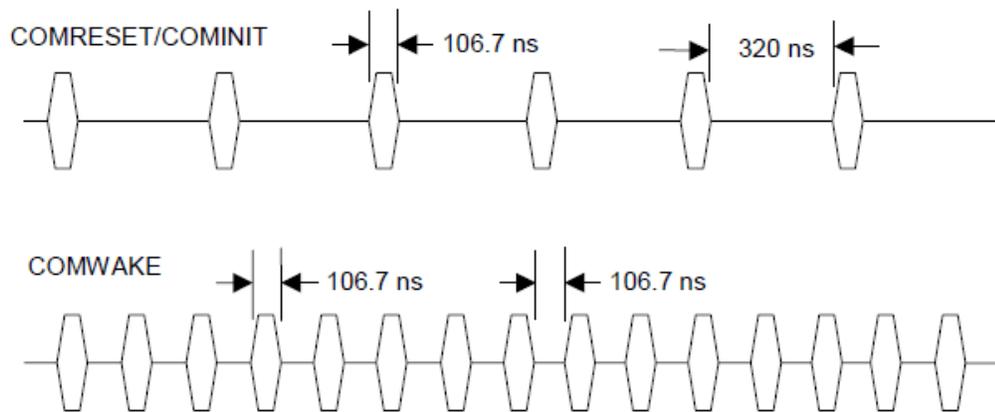


Fig. 2.8 OOB signal

2.5 Cyclic Redundancy Check (CRC)

In SATA error detection is done with Cyclic Redundancy Check (CRC). It ensures that the received data is the same as the transmitted data for each bit in the Dword. When the SATA host is transmitting a frame, a 32 bit check value, often called the CRC value, is calculated over the contents of the FIS. The CRC value is then appended to the FIS. The 32 bit CRC check sum is calculated with a simple algorithm that resembles mathematic

polynomial division. The polynomial that is used as a divisor is called the generator polynomial $G(x)$. The generator polynomial that is defined for SATA is represented in equation 2.5.1. This is an IEEE standard polynomial used for CRC calculation.

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + 1 \quad (2.5.1)$$

If the input data stream is denoted as the polynomial $M(x)$, the CRC check sum $C(x)$ can be expressed as

$$C(x) = M(x) x^{32} \text{ mod } G(x) \quad (2.5.2)$$

$$C(x) = \text{remainder of } [M(x) x^{32} / G(x)] \quad (2.5.3)$$

The algorithm is implemented with a LFSR (Linear Feedback Shift Register). This means that the flip flops on position 32,26,23,22,16,12,11,10,8,7,5,4,2 and 1 in a hardware LFSR is connected to an EXOR gate.

To decide if the appended CRC value in a received frame is correct, a check is required by the SATA host. If the CRC value is not correct, there is at least one bit error. One might think that the check is made by calculating the CRC value over the FIS and then compare this value with the appended CRC value. In practice, the CRC value is calculated over both the FIS and the appended CRC value. Due to the mathematics of CRC algorithms, the resulting CRC value will be zero if there are no bit errors.

2.6 Scrambler

The payload data in a FIS might have long sequences of repeated data. To avoid long run length and EMI (Electromagnetic Interference) the data is encoded as a pseudo random bit pattern. This method is called scrambling and is common in many digital systems. To

create randomized data, so called scrambled data, there are a few different algorithms that can be used. The algorithm that is specified for SATA uses a LFSR, just like the CRC algorithm. The SATA protocol specifies that the LFSR should use the polynomial in (2.6.1)

$$G(x) = x^{16} + x^{15} + x^{13} + x^4 + 1 \quad (2.6.1)$$

Since the dwords are continuously shifted and EXORed, the output from the LFSR appears to be random data. However, it is easy to recreate the dwords by simply running them through a descrambler that uses the exact same algorithm. The SATA specification [2] demands that all data between the SOF and EOF is scrambled before transmission to the Phy layer and descrambled when received from the Physical layer. The protocol also states that the LFSR should be initialized with a seed value of all 1 s (FFFF in hex). The initialization should be made each time a SOF is transmitted (for scrambling) or received (for descrambling).

