A VHDL Implementation of WiMAX Convolutional Turbo Code

Ailton Akira Shinoda

***Abstract*—this article describes the VHDL implementation of convolutional turbo code of an OFDM system based on adaptive data rate control of physical layer according to the WiMAX standard IEEE 802.16.**

***Index Terms*— convolutional, turbo code, WiMAX, VHDL.**

I. Introduction

The prediction made by the ITU (International Telecommunication Union) shows that user demand for broadband services, terminal mobility and roaming availability is becoming increasingly intensive. The operation trend of new networks is focusing more on access and wireless coverage. And the wireless local networks (WLAN) have reached a great success in this field.

But at the moment, the WLAN has several limitations such as small coverage areas (transmission range) and interference problems due to the use of ISM (Industrial, Scientific and Medical) band frequency. To overcome these limitations, the IEEE committee developed 802.16x standards, which specified wireless broadband access (WBA) technologies. The standard defines the physical layer and MAC (Medium Access Control) WBA system in the range 2-66 GHz. It can operate with a maximum data rate of 75 Mbps and wireless coverage of up to 50 km.

The standard IEEE 802.16x consists of several sub-standards: 802.16, 802.16a, 802.16d and 802.16e [2-4]. The IEEE 802.16 defines fixed wireless broadband access (FWBA) in the range 10-66 with LOS (line of sight); IEEE 802.16a extends for 2-11 GHz without LOS. IEEE 802.16d is the revision of both with some improvements in reverse link; IEEE 802.16e supports mobility up to a speed of 120 km/h and a structure of asymmetric link.

Section 2 presents the architecture and WiMAX network topology. Section 3 describes the main elements of the physical layer (PHY) and adaptive modulation of a WiMAX simulator. Section 4 presents the operation of convolutional turbo coding employed in WiMAX. Section 5 shows the results and validation of convolutional turbo code implemented in VHDL.

Ailton Akira Shinoda is with Faculdade de Engenharia de Ilha Solteira, Universidade Estadual Paulista, São Paulo, Brazil (e-mail: shinoda@dee.feis.unesp.br).

This work is supported by CNPq through the process 472686/2008-9.

II. WiMAX Architecture and Network

The topology and network architecture specified by IEEE 802.16 is illustrated in Fig. 1.

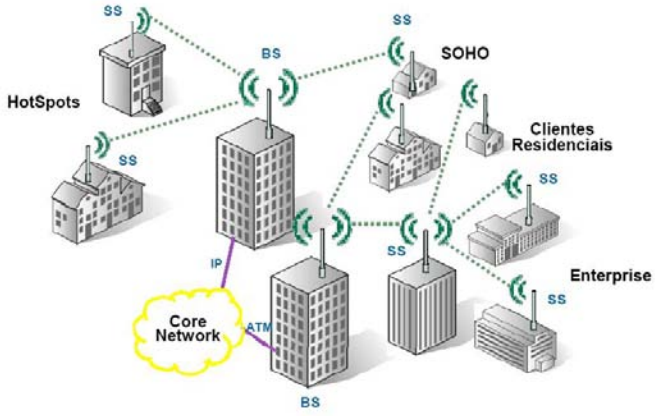

Fig. 1. WiMAX network architecture and topology

WiMAX system consists of Base Station (BS) and Subscriber Station (SS). The BS performs the interface between the wireless network and a core network, supporting IP, ATM, Ethernet or E1/T1 interface [2]. The SS allows the user to access the network through the establishment of links with BS in a point-multipoint topology.

As an alternative to point-multipoint topology, the standard specifies the mesh topology (optional), in which an SS can connect to one or more intermediate SS to reach BS. In this case, it is a multihop network, which represents an interesting strategy to expand the coverage area of the network without the need for a proportional increase in the number of BS´s, which represents a significant saving in deployment costs, since the SS´s should cost well below that of BS´s.

III. Physical Layer

The standard 802.16a/d defines seven combinations of modulation and coding rate that can be employed to achieve the data rate more appropriate, based on channel conditions and interference. Table I illustrates these possible combinations.

