International Journal of Research and Review
Vol. 9; Issue: 11; November 2022
Website: www.ijrrjournal.com
Research Paper *E-ISSN: 2349-9788; P-ISSN: 2454-2237*

Design, Verification and Testing of Comprehensive Serial Peripheral Interface

Arjumanth Farraj R[1], Dr. Kiran V[2]

[1]Master's student, Department of Electronics and Communication Engineering, RV College of Engineering, Bengaluru, India
[2]Associate Professor, Department of Electronics and Communication Engineering, RV College of Engineering, Bengaluru, India

Corresponding Author: Arjumanth Farraj R

DOI: https://doi.org/10.52403/ijrr.20221107

ABSTRACT

Synchronous serial interfaces are frequently employed to offer cost-effective board-level interfaces between various devices, including microcontrollers, DACs, ADCs, and others. Components that are compatible with SPI and Microwire are made by numerous IC makers. An interface called Serial Peripheral Interface (SPI) makes it easier to send synchronous serial data. In master/slave mode, data frames are initiated by the master device. With a separate slave choose line, multiple slave devices are permitted. It is necessary to guarantee that the given hardware design operates as intended and produces the desired outcomes. The design phases would need to be repeated after the functional flaws were discovered. Verification is thus a process that is time and money efficient. Due to this, a strong testbench structure is required, one that includes large broad verification elements that are easily extensible across designs and are extensively reusable. To gain good control over stimulus creation, functional coverage, and other factors, an industry standard verification technique is needed. The primary goal of this work is to design the Master SPI Core and use System Verilog to validate the code. Serial Peripheral Interface of symmetrical structure was synthesized using Cadence genus, and then simulated using Questa IDE. The testing of the SPI core was done by inserting the scan chains using the Cadence Genus tool and finally the code coverage of the SPI core was captured using the Questa IDE.

Keywords: SPI, Coverage, Protocol, I2C, Functional Verification

INTRODUCTION

To establish a communication within the system, protocols are utilised. Motorola created the SPI Protocol. Comparing SPI Protocol implementation to other protocols, it is straightforward. It is synchronous and offers full-duplex mode operation, allowing for simultaneous data transmission and reception. It is employed for close-range communication. When compared to the I2C protocol, the data transfer speed is far faster—about 10Mbps—and there won't be any data loss. It has many benefits over the parallel protocol, including high speed, a straightforward design, and less pin usage. However, the Slave device is obligated to comply with the Master device's instructions at all times. Combining a single Slave with a single Master is the simplest configuration for the Serial Peripheral Interface (SPI). Functional verification is a technique used to ensure that our design adheres to a certain specification. The protocol is designed using the Verilog and System Verilog hardware description languages. It is designed in accordance with the data sheet. System Verilog is used for the design and verification of the protocol whereas for testing of the protocol by inserting the necessary scan chains was implemented using Verilog in Cadence genus.

LITERATURE REVIEW

Priyanka B et al. [1] has developed which involves 8-bit of the data transfer and all necessary incorporates features that are required for modern applications like Application Specific Integrated Circuit or System on chip. Maximillian Holliday et al. [2] has mentioned that each node on a communication bus should have a straightforward external circuit built so that, in the event of a device failure, the node is immediately isolated. The security of both the bus is maintained through autonomous component isolation, needing no essential feedback or network interface operations. The overall effectiveness of I^2C and SPI separation circuitry in sustaining bus stability in the case of accessory hardware failures have been modelled, constructed, and empirically tested. Kiran.V et al. [3] focused on converting the SPI to I2C communication using 'read' and 'write' instructions. Additionally, this design supports efficient low-power techniques like clock_gating_insertion, harmonic current gated, and the use of multi-threshold cells to minimize leakage power.

