Digital Design Simulation and Performance Analysis of Scrambler and Descrambler

Tanveer Singh Gaga[1], Mohammed Anwar[2]
[1]*Assistant Professor, Vyas Institute of Engineering & Technology, Jodhpur*
[2]*Assistant Professor, Mahaveer Institute of Technology & Science, Pali*

Abstract— Scramblers are a classified for security reasons and are useful in areas of Entertainment services. A Scrambler is an operation which basically randomizes the data streams. In addition to its use as a stream cipher, a scrambler is commonly used to avoid continuous 0's and 1's which are responsible for DC wander and synchronization problems in communication circuits. Scramblers are required to encrypt video and audio signals for broadcasting and many other applications. Less cost and complexity, greater speed of operation and easy to use are the main features of scramblers. Scrambler is gainfully used in data communication systems to add redundancy in the transmitted data stream so that at the receiver end, that information can be retrieved to aid the synchronization between data terminals.

Keywords— Scrambler, Descrambler, VHDL, Xilinx

I. INTRODUCTION

A simple method for ensuring security is to encrypt the data. The pseudo-noise (PN) key generation is commonly used for any secure communication system. PN sequences based on Linear Feedback Shift Registers (LFSR) and non linear combination based implementations are easiest to give moderate level of security. Chaos based encryption techniques have proved accurate, but complexity of such systems is high.

The alternative solution for data encryption with efficient message security at low cost is the use of Scrambler/ Descrambler.

II. PREVIOUS WORK

In another figure 1.1 we see that All the bits transmitted in the data portion are scrambled using a frame synchronous 8 bits sequence generator. Scrambling is used to randomize the data sequences, which may contain long strings of binary 1s or 0s. The tail bits are not scrambled. The octets of the data are placed in the transmitted serial bit stream, bit by bit from 0 to 7. The frame synchronous scrambler uses the generator polynomial *S(x)* as follows:

The 8 bits sequence is generated repeatedly by the scrambler (leftmost used first),(with 8 one's as input). This scrambler is used to scramble transmit data and to de-scramble receive data. When transmitting, the initial state of the scrambler will be set to a pseudo random defined non-zero state. The seven LSBs of the data will be reset to all zeros prior to scrambling to enable prediction of the initial state of the scrambler in the receiver.

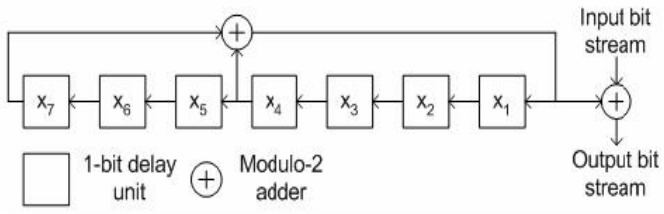

Figure 1.1 Structure of Scrambler

III. PROPOSED METHODOLOGY/ SYSTEM MODEL PARALLELIZED SCRAMBLER

Fig. 1.2 shows the scrambler structure defined. The value of the bit string {a7,a6,a5,a4,a3,a2,a1} is called the values of the scrambler. Given a non-zero initial state, the scrambler forms a pseudo-random sequence used to randomize the input bit stream.

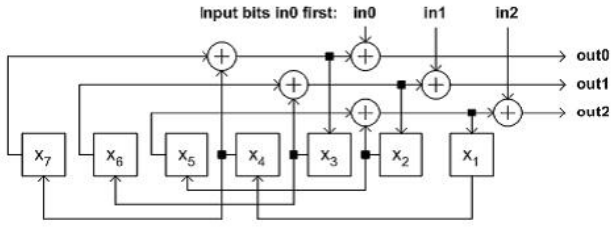

Figure 1.2 Parallelized scrambler to generate three bits at a time

The scrambler in Figure 1.1 processes one input bit at a time. By observing the operation of the scrambler defined for consecutive input bits, we can find that three consecutive output bits can be generated based on solely the present state of the scrambler as shown in Fig. 1.2

Based on this observation, we can parallelize the scrambler to take three input bits together at a time and generate three output bits accordingly as shown in Fig. 1.2.

