SATA-IP reference system on Linux manual

Rev1.1 22-Feb-12

1. Introduction

Serial ATA (SATA) is an evolutionary replacement for the Parallel ATA (PATA) physical storage interface. SATA interface increases speed transfer to be 1.5 Gbps for SATA-I and 3.0 Gbps for SATA-II. To communication by SATA protocol, there are four layers in its architecture, i.e. Application, Transport, Link, and Phy.

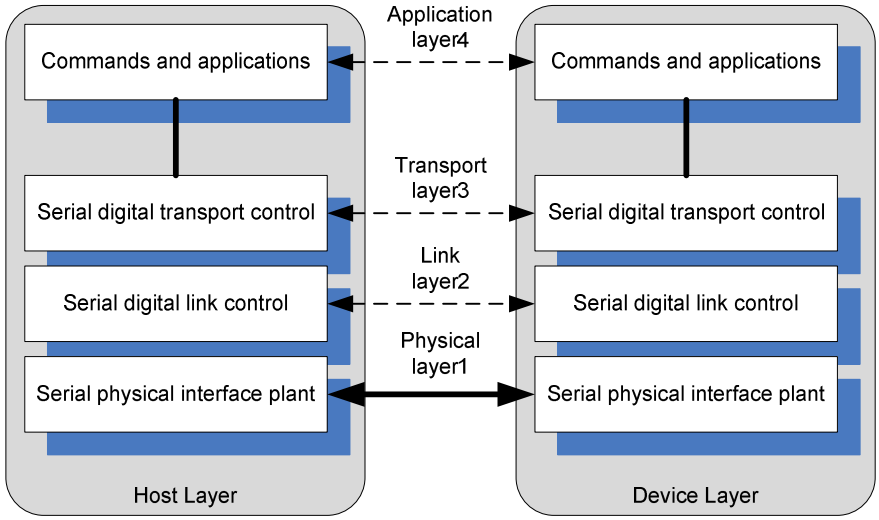

Figure 1 SATA Communication Layer

The Application layer is responsible for overall ATA command execution, including controlling Command Block Register accessed. The Transport layer is responsible for placing control information and data to be transferred between the host and device in a packet/frame, known as a Frame Information Structure (FIS). The Link layer is responsible for taking data from the constructed frames, encoding or decoding each byte using 8b/10b, and inserting control characters such that the 10-bit stream of data may be decoded correctly. The Physical layer is responsible for transmitting and receiving the encoded information as a serial data stream on the wire.

SATA-IP is implemented for Link layer protocol and some Transport layer. Physical layer is implemented by using GTX transceiver of the Virtex-5 platform with some logic design. The other Transport layer and Application layer also be designed in this reference design by using logic design and software on Linux system running by PowerPC of Virtex-5 FXT. More details about this reference design are described as follows.

2. Environment

This reference design is based on the following environment as shown in Figure2.

- ML507 Platform
- ISE 11.4 / EDK 11.4
- SATA Peripheral (SATA-I/II HDD) connect SATA cable to J40 on ML507
- Serial (RS232C) communication, connect RS232C cable to P3 on ML507 (Set baud rate=115,200 / data=8bit / Non-Parity / Stop=1bit)