MATERIALS & METHODS

SPI BUS PRINCIPLE

Unlike other protocols, SPI only has one master device and many slaves. There are four signaling pins used in the SPI bus protocol :

- Master Out Slave In
- Master In Slave Out
- Serial Clock / Master clock
- Slave Select pin / Chip Select pin
- Here, each signal pin's operational functionality is described.
- Serial clock / Master clock: - Only the Master can control the clock signal that is provided to the slave or slaves by this pin. This pin, however, is still in the idle position. If no operation is carried out, the device is inactive (tri-state).
- Slave Select / Chip Select: - Which Slave the Master module will communicate with or transfer data to is determined by this pin.
- MOSI (Master Out / Slave In) : - This functions as the Master output pin as well as the Slave input pin. This pin is used to transmit data from the Master module to the Slave module. It has one direction, like a pin.
- MISO (Master In / Slave Out) : - This pin serves as a master input pin as well as a slave output pin. This pin is used to transmit data from the Slave Module to the Master Module. Additionally, it is a unidirectional pin.

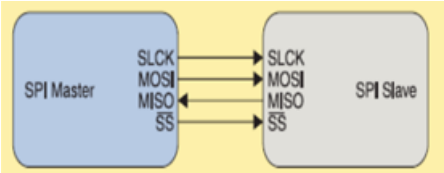

Fig1: SPI master with a single slave

DESIGN PRINCIPLES

In the design we use the ports – '*clk'* which is global synchronising signal, '*newd'* where user will specify to the system that it has new data that it wants to communicate to the DAC, RST which is active high type, Din is the input which we want to communicate to the DAC is of size 12 bits. The series of output pins which are required are sclk, CS is the chip select which is of active low type and MOSI.

For our design, a state machine is utilized.

1. First state is to check whether we have any new data available.
2. Second state is to start the transaction.
3. Third state is to send the data serially to the DAC.
4. Forth state is to complete the transaction.

countc is use to generate the clock of lower frequency assuming that we have a clock of 100MHZ and the sclk should be 1MHZ where the assumed clock ticks that we need to wait is 100 and half clock period is 50. If

reset RST is 0 then both countc and sclk are both 0. When countc is less than 50 which is half clock period, we'll be incrementing countc by 1 else we will be making countc as 0 and complementing sclk. Once 50 clock ticks are reached sclk will be complemented and countc will be 0 where we will be waiting for another 50 clock ticks. Total clock ticks are 100, the clock is operating at 100MHZ and this gives 1MHZ frequency. This is how we generate the clock divider.

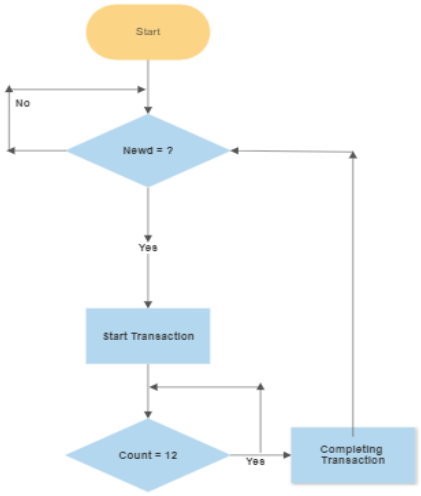

Fig2: Flowchart of the design

The state machine of the DAC will be working on sclk. When reset is high, we'll be making both CS and MOSI to their initial value i.e.1 & 0. It'll be in the idle state untill user adds a new data to the system. If new data (newd) is 1 then we go to the next state and we also make chip select 0 which means we started a transaction as soon as we sense new data else we stay in the same state.

In the send state, if the count is less than 11 then we'll be sending the data serially else if the count is less than equal 0 it remains in the idle state. This is the working of serial peripheral interface.

DIGILENT PMOD DA2

Through an SPI-like protocol, the Pmod DA2 connects with the host board. Users can send a string of 16 clock pulses on the Serial Clock line by lowering the voltage on the Chip Select line (SCLK). On the last 12 clock pulses, the data is transmitted with the most significant bit (MSB) coming first.

