Verification of Spread Spectrum Correlator

Akhhiila Gurram, Navika Iyer, and Lili He, *Member, IAENG*

***Abstract*- Since the early 1940s, military communication systems, regular networking, and wireless communication systems use spreading techniques. One such technology is Spread Spectrum, where the waveform is modulated by spreading the bandwidth using a pseudorandom noise, thereby encrypting the signal. The project verifies a spread spectrum correlator and checks for its verification properties. It aims to check the correlators through 14 different designs under tests (DUTs) while correlating to the gold code (PRN), frequency, and phase. It further aims to check the correlation properties with frequency errors and phase errors. This project will allow upgrading systems with spread spectrum techniques to send and receive information with finer encryption. The project aims to provide full functional verification of the design and check if the design meets the specification requirements. The results of the project show plot of correlation characteristics with frequency and phase errors.**

***Index Terms*— Gold Code, Pseudorandom Noise, Spread Spectrum Technique, Universal Verification Methodology**

I. Introduction

The Spread Spectrum technique is used in communication systems by increasing the transmission bandwidth to secure and reliable communication [1]. Narrowband signals transmit information; however, they are intercepted easily. Any signal in the same band could easily jam the narrowband signals leading to loss or delicate information corruption. If there is a low S/N ratio due to interference, communication performance (C) increases the bandwidth by injecting a higher frequency signal.

The Shannon and Hartley channel-capacity theorem explains Spread Spectrum using equation [2]:

$$C = B \times Log_2\left(1 + \frac{S}{N}\right) \quad (1)$$

Where C stands for Channel Capacity (in bits/second), B stands for required channel Bandwidth (in Hz), and S/N is the Signal to Noise power Ratio. To apply the spread spectrum technique to any channel; before the receiver antenna, insert the spread spectrum code; this is known as the spreading operation leading to the diffusion of information to a larger bandwidth. The Spread Spectrum technique applies on top of another modulation technique such as BPSK[3]. Verification is the process of determining the correctness of a design. It checks if the design developed is according to the specifications before process installation and optimization.

Akhhiila Gurram is a student at San Jose State University, San Jose, CA. USA. phone: (925)-520-5274. email: akhhiila.gurram@gmail.com
Navika Iyer is a student at San Jose State University, San Jose, CA, USA. phone: (408)-797-4212, email: navika.iyer20@gmail.com

The design verification process is not simply linear, but it is a continuous process to test until it meets the requirements.

Universal Verification Methodology is a standard verification methodology consisting of class libraries required to build reusable test benches for the System Verilog-based environment. It works on a message-based programming model where the UVM components communicate with messages.

II. Methodology

UVM is a standard methodology used in verification to test functionality, timing, and performance. Types of verification: Simulation, assertions, formal, semi-formal, and HW/SW. Verification of spread spectrum correlator tests functionality by building class libraries to test the System Verilog-based environment. It uses messages, data collection, and class objects to communicate with multiple UVM components through TLM interfaces, ports, exports, and imps. UVM classes have UVM transactions, components, and objects where each of them performs different tasks. The verification runs in 9 different phases.

A. UVM Components

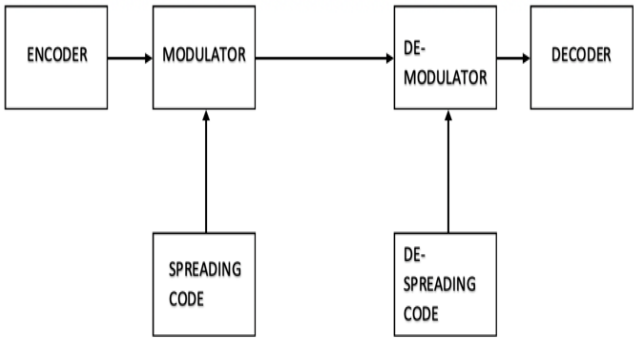

Fig. 1. Block diagram of Spread Spectrum Communication

Major UVM components are sequencer, driver, monitor, scoreboards, agent, environment, test, and top module, and each of these components extends to a base class. UVM Sequencer sends the sequence items generated by the sequence to and from the driver with a built-in export called seq_item_export. UVM driver collects the sequence items sent by the sequencer and pushes them into the design under test. It uses a configuration database to access the virtual interface, which connects to the Design under test. During the driver's run phase, the sequence item fields map to the virtual interface handle. Therefore, the sequence items convert to pin wiggles and apply to DUT. It continuously drives the interface signals to the DUT throughout the simulation time.