Table I Modulation and coding schemes for WiMAX

TRate (ID)	Modulation	Rate Code	Inf. bits/ Symbol	Inf. bits/ OFDM symbol	Peak rate at 5 MHz (Mbps)
0	BPSK	1/2	0.5	88	1.89
1	QPSK	1/2	1	184	3.95
2	QPSK	3/4	1.5	280	6.00
3	16QAM	1/2	2	376	8.06
4	16QAM	3/4	3	568	12.18
5	64QAM	2/3	4	760	16.30
6	64QAM	3/4	4.5	856	18.36

Fig. 2 shows the various functional stages of the WiMAX PHY.

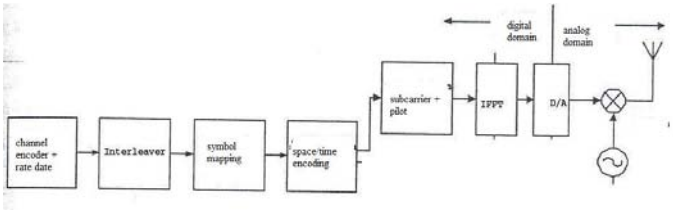

Fig. 2. Functional stages of the WiMAX PHY

The first set of stages is related to error correcting codes (FEC), which includes channel coding (this work utilizes the convolutional turbo code) and the matching rate data (repetition or puncturing), interleaving and mapping symbols. The next set of functional stages is the construction of OFDM symbols in the frequency domain. During this stage, the data are mapped into sub-channels and sub-carriers most suitable. The following symbols are inserted pilots, responsible for estimating and tracking of channel conditions at reception. This stage is also responsible for any encoding temporal/spatial diversity in the transmission, if required. The final set of functions is related to the conversion of OFDM symbols in the frequency domain to time domain.

In the WiMAX technology, in addition to multiplexing OFDM scheme, it is adopted a scheme of adaptive modulation as shown in Fig. 3.

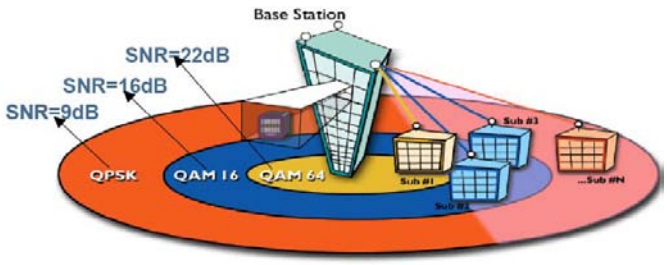

Fig. 3. Adaptive modulation scheme [1]

The selection of modulation to be used in physical layer (QPSK, 16-QAM, 64-QAM) depends on the level of signal to noise ratio at the receiver output. From the negotiation between base stations and user, the modulation to be taken is dynamically adapted to the conditions of the radio link. This technique gives greater robustness and flexibility to the system, used widely in Wi-Fi [2].

IV. Convolutional Turbo Code

The basic idea of turbo codes is the use of two convolutional codes in parallel with interleaving between them. Convolutional codes are used to encode a continuous stream of data, but in this case it is assumed that the data are configured in fixed blocks, associated with the size of the interleaver. Fig. 4 shows a general turbo encoder.

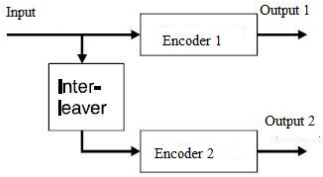

Fig. 4. Turbo encoder

Convolutional turbo code (CTC) implemented in fixed point was based on the IEEE Std 802.16 specification [4] shown in Fig. 5.

