DESIGN OF A SECURE DIGITAL CARD IO TRANSACTOR AND CARD MODEL

Dr. Arshia Azam,
Associate Professor, ECE Department, Polytechnic
Maulana Azad National Urdu University
aazam_04@yahoo.co.in

Abstract:
This paper describes the design of secure digital IO transactor, secure digital card model, and verification of the functionality of secure digital card model. The Secure Digital IO protocol for the Transactor is implemented very easily in System Verilog, With the object-oriented features provided by System Verilog, we made our verification environment more manageable and reusable and The simulation is carried out with synopsis VCS tool.

Index Terms: *Secure digital card, Transaction Level Modeling (TLM), System verilog, Verification.*

I. Introduction

SD Secure Digital Memory Card is a memory card that is specifically designed to meet the security, capacity, performance, and environment requirements inherent in newly emerging audio and video consumer electronic devices. SD Cards can be used in a variety of digital products, digital music players, cellular phones, handheld PCs (HPCs), digital cameras, digital video camcorders, smart phones, car navigation systems and electronic books. SD Host Controller acts as the interface between SD Memory card and the host system, understand the SD Protocol and AHB Protocol. Transaction Level abstraction is used for verification as this is much more efficient from both the performance and debugging points of view in terms of simulation time as compared to performing all of the low level bit _twiddling operations on individual's signals in RTL. A properly written transaction-level model only takes a fraction of the time it takes to write an RTL model, thus enabling the design and the verification team to work in parallel. It also executes orders of magnitude faster than the RTL model, allowing the verification environment and tests to be developed and debugged much faster. Design methodologies that require or produce a transaction-level model of the design often use System Verilog as the modeling language, as it provides all of the necessary high-level constructs that make writing transaction-level models more efficient.

Transaction: A **transaction** abstracts the set of data (logical set) that is transferred between the blocks. It has no notion of time and triggers an event once it is put on the bus. A transactor concurrently executes the tasks that monitor (watch) and drive a transaction (assign). A **driver** actively supplies data to the DUT and can be either a proactive driver or a reactive driver. A **proactive driver** is in control of the initiation and type of transaction. Whenever a new transaction is supplied by the higher layers of verification environment, the transaction is immediately executed on the physical interface. A **reactive driver** is not in control of initiation or type of transaction, but may be in control of some aspect of timing of its execution, such as

introduction of wait states. The transaction is initiated by the DUT, and the reactive driver supplies the required data to successfully complete the transaction.

A **monitor** reports observed high-level transaction timing and data information. A **reactive monitor** includes elements to generate the required low-level handshaking signals to successfully complete a transaction. A **passive monitor** observes all signals involved in transaction without interference. A passive monitor is suitable for monitoring transactions on an interface between two DUT blocks in a system-level verification environment.

Transactor is used to identify components of the verification environment that interface between two levels of abstraction for a particular protocol or to generate protocol transactions. Thousands of transactions get created by the generators, flow through transactors, get recorded and compared then freed. Transactors are better modeled using a transaction descriptor. A transaction descriptor contains all of the necessary information to call the appropriate procedure that implements the transaction.

II. Design of Secure Digital IO Transactor:

The Paper involves the design of SD IO Transactor and SD memory card model. The SD IO protocol is implemented as a 9-pin serial interface and supports up to 4 data pins. It is a synchronous interface with the clock being provided by the host device. Bi-directional command and data buses are used and are implemented in a dominant-recessive manner such that the buses are pulled-high when not active. The maximum clock frequency is 25MHz and as such the maximum data transfer rate is 100Mbps. Commands and data are transferred in frames with start-bit, stop-bit and CRC fields as shown in Figure 1. The protocol is implemented very easily in System Verilog, The Protocol is implemented from the SD_Card_Physical_Layer_v2.00 [1].

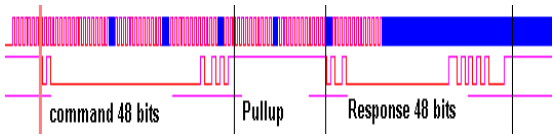

Figure 1 command response signals

Based on the functionalities the SDIO Transactor has been divided into two types:

- Monitor
- Drive

The monitoring function is again divided in to three tasks

- Monitor command response
- Monitor data0-data3 lines for read and write data and crc status
- Monitor interrupt on data1 line

Monitor Command Response Task: In the absence of command and response, the command response line is pulled high. This task always monitors command response line. If the line goes low detects the start bit, and then captures the second bit (direction bit) if it is 1 starts capturing first 6 bits (command Index), 32 bits argument, 7 bits crc, then calculates crc and compares with the received crc . Then sets the crc Validated bit if matched. It finally captures end bit and checks whether it is 1 or not If it is not 1 it is the indication end bit failed. The Figure 2 gives a detailed flow chart representation of the command response task.