ISBN: 978-988-14049-2-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

UVM Monitor collects the signal level activity from the design through the virtual interface, converts it to transaction items, and sends it to the other UVM components. It uses a configuration database to access the virtual interface connected to the Design under test. The data transfers through TLM analysis ports. UVM Scoreboard component checks the functionality of the DUT, whether the design is behaving correctly and according to the requirements or not. It receives the data from the monitor through analysis ports and stores the data in analysis FIFOs.

UVM Agent is an abstract container class used to emulate and verify the Design under Test. It encapsulates a driver, sequencer, and monitor. All the components are connected using TLM interfaces. The connect phase is the most critical phase as it connects all the agent's child classes.

UVM Environment is a container class for multiple agents and other components like scoreboards, top-level monitors.

UVM test contains the environment which, acts as a container class. The top module contains all the components like the test, packages, interface, and DUT. Import UVM packages using "import uvm_pkg::*:" and run_test is declared invoking to build the UVM test component responsible for building the UVM hierarchy.

B. UVM Phases

There are nine phases in UVM. Build phase, connect phase, end of elaboration phase, the start of simulation phase, run phase, extract phase, check phase, report phase, and final phase. These phases are essential to build, connect and run the UVM components to act in a synchronizing mechanism. The build phase is top-down, while the rest of them are bottom-up. The extract phase, check phase, report phase, and final phase are together called clean-up phases. The run phase is the only phase with a method-type task while the rest of them use functions [4-5].

C. UVM Objects

UVM object, unlike the UVM component, is not required to stay alive during the entire simulation. Sequence and Sequence items are the only two objects, and they extend to a base class. UVM Sequence and Sequence item classes are built with macros, registered with a factory, and provided stimulus, triggered during UVM phases. The classes use item methods like create (), copy (), clone (), print ().

III. Spread Spectrum Technique

Spread Spectrum's idea is to increase the bandwidth of the signal transmitted but, at the same time, maintaining the power level, and this leads to indistinguishable peaks in a spectrum, which looks like noise. Since the information cannot be differentiated from noise unless the receiver has proper decoding techniques, it is tough to intercept or jam these signals. The spread spectrum has emerged as one of the most secure ways of transmitting the information.[6]

The 6 components of the block diagram are explained as follows

A. Encoder

The encoder is responsible for converting messages into bits (or signals). It removes redundancy to compress the message and keep in mind that the message sent will encounter various noise types. [7]

B. Modulator

The modulator is responsible for converting the message bits into signals to send through a channel. There are different types of modulation techniques used in the industry, such as Phase Shift Keying (PSK), Frequency Shift Keying (FSK), and Amplitude Shift Keying (ASK), among others.[7]

C. Channel

The channel is the physical medium of message transportation. A channel also includes attenuation, noise, and interference to the message signal.

D. De-modulator

The de-modulator is responsible for converting the signals received through the channel back to bits. The demodulation technique used is the same as the modulation technique used to transmit the signal at the sender side.

E. Decoder

A decoder is finally responsible for converting the bits into understandable and readable messages as sent by the sender.

F. Spreading and de-spreading code

This additional code is added to the signal while modulating and demodulating respectively to spread the signal's bandwidth and make it immune to interceptions.[8]

IV. The Design Under Test

Of the six inputs, Din is the input data signal, while the address register is to determine the use of Din. A strobe signal is a one-bit signal used to determine when to write or check the address register. The fourth input signal is Samp and it is used to get the sample value.

The last input signal is push_samp which determines when the Sample value is available.

There are four outputs; Dout is the same value as Din. The second output is the correlator which is the actual output, and the third output is just a one-bit flag which is the push_correlator which signifies that the correlator output is ready.

Table I. Design Specifications

Name	Size	Direction
Din	32 bits	Input
Addr	4 bits	Input
Strobe	1 bit	Input
Sync	1 bit	Input
Samp	12 bits	Input
Push_samp	1 bit	Input
Dout	32 bits	Output
Corr	32 bits	Output
Push_corr	1 bit	Output

The first module is called the Register Select Module. As the name suggests, the responsibility of this module is to select the register for input data. The first signal checked is

ISBN: 978-988-14049-2-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

push_Samp. If push_Samp is high, the sample input is then read.

The address signal has only four values considered in the design. If the address is 0, 4, 8, 12, that means the input data is equal to frequency, phase, S1, S2 register, respectively. S1 and S2 registers are responsible for selecting tapping bits for polynomials in the Gold Code. If the sync is high in the accumulator module, it sets the accumulator equal to phase.

If the push_samp is high and sync is 0, then the accumulator keeps adding frequency to itself. Once the accumulator value crosses 100 Million, 100,000,000 subtracted from it, and raise a flag to indicate the Gold Code to advance to the next chip.

