Angelia WIFI - IEEE 802.11a - No 001 v1.0

WIFI Datasheet - Software-Hardware Implementation of IEEE 802.11a Wifi Standard, Verilog + Matlab

IP CORE SERIAL NUMBER 2

IP TYPE 2

STANDARD 2

LANGUAGE 2

TECHNOLOGY 2

MATURITY / STATUS 2

DELIVERABLES 3

OVERVIEW 3

COMPONENTS 3

TRANSMITTER 3

Scrambler *4*

ConvEncoder *4*

Interleaver *6*

RECEIVER 7

DeScrambler *8*

ViterbiDecoder *8*

DeInterleaver *8*

FUNCTIONAL DIAGRAM 10

CODE HIERARCHY 10

ROOT DIRECTORY 10

RECEIVER DIRECTORY 11

DeInterleaver *11*

DeScrambler *12*

ViterbiDecoder *13*

TRANSMITTER DIRECTORY 13

ConvEncoder *14*

Interleaver *14*

Scrambler *15*

INTEGRATION GUIDE 15

DEVELOPER 15

LICENSE 16

IP Core Serial Number

Angelia WIFI - IEEE 802.11a - No 001 v1.0

IP Type

Soft IP

Standard

WIFI - IEEE 802.11a

Language

Verilog + Matlab

Technology

FPGA

Maturity / Status

Pre-Silicon

Deliverables

Verilog codes and Matlab test codes

Datasheet

Customer Support

Overview

In the hardware implementation of IEEE 802.11a Wifi Standard we feature Verilog for our HDL language and Matlab for our high-level testing language.

Hardware simulations are carried out by Modelsim.

Components

Transmitter

Transmitter module has the rule of converting input data to WIFI frame and send it to antenna.

User should set the start for a clock and then wait for 12 * 8 + 4 + 1 + 12 + 1 + 6 + 16 = 136 clocks and then send in the input data which should be Transmitter.LENGTH octets (Bytes) , and then for resending you should set start for a clock again.

This part contains transmitter part of the protocol which simply breaks into 3 part:

- Scrambler
- ConvEncoder
- Interleaver

```
              +-----------+      +-------------+      +-------------+
              |           |      |             |      |             |
Input Data --> | Scrambler + ---> | ConvEncoder + ---> | Interleaver | ------> to
Antenna
              |           |      |             |      |             |
              |           |      |             |      |             |
              +-----------+      +-------------+      +-------------+
```

Scrambler

Refer to 802.11a IEEE Wifi PLCP DATA Scrambler (https://pdos.csail.mit.edu/archive/decouto/papers/802.11a.pdf - Page 16)

S(x) = x ^ 7 + x ^ 4 + 1

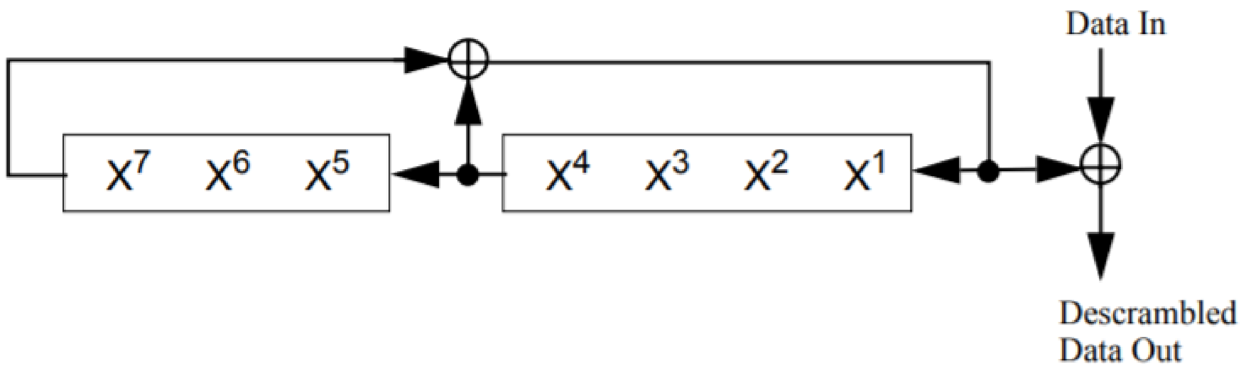
With initial value of (x_7, x_6, x_5, x_4, x_3, x_2, x_1) = (1, 1, 1, 1, 1, 1, 1)

The DATA field, composed of SERVICE, PSDU, tail, and pad parts, shall be scrambled with a length-127 frame-synchronous scrambler. The octets of the PSDU are placed in the transmit serial bit stream, bit 0 first and bit 7 last. The frame synchronous scrambler uses the generator polynomial S(x) as follows:

S(x) = x^7 + x^4 + 1

The 127-bit sequence generated repeatedly by the scrambler shall be (leftmost used first), 00001110 11110010 11001001 00000010 00100110 00101110 10110110 00001100 11010100 11100111 10110100 00101010 11111010 01010001 10111000 1111111, when the "all ones" initial state is used.