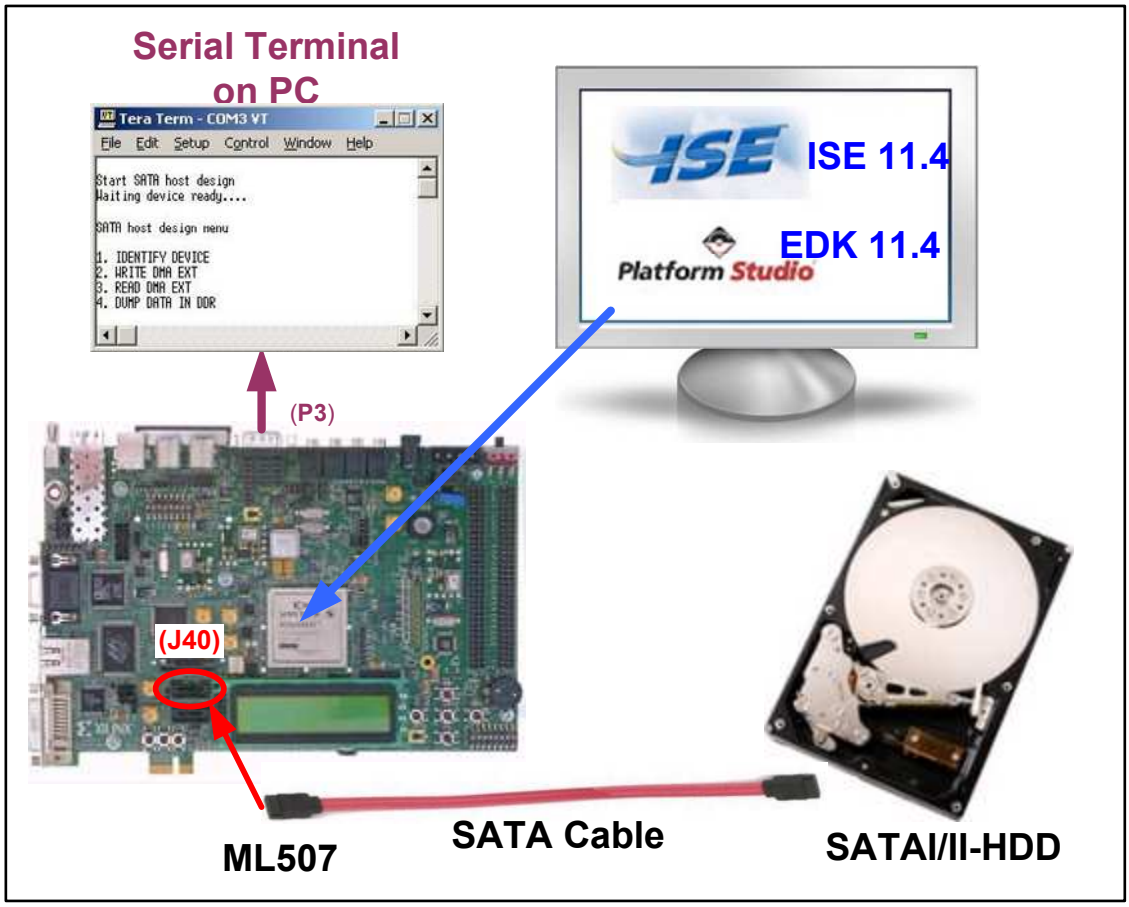

Figure 2 Reference design environment

Refer to “SATA_IP Linux Demo Instruction” for operation procedure of this reference design.
For evaluation version, IP-Core has 1-hour time limitation to use. After 1-hour use, IP-core will stop any data transfer.

3. Hardware description

- SATA IP Host design implementation on Virtex5 FPGA

As shown in Figure3, SATA IP consists of only Link Layer and some part of Transport Layer, so users need to prepare other Layer such as PHY Layer and Transport Layer by themselves. This reference design provides the example hardware logic design for PHY and Transporter Layer. The special SATA device driver on Linux system is also implemented to support Application Layer. Linux OS on this reference design use only the lower 128 MB area in DDR2, so 128 MB remaining area can be implemented to store special data to transfer between software on PowerPC and hardware logic control. The details of each block are described as follows.

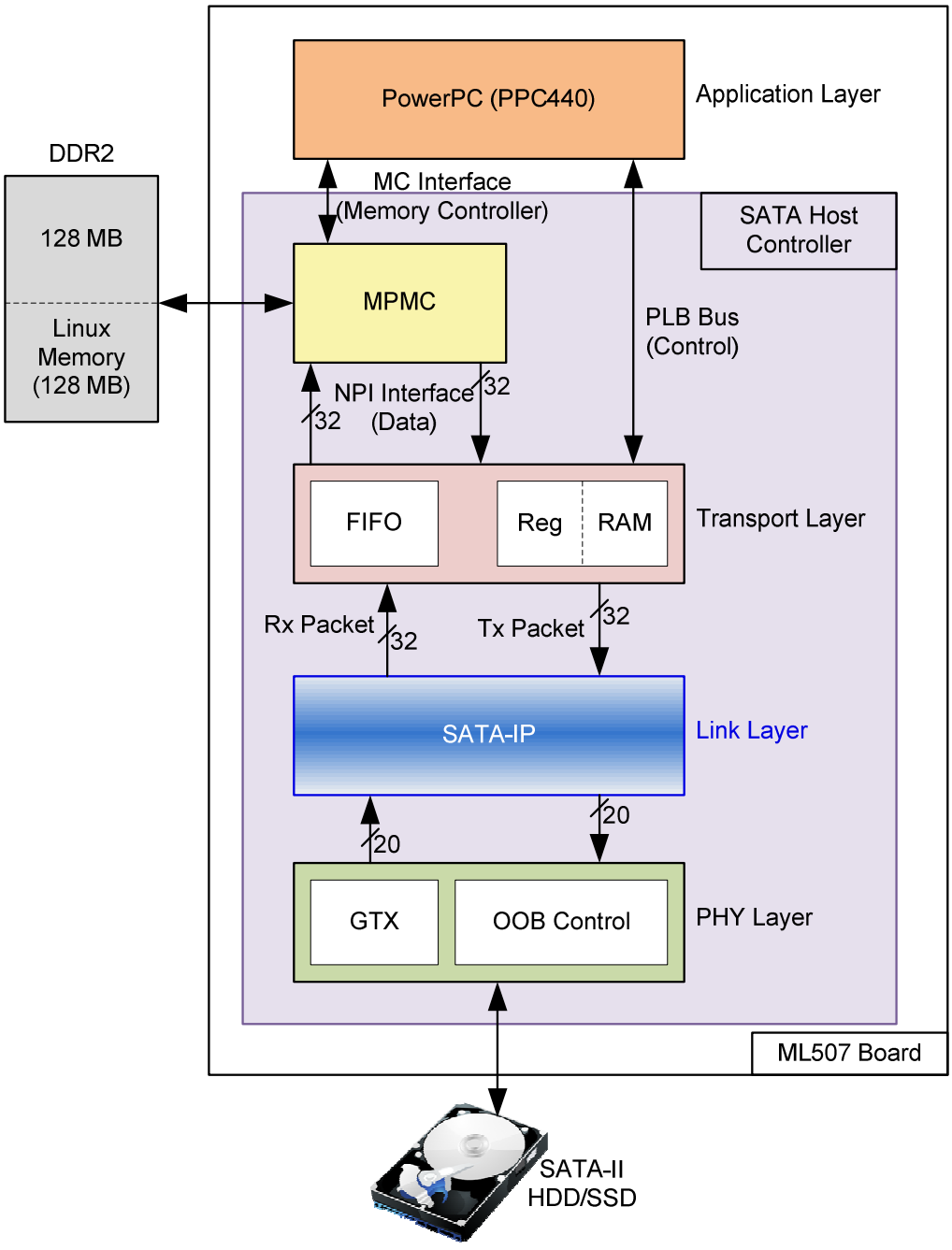

Figure 3 Hardware block diagram in reference design on ML507 board

- PHY Layer

PHY Layer in this reference design is implemented by using built-in high-speed serial circuit of Virtex-5 FXT device, called GTX transceiver, which is controlled by OOB (Out-of-Band) Control logic (oob_control.v). OOB Control logic is modified from Xilinx application note (XAPP870), available on Xilinx website, to support GTX transceiver instead of GTP transceiver. PHY layer source code is stored in "sata2phy_ml507.v" of "pcores/npi_plb_sata_v1_00_a/hdl/verilog" folder. It can support Automatic Speed Negotiation to change GTX parameter for SATA-I/II speed is designed by "speed_neg_control.v". Since GTX transceiver is set to interface with 16-bit data size, data input/output interface with this Layer is equal to 20-bit data size (4-bit additional signals are control bit of 16-bit data signal).