Every time the accumulator crosses 100 million and raises the flag, there is a counter to increment in the gold code module. The Gold Code [9-10] module consists of two polynomials. Both of them are set to 10'b1111111111 when the sync signal is high. Four temporary registers aid in the calculation of the gold code output. The 10th bit is taken from the first polynomial and stored in the first register. Then 2 bits are taken from the second polynomial, XORed together, and stored in the second register. The output of the gold code is the modulo two addition of register one and register 2, and the gold code output becomes 0 [11]. Once the accumulator raises its flag, move on to the next chip. The two taps on polynomial 1 are modulo two added together, and the polynomial is left-shifted by one and then repeat the same process.

Once the counter reaches 1023, the push_corr flag goes high. Gold code output determines whether the sample value is added or subtracted to the correlator value and gives the final output. Din directly goes into Dout.

V. UVM TESTBENCH

A. Interface

The interface has input and output signals with a clocking block, and the modport is a named port where the direction of the signals is specified. The interface acts as a medium between the design and the uvm testbench to exchange the signlas.

B. Sequence Item

The class is registered with the factory “uvm_ object_ utils” and constructed using a class constructor. The items are created in the body using type_id:: create. The body of the sequence calls for multiple tasks. These tasks include directed test case by specifying the Din value as 100 Million, push samp = 1, Samp = 1388 and address= 4'h0, 4'h4, 4'h8, 4'hC, and strobe = 1.

C. Sequencer

The class registers with the factory UVM “component utils.”

D. Driver

The driver class's build phase has the configuration database to access the design through a virtual interface. Failing to connect will show a fatal error. The connect phase connects the ports. The interface signals take sequence items into at the positive edge of the clock. The sync signal is set high for every alternative clock cycle, and the driver uses its inbuilt seq_item_port to get signals from the sequencer and item.done method use the items[11].