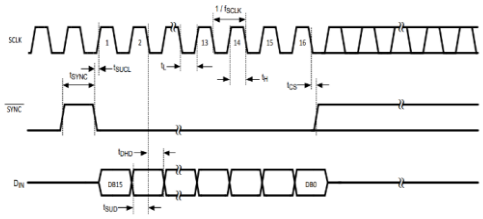

Fig3: Input Register Contents of PMOD DA2

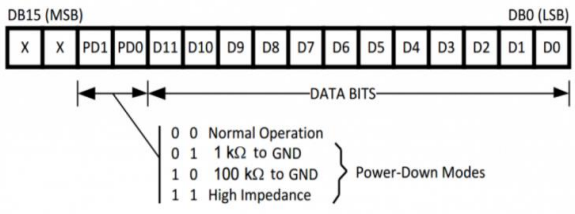

Fig4: DAC121S101 Timing

Advantages of SPI:

1. Full duplex communication is the protocol's standard configuration.
2. Push-pull drivers offer fast speed and exceptional signal integrity (as compared to open drain). higher throughput compared to I2C.
3. Adaptability of the protocol to the bits sent in full.
4. Extends the capability beyond 8_bit words.

Disadvantages of SPI:

1. No in-band addressing is permitted on shared buses; out-of-band chip select signals are necessary.
2. The slave does not use hardware flow control (nonetheless, the master can postpone the following clock edge to reduce the transfer rate.).
3. No acknowledgement of the slave hardware.
4. Only allows for one master device.

SCAN INSERTION

By incorporating more multiplexers into the design to verify the design, scan insertion

aims to transform sequential cells of the original design into scan cells. Multiplexer will separate either test input or data. Each node can be made by this process. Design is more observable and subject to control. Every scan using stitched cells, a scan chain is created.

VERIFICATION PLAN

System Verilog Hardware Verification Language serves as the foundation for the verification plan. Constraint random coverage driven verification is the method used for verification. The verification phase of verification is monitored for progress and completion using the verification plan, which serves as the central point for outlining precisely what has to be checked. It will functionally test the design in all conceivable special circumstances. The structure of the verification environment is contained in the verification plan. The following factors are taken into account when building architecture based on the project needs*:*

1. Reusability, Is it an IP for verification.
2. The categories of blocks the verification language is capable of handling
3. Controllability in the creation of the stimulus, etc.
4. Building the Verification environment comes next.
5. Verifying the DUT in the created environment is the last step.

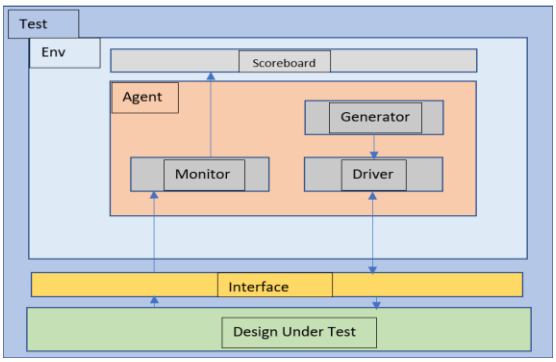

Fig5: Verification Environment

The following are the blocks required for the verification environment.

1. Interface Block

We include each port that we have in the design inside the interface block.

2. Transaction Block

Din is the only input for which we want to produce random values (input data). Every time the user wants to provide new data, the value must be 1. The other method involves selecting a value at random from those that we have for new data and applying it to the DAC. The design itself will be the source of the clock (sclk).

3. Generator Block

The generator block consists of the transaction, mailbox for receiving the transaction and followed by 3 events – '*done*' which tells that the generator has completed sending requested number of transactions, '*drvnext*' to indicate that the driver has completed its work and '*sconext*' to indicate that scoreboard has completed its work.

4. Driver Block

Driver consists of an interface. Transaction is required inside the driver and along with this mailbox is needed to receive the transactions from the generator. The data which we get from MOSI will be sent to the scoreboard which is a bit-by-bit data (12 bits). This data is nothing but the reference data from the driver. Another mailbox is needed to send the data from the driver to the monitor.

Add a reset task to reset the system, all the signals will be in its initial values. Create another task to generate the transactions where we'll be creating the actual transactions with the help of pseudo random generator that will be applied to the Design under Test (DUT). This data will also be used by the scoreboard for the comparison. This will continue until chip select goes high indicating the *end* of transaction.