VHDL Coding:
VHDL Code of scrambler

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity scrambl is
port(din, clk: in std_logic;
dout: out std_logic);
end scrambl;

architecture Behavioral of scrambl is
signal sc_reg: std_logic_vector(4 downto 1):="0000";
signal xr: std_logic;
begin
process(clk)
begin
xr<= din xor sc_reg(1) xor sc_reg(2) xor sc_reg(4);
if clk'event and clk='1' then
sc_reg<= sc_reg(3 downto 1)& xr;
dout<= xr;
end if;
end process;
end Behavioral;
```

VHDL Code of Descrambler

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity descram is
   Port ( sc_in : in  STD_LOGIC;
          clk : in  STD_LOGIC;
          uns_out : out  STD_LOGIC);
end descram;
architecture Behavioral of descram is
signal sc_reg: std_logic_vector(4 downto 1):="0000";
signal xr: std_logic;
begin
xr<= sc_reg(1) xor sc_reg(2) xor sc_reg(4);
process(clk)
begin
if clk'event and clk='1' then
uns_out<= xr xor sc_in;
sc_reg<= sc_reg(3 downto 1) & sc_in;
end if;
end process;
end Behavioral;
```

VHDL testbench program for Scrambler

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTIO.ALL;

ENTITY tbbb IS
END tbbb;

ARCHITECTURE testbench_arch OF tbbb IS
   COMPONENT scrambl
      PORT (
         din : In std_logic;
         clk : In std_logic;
         dout : Out std_logic
      );
   END COMPONENT;
   SIGNAL din : std_logic := '0';
   SIGNAL clk : std_logic := '0';
   SIGNAL dout : std_logic := '0';
   constant PERIOD : time := 200 ns;
   constant DUTY_CYCLE : real := 0.5;
   constant OFFSET : time := 100 ns;
   BEGIN
      UUT : scrambl
      PORT MAP (
         din => din,
         clk => clk,
         dout => dout );
      PROCESS    -- clock process for clk
      BEGIN
         WAIT for OFFSET;
         CLOCK_LOOP : LOOP
            clk <= '0';
            WAIT FOR (PERIOD - (PERIOD *
DUTY_CYCLE));
            clk <= '1';
            WAIT FOR (PERIOD * DUTY_CYCLE);
         END LOOP CLOCK_LOOP;
      END PROCESS;
      PROCESS
         BEGIN
            -- -------------  Current Time:  185ns
            WAIT FOR 185 ns;
            din <= '1';
            -- ------------------------------------
            -- -------------  Current Time:  385ns
            WAIT FOR 200 ns;
            din <= '0';
```

```
            -- ------------------------------------
            -- -------------  Current Time:  785ns
            WAIT FOR 400 ns;
            din <= '1';
            -- ------------------------------------
            WAIT FOR 415 ns;
         END PROCESS;
   END testbench_arch;
```

VHDL testbench program for Descrambler

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTIO.ALL;

ENTITY destb IS
END destb;

ARCHITECTURE testbench_arch OF destb IS
   COMPONENT descram
      PORT (
         sc_in : In std_logic;
         clk : In std_logic;
         uns_out : Out std_logic
      );
   END COMPONENT;

   SIGNAL sc_in : std_logic := '0';
   SIGNAL clk : std_logic := '0';
   SIGNAL uns_out : std_logic := '0';

   constant PERIOD : time := 200 ns;
   constant DUTY_CYCLE : real := 0.5;
   constant OFFSET : time := 100 ns;

   BEGIN
      UUT : descram
      PORT MAP (
         sc_in => sc_in,
         clk => clk,
         uns_out => uns_out
      );
      PROCESS    -- clock process for clk
      BEGIN
         WAIT for OFFSET;
         CLOCK_LOOP : LOOP
            clk <= '0';
            WAIT FOR (PERIOD - (PERIOD *
DUTY_CYCLE));
            clk <= '1';
            WAIT FOR (PERIOD * DUTY_CYCLE);
         END LOOP CLOCK_LOOP;
      END PROCESS;
      PROCESS
         BEGIN
            -- -------------  Current Time:  100ns
            WAIT FOR 100 ns;
            sc_in <= '1';
            -- ------------------------------------
            -- -------------  Current Time:  585ns
            WAIT FOR 485 ns;
            sc_in <= '0';
            -- ------------------------------------
            WAIT FOR 615 ns;

         END PROCESS;
   END testbench_arch;
```