The same scrambler is used to scramble transmit data and to descramble receive data. When transmitting, the initial state of the scrambler will be set to a pseudo random non-zero state (in this project 1111111). The seven LSBs of the SERVICE field will be set to all zeros prior to scrambling to enable estimation of the initial state of the scrambler in the receiver.

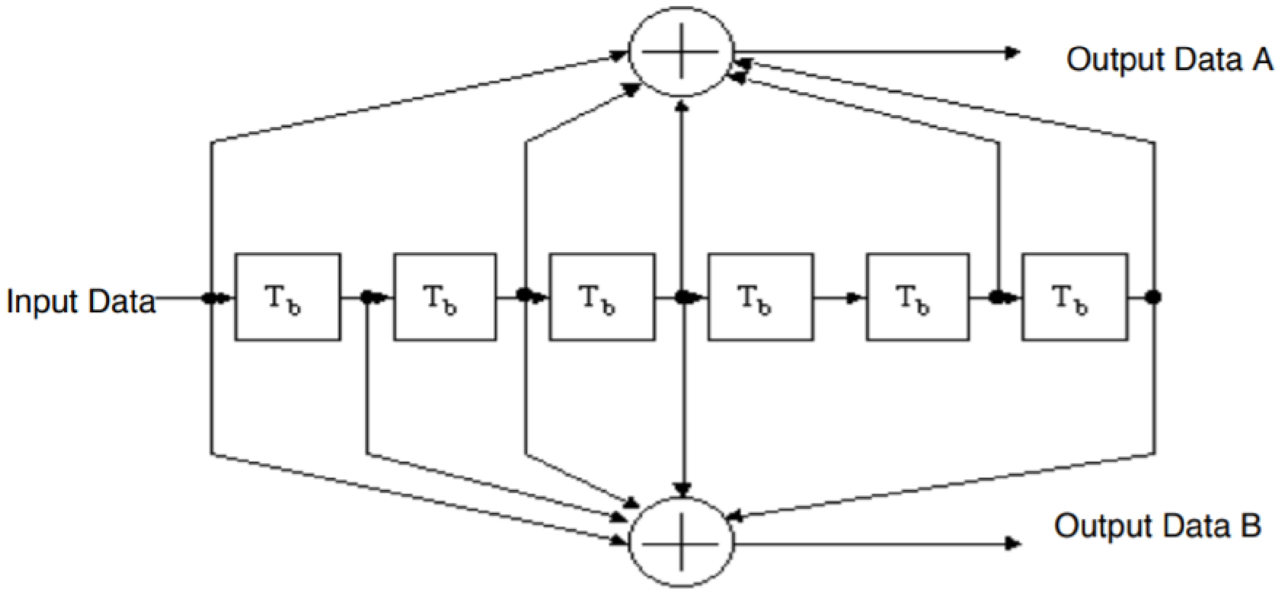


ConvEncoder

Refer to 802.11a IEEE Wifi PLCP DATA Convolutional Encoder (https://pdos.csail.mit.edu/archive/decouto/papers/802.11a.pdf - Page 16)

The DATA field, composed of SERVICE, PSDU, tail, and pad parts, shall be coded with a convolutional encoder of coding rate R = 1/2, 2/3, or 3/4, corresponding to the desired data rate. The convolutional encoder shall use the industry-standard generator polynomials, g0 = (133)_8 and g1 = (171)_8, of rate R = 1/2, as shown in Figure bellow.

The bit denoted as "A" shall be output from the encoder before the bit denoted as "B." Higher rates are derived from it by employing "puncturing." Puncturing is a procedure for omitting some of the encoded bits in the transmitter (thus reducing the number of transmitted bits and increasing the coding rate) and inserting a dummy "zero" metric into the convolutional decoder on the receive side in place of the omitted bits.