5. Monitor Block

Monitor also consists of an interface. We require a mailbox that allows us to send the

data to the scoreboard. In the driver block we are sending the data serially on a MOSI pin so that we can use it for comparison where we have a task to enable the start of transaction by making use of the chip select signal. SOT is active by making chip select low before we start collecting the samples and finally the EOT (End of Transaction) is triggered by chip select high i.e. (CS = 1'b1).

6. Scoreboard Block

Scoreboard will be receiving 12 bits of data from both driver and monitor and therefore we have 2 mailbox declarations inside the scoreboard block. We also require 2 data containers for storing the data from the driver and monitor. Add a constructor to the scoreboard block which is capable of handling 2 mailbox arguments.

The main task of a scoreboard is to compare the data received from the monitor with the reference data from driver and trigger an event to allow generator to send the stimulus.

7. Environment Block

We have to instantiate all the above blocks i.e. (Generator, Driver, Monitor & Scoreboard) in the environment block followed by declaring the constructor for all the blocks. Specify the connection of interface for the blocks which means that whatever interface that we declare in the environment block will be connecting to an interface argument that the user has specified while adding a constructor to the environment block.

8. Testbench Top block

With the help of the testbench top block we'll be able to verify whether the data we are sending from the driver is same which will be key for the scoreboard development. We have to create the instance of the generator, driver, monitor followed by creating events – *'drvnext'* , *'sconext'*, *'done'* . Mailbox is needed in the block for communication between generator – driver (mbxgd), driver – scoreboard (mbxds) & monitor – scoreboard (mbxms). Declare the constructors and make the interface connections that is used in all the three components followed by running the main task.

STATISTICAL ANALYSIS

The statistical analysis was carried out by performing synthesis in Cadence genus using 45nm GPDK technology where total number of cells for the spi master instance is 122 and consumed a leakage power of 79.423nW, dynamic power of 81350.51nW and a total power of 81429.934nW.

RESULT

The verification of the protocol was done using both Questa and Aldec Riviera Pro 2022.04

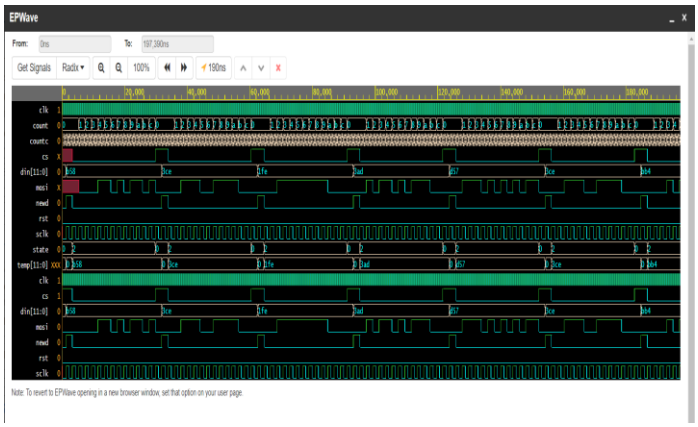

Fig6: Simulation Results - Aldec Riviera Pro 2022.04.

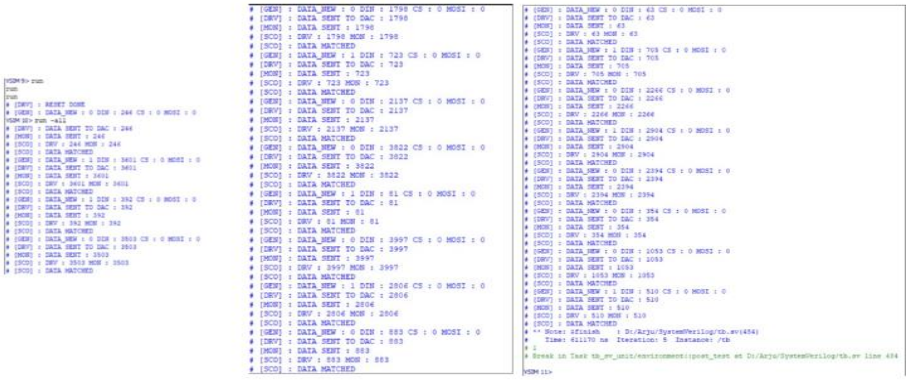

Fig7: Simulation Results - Questasim 10.4e

The DFT scan chain insertions were implemented using Cadence genus tool.