E. Input and Output Monitor

The monitor's build phase has the configuration

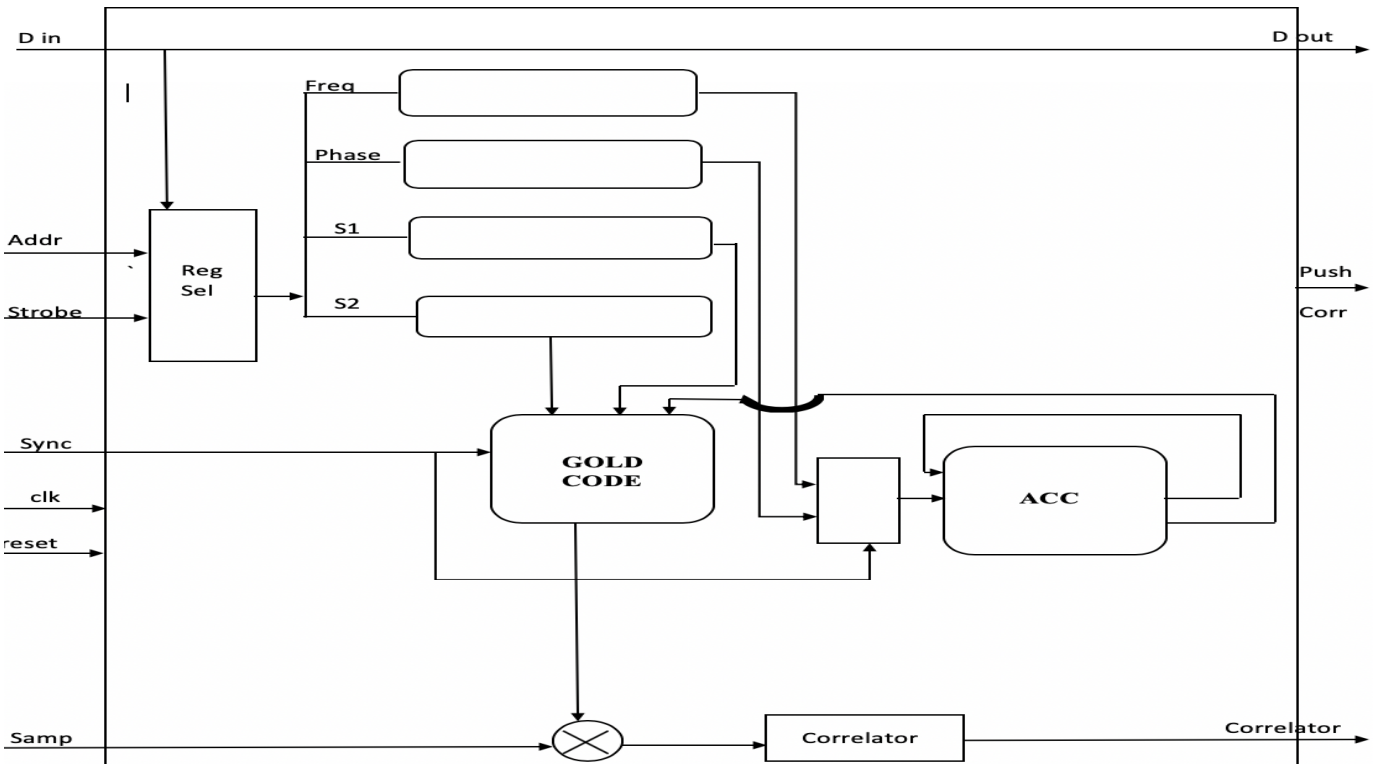

Fig. 2 Block Diagram of Spread Spectrum

ISBN: 978-988-14049-2-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

database, so the monitor accesses the design through a virtual interface.

.

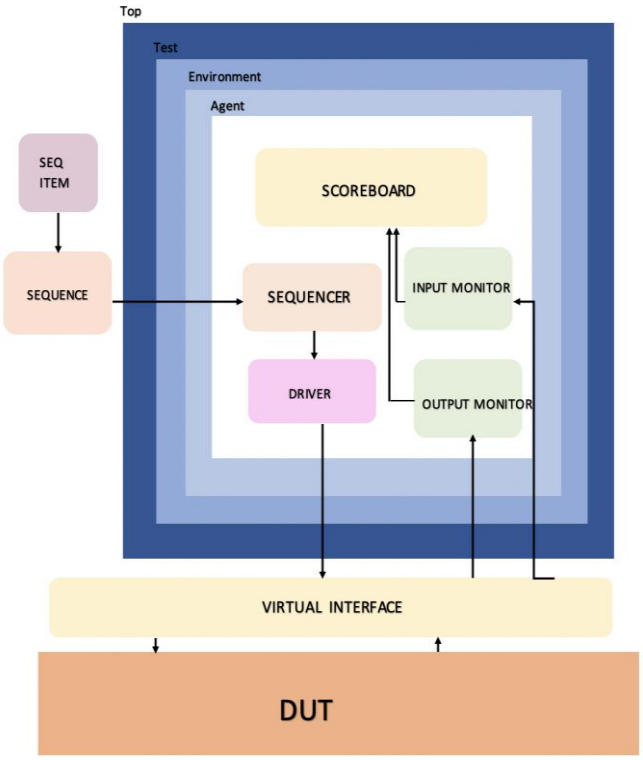

Fig. 3. UVM Testbench Block Diagram

Analysis ports use messages for communication between components. The task enabling the run phase has the input interface signals that push into sequence items at the clock's positive edge.

F. Scoreboard

There are five different scoreboards used in this project. Each checks a separate module of the test and sends information to each other through messages.

The four scoreboards send and receive messages to imitate the correct function of the Spread Spectrum Correlator. The reg select scoreboard performs a register select operation. The accumulator scoreboard performs the frequency addition and raises a flag. The gold code scoreboard checks the tapping polynomials in register S1 and S2 and performs the operations. The correlator scoreboard receives the output, which checks for the accumulator flag and increments the correlator counter. Once the counter reaches 1023, check the gold code output. According to the gold code output, the correlator adds or subtracts the previous value to the sample and gives it.

The 5th scoreboard, which is the checker scoreboard, compares the output achieved using the reference model to the actual output from the DUT. If it matches, the design passes the test; otherwise, it fails the test.

G. Agent

The agent connects all the child classes like the scoreboards, monitors, driver, and sequencers after the build phase achieves proper hierarchy.

H. Environment

The environment class connects to the agent and builds a hierarchy. The agent class item is created in the build phase using type_id::create. The connect phase connects its child classes, and it is displayed.