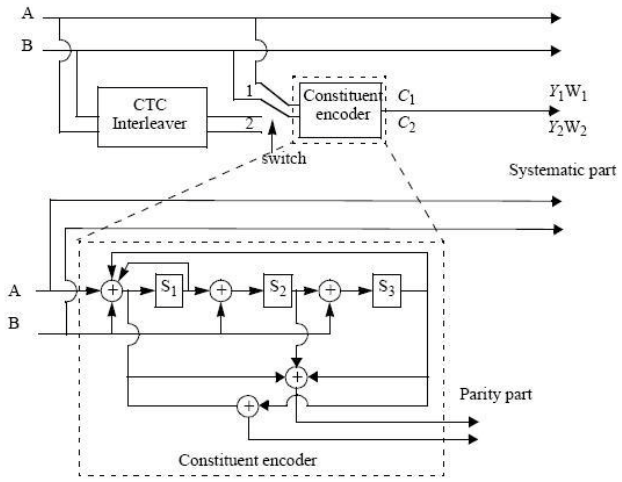

Fig. 5. CTC coding

The bits are sent alternately to *A* and *B*, starting with the most significant bit (MSB) of the first byte being transmitted by the *A*. The encoder is fed by blocks in *k* bits or *N* tulles (*k=2*N* bits). For any frame size, k is a multiple of 8 and *N* is a multiple of 4. Moreover, $8 \leq N/4 \leq 1024$.

The polynomials that describe the connections in the symbolic notation are:

- Feedback: $1+D+D^3$
- *Y* parity bit : $1+D^2+D^3$
- *W* parity bit: $1+D^3$

Initially, the encoder is fed by a natural sequence (position 1) with incremental address *i=0...N-1*. The first encoder is known as the code C_1. Then the encoder is fed by a sequence

of interleaver (switch in position 2) with incremental address $j=0,...N-1$. This second encoder is known as the code C_2.

The order in which the bit is encoded in the output is:

$$A,B,Y_1,Y_2,W_1,W_2=$$
$$A_0,B_0,\ldots, A_{N-1},B_{N-1},Y_{1,0},Y_{1,1},\ldots,Y_{1,N-1}, Y_{2,0},Y_{2,1},\ldots,Y_{2,N-1},$$
$$W_{1,0},W_{1,1},\ldots,W_{1,N-1}, W_{2,0},W_{2,1},\ldots,W_{2,N-1}$$

The encoding block size shall depend on the number of subchannels allocated and the modulation specified for the current transmission. Concatenation of a number of subchannels shall be performed in order to make larger blocks of coding where it is possible, with the limitation of not passing the largest block under the same coding rate (the block defined by 64-QAM modulation). Table II [4] specifies the concatenation of subchannels for different allocation and modulation.

Table II Optimal CTC channel coding per modulation

Modulation	Date (bytes)	Encoded Date (bytes)	Rate	N	P0	P1	P2	P3
QPSK	6	12	1/2	24	5	0	0	0
QPSK	12	24	1/2	48	13	24	0	24
QPSK	18	36	1/2	72	11	6	0	6
QPSK	24	48	1/2	96	7	48	24	72
QPSK	30	60	1/2	120	13	60	0	60
QPSK	36	72	1/2	144	17	74	72	2
QPSK	48	96	1/2	192	11	96	48	144
QPSK	54	108	1/2	216	13	108	0	108
QPSK	60	120	1/2	240	13	120	60	180
QPSK	9	12	3/4	36	11	18	0	18
QPSK	18	24	3/4	72	11	6	0	6
QPSK	27	36	3/4	108	11	54	56	2
QPSK	36	48	3/4	144	17	74	72	2
QPSK	45	60	3/4	180	11	90	0	90
QPSK	54	72	3/4	216	13	108	0	108
16-QAM	12	24	1/2	48	13	24	0	24
16-QAM	24	48	1/2	96	7	48	24	72
16-QAM	36	72	1/2	144	17	74	72	2
16-QAM	48	96	1/2	192	11	96	48	144
16-QAM	60	120	1/2	240	13	120	60	180

V. RESULTS

The implementation of the CTC was separated into three blocks, shown in Fig. 6.

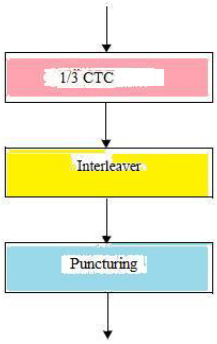

Fig. 6. Block diagram of CTC subpacket generation.

The interleaving scheme used, according to the specification [4], is illustrated in Fig. 7.