IV. SIMULATION/EXPERIMENTAL RESULTS

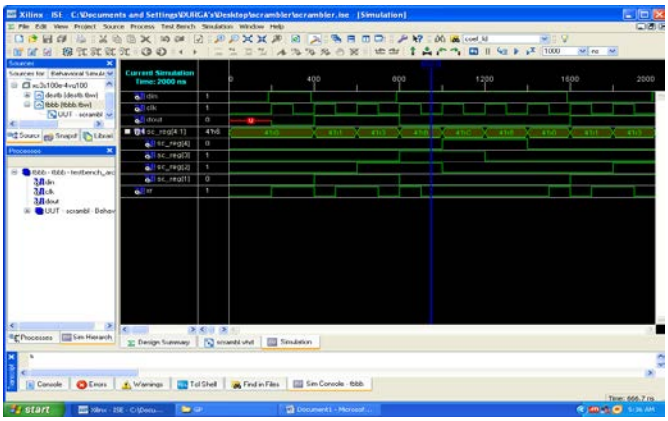

Figure 1.3: Simulated results of scrambler din represents the data_in given to scrambler module dout represents the scrambled data out of it. The signal xr represents the feedback signal out of scrambler register sc_reg

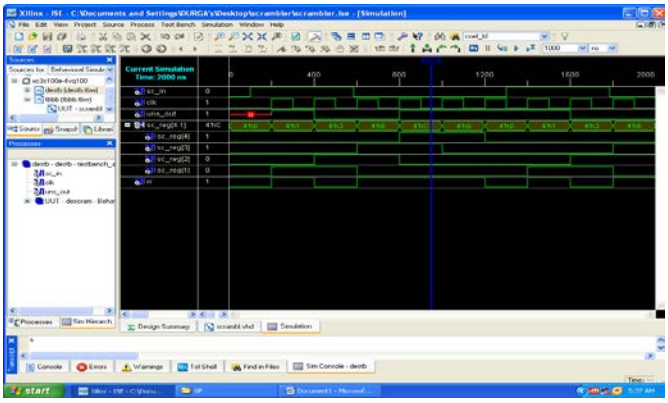

Figure 1.4: Simulated results of descrambler din represents the sc_in given to descrambler module uns_out represents

the descrambled data out of it. xr represents the feedback signal out of scrambler register sc_reg

