in the name of God

FPGA Project
Phase1
Scrambling & Descrambling of LAN 802.11a-1999
Sharif University of Technology

Name: Sajjad Akherati
id : 96101154

June 12, 2021

1 Introduction

In this project, we are trying to implement the structure of LAN standard 802.11a-1999 based on a FPGA system.
Our goal in this project is to implement the first blocks of this system including scrambling, convlutional encoder and data interleaver from the Tx part and their corresponding part in Rx part including descrambling, viterbi decoder, data deinterleaver.
In this project, we had assumed that the datalink layer in transmitter build a frame and sends it to the phy in Tx part serially. the physical layer that is a hardware based on FPGA system change the data such that it can be passed from the channel such that avoid noisy data in receiver as much as possible. to simulating a frame in our system, we build a frame base of the structure of the standard in save it in a .txt file and then we can test our design for the physical layer.

2 Framing

As we can see in the standard, the frame that the physical layer receives from the upper layer has a specific structure shown in the following figure:
we build out frame with this structure in MATLAB and save the data in a txt file and in verilog modules,

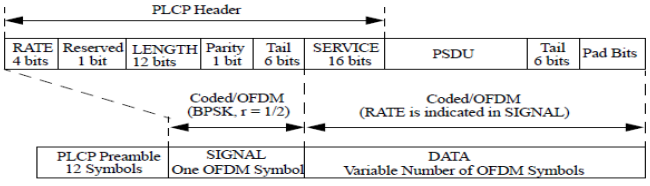

Figure 1: frame structure

we can test thei functionality in testbench.
the preamble part is to synchronize the receiver eith the transmitter; it use an special pattern of OFDM symbles and append it to the first part of the frame (in OFDM block); so when the receiver, receives this pattern, it finds out that a new frame has been transmitted by ransmitter and the receiver must synchronize itself with the transmitter at this time interval.
as we don't want to implement the OFDM block in this project, we will use and special pattern of zeros and ones in transmitter such that the receiver can sychronize itself with it. we will avoid to scrambling, coding and interleaving of this part. so i decide to implement this part in Phase3, after that the interleave had been implemented. this is because we don't want to scramble, code and interleave the preamble bits and notice to this approach, we can just use a shift register to add the preamble feature to our design. i has coinsider the following pattern for preamble. this pattern is can not be seen in our data as we use scramble in transmitter:

$$\overbrace{0101010101010101010}^{10\ \text{short symbols}}\ \overbrace{00000000}^{\text{first long symbol}}\ \underbrace{11111111}_{\text{second long symbol}}$$

building frames can be done in MATLAB. this part is implemented in MATLAB code attached with *main.m* in the first section; i bulid a frame to use in all blocks and so i has commented them in my files. but you can uncomment them and bulild another frame.

3 Scrambling

There is a coding theory that says that if we try to create a randomness in the data that is send in transmitter, there is fewer probability of effecting on data and creation of noise. the module scrambling in standard LAN 802.11a-1999 creat this randomness in our data to improve the efficiency of our system. base of the standard, we scramble our data with the following strucure: we avoid scrambling of the signal field and tail bits in

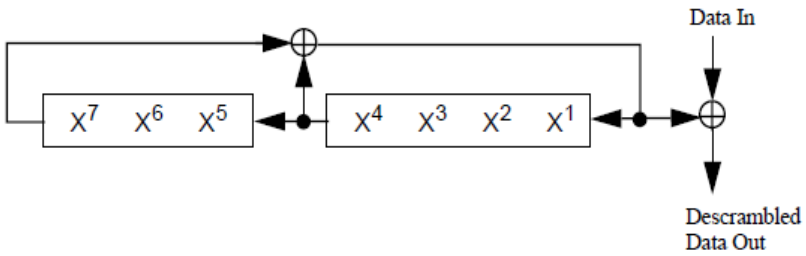

Figure 2: Data Scrambler

data field of the frame in transmitter. this is because the structure of the frames. the receiver must be able to find the length of data and detects some other parameter in the feauture blocks that determines with rate field. so there is needed to do not scramble them and loss their values. there is also necessary that the receiver be able to synchronize itself eith transmitter and detect the initial seed to scrambling. this can be done with the first 7 bit of service field of data. the other nine bits are left and has been reserved for the feutur. The annex G of the standard has been scramble them. so is also scrambled them in my design to be the same as the standard.