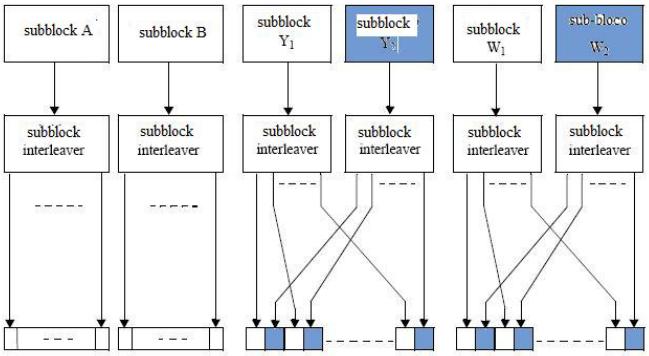

Fig. 7. Block diagram of the interleaving scheme

The platform used to implement the blocks was the Simulink®[5], shown in Fig. 8, the CTC and the parameters used in this work are hatched in red in Table II.

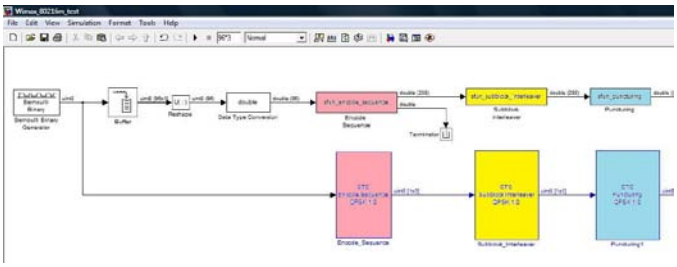

Fig. 8. Development Platform

The upper blocks in Fig. 8 were implemented in Simulink®, according to the diagram in Fig. 6, employing the C mex s-function [5].

```
static void mdlOutputs(SimStruct *S, int_T tid)
{
    uint_T i,initial_state;
    uint_T * output_y1w1,* output_y2w2,* output,* data_interleaved;

    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    const real_T *N    = mxGetPr( N(S) );
    const real_T *P0   = mxGetPr( P0(S) );
    const real_T *P1   = mxGetPr( P1(S) );
    const real_T *P2   = mxGetPr( P2(S) );
    const real_T *P3   = mxGetPr( P3(S) );
    real_T            *y    = ssGetOutputPortRealSignal(S,0);
    real_T            *y1    = ssGetOutputPortRealSignal(S,1);

    output_y1w1     = mxCalloc(2*(uint_T)(*N),sizeof(uint_T));
    output_y2w2     = mxCalloc(2*(uint_T)(*N),sizeof(uint_T));
    data_interleaved= mxCalloc(2*(uint_T)(*N),sizeof(uint_T));


   // Encoder 1
   //initial_state=determine_circular_state(data_block,N);
   //unsigned int determine_circular_state(unsigned int * data, unsigned int N)
```

Fig. 9 Piece of code sfun_encode_sequence

Fig. 9 shows a part of the code sfun_encode_sequence, referring to the first block (pink) of Fig. 6.

The bottom blocks in Fig. 8 were implemented in fixed-point, and it was used eight bits as the standard unit.

```
if index < 96
    index= index+1;
else
 %*************************************************************************
    %Encoder 1
    %initial_state=determine_circular_state(data_block,N);
    state       =       fi(1:3,0,8,0);
    next_state =        fi(1:3,0,8,0);
    output_y1w1=        fi(1:96,0,8,0);
    output_y2w2=        fi(1:96,0,8,0);
    data_interleaved= fi(1:96,0,8,0);
    for i=1:3
        state(i)= uint8(0);
        next_state(i)=uint8(0);
    end
    for i=0:47
        next_state(1)=getlsb(in_r(2*i+1)+in_r(2*i+2)+state(1)+state(3));
        next_state(2)=getlsb(in_r(2*i+2)+state(1));
        next_state(3)=getlsb(in_r(2*i+2)+state(2));

        for j=1:3
            state(j)=next_state(j);
        end
    end
    final_state= state(3)+uint8(2)*state(2)+uint8(4)*state(1);


    %index=((uint_T)(*N)%7)-1;
    initial_state= Circulation_State_Lookup_Table(int(getfi(final_state))+1);
```

Fig. 10. Piece of code Encode_Sequence

Fig. 10 shows a part of the code Encode_Sequence fixed-point implemented in Simulink® [5].