Before building user board, user must read carefully and must follow design guide line described in UG198 (Virtex-5 FPGA RocketIO GTX Transceiver User Guide).

- Link Layer

This layer protocol is implemented by using SATA-IP of Design-Gateway Company. The details of SATA-IP signal interface is shown in Table 1. The interface signals with Transport Layer are separated into two groups, i.e. transmit and receive. Signal waveform of data transmit and data receive transaction are shown in Figure 4 and 5 respectively.

Signal	Signal Direction	Description
Common Interface Signal		
trn_reset	In	Reset SATA IP core. Active high. Assert at least 4 clock period of core_clk for reset SATA-IP.
trn_link_up	Out	Transaction link up is asserted when the core establish the communication with SATA PHY.
trn_clk	In	Clock which is synchronized with trn_xxx signal for interface with the Host. There is no global clock buffer inside SATA IP core for this signal, so external global clock buffer should be inserted This clock frequency is required to be higher than core_clk frequency.
core_clk	In	IP Core operating frequency output (37.50MHz for SATA-I, 75.00MHz for SATA-II). This clock is generated from SATA PHY.
dev_host_n	In	Device or Host design assignment. '0': ATA Host IP Core, '1': ATA Device IP Core (Use '0' for the host reference design)
Transmit Transaction Interface		
trn_tsof_n	In	Transmit Start-Of-Frame (SOF): Indicate start each SATA FIS packet. Active low. Not used now.
trn_teof_n	In	Transmit End-Of-Frame (EOF): Indicate end each SATA FIS packet. Active low.
trn_td[31:0]	In	Transmit Data: SATA FIS packet data to be transmitted.
trn_tsrc_rdy_n	In	Transmit Source Ready: Indicates that trn_td[31:0] from the Host is valid. Active low.
trn_tdst_rdy_n	Out	Transmit Destination Ready: Indicate that the core is ready to accept data on trn_td[31:0]. Active low. trn_tsrc_rdy_n must be de-asserted within 4 period of trn_clk after trn_tdst_rdy_n is de-asserted. So the core can accept 4 DWORD of trn_td[31:0] after trn_tdst_rdy_n is de-asserted.
trn_tsrc_dsc_n	In	Transmit Source Abort: Assert 1 clock period of trn_clk during operation (between tsof and teof) when the Host requires to cancel current write operation. Active low. After asserted, the Core will send SYNC primitive to SATA-PHY for abort the current transfer. The Host needs to wait until trn_tdst_rdy_n ready again before sending next packet. See Figure 6 for more details.
trn_tdst_dsc_n	Out	Transmit Destination Abort: Assert 1 clock period of trn_clk from the Core to cancel current write operation when SYNC primitive is received during data write operation. Active low. See Figure 8 for more details.

Table 1 SATA IP Interface Signal Definition

Signal	Signal Direction	Description
Receive Transaction Interface		
trn_rsof_n	Out	Receive Start-Of-Frame (SOF): Indicate start each SATA FIS packet. Active low.
trn_reof_n	Out	Receive End-Of-Frame (EOF): Indicate end each SATA FIS packet. Active low.
trn_rd[31:0]	Out	Receive Data: SATA FIS packet data to be transmitted.
trn_rsrc_rdy_n	Out	Receive Source Ready: Indicates that trn_rd[31:0] from the core is valid. Active low.
trn_rdst_rdy_n	In	Receive Destination Ready: Indicate that the Host is ready to accept data on trn_rd[31:0]. Active low. trn_rsrc_rdy_n will be de-asserted within 4 period of trn_clk after trn_rdst_rdy_n is de-asserted. So Host should be supported to accept 4 DWORD of trn_rd[31:0] after trn_rdst_rdy_n is de-asserted.
trn_rsrc_dsc_n	Out	Receive Source Abort: Assert 1 clock period of trn_clk from the Core to cancel current read operation when SYNC primitive is received during data read operation. Active low. See Figure 9 for more details.
trn_rdst_dsc_n	In	Receive Destination Abort: Assert 1 clock period of trn_clk during read operation (between rsof and reof) when the Host requires to cancel current read operation. Active low. After asserted, the core will send SYNC primitive to SATA-PHY for abort the current transfer. The Host needs to wait until trn_tdst_rdy_n ready again before sending next packet. See Figure 7 for more details.
SATA PHY Interface for Virtex5 GTX		
PHYCLK	In	Reference Clock for 16-bit SATA PHY (Virtex5 GTX) - 75MHz for SATA-I - 150MHz for SATA-II This clock is generated from DCM inside SATA PHY. It's used for both both TX and RX data by elastic buffer in GTX of SATA PHY.
LINKUP	In	Indicates that SATA link communication is established. Active high.
PLLLOCK	In	Indicates that DCM of GTX is locked. Active high.
TXDATA[15:0]	Out	16-bit transmit data from the core to the GTX
TXDATAK[1:0]	Out	2-bit Data/Control for the symbols of transmitted data. ("00": data byte, "01": control byte, "1X": undefined).
RECCLK	In	Clock Recovery to synchronous with received data from GTX
RXDATA[15:0]	In	16-bit receive data from the GTX to the core.
RXDATAK[1:0]	In	2-bit Data/Control for the symbols of received data. ("00": data byte, "01": control byte, "1X": undefined)
RXDATAVALID	In	Indicate that RXDATA from SATA PHY is valid.
RXDATAOUT	Out	RXDATA signal after Elastic buffer and sychronous with PHYCLK
RXDATAKOUT	Out	RXDATAK signal after Elastic buffer and sychronous with PHYCLK
RXDATAVALIDOUT	Out	Indicate that RXDATAOUT is valid.