I. Test

The test class is the last in the hierarchy, which connects the environment and the sequence. The items for the environment and sequence class are created in the build phase using type_id::create, and the run phase has phase objections to signify the start and end of the test.

```
      (Specify +UVM_NO_RELNOTES to turn off this notice)

UVM_INFO @ 0: reporter [RNTST] Running test test...
CONNECT PHASE
UVM_INFO seq.sv(16) @ 0: uvm_test_top.env1.agt.seqr1@@seq1 [ : sequence : ] Starting patterns
******************** TESTING GENERAL SCENARIOS ********************
UVM_INFO checker.sv(50) @ 20582000: uvm_test_top.env1.agt.cmp [Passed] Recieved        1388 and Expected        1388
UVM_INFO checker.sv(50) @ 20591000: uvm_test_top.env1.agt.cmp [Passed] Recieved        1388 and Expected        1388
UVM_INFO checker.sv(50) @ 30812000: uvm_test_top.env1.agt.cmp [Passed] Recieved        1388 and Expected        1388
UVM_INFO checker.sv(50) @ 30821000: uvm_test_top.env1.agt.cmp [Passed] Recieved        1388 and Expected        1388
UVM_INFO checker.sv(50) @ 41042000: uvm_test_top.env1.agt.cmp [Passed] Recieved        1388 and Expected        1388
UVM_INFO checker.sv(50) @ 41051000: uvm_test_top.env1.agt.cmp [Passed] Recieved        1388 and Expected        1388
UVM_INFO checker.sv(50) @ 51272000: uvm_test_top.env1.agt.cmp [Passed] Recieved        1388 and Expected        1388
UVM_INFO checker.sv(50) @ 51281000: uvm_test_top.env1.agt.cmp [Passed] Recieved        1388 and Expected        1388
UVM_INFO checker.sv(50) @ 61502000: uvm_test_top.env1.agt.cmp [Passed] Recieved        1388 and Expected        1388
UVM_INFO checker.sv(50) @ 61511000: uvm_test_top.env1.agt.cmp [Passed] Recieved        1388 and Expected        1388
UVM_INFO checker.sv(50) @ 71732000: uvm_test_top.env1.agt.cmp [Passed] Recieved        1388 and Expected        1388
UVM_INFO checker.sv(50) @ 71741000: uvm_test_top.env1.agt.cmp [Passed] Recieved        1388 and Expected        1388
UVM_INFO checker.sv(50) @ 81962000: uvm_test_top.env1.agt.cmp [Passed] Recieved        1388 and Expected        1388
UVM_INFO checker.sv(50) @ 81971000: uvm_test_top.env1.agt.cmp [Passed] Recieved        1388 and Expected        1388
UVM_INFO checker.sv(50) @ 92192000: uvm_test_top.env1.agt.cmp [Passed] Recieved        1388 and Expected        1388
UVM_INFO checker.sv(50) @ 92201000: uvm_test_top.env1.agt.cmp [Passed] Recieved        1388 and Expected        1388
UVM_INFO /home/morris/uvm-1.2/src/base/uvm_objection.svh(1271) @ 95091000: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
UVM_INFO /home/morris/uvm-1.2/src/base/uvm_report_server.svh(847) @ 95091000: reporter [UVM/REPORT/SERVER]
--- UVM Report Summary ---

** Report counts by severity
UVM_INFO :   20
UVM_WARNING :    0
UVM_ERROR :    0
UVM_FATAL :    0
** Report counts by id
[ : sequence : ]     1
[Passed]    16
[RNTST]     1
[TEST_DONE]     1
[UVM/RELNOTES]     1

$finish called from file "/home/morris/uvm-1.2/src/base/uvm_root.svh", line 517.
$finish at simulation time                95091000
           V C S   S i m u l a t i o n   R e p o r t
Time: 95091000 ns
CPU Time:      0.680 seconds;       Data structure size:   0.2Mb
```