3. DESIGN

3. The CRC and scrambler modules were designed using VHDL. The design methodology is as follows:

3.1. Steps taken to create a parallel CRC generator [3]:

1. Data width was denoted by N and CRC width was denoted by M as shown in Fig 3.1.
2. Serial CRC generator routine was implemented using given polynomial.

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + 1$$

3. Using the routine from (2) calculate CRC for the N values when Min=0. Each value is one-hot encoded, that is there is only one bit set. $Mout = F(Nin, Min=0)$
4. Build NxM matrix. Each row contains the results from (3) in increasing order.
5. Each column in this matrix, and that's the interesting part, represents an output bit $Mout[i]$ as a function of Nin .
6. Using the routine from (2) calculate CRC for the M values when $Nin=0$. Each value is one-hot encoded, that is there is only one bit set.
7. Build MxM matrix. Each row contains the results from (6) in increasing order.
8. Now, build an equation for each $Mout[i]$ bit: all $Nin[j]$ and $Min[k]$ bits in column [i] participate in the equation. The participating inputs are XORed together.

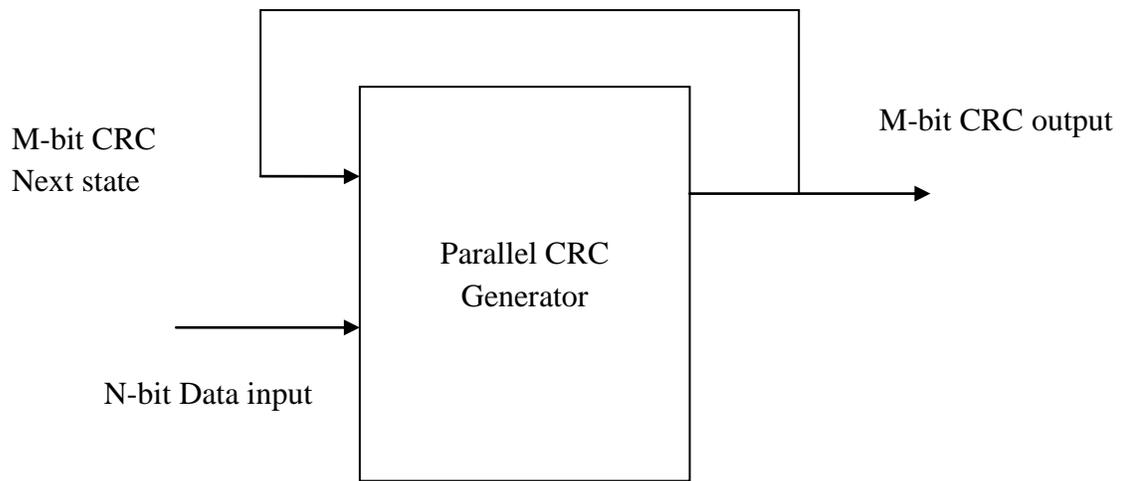


Fig 3.1 Parallel CRC block

3.2 Steps taken to create a parallel scrambler generator:

1. Data width and generator polynomial width were denoted by N and M respectively as shown in fig 3.2.
2. Serial scrambler generator was implemented using given polynomial

$$G(x) = x^{16} + x^{15} + x^{13} + x^4 + 1$$

3. Parallel Scrambler implementation is a function of N-bit data input as well as M-bit current state of the polynomial. Three matrices were built:
 - Mout (next state polynomial) as a function of Min(current state polynomial) when N=0
 - Nout as a function of Nin when Min=0
 - Nout as a function of Min when Nin=0
4. Using the routine from (3) scrambled data was calculated for the Mout values given Min, when Nin=0. Each Min value was one-hot encoded, that is there is only one bit set.
5. An MxM matrix was built. Each row contained the results from (4) in increasing order. The output was M-bit wide, which is the polynomial width.
6. The Nout values were calculated given Nin, when Min=0. Each Nin value was one-hot encoded, that is there is only one bit set.

7. An $N \times N$ matrix was built. Each row contains the results from (6) in increasing order. The output was N -bit wide, which is the data width.
8. The N_{out} values was calculated given Min , when $N_{in}=0$. Each Min value was one-hot encoded, that is there is only one bit set.
9. An $M \times N$ matrix was built. Each row contained the results from (7) in increasing order. The output was N -bit wide, which is the data width.
10. Finally an equation was built for each $N_{out}[i]$ bit: all $N_{in}[j]$ and $Min[k]$ set bits in column $[i]$ from three matrices participated in the equation. The participating inputs were XORed together.