Table 1 SATA IP Interface Signal Definition (Cont'd)

Bit	Signal Name	Description
[31:27]	Reserved	Always zero
[26]	Dir	Current transfer direction flag. '0': From the Host to SATA IP, '1': From SATA IP to the Host
[25:24]	Error	Error code flag. "00": No error "01": Bad/Unknown SATA FIS packet. WTRM primitive is received during read operation or R_ERR primitive is received at the end of write operation. "10": CRC error "11": Reserved
[23:8]	Reserved	Always zero
[7:0]	FIS Type	This byte indicates the header of error code packet. "0xEF" is defined to be different from other SATA FIS.

Table 2 Error Code Description

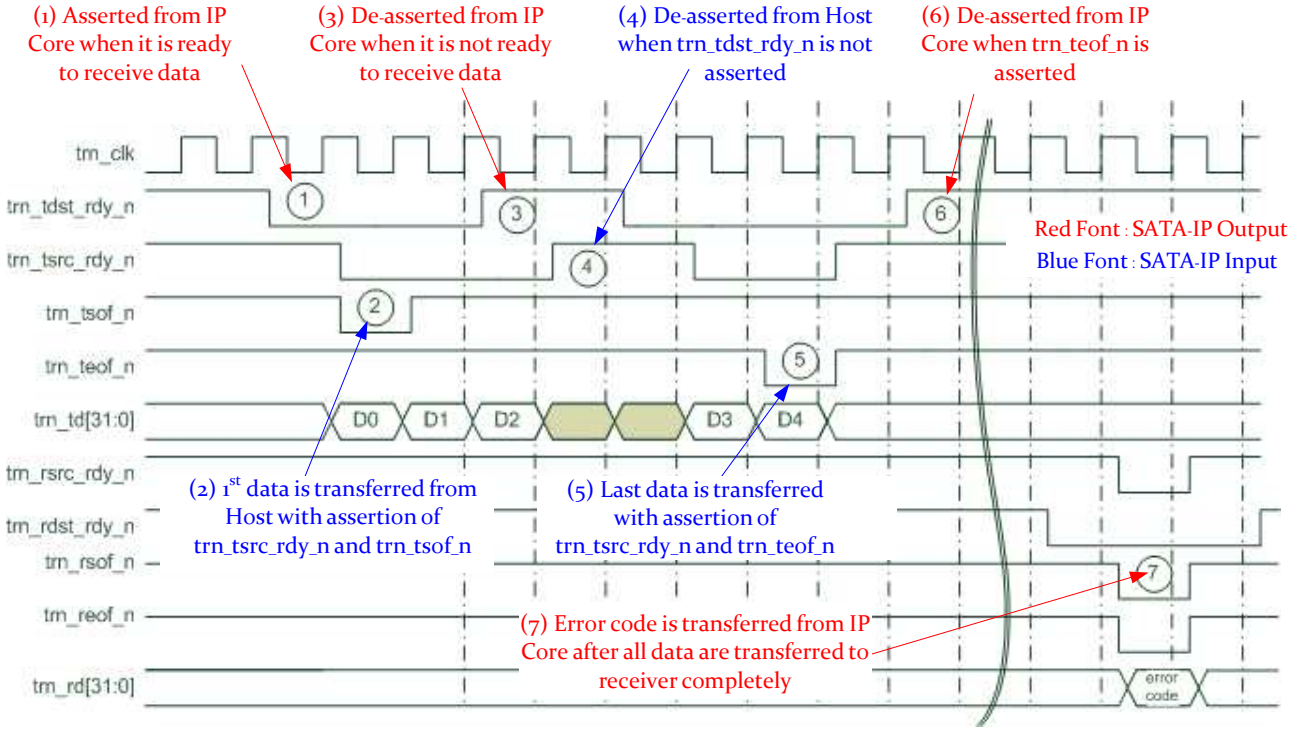

Figure 4 Waveform of data transmit transaction

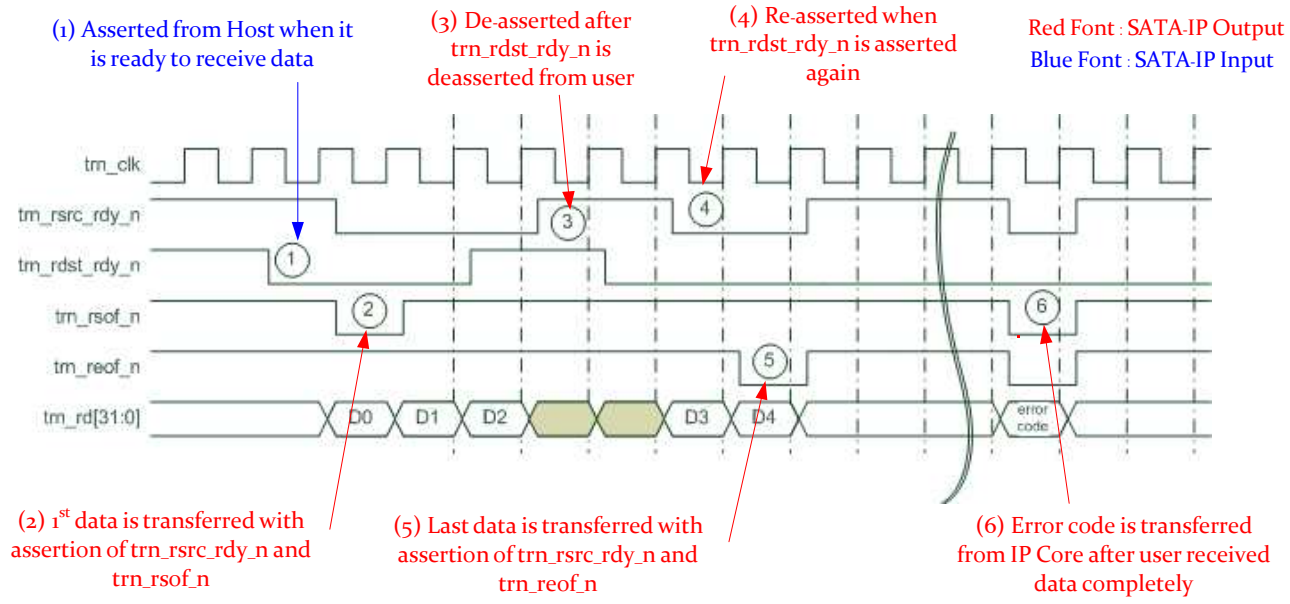

Figure 5 Waveform of data receive transaction

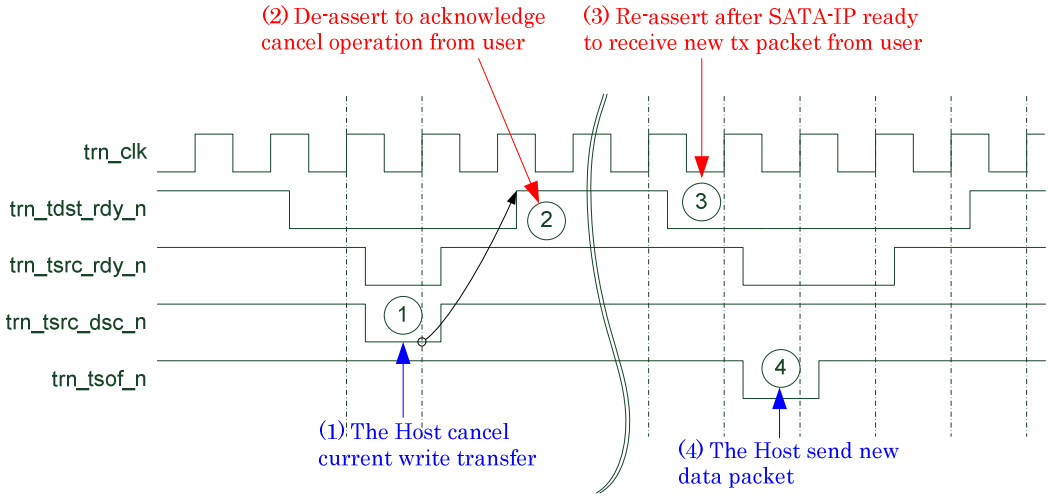

Figure 6 trn_tsrc_dsc_n Timing diagram

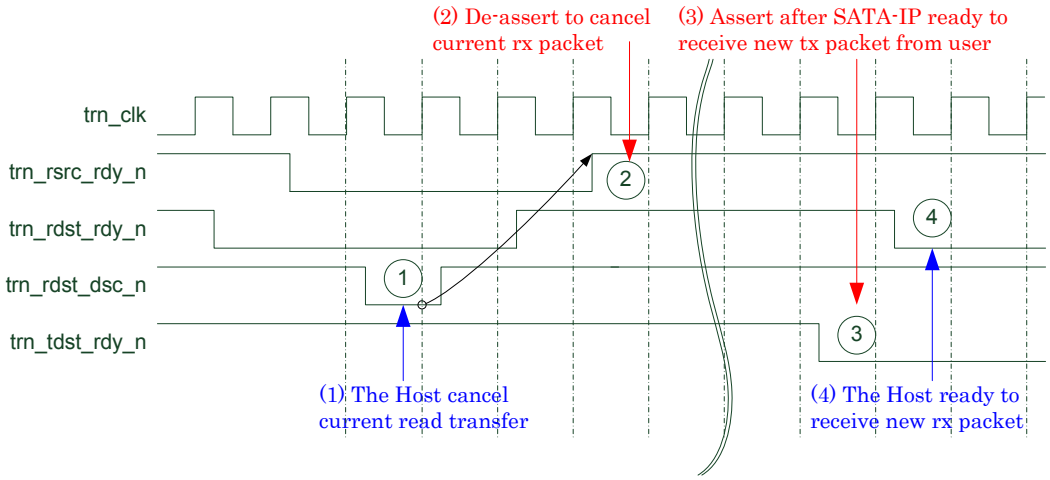

Figure 7 trn_rdst_dsc_n Timing diagram

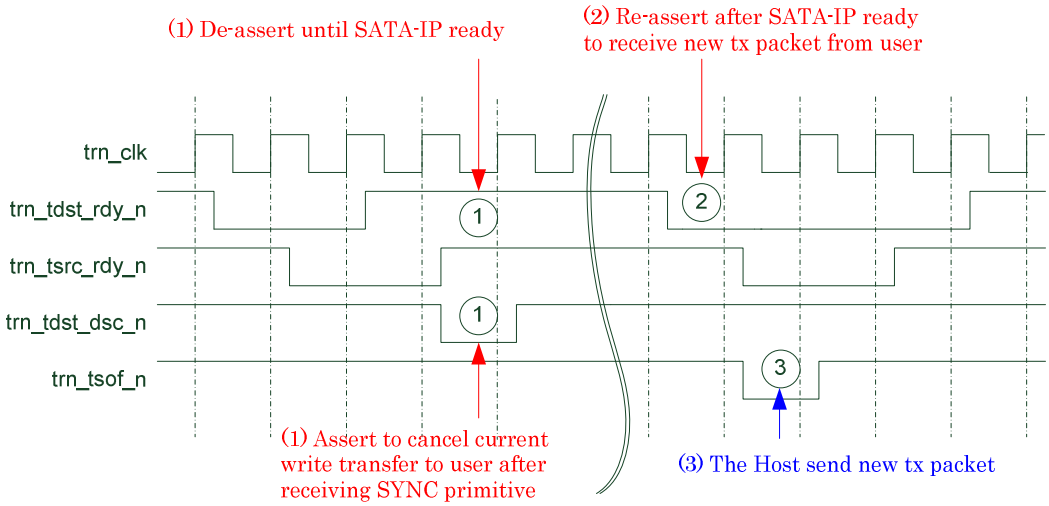

Figure 8 trn_tdst_dsc_n Timing diagram

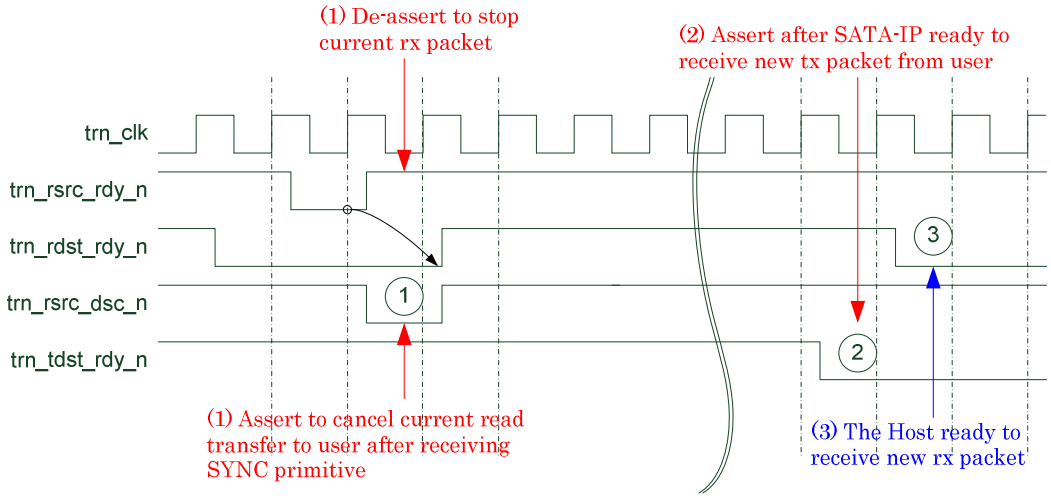

Figure 9 trn_rsrc_dsc_n Timing diagram

- Transport Layer

User need to build Transport Layer by themselves because this Layer structure depends on hardware architecture and user application. In reference design, FIS content is built by Software and store in DDR2, and then transferred to Transporter