Fig.4. Simulation results of passed DUTs

ISBN: 978-988-14049-2-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

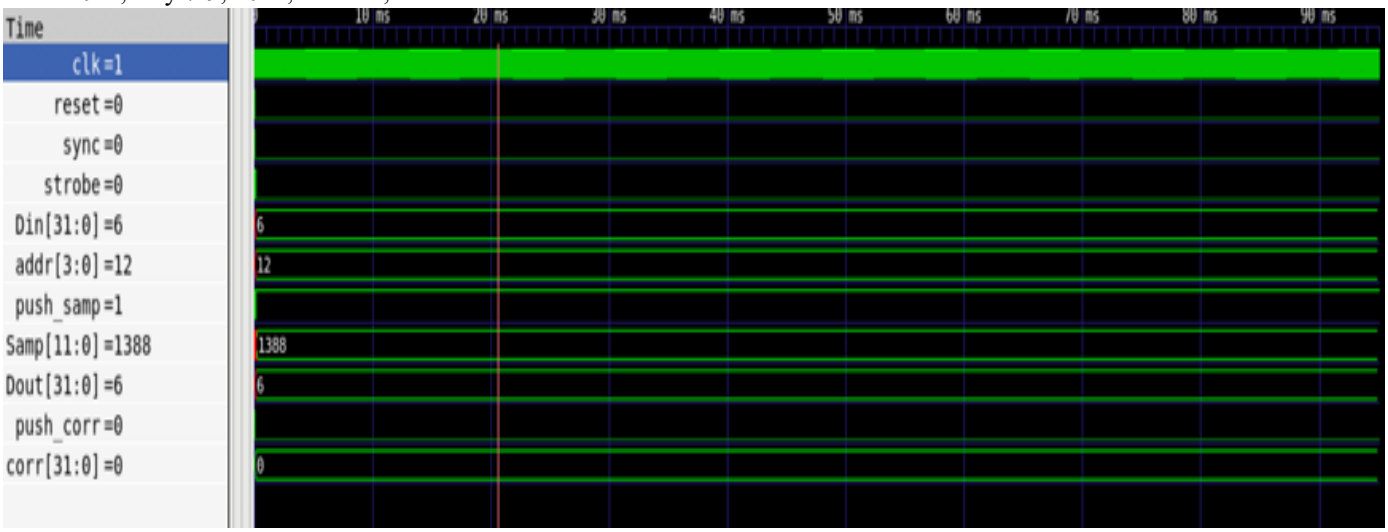

Fig. 5. Waveform of DUTS that failed the test

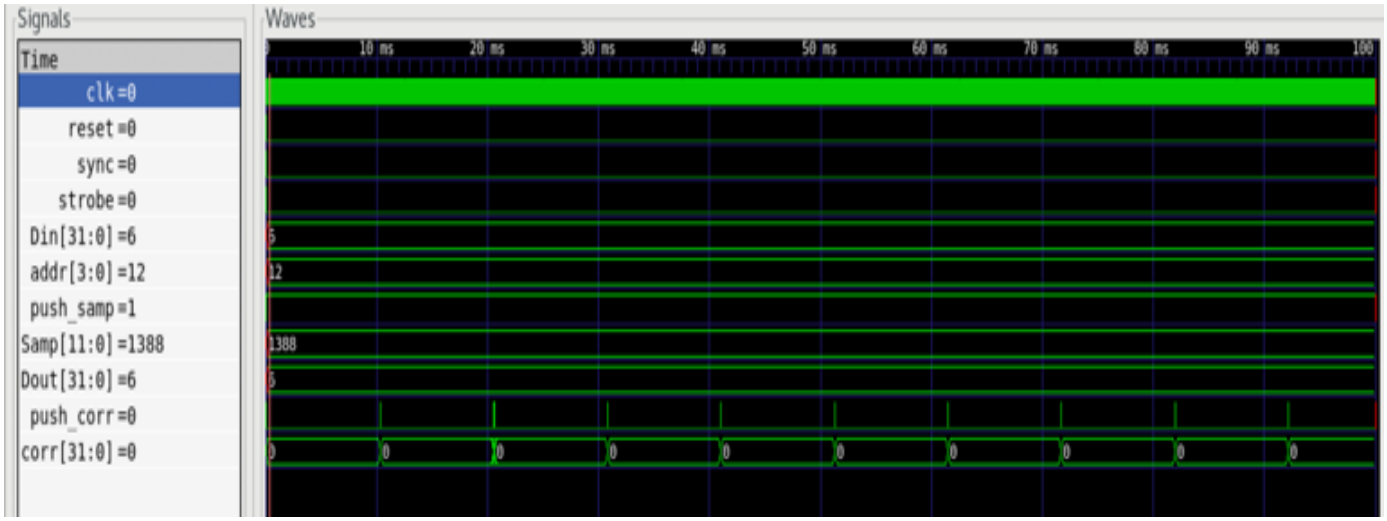

Fig. 6. Zoomed waveform of the DUTs

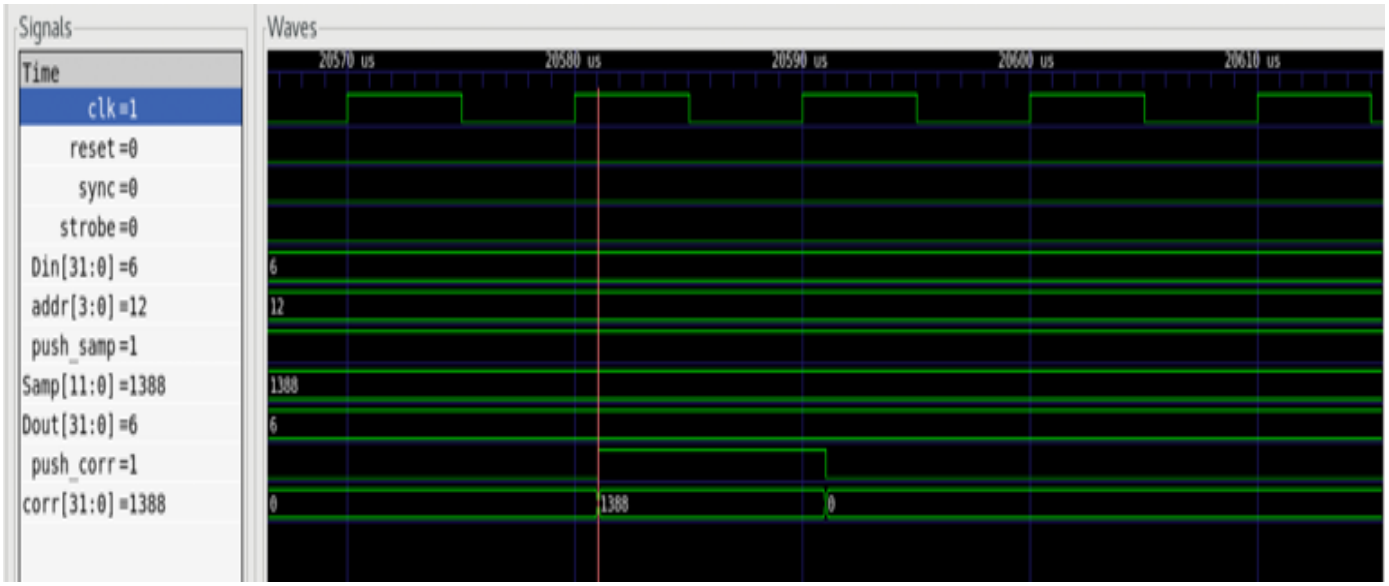

Fig. 7. Waveform of the DUTs that passed the test

J. Top Module

The top module encapsulates the entire uvm environment. We include all uvm component and object files at the top to be built and used. After initializing the clock and reset, provide the virtual interface handle, instantiate the DUT, and set the configuration database. For a better representation, create waveform files using dump files [12]. The virtual interface is present in configuration database allows us to access from uvm components like driver and monitor.

ISBN: 978-988-14049-2-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

VI. Results

14 DUTs were tested 7 DUTs failed the test and 8 DUTs Passed. This section shows screen captures of the simulation results as well as the waveforms achieved by DUTs.

Fig. 4 shows the simulation results of the DUTs that passed the test. The expected and received values match; therefore, the DUTs correlated correctly.Fig. 5 shows the simulation results of DUTs that failed. Fig. 6 shows the zoomed-in waveform of the DUT, which passed the testbench. When the push corr is high, push the value of the corr. Fig. 7 shows the waveform of the DUT till the end of simulations which passed the testbench. When the push corr is high, push the value of the corr, as seen in fig. 7.