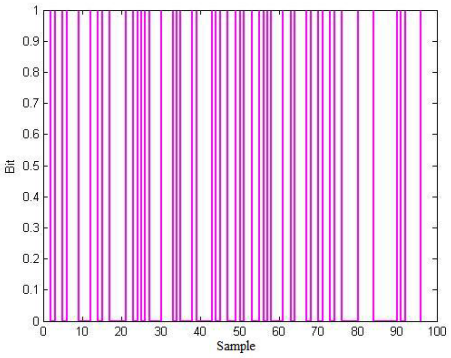

Fig. 11. Frame of 96 bits

Fig. 11 shows a frame random 96-bit block as input for sfun_encode_sequence and Encode_Sequence (Fig. 8).

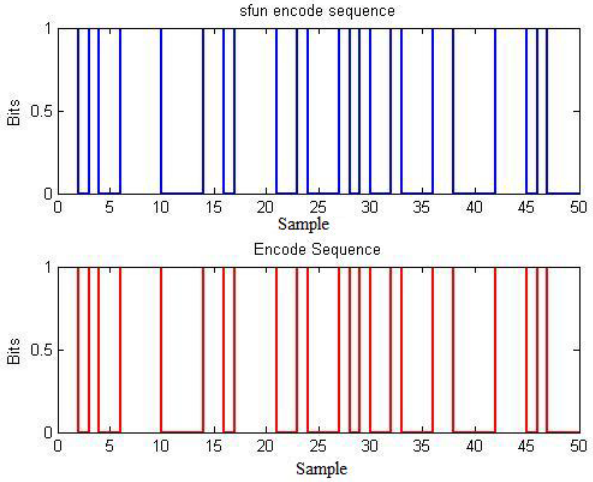

Fig. 12. Partial output of sfun_encode_sequence (Encode_Sequence)

Fig. 12 shows partial output of the block Encode_Sequence in fixed-point and its validation block sfun_encode_sequence.

From the code Encode_Sequence fixed-point was generated the VHDL code. Fig. 13 shows an excerpt from Encode_Sequence in VHDL.

```
IF index < 96.0 THEN
    index_next <= index + 1.0;
ELSE
    --*************************************************************************
    --Encoder 1
    --initial_state=determine_circular_state(data_block,N);
    output_y1w1 := e;
    output_y2w2 := e;
    state := (0 TO 2 => to_unsigned(0, 8));

    FOR f_i IN 0 TO 47 LOOP
        add_temp(f_i) := resize(resize(resize(in_r_temp(2 * f_i), 9) +
                         resize(in_r_temp((2 * f_i) + 1), 9), 10) + resize(state(0), 10), 11) +
                         resize(state(2), 11);
        next_state(0) := '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & add_temp(f_i)(0);
        add_temp_1(f_i) := resize(in_r_temp((2 * f_i) + 1), 9) + resize(state(0), 9);
        next_state(1) := '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & add_temp_1(f_i)(0);
        add_temp_2(f_i) := resize(in_r_temp((2 * f_i) + 1), 9) + resize(state(1), 9);
        next_state(2) := '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & add_temp_2(f_i)(0);
        state := next_state;
    END LOOP;
```

Fig. 13. Excerpt from Encode_Sequence in VHDL

The validation of the VHDL code generated was performed using the package from Mentor Graphics ModelSim ®[6]. Fig. 14 shows the result of simulation through ModelSim ®.