Layer through NPI interface which is DMA channel of MPMC (Multi-Port Memory Controller). The DDR2 address of FIS content, status and necessary control signal are implemented by using RAM and register which can be accessed through PLB bus interface.

This reference design implements only basic FIS type to demonstrate SATA-IP core. The supported FIS types are categorized to be three groups, i.e. CFIS, SFIS, and DFIS. Register FIS from Host to Device (FIS value = 27h) which is used to transfer ATA commands and contents to the device is in CFIS group. Register FIS from Device to Host (FIS value = 34h) which is used to indicate command completion status and PIO Setup FIS (FIS value = 5Fh) which is returned from the Device in PIO data transfer are in SFIS group. Data FIS from/to Host to/from Device (FIS value = 46h) is in DFIS group. DMA Setup FIS (FIS value = 41h), BIST Activate FIS (FIS value = 58h), and Set Device Bits FIS (FIS value = A1h) are not implemented in this reference design.

Memory map of each register is shown in Table3 and Table4.

Address	Register Name	Description (Bit order is little endian)
Register map when PowerPC write		
BA+0x00	CFIS Address	DDR2 address to store CFIS content
BA+0x04	SFIS Address	DDR2 address to store SFIS content
BA+0x08	Total DFIS Length	Total transfer size of DFIS in current command (Max size = 2000000h or 32 MB)