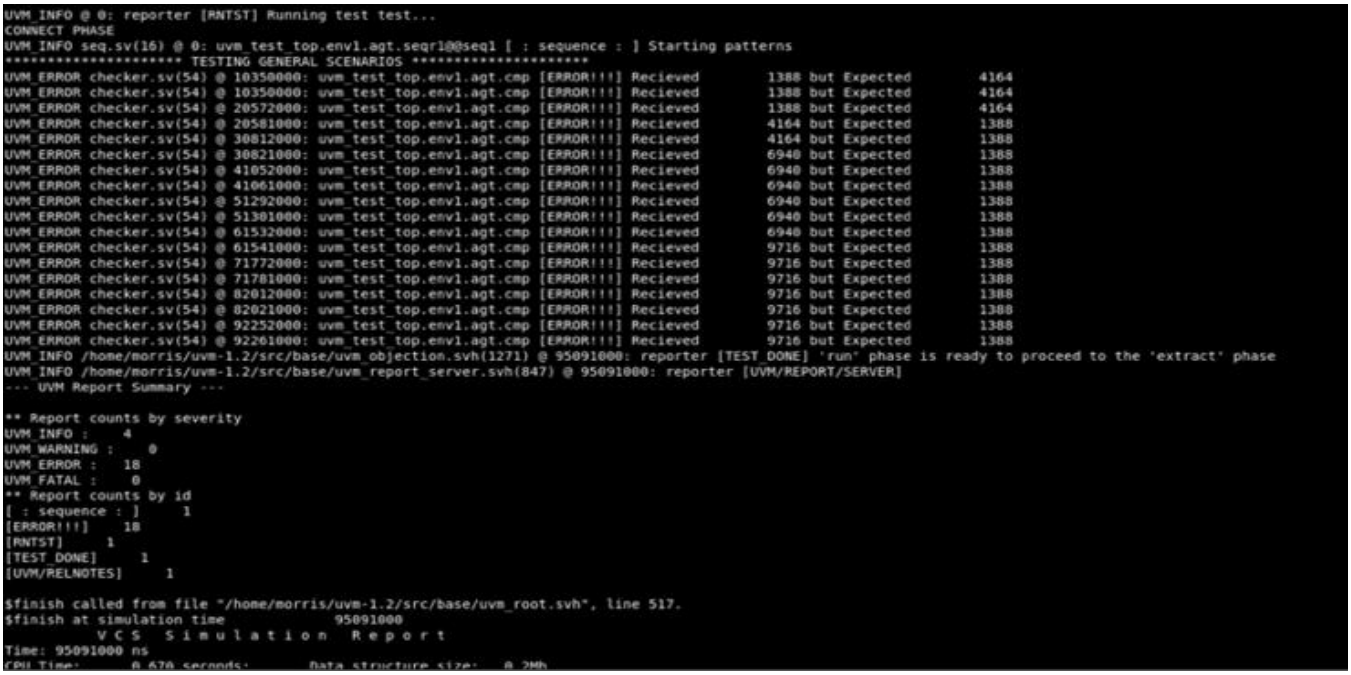

Fig. 8. Simulation results of DUTs that failed the test

As shown in the figure, the expected and received outputs from the correlator do not match; therefore, the DUTs failed the test. Fig. 8 shows the waveform of the result which failed. The other DUTs never gave out any outputs because the push corr never went high, implying the corr value was never given out.

VII. Conclusion

The project's objective was to provide functional verification for the spread spectrum correlator and plot its correlation characteristics. It has successfully achieved functional verification by building an environment consisting of agents, drivers, monitors, and Scoreboards of the main blocks like Accumulator, Correlator, and the Direct Digital Synthesizer. The correlator output was verified and shown in the results.

References

[1] "An Introduction to Spread-Spectrum Communications - Maxim", Maximintegrated.com, 2003. [Online]. Available: https://www.maximintegrated.com/en/design/technicaldocuments/tutorials/1/1890.html. [Accessed: 02- Oct- 2019].

[2] V. Eerola and T. Ritoniemi, "Signal Acquisition System for Spread Spectrum Receiver", US6909739B1, 2005.

[3] S. Kulkarni, P. Mazumder and G. Haddad, "A high-speed 32-bit parallel correlator for spread spectrum communication", IEEE, Bangalore, India, 1996.

[4] A. Goiser and M. Sust, "Spread Spectrum communication using CMOS digital correlator", IEEE, Lisbon, Portugal, 1989.

[5] A. Hendrickson, "Verification of PN Synchronization in a Direct-Sequence Spread Spectrum Digital Communication System", US6002709, 1999

[6] J. Yun, "Adaptive acquisition method and apparatus for CDMA and spread spectrum systems compensating for frequency offset and noise", KR100473679B1, 2005

[7] V. Nath and A. Kumar, "A Comparative Study of Spread Spectrum Technique based on Various Pseudorandom Codes", Global Journal of research in engineering, vol. 12, no. 6, 2012. Available:https://pdfs.semanticscholar.org/b821/b35033237efd3c31bb016e93bb9d87e49e46.pdf. [Accessed 16 September 2019].

[8] "Exploring communications technology", OpenLearn, 2020. [Online]. Available:https://www.open.edu/openlearn/science-maths-technology/exploring-communicationstechnology/content-section-1.4. [Accessed: 03- Feb- 2020].

[9] "Understanding Spread Spectrum for Communications - National Instruments", Ni.com, 2020. [Online]. Available: https://www.ni.com/en-us/innovations/whitepapers/06/understanding-spread-spectrum-for-communications.html. [Accessed: 07- Mar- 2020].

[10] "The GPS PRN (Gold Codes)", Natronics.github.io, 2020. [Online]. Available:https: //natronics.github.io/blag/ 2014/gps-prn/. [Accessed: 13- Feb- 2020].

[11] Huang, Kai & Zhu, Peng & Yan, Rongjie & Yan, Xiaolang. (2015). “Functional Testbench Qualification by Mutation Analysis”. VLSI Design. 2015. 10.1155/2015/256474.

[12] Saponara, Sergio & Vitullo, Francesco & Petri, Esa & Fanucci, Luca & Coppola, Marcello & Locatelli, Riccardo. (2011). “Coverage-Driven Verification of HDL IP Cores”. 10.1007/978-94-007-0638-5_8.

ISBN: 978-988-14049-2-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)