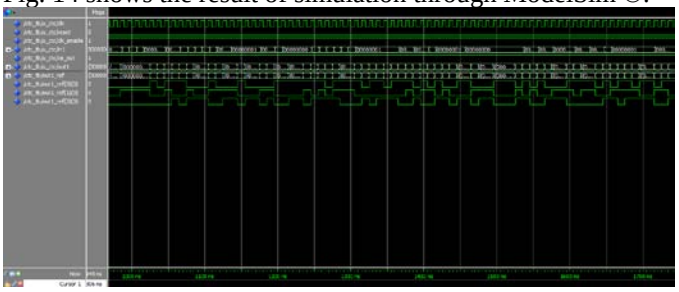

Fig. 14. Simulation of VHDL code Encode_Sequence

Fig. 15 shows partial output of the block Subblock_Interleaver in fixed-point and its validation block sfun_subblock_interleaver.

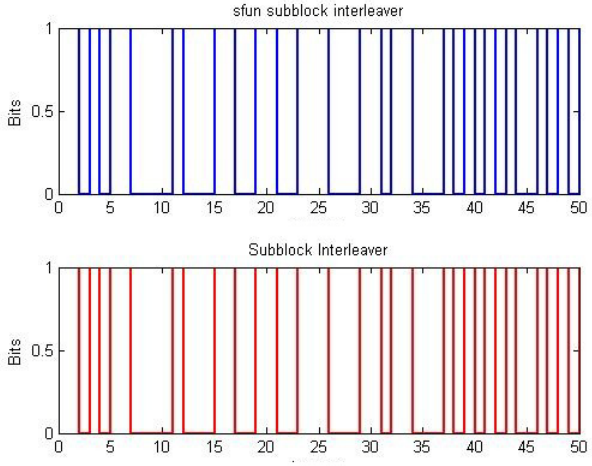

Fig. 15. Partial output of sfun_subblock_interleaver (Subblock_Interleaver)

Fig. 16 shows an excerpt from Subblock_Interleaver in VHDL.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
USE work.Subblock_Interleaver_pkg.ALL;

ENTITY Subblock_Interleaver IS
  PORT( clk                               :   IN    std_logic;
        reset                             :   IN    std_logic;
        clk_enable                        :   IN    std_logic;
        In1                               :   IN    vector_of_std_logic_vector8(0 TO 2);  -- uint8 [3]
        ce_out                            :   OUT   std_logic;
        Out1                              :   OUT   vector_of_std_logic_vector8(0 TO 2)  -- uint8 [3]
        );
END Subblock_Interleaver;


ARCHITECTURE rtl OF Subblock_Interleaver IS

  -- Component Declarations
  COMPONENT eml_5ubblock_Interleaver
    PORT( clk                             :   IN    std_logic;
          clk_enable                      :   IN    std_logic;
          reset                           :   IN    std_logic;
          tx                              :   IN    vector_of_std_logic_vector8(0 TO 2);  -- uint8 [3]
          out_rsvd                        :   OUT   vector_of_std_logic_vector8(0 TO 2)  -- uint8 [3]
          );
  END COMPONENT;
```

Fig. 16. Excerpt from Subblock_Interleaver in VHDL

Fig. 17 shows the result of test bench generated by Subblock_Interleaver in VHDL with ModelSim®.

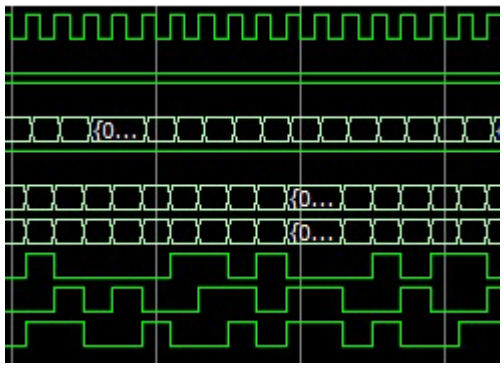

Fig. 17. Simulation of VHDL code Subblock_Interleaver

Fig. 18 shows the output of Puncturing and its validation block sfun_puncturing when the data flow is given by Fig. 11.

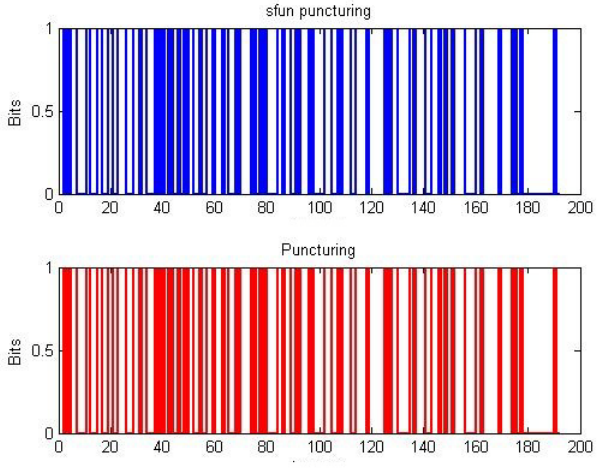

Fig. 18. Output of sfun_puncturing (Puncturing)

Fig. 19 shows an excerpt from Puncturing in VHDL.