BA+0x0C	CFIS Control Reg	[4:0] CFIS Length (Not used in current design) [5] Non-data command (‘0’: data command, ‘1’: Non-data command) [6] PIO/DMA command (‘0’: PIO command, ‘1’: DMA command) [7] Transfer direction (‘0’: data-out command, ‘1’: data-in command) [11:8] Total PRD (Not used in current design)
BA+0x10	Control Reg	Write this register to start transfer command/data
BA+0x40	Clear HStatus Reg	[0] Clear Command Complete flag (‘0’: Not clear, ‘1’: Clear flag) [3] Clear Signature Updated flag (‘0’: Not clear, ‘1’: Clear flag) [5] Clear Fatal Error flag (‘0’: Not clear, ‘1’: Clear flag)
BA+0x48	Clear SError Reg	[0] Clear write transfer fail flag (‘0’: Not clear, ‘1’: Clear flag) [1] Clear read transfer fail flag (‘0’: Not clear, ‘1’: Clear flag) [2] Clear unknown FIS error flag (‘0’: Not clear, ‘1’: Clear flag)
BA+0x7C	Reset Reg	[0]: SATA Reset (‘0’: No reset, ‘1’: Reset SATA)
BA+0x80	PRD0 Address	DDR2 address of PRD0
BA+0x84	PRD0 Length	PRD0 data transfer length (Max size = 200000h or 2 MB)
BA+0x88	PRD1 Address	DDR2 address of PRD1
BA+0x8C	PRD1 Length	PRD1 data transfer length (Max size = 200000h or 2 MB)
…	…	…
BA+0xF0	PRD14 Address	DDR2 address of PRD14
BA+0xF4	PRD14 Length	PRD14 data transfer length (Max size = 200000h or 2 MB)
BA+0xF8	PRD15 Address	DDR2 address of PRD15
BA+0xFC	PRD15 Length	PRD15 data transfer length (Max size = 200000h or 2 MB)

(BA : Base Address)