As we mentioned about generator polynomials, after defining a que structure for input buffer like bellow:

```
               _______________________________
               |    |    |    |    |    |    |    |
Input ---> | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
               |    |    |    |    |    |    |    |
               _______________________________
```

Then we have:

```
g0 = (1011011)_2   ---> g_0(x) = x^1 + x^3 + x^4 + x^6 + x^7
g1 = (1111001)_2   ---> g_1(x) = x^1 + x^2 + x^3 + x^4 + x^7
```

so output is a sequence (y) which is defined like bellow:

```
         {     k % 2 == 0   :    x[1] + x[3] + x[4] + x[6] + x[7]
y[k] = {
         {     k % 2 == 1   :    x[1] + x[2] + x[3] + x[4] + x[7]
```

Interleaver

All encoded data bits (Signal and Data subframe) shall be interleaved by a block interleaver with a block size corresponding to the number of bits in a single OFDM symbol, NCBPS. The interleaver is defined by a two-step permutation. The first permutation ensures that adjacent coded bits are mapped onto nonadjacent subcarriers. The second ensures that adjacent coded bits are mapped alternately onto less and more significant bits of the constellation and, thereby, long runs of low reliability (LSB) bits are avoided.

We shall denote by k the index of the coded bit before the first permutation; i shall be the index after the first and before the second permutation, and j shall be the index after the second permutation, just prior to modulation mapping.

The first permutation is defined by the rule:

$i = (NCBPS/16)\ (k \bmod 16) + \mathrm{floor}(k/16)$ for $k = 0, 1, \ldots, NCBPS-1$

The function floor (.) denotes the largest integer not exceeding the parameter.

The second permutation is defined by the rule:

$j = s \times \mathrm{floor}(i/s) + (i + NCBPS - \mathrm{floor}(16 \times i/NCBPS)) \bmod s$ for $i = 0,1,\ldots\ NCBPS - 1$

The value of s is determined by the number of coded bits per subcarrier, NBPSC, according to:

$s = \max(NBPSC/2,1)$

This module will put inputs into a buffer which has a length relative to NCBPS and start to output values after a dedicated pulses of clock. The output shall be interleaved values of input.

Some permutation for NCBPS = 48 have been came here:

0 --> 0
1 --> 3
2 --> 6
3 --> 9
4 --> 12
5 --> 15
6 --> 18
7 --> 21
8 --> 24
9 --> 27
10 --> 30
11 --> 33
12 --> 36
13 --> 39
14 --> 42

15 --> 45
16 --> 1
17 --> 4
18 --> 7
19 --> 10
20 --> 13
21 --> 16
22 --> 19
23 --> 22
24 --> 25
25 --> 28
26 --> 31
27 --> 34
28 --> 37
29 --> 40
30 --> 43
31 --> 46
32 --> 2
33 --> 5
34 --> 8
35 --> 11
36 --> 14
37 --> 17
38 --> 20
39 --> 23
40 --> 26
41 --> 29
42 --> 32
44 --> 35
45 --> 38
46 --> 41
47 --> 44

```
                 +-----------+        +-------------+        +-------------+
                 |           |        |             |        |             |
Input Data --> | Scrambler + ---> | ConvEncoder + ---> | Interleaver | ------> to Antenna
                 |           |        |             |        |             |
                 |           |        |             |        |             |
                 +-----------+        +-------------+        +-------------+
```

Receiver

Receiver is waiting for the preamble set to start the process and it pushes it into a FIFO que named **Input_buffer** and then wait for 152 clocks and then start to pick up LENGTH * 8 bits of input.

This part contains receiver part of the protocol which simply breaks into 3 part:

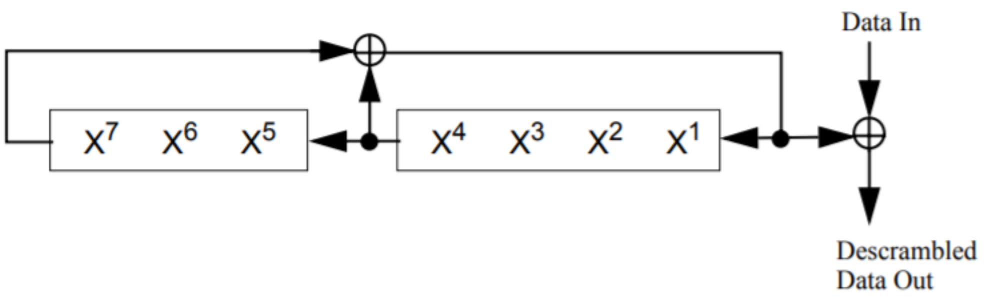
DeScrambler

Refer to 802.11a IEEE Wifi PLCP DATA DeScrambler (https://pdos.csail.mit.edu/archive/decouto/papers/802.11a.pdf - Page 16)