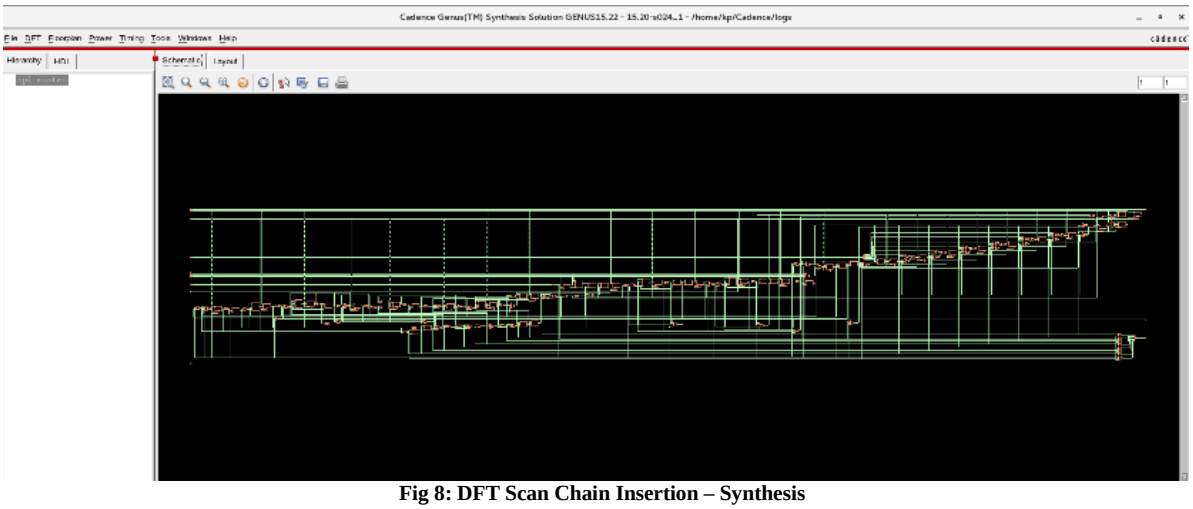

Fig 8: DFT Scan Chain Insertion – Synthesis

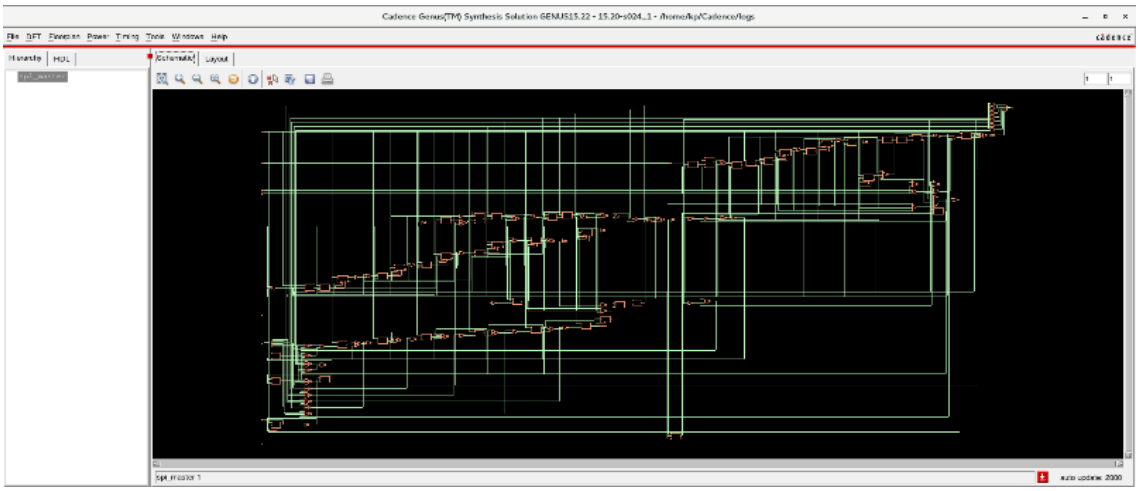

Fig 9: Synthesis of SPI master – Cadence genus

TABLE 1. Power Report

Power Report				
Instance	**Cells**	**Power (Leakage)**	**Power (Dynamic)**	**Power (Total)**
spi_master	122	79.423nW	81350.51nW	81429.934nW