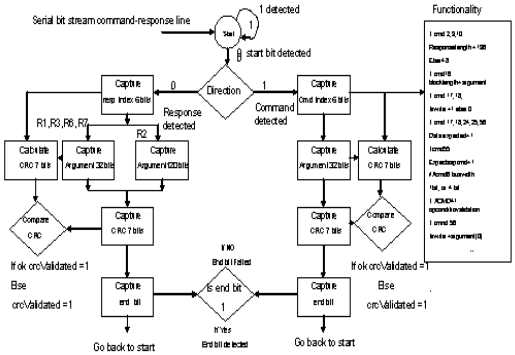

Figure 2. Flow chart Command Response Task.

After capturing command index the parameters needed to capture response and data are set according to the commands received . for example if cmd2, cmd9, cmd10 received then response length is set to 136, else 48 bits. For data commands data expected bit is set, read/ write data is also identified from the commands. Data bus width 1bit or 4 bit mode is set if Acmd 6 is received . The functionality is implemented from the card specifications [1]. Now if the captured direction bit is 0 , starts capturing first 6 bits response index, argument either 32 bits or 120 bits set by command, 7 bits crc, calculates crc for direction, index ,argument concatenated and compares with the received crc sets the *crcValidated* bit to 1 if matched, Finally checks for end bit if 1 gives message response end bit detected, else gives message response end bit failed. Again starts monitoring for start bit.

Monitor data0-data3 lines for read and write data and crc status Task:

In the absence of data on data lines all lines are pulled high, data starts with start bit 0,16 bit crc followed by the data, finally end bit 1 comes last and the data line pulled high.

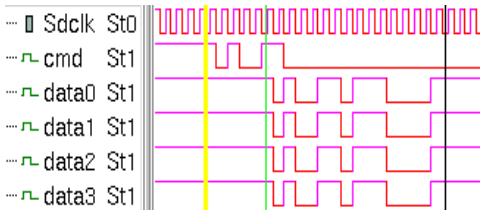

Figure 3 data signals on 4 data lines

The figure 4 shows the flow chart for monitoring the data on the four data lines as the data lines are connected to pull up in the absence of data ,

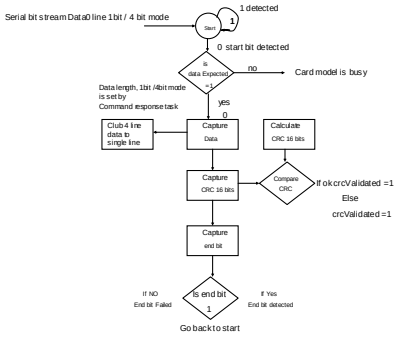

Figure 4. Flow chart Monitor data0-data3 lines for data and crc status

The data monitor task Starts looking for a 0 (low start bit) on the data line , checks for data expected bit, If high starts capturing 1bit/ 4 bit data , depending on the 1 bit or 4 bit mode , data length to be captured is set by command response task. Once data capture over starts capturing crc 16 bits for every data line, calculates the crc , and compares with the received crc set *crc Validated* bit to 1 if both are matched , for 4 bit mode after capturing data 4 lines of data is clubbed together to get the single line data. Identifying read or write data is done from the bit *is write data* set by the command response task. As after capturing data all data lines are connected to pullup, for write data secure

digital card sends back either 010, or 101, preceded by start bit followed by end this is also captured by the task.

Driving response Task: The flow chart for the drive response task is shown in figure 5 . After receiving the transaction from the card model checks the data type , if it is 01 then starts driving start bit, response, calculates crc and drives , finally drives the end bit, and go back to wait until transaction received.

Drive Read data and CRC status : The flow chart for the drive read data and crc status is as shown, The task stays in the loop until the transaction received from the card model , after receiving transaction checks the data type if 10 starts driving the start bit for read data , 1bit mode or 4 bit mode is set by the command response task, drives the read data, the data length to be driven comes along with the transaction, calculates the crc for the data , drives the crc, end bit and comes back to the begin of the task. If data type received is 11 then drives the start bit then crc status and end bit on to the data0 line for both 1 bit or 4 bit mode.