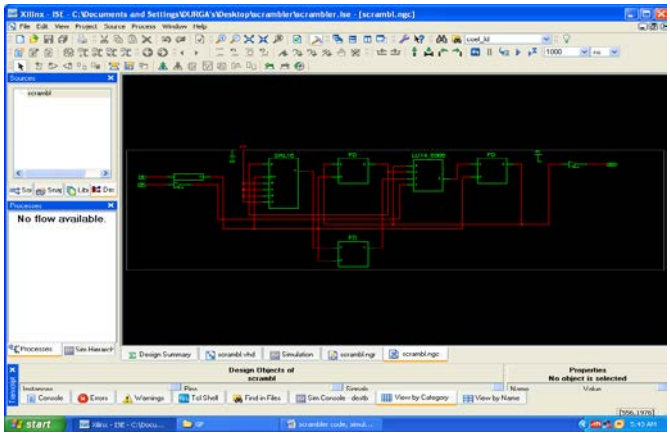

Figure 1.5: RTL view of the synthesized scrambler module

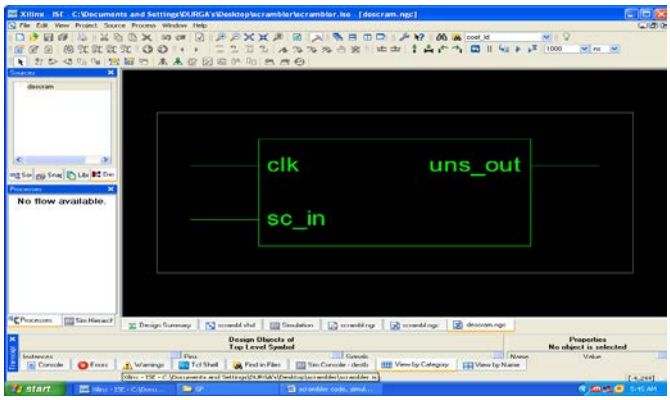

Figure 1.6: Block diagram representation of descrambler module

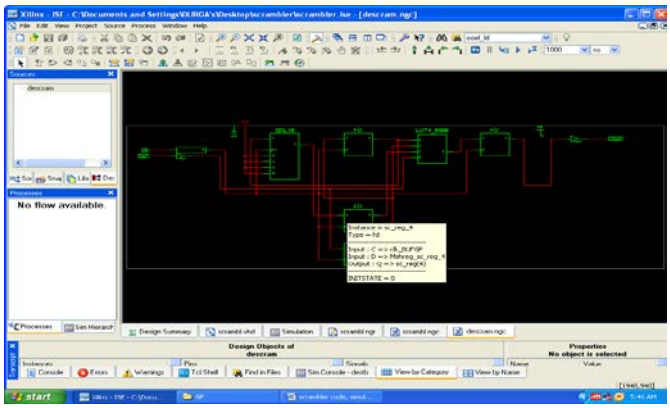

Figure 1.7: RTL view of the synthesize descrambler module

V. CONCLUSION

The designed digital circuits are simulated in Xilinx 9.2i, ModelSim XE III 6.0 a and the designed blocks are Scrambler, descrambler have been designed in this dissertation. In this Paper work the digital design and synthesis of Scrambler, Descrambler is presented. With 1 bit scrambler first then modifying for 3 bit parallelized scrambler. These blocks are simulated using Xilinx 9.2i, Model Sim XE 6.0.In further work these designs could be used for real time operation for security in data communication.

VI. FUTURE SCOPES

In this paper work the digital design and synthesis of Scrambler, Descrambler. With 1 bit scrambler first then modifying for 3 bit parallelized scrambler. These blocks are simulated using Xilinx 9.2i, Model Sim XE 6.0. In further work these designs could be used for real time operation for security in data communication.

VII. REFERENCES

[1] WLAN MAC and PHY Specifications: High-speed Physical Layer in the 5GHz Band_, IEEE Std 802.11a Supplement to IEEE Std Part 11, Sept.1999.

[2] Yiyan Tang, Lie Qian, and Yuke Wang Optimized Software Implementation of a Full-Rate IEEE 802.11a Compliant Digital Baseband Transmitter on a Digital Signal Processor

[3] VHDL modeling and simulation of data scrambler and descrambler for secure data communication. G.M. Bhat, M. Mustafa, Shabir Ahmad and Javaid Ahmad

[4] DSP Based Implementation of Scrambler for 56Kbps Modem, Davinder Pal Sharma, Department of Physics, University of the West Indies St. Augustine, Tinidad & Tobago, Jasvir Singh, Department of Electronics Technology, Guru Nanak Dev University, Amritsar -143105, India

[5] A VHDL Primer, J. Bhasker