S(x) = x ^ 7 + x ^ 4 + 1

With initial value of (x_7, x_6, x_5, x_4, x_3, x_2, x_1) = (1, 1, 1, 1, 1, 1, 1)

```
                   +---------------+     +-----------------+
                   |               |     |                 |
Output Data <--    | DeScrambler   | <-- | ViterbiDecoder  | <---- from Antenna
                   |               |     |                 |
                   |               |     |                 |
                   +---------------+     +-----------------+
```



ViterbiDecoder

As Document issued decoding the received sequence by the Viterbi algorithm is recommended.

DeInterleaver

The deinterleaver, which performs the inverse relation, is also defined by two permutations

This module will put inputs into a buffer which has a length relative to NCBPS and start to output values after a dedicated pulses of clock. The output shall be interleaved values of input.

Some permutation for NCBPS = 48 have been came here:

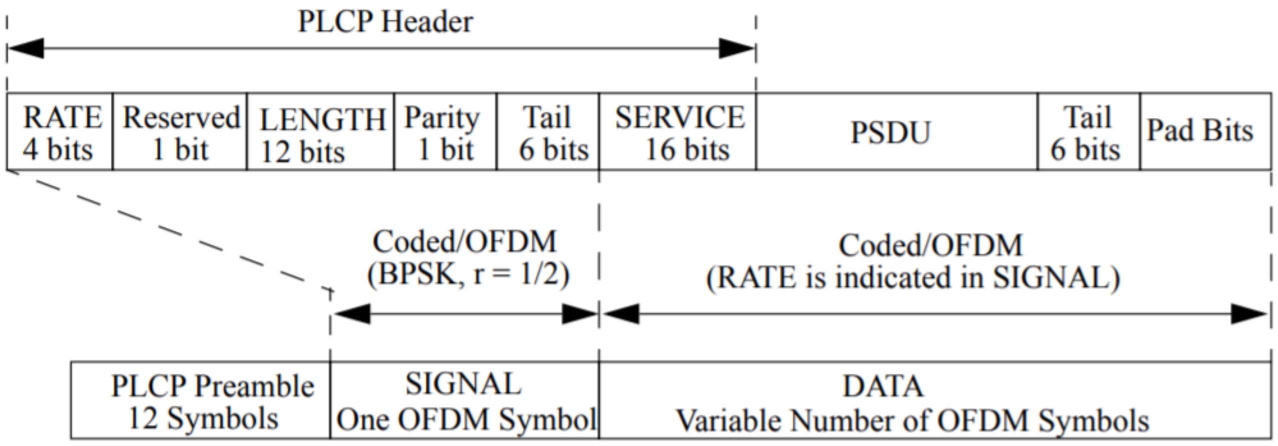
```
0 <-- 0
1 <-- 3
2 <-- 6
3 <-- 9
4 <-- 12
5 <-- 15
6 <-- 18
7 <-- 21
8 <-- 24
9 <-- 27
10 <-- 30
11 <-- 33
12 <-- 36
13 <-- 39
14 <-- 42
15 <-- 45
16 <-- 1
17 <-- 4
18 <-- 7
19 <-- 10
20 <-- 13
21 <-- 16
22 <-- 19
23 <-- 22
24 <-- 25
25 <-- 28
26 <-- 31
27 <-- 34
28 <-- 37
29 <-- 40
30 <-- 43
31 <-- 46
32 <-- 2
33 <-- 5
34 <-- 8
35 <-- 11
36 <-- 14
37 <-- 17
38 <-- 20
39 <-- 23
40 <-- 26
41 <-- 29
42 <-- 32
44 <-- 35
45 <-- 38
46 <-- 41
```

```
47 <-- 44

                     +--------------+      +----------------+      +---------------+
                     |              |      |                |      |               |
Output Data <-- | DeScrambler   | <-- | ViterbiDecoder | <-- | DeInterleaver | <---- from Antenna
                     |              |      |                |      |               |
                     |              |      |                |      |               |
                     +--------------+      +----------------+      +---------------+
```

Functional Diagram

The structure below was implemented for this IP core which is by the way an optimal subset of 802.11a standard. WIFI frame which is going to be used here is as follows:



Code Hierarchy

Root directory

This directory contains different files:

DUT.v

Whole design test module.