Table 3 Register mapping from CPU Write side

Address	Register Name	Description (Bit order is little endian)
Register map when PowerPC read		
BA+0x00	CFIS Address	DDR2 address to store CFIS content
BA+0x04	SFIS Address	DDR2 address to store SFIS content
BA+0x08	Total DFIS Length	Total transfer size of DFIS in current command
BA+0x0C	CFIS Control Reg	[4:0] CFIS Length [5] Non-data command [6] PIO/DMA command [7] Transfer direction [11:8] Total PRD
BA+0x14	Remain write data	[25:13] Remaining data in that write transaction in 8kByte unit, [12:0]=0
BA+0x18	Remain write data	[15:0] Remaining write data request to NPI in that Tx packet [31:16] Remaining write data from NPI in that Tx packet (Max packet size = 8kByte)
BA+0x1C	Remain read data	Remaining data in that read transaction
BA+0x20	Error code	Returned error code from SATA-IP
BA+0x40	HStatus Reg	[0] Command Complete flag (‘0’: Not complete, ‘1’: Command complete) [3] Signature Updated flag (‘0’: Not updated ‘1’: Signature Reg updated) [5] Fatal Error flag (‘0’: No error, ‘1’: Fatal error detected) [31] SATA Link-up (‘0’: SATA device not detected, ‘1’: SATA device detected)
BA+0x44	SStatus Reg	00000000h when SATA not Link-up When SATA Link-up, [3:0] = “0011”, [11:8] = “0001” [7:4] = “0001”: SATA-I, “0010”: SATA-II
BA+0x48	SError Reg	[0] Write error (‘0’: No error, ‘1’: Write transfer failed) [1] Read error (‘0’: No error, ‘1’: Read transfer failed) [2] FIS error (‘0’: No error, ‘1’: Unknown FIS error)
BA+0x4C	Signature	Signature value
BA+0x80	PRD0 Address	DDR2 address of PRD0
BA+0x84	PRD0 Length	PRD0 data transfer length
BA+0x88	PRD1 Address	DDR2 address of PRD1
BA+0x8C	PRD1 Length	PRD1 data transfer length
…	…	…
BA+0xF0	PRD14 Address	DDR2 address of PRD14
BA+0xF4	PRD14 Length	PRD14 data transfer length
BA+0xF8	PRD15 Address	DDR2 address of PRD15
BA+0xFC	PRD15 Length	PRD15 data transfer length

(BA : Base Address)

Table 4 Register mapping from CPU Read side

Note: BA+0x80 – BA+0xFF can read after write that register. Please don’t read these register during data transfer because it will set PRD pointer to wrong position.

From Table3 and Table4, it shows that there are 16 PRDs (Physical Region Descriptor) inside the reference design to be pointer of data buffer. Each PRD stores Start physical address of data buffer and buffer size. From SATA specification, the maximum sector count of 48-bit write/read ATA command is equal to 65536 sectors or 32 MB, so the maximum size of each PRD in this design is limited to 2 MB (32 MB/16 PRDs). By using this architecture, PowerPC will set all control registers only one time per command which helps PowerPC can handle other tasks during this time and also increase performance by not much h/w and s/w handshake process.

The example of PRD function is shown in Figure10. Data buffer size and position is independent with other PRDs. So, Linux system can manage memory with high efficiency.

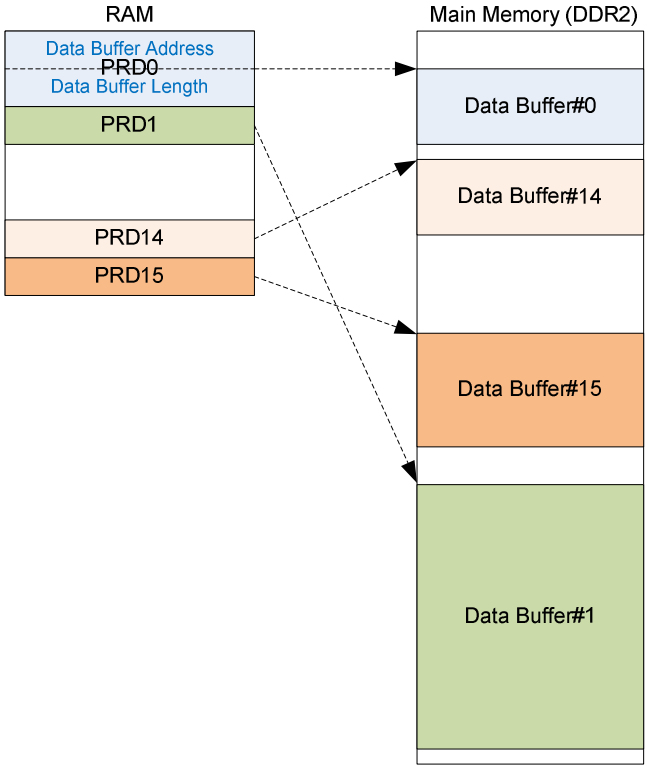

Figure 10 The example of PRD function

In write transaction, logic control will wait DMA Activate FIS (FIS value = 39h) from Device, read data from data buffer as defined in PRD, arrange data packet to 8kByte size, add DFIS header (FIS value = 46h) to each data packet, and then send it to SATA-IP. This step will repeat until total data transfer size is equal to setting value in Total DFIS Length register. After that, Interrupt and other status flag will be send to PowerPC and go to Idle state to wait next command from PowerPC.

In read transaction, the process will be reversed. Logic control will receive DFIS packet from SATA-IP, remove DFIS header of each packet, store data to internal FIFO, send data from internal FIFO to data buffer as defined in PRD through NPI interface. This step will repeat until Register Device to Host is detected which means data transfer complete. Interrupt and no error status will be sent to PowerPC if total received data is equal to setting value in Total DFIS Length register. Error will be set if transfer size is not matched.