```
USE IEEE.numeric_std.ALL;
USE work.Puncturing_pkg.ALL;

ENTITY Puncturing IS
  PORT( clk                               :   IN    std_logic;
        reset                             :   IN    std_logic;
        clk_enable                        :   IN    std_logic;
        In1                               :   IN    vector_of_std_logic_vector8(0 TO 2);  -- uint8 [3]
        ce_out                            :   OUT   std_logic;
        Out1                              :   OUT   vector_of_std_logic_vector8(0 TO 1)  -- uint8 [2]
        );
END Puncturing;


ARCHITECTURE rtl OF Puncturing IS

  -- Component Declarations
  COMPONENT eml_puncturing
    PORT( clk                             :   IN    std_logic;
          clk_enable                      :   IN    std_logic;
          reset                           :   IN    std_logic;
          in_rsvd                         :   IN    vector_of_std_logic_vector8(0 TO 2);  -- uint8 [3]
          out_rsvd                        :   OUT   vector_of_std_logic_vector8(0 TO 1)  -- uint8 [2]
          );
  END COMPONENT;

  -- Component Configuration Statements
  FOR ALL : eml_puncturing
    USE ENTITY work.eml_puncturing(fsm_SFHDL);
```

Fig. 19. Excerpt from Puncturing in VHDL

Fig. 20 shows the result of test bench generated by Puncturing in VHDL with ModelSim®.

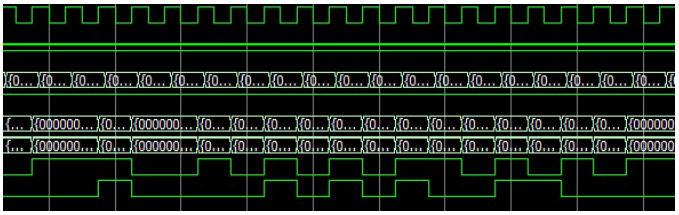

Fig. 20. Simulation of VHDL code Puncturing

VI. CONCLUSION

This work described the generation of the convolutional turbo code, in fixed-point and its conversion to VHDL, of an OFDM system based on adaptive control of the data rate physical layer of WiMAX in accordance with the IEEE 802.16.

With the intent to validate the generated code, the CTC was developed in s-function [5] and the results were identical for the same input stream of bits. The next step it will be an implementation on a FPGA device.

REFERENCES

[1] A. Ghosh et al, “Broadband Wireless Access with WiMAX/802.16: Current Performance Benchmarks and Future Potential,” IEEE Communication Mag., vol. 43, no.2, pp. 129-136, 2005.

[2] IEEE 802.16-2001, “IEEE Standard for Local and Metropolitan Area Network – Part 16: Air Interface for Fixed Broadband Wireless Access Systems”, Apr. 8, 2002.

[3] IEEE 802.16a, “IEEE Standard 802.16a, Amendment 2: Medium Access Control Modifications and Additional Physical Layer Specifications for 2-11 GHz”, 2003.

[4] IEEE 802.16e, “IEEE 802.16e TGe Working Document, (Draft Standard) – Amendment for Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licenses Bands, 802.16e/D4”, 2004.

[5] MathWorks Inc., “Simulink User Manual”, 2009.

[6] Mentor Graphics Corp., “ModelSim SE User Guide”, 2010.