DUT_tb.v

Design Verilog test bench.

DUT_tb.txt

Design Verilog test bench records

DUT.m

Transmitter matlab test script.

Receiver directory

This directory contains different files:

Receiver.v

Receiver Verilog module.

Receiver_tb.v

Receiver Verilog test bench which will be used as a unit test.

Receiver_tb.txt

Receiver Verilog test bench result.

Receiver.m

Receiver matlab function which will be used as a software simulation for HDL module.

Receiver_test.m

Receiver matlab function test.

DeInterleaver

This directory contains different files:

DeInterleaver.v

DeInterleaver Verilog module which will be used as a last layer in Receiver.

DeInterleaver_tb.v

DeInterleaver Verilog test bench which will be used as a unit test.

DeInterleaver_tb.txt

DeInterleaver Verilog test bench result.

DeInterleaver.m

DeInterleaver matlab function which will be used as a software simulation for HDL module.

DeInterleaver_test.m

DeInterleaver matlab function test which is used as a proved test which is in IEEE standard.

DeScrambler

This directory contains different files:

DeScrambler.v

DeScrambler Verilog module which will be used as a first layer in receiver.

DeScrambler_tb.v

DeScrambler Verilog test bench which will be used as a unit test.

DeScrambler_tb.txt

DeScrambler Verilog test bench result.

DeScrambler.m

DeScrambler matlab function which will be used as a software simulation for HDL module.

DeScrambler_test.m

DeScrambler matlab function test which is used as a proved test which is in IEEE standard.

ViterbiDecoder

This directory contains different files:

ViterbiDecoder.v

ViterbiDecoder Verilog module which will be used as a second layer in receiver.

ViterbiDecoder_tb.v

ViterbiDecoder Verilog test bench which will be used as a unit test.

ViterbiDecoder_tb.txt

ViterbiDecoder Verilog test bench result.

ViterbiDecoder.m

ViterbiDecoder matlab function which will be used as a software simulation for HDL module.

ViterbiDecoder_test.m

ViterbiDecoder matlab function test which is used as a proved test which is in IEEE standard.

Transmitter directory

This directory contains different files:

Transmitter.v

Transmitter Verilog module.

Transmitter_tb.v

Transmitter Verilog test bench which will be used as a unit test.

Transmitter_tb.txt

Transmitter Verilog test bench result.

Transmitter.m

Transmitter matlab function which will be used as a software simulation for HDL module.

Transmitter_test.m

Transmitter matlab function test.

ConvEncoder

This directory contains different files:

ConvEncoder.v

ConvEncoder Verilog module which will be used as a first layer in transmitter.

ConvEncoder_tb.v

ConvEncoder Verilog test bench which will be used as a unit test.

ConvEncoder_tb.txt

ConvEncoder Verilog test bench result.

ConvEncoder.m

ConvEncoder matlab function which will be used as a software simulation for HDL module.

ConvEncoder_test.m

ConvEncoder matlab function test which is used as a proved test which is in IEEE standard.

Interleaver

This directory contains different files:

Interleaver.v

Interleaver Verilog module which will be used as a last layer in receiver.

Interleaver_tb.v

Interleaver Verilog test bench which will be used as a unit test.

Interleaver_tb.txt

Interleaver Verilog test bench result.

Interleaver.m

Interleaver matlab function which will be used as a software simulation for HDL module.

Interleaver_test.m

Interleaver matlab function test which is used as a proved test which is in IEEE standard.

Scrambler

This directory contains different files:

Scrambler.v

Scrambler Verilog module which will be used as a first layer in transmitter.

Scrambler_tb.v

Scrambler Verilog test bench which will be used as a unit test.

Scrambler_tb.txt

Scrambler Verilog test bench result.

Scrambler.m

Scrambler matlab function which will be used as a software simulation for HDL module.

Scrambler_test.m

Scrambler matlab function test which is used as a proved test which is in IEEE standard.

Integration Guide

See INTEGRATION GUIDE.pdf

Developer

Mark Chen

Angelia Technology / ICTech
http://www.angelia.eu.org

License

See “License Agreement”.