3.1 Results

I implemented the scrambler in MATLAB in a function name *scrambler.m* in MATLAB directoy of attached files.

then i implemented the module scrambler in verilog. this module has been uses the following recourses in my design: i wrote a testbench for this module and compare its .txt output file with my MATLAB code and

Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	43	54,576	1%	
Number used as Flip Flops	42			
Number used as Latches	0			
Number used as Latch-thrus	0			
Number used as AND/OR logics	1			
Number of Slice LUTs	72	27,288	1%	
Number used as logic	71	27,288	1%	
Number using O6 output only	23			
Number using O5 output only	26			
Number using O5 and O6	22			
Number used as ROM	0			
Number used as Memory	0	6,408	0%	
Number used exclusively as route-thrus	1			
Number with same-slice register load	0			
Number with same-slice carry load	1			
Number with other load	0			
Number of occupied Slices	25	6,822	1%	
Number of MUXCYs used	48	13,644	1%	
Number of LUT Flip Flop pairs used	84			
Number with an unused Flip Flop	44	84	52%	
Number with an unused LUT	12	84	14%	
Number of fully used LUT-FF pairs	28	84	33%	
Number of unique control sets	6			
Number of slice register sites lost to control set restrictions	22	54,576	1%	
Number of bonded IOBs	35	218	16%	
Number of bonded IOBs	35	218	16%	
Number of RAMB16BWERs	0	116	0%	
Number of RAMB8BWERs	0	232	0%	
Number of BUFIO2/BUFIO2_2CLKs	0	32	0%	
Number of BUFIO2FB/BUFIO2FB_2CLKs	0	32	0%	
Number of BUFG/BUFGMUXs	1	16	6%	
Number used as BUFGs	1			
Number used as BUFGMUX	0			
Number of DCM/DCM_CLKGENs	0	8	0%	
Number of ILOGIC2/ISERDES2s	0	376	0%	
Number of IODELAY2/IODRP2/IODRP2_MCBs	0	376	0%	
Number of OLOGIC2/OSERDES2s	0	376	0%	
Number of BSCANs	0	4	0%	
Number of BUFHs	0	256	0%	
Number of BUFPLLs	0	8	0%	
Number of BUFPLL_MCBs	0	4	0%	
Number of DSP48A1s	0	58	0%	
Number of ICAPs	0	1	0%	
Number of MCBs	0	2	0%	
Number of PCILOGICSEs	0	2	0%	
Number of PLL_ADVs	0	4	0%	
Number of PMVs	0	1	0%	
Number of STARTUPs	0	1	0%	
Number of SUSPEND_SYNCs	0	1	0%	
Average Fanout of Non-Clock Nets	3.16			

Figure 3: Recourse used by Scrambler

you can see the check part of my *main.m* MATLAB code and run it to check any error. you can see some the result of the outputs in the following figures:

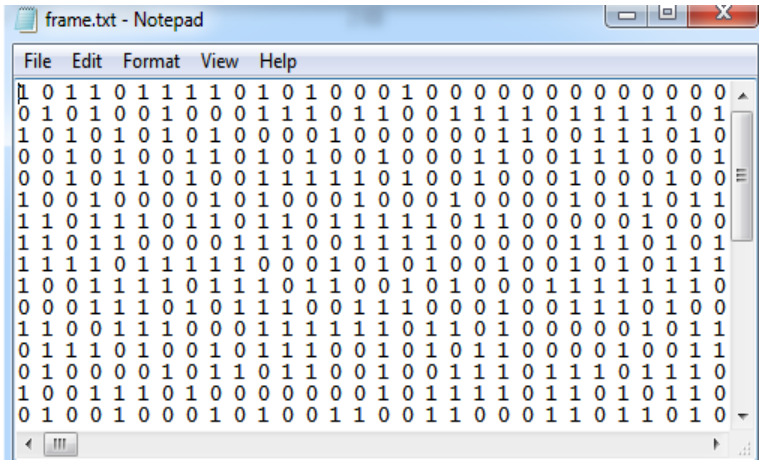

Figure 4: Frame

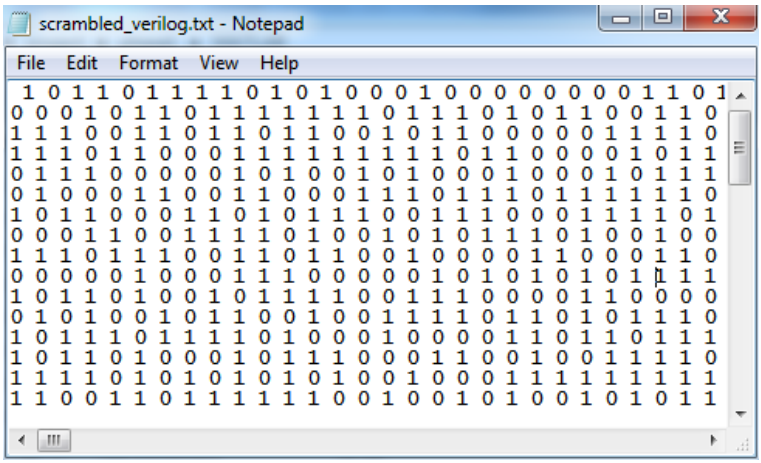

Figure 5: Scrambled Frame Verilog

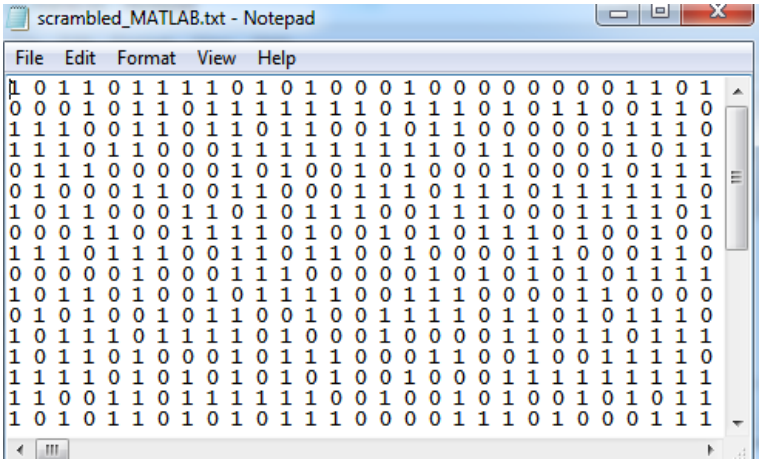

Figure 6: Scrambled Frame MATLAB

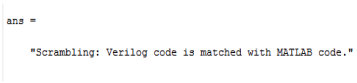

Figure 7: compare result of MATLAB with Verilog

4 Descrambler

As i sayed in the previous section, scrambling data in transmitter cause that there be less probability that the data in passsign tha channel be noised. this effect in the receiver must be delete as we want to deliver the main data on MAC layer. to do this, we must design a descrambler in receiver to delets these effects; the only thing that we need to do is that we do XOR the same bit was scrambled with data in transmitter. to do so we need to detect the initial seed by the receiver; as i sayed in the previous section, this can be done by the lower 7 bits of the service field. by reading the standard, i concluded two thing that it helped me to

use more little recources in receiver in descramble module:

1. we add tail bit at the end of the transmitter to increase the efficiency of the convlotional encoder.
2. we add pad bit at the end of the data field after tail bits, because we need that the OFDM mpdule in transmitter needs to receive a frame that its data field length is divisable by N_{DBPS}. this enable the interleaver in transmitter and deinterleave in receiver to work properly.
3. The MAC layer is a CPU based system that can calculate the number of the pad bits with the following formula: In transmitter, this proccess can be done with the cpu of the Mac layer. so the scrambler

$$N_{SYM} = \text{Ceiling}\left((16 + 8 \times \text{LENGTH} + 6)/N_{DBPS}\right)$$

$$N_{DATA} = N_{SYM} \times N_{DBPS}$$

$$N_{PAD} = N_{DATA} - (16 + 8 \times \text{LENGTH} + 6)$$

Figure 8: Number of Pad bits Formula

   module can dtetct the number of the pad bits by is upper MAC layer as as input.
   there is also no need for receiver to send back the data tail bits and pad bits for MAC layer. the MAC layer system can receive the original data by the length that is in signal field. after that the receiver does not need to tranmite them to MAC layer, so there is no need caculate N_PAD in receiver.

notice to the above approachs, i designed the descrambler module. the *descramble.m* is also do descrambling in MATLAB. it is attached in MATLAB directory.

4.1 Result

you can see the number of recources was used by mo descrambler module in the following figures: also you

Slice Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Registers	33	54,576	1%	
Number used as Flip Flops	33			
Number used as Latches	0			
Number used as Latch-thrus	0			
Number used as AND/OR logics	0			
Number of Slice LUTs	67	27,288	1%	
Number used as logic	60	27,288	1%	
Number using O6 output only	24			
Number using O5 output only	14			
Number using O5 and O6	22			
Number used as ROM	0			
Number used as Memory	2	6,408	1%	
Number used as Dual Port RAM	0			
Number used as Single Port RAM	0			
Number used as Shift Register	2			
Number using O6 output only	2			
Number using O5 output only	0			
Number using O5 and O6	0			
Number used exclusively as route-thrus	5			
Number with same-slice register load	4			
Number with same-slice carry load	1			
Number with other load	0			
Number of occupied Slices	21	6,822	1%	
Number of MUXCYs used	40	13,644	1%	
Number of LUT Flip Flop pairs used	67			
Number with an unused Flip Flop	38	67	56%	
Number with an unused LUT	0	67	0%	
Number of fully used LUT-FF pairs	29	67	43%	
Number of unique control sets	5			
Number of slice register sites lost to control set restrictions	21	54,576	1%	
Number of bonded IOBs	4	218	1%	
Number of RAMB16BWERs	0	116	0%	
Number of RAMB8BWERs	0	232	0%	
Number of BUFIO2/BUFIO2_2CLKs	0	32	0%	
Number of BUFIO2FB/BUFIO2FB_2CLKs	0	32	0%	
Number of BUFG/BUFGMUXs	1	16	6%	
Number used as BUFGs	1			
Number used as BUFGMUX	0			
Number of DCM/DCM_CLKGENs	0	8	0%	
Number of ILOGIC2/ISERDES2s	0	376	0%	
Number of IODELAY2/IODRP2/IODRP2_MCBs	0	376	0%	
Number of OLOGIC2/OSERDES2s	0	376	0%	
Number of BSCANs	0	4	0%	
Number of BUFHs	0	256	0%	
Number of BUFPLLs	0	8	0%	
Number of BUFPLL_MCBs	0	4	0%	
Number of DSP48A1s	0	58	0%	
Number of ICAPs	0	1	0%	
Number of MCBs	0	2	0%	
Number of PCILOGICSEs	0	2	0%	
Number of PLL_ADVs	0	4	0%	
Number of PMVs	0	1	0%	
Number of STARTUPs	0	1	0%	
Number of SUSPEND_SYNCs	0	1	0%	

Figure 9: Recourse used by Descrambler

can see the result of descramblein of the frame was scrabled in te previous part in the following figures:

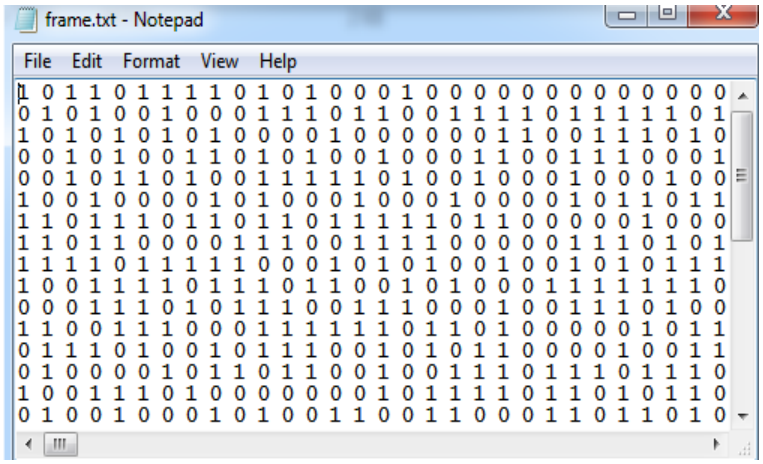

Figure 10: Frame

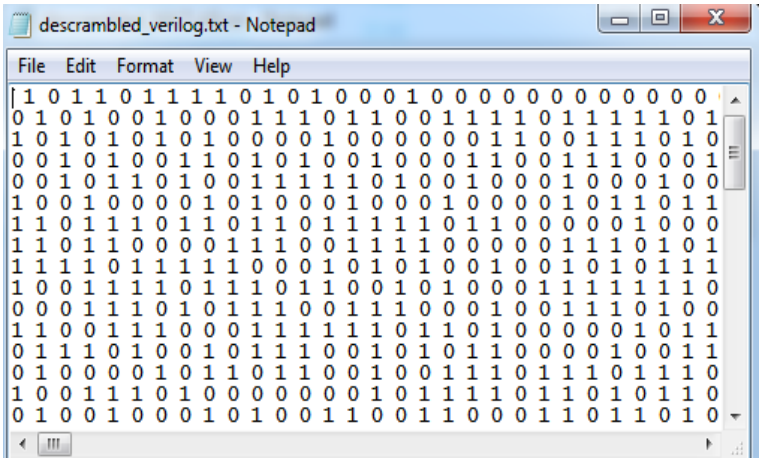

Figure 11: Descrambled Frame Verilog

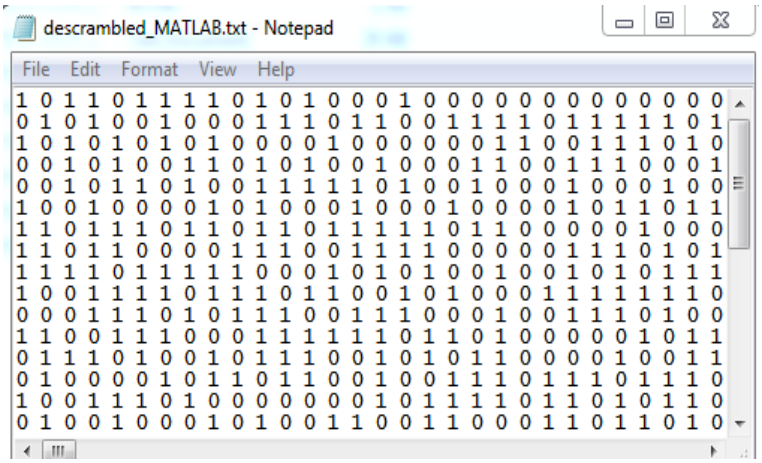

Figure 12: Descrambled Frame MATLAB

```
ans =

    "deScrambling: Verilog code is matched with MATLAB code."
```

Figure 13: compare result of MATLAB with Verilog