SD Device Model: SD Device Model is developed by reusing SDIO transactor, The transactor is used to drive the response for the command from the host it also drives the Read data and receives the write data. This transactor also implements the SD_Card_Physical_Layer_v2.00 [1]. In this model the card registers are defined and updated, also the state of the card is maintained according to the specifications [1]. This card also implements the memory read and write operations.

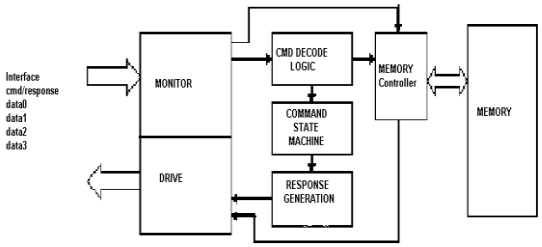

Figure 5. Block Diagram of Card model

Card Model receives the command from the monitor task of IO transactor decodes it , generates the response for the command received and send the transaction to the drive task of IO transactor to drive on to the bus. The state of the command state machine gets updated according to the received command.

If command received for write operation memory controller gets address block length from Command Decode Logic block, writes the data received in to the memory. It checks the *crcvalidated* bit in the transaction and sends the crc status to the drive task. hen card model receives the command for read operation memory controller gets address, block length from command decode logic sends the Data transaction to the drive task of IO transactor.

Verification of functionality. Transaction Level abstraction is used for verification as this is much more efficient from both the performance and debugging points of view in terms of simulation time as compared to performing all of the low level bit _twiddling operations on individual's signals in RTL. A properly written transaction-level model only takes a fraction of the time it takes to write an RTL model, thus enabling the design and the verification team to work in parallel. It also executes orders of magnitude faster than the RTL model, allowing the

verification environment and tests to be developed and debugged much faster. Design methodologies that require or produce a transaction-level model of the design often use System Verilog as the modeling language, as it provides all of the necessary high-level constructs that make writing transaction-level models more efficient.

III. Simulation Results:

The results of simulation are stored in the log file with reference to the simulation time for different test cases. Also wave forms of the signals are dumped which are further useful for analyzing the failure cases. Simulation results for some of the test cases are shown in the Figure below.

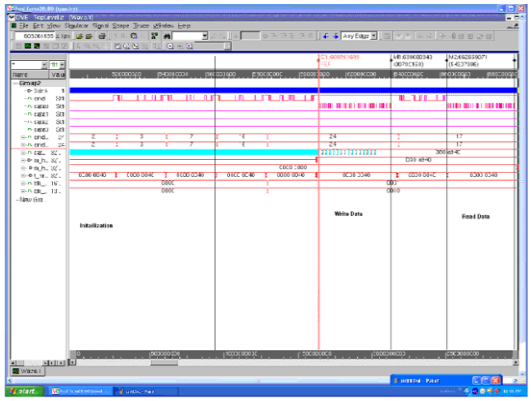

Figure 1.12 Single block write and read form the SD card

IV. **Conclusion:**

The powerful feature in System Verilog are exploited to make our verification environment more manageable and reusable, and the scheme of constrained-randomization stimulus generation can help us reduce a lot of redundant man efforts without any quality damage. The most of the bugs are discovered in randomization phase itself, and only a small amount of bugs occurred in corner cases are detected by the directed tests. The most importunately the information generated from our verification environment helps the design engineers to fix these bugs quickly thus helps in reducing the time to market.

Raising the level of design abstraction can be achieved by means of transaction level models (TLMs). Such models communicate with each other at the transaction level. Working with TLMs facilitates architectural exploration and allows system engineers to perform hardware/software integration ,validation , and co-verification.

Though most of the verification tasks are accomplished here, still some important improvements are to be added in the future. Assertion statements are to be added separately to help monitor components for all kinds of sequences that appear on the interface and also some critical timing relationships, which will help to ensure whether the interface is compliant to its own specifications or not.

V. References :

[1] SD_Card_Physical_Layer_v2.00. www.sdcard.org

[2] SD Host controller Standard Simplified Spec. www.sdcard.org

[3]System Verilog LRM http://www.eda.org/sv/SystemVerilog_3.1a.pdf

[4] Verification Methodology Manual for SystemVerilog anick Bergeron, Eduard cerny, Alan Hunter, Andrew Nightingale Springer Publisher 2005 Edn.

[5] Writing Test benches using System Verilog Janick Bergeron , Springer Publisher 2006 Edn . Advanced Digital Design

[6] Advanced Digital Design with VERILOG HDL. Michael D. Ciletti

[7] Functional Verification of HDL Models. Janick Bergeron 2nd Edition