Note: Transport Layer in this reference design support Read/Write DMA command, Identify command, but not support Read/Write command in PIO mode.

- MPMC

MPMC is used to be memory controller to control memory access from MC interface of PowerPC and NPI interface of hardware logic. MPMC can support many interfaces and support up to 8 channels. For more detail of PowerPC, NPI, or MPMC, please refer to Xilinx technical document.

- Application Layer

This layer is designed by using software running on PowerPC. More details about software structure and function are described in next topic.

4. Software description

- SATA Software design on ML507

SATA Host Driver in this reference design is developed on Xilinx Open Source Linux system. More details about this Linux system can be found from http://xilinx.wikidot.com. The software structure for SATA application is shown in Figure 11.

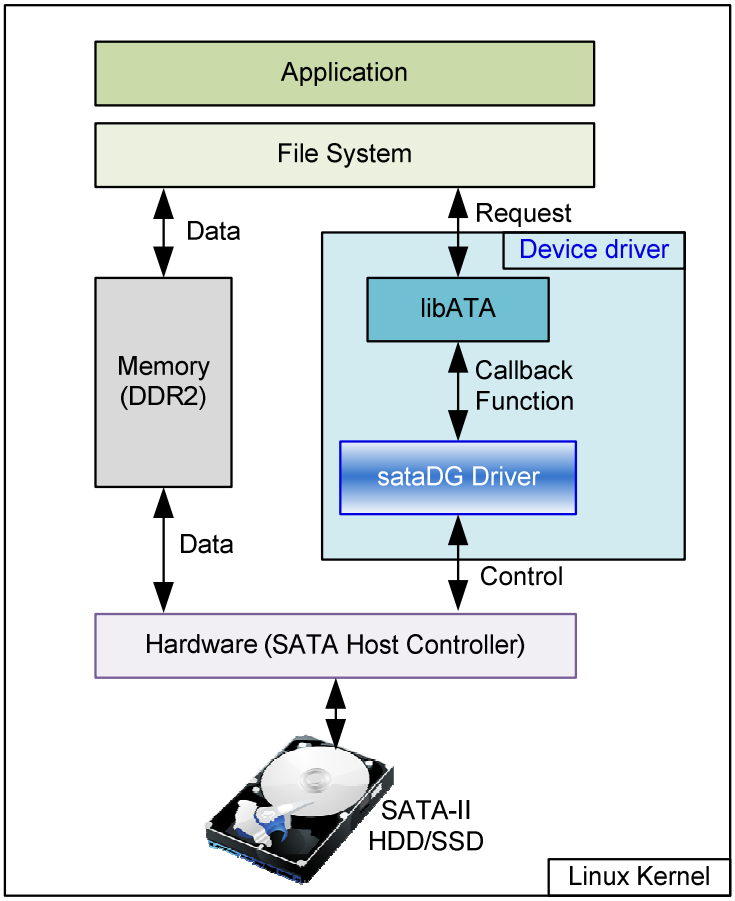

Figure 11 Software Structure for SATA Application

When user application transfers data with SATA device, data will be managed by File System and then send request to device driver. After that, device driver which consists of libATA and sataDG Driver will allocate memory to store CFIS (ATA command packet), SFIS (status packet returned from SATA device), and set the address of these FIS packets to internal register of hardware system. Since data is managed by File System, device driver only gets the address of data and set them to PRD RAM inside hardware system.

libATA is high-level device driver which provides many functions to support Linux kernel to control the SATA host controller. But SATA host controller interface in each platform may be different, low-level device driver is developed to convert the general function of libATA to match with each hardware protocol. More details about libATA function can be found from http://www.kernel.org/doc/htmldocs/libata.html. This reference design also develops the special SATA host driver to match with the hardware design. The details are described as follows.

- SATA Host Driver on reference design

The driver is developed to support the basic function of libATA. Some advance features in libATA function isn't supported because of hardware limitation such as speed auto-negotiation function, power management function, multiple device function.

After power-on system, the driver will initial hardware system by setting register to be ready for transferring data with SATA device. The driver will receive interrupt with SFIS packet from SATA device to confirm that the initialize process of SATA device is completed.

To interface with hardware system, the driver needs to decode the ATA command to check command types such as Non-data command, PIO data-in command, PIO data-out command, or DMA command and then set the types of command to control register in hardware before sending start signal. If command is Non-data command, PRD is not required to set.

In case of data transfer command, if data size is big, data will be scattered to smaller size in different address of main memory by File System management. The address and its length of each data will be set to PRD RAM by the driver. Because PRD RAM in this reference design is limited at 16 areas for one ATA command, more than one command will be generated by device driver when data is split more than 16 parts.

After each command complete, the driver will check status and error register of hardware system and then converts all information back to libATA.

5. Summary

- Test performance

Table5 shows performance of SATA device by using Bonnie++ benchmark software. The performance of SATA device in this reference design is less than raw data test performance without OS and file system.

HDD Model	Sequential Output			Sequential Input		Random Seeks
	Per Char (K/s)	Block (K/s)	Rewrite (K/s)	Per Char (K/s)	Block (K/s)	(/s)
HDP725025GLA380	3,787	27,124	19,382	4,335	48,131	137.9
TS32GSSD25S-M	4,428	55,758	17,686	4,337	48,634	1083.6
	Sequential Create			Random Create		
	Create (/s)	Read (/s)	Delete (/s)	Create (/s)	Read (/s)	Delete (/s)
HDP725025GLA380	1,267	-	5,593	1,267	-	3,157
TS32GSSD25S-M	1,775	-	-	1,790	-	8,432

Table 5 Disk performance by using benchmark

6. Revision History

Revision	Date	Description
1.0	22-Jul-09	Initial draft
1.1	22-Feb-12	Change SATA-IP version to bypass elastic buffer, ISE/EDK version, and support auto-speed negotiation.