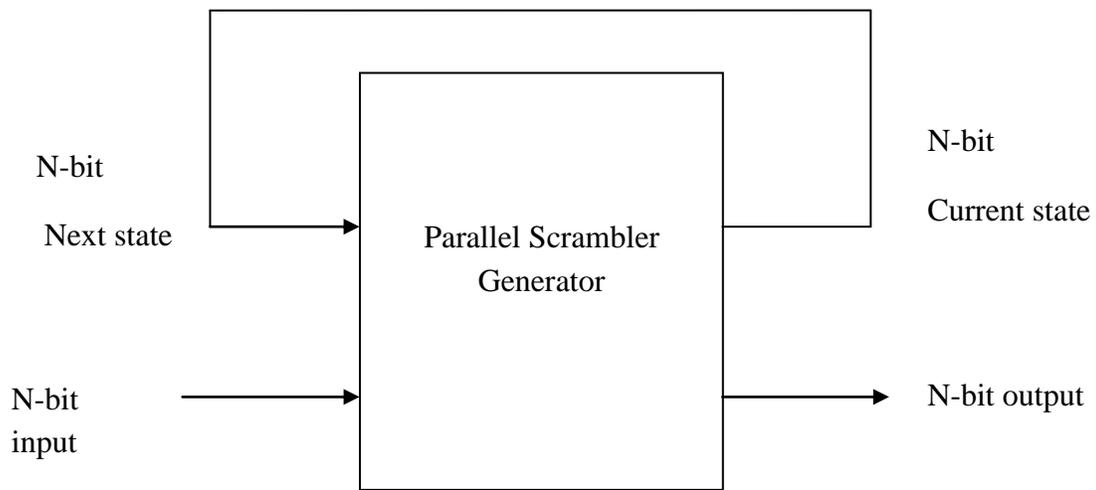


Fig 3.2 Parallel Scrambler Block

4. SIMULATION and RESULT

4.1 CRC simulation

The testbench wavefor of CRC module is given in fig 4.1. As per the requirement of SATA specification the CRC module accepts a dword of data. After the block is reset it is initialised (0x52325032) and is ready to start calculating CRC. But as long as the enable is low, no calculation takes place. As soon as the enable is pulled high the CRC calculation starts on the next clock cycle. The calculation goes on for each dword at each clock cycle till the enable is pulled low. The CRC output after the enable becomes low is the final CRC value for the three input dwords.

4.2 CRC Result

Input Dwords:

0x00308027
0xE1234567
0x00000000

CRC output:

0x3D14369C

4.3 Scrambler simulation

The testbench wavefor of scrambler module is given in fig 4.2. As per the requirement of SATA specification the scrambler module accepts a dword of data. After the block is reset it is initialised (0xFFFF) and is ready to start scrambling . But as long as the enable is low, no scrambling takes place. As soon as the enable is pulled high scrambling starts on the next clock cycle. The calculation goes on for each dword at each clock cycle and each dword is scrambled till the enable is pulled low. Fig 4.2 shows both the scrambler output, as well as the scrambled data.

4.4 Scrambler Result

Input Dword	Scrambler Output	Scrambled Data
0xC2D2768D	0x4BA08972	0x8972FFFF
0xC2D2768F	0xE7EEF8C8	0x253C8E47



Fig. 4.2 Testbench waveform of Scrambler block

5. CONCLUSION

Thus we studied the SATA protocols and gained a great insight into this powerful bus standard. The various layers of SATA were studied and the inherent mechanism of data transfer was understood.

Also we designed, synthesized and simulated two important parts of SATA protocols, CRC and Scrambler.

References

[1] Don Anderson, *SATA Storage Technology*, MindShare Inc., 2007, ISBN

978-0-9770878-1-5

[2] Serial ATA International organization, *Serial ATA Revision 1.0a*, 7 January 2003

[3] Evgeni Stavinov, *A Practical Parallel CRC Generation Method*, Circuit Cellular,

Issue No. 234, January 2010