TABLE 2. Area Report

Area Report			
Type	**Instances**	**Area**	**Percentage (Area)**
sequential	24	1500.206	53.2
inverter	13	86.486	3.1
logic	85	1230.768	43.7
physical_cells	0	0.00	0.00

Coverage Report

Scope	TOTAL	Assertion
TOTAL	100.00	100.00
tb_sv_unit	100.00	100.00
generator/run/#ublk#180432228#151	100.00	100.00

Fig10: Coverage Summary by Instance

Total Coverage:					100.00%	100.00%
Coverage Type	**Bins**	**Hits**	**Misses**	**Weight**	**% Hit**	**Coverage**
Assertions	1	1	0	1	100.00%	100.00%

Fig11: Local Instance Coverage Details

Total Coverage:					100.00%	100.00%
Coverage Type	**Bins**	**Hits**	**Misses**	**Weight**	**% Hit**	**Coverage**
Assertions	1	1	0	1	100.00%	100.00%

Fig12: Recursive Hierarchical Coverage Details

DISCUSSION

When we observe the above waveforms, from figure 7- the first thing which the design wants to do is the RESET done, generator sends the randomized data and the same data is applied to the design under test. After receiving bit by bit data in the monitor we are capturing the same data and hence the data from both driver and monitor is checked for its data consistency and it found out to be same.

On analysing the design under the test data from figure 6 displays the transactions with random transactions on MISO (Master In / Slave Out) and MOSI (Master Out / Slave In), reset, clock and new data and we are capturing the right data as we were able to capture using Questa.

CONCLUSION

The Serial Peripheral Interface (SPI) was successfully designed with a Single Master and Single Slave configuration, demonstrating that it operates in Full Duplex Mode. The designed SPI was also verified using System Verilog's constrained random methodology and a 100% Code Coverage report with all functional coverage.

Acknowledgement: None

Conflict of Interest: None

REFERENCES

1. M. Holliday, Z. Manchester and D. G. Senesky, "On-Orbit Implementation of Discrete Isolation Schemes for Improved Reliability of Serial Communication Buses," in *IEEE Transactions on Aerospace and Electronic Systems*, vol. 58, no. 4, pp. 2973-2982, Aug. 2022, doi: 10.1109/TAES.2022.3142713.
2. Vinilanagraj and Kiran.V, “Design of SPI to I2C Protocol Converter and Implementation of Low Power Techniques,” in *International*

Journal of Advanced Research in Computer and Communication Engineering Vol. 2, Issue 10, October 2013

3. Z. Zhou, Z. Xie, X. Wang and T. Wang, "Development of verification envioronment for SPI master interface using SystemVerilog," *2012 IEEE 11th International Conference on Signal Processing*, 2012, pp. 2188-2192, doi: 10.1109/ICoSP.2012.6492015.
4. Y. Guo *et al*., "A SPI Interface Module Verification Method Based on UVM," *2020 IEEE International Conference on Information Technology, Big Data and Artificial Intelligence (ICIBA)*, 2020, pp. 1219-1223, doi: 10.1109/ICIBA50161.2020.9277156.
5. "Design and Implementation of a Reused Interface" 978-0-7695- 3887-7/09/$26.00 ©2009 IEEE.

How to cite this article: Arjumanth Farraj R, Kiran V. Design, verification and testing of comprehensive serial peripheral interface. *International Journal of Research and Review*. 2022; 9(11): 46-53.
DOI: *https://doi.org/10.52403/ijrr.20221107*

******