Ministry of Education and Science of Ukraine
National University "Zaporizhzhia Polytechnic"

GUIDELINE
for laboratory work
on the discipline "Methods for the synthesis of energy-efficient systems on FPGAs"
on the topic "Developing a Convolutional Encoder"

For Master's students
of the specialty F7 Computer Engineering
According to the educational program
"Specialized Computer Systems"

2025

Guideline for laboratory work on the discipline "Methods for the synthesis of energy-efficient systems on FPGAs" for Master`s students of specialty F7 "Computer Engineering" according to the educational program "Specialized Computer Systems" for all forms of education / Compl.: I. Zelenova, T. Holub, S. Hrushko – Zaporizhzhia: NU "Zaporozhzhia Polytechnic", 2025. – 17 p.

Compilers: Irina Zelenova,
Ph.D., Associate Professor
Tetiana Holub,
Ph.D., Associate Professor
Svitlana Hrushko,
Ph.D., Associate Professor

Reviewer: Ravil Kudermetov,
Ph.D., Associate Professor

Responsible
for issue: Irina Zelenova, Ph.D., Associate Professor

Approved
at the meeting of the Department
"Computer Systems and Networks"
Protocol № 2 from 29.08.2025

Recommended for publication
SMC of Faculty CST
Protocol № 2 from 10.09.2025

CONTENTS

INTRODUCTION

Convolutional codes are highly relevant due to their efficient implementation and good performance at a relatively low decoding complexity compared to block codes, especially in channels with random errors.

Systematic vs. Non-systematic: In systematic codes, the input bits appear directly in the output stream, while non-systematic codes mix the input for maximum error-correction capability [1, 3].

Performance: The error-correcting capability is characterized by the free distance (d free), which determines the code's minimum distance and thus its asymptotic performance [1].

Convolutional codes have found broad application in scenarios where reliable data transmission over noisy channels is critical [2]. Notable applications include:

- Wireless Communications: Essential in many wireless standards, including 3G (UMTS) and earlier standards like GSM, and often used as the inner code in concatenated coding schemes [2].

- Satellite Communications: Employed in deep-space missions (e.g., Voyager, Galileo) and commercial satellite systems due to their robustness against faint signals and long-distance noise.

- Digital Video Broadcasting (DVB): Used in terrestrial, cable, and satellite television standards for robust signal reception.

- Turbo Codes and LDPC Codes: Convolutional codes serve as the building blocks for modern, high-performance Turbo codes (using parallel concatenated CCs) and are sometimes used in conjunction with Low-Density Parity-Check (LDPC) codes in hybrid schemes, highlighting their enduring fundamental role in modern coding theory [4-5].

In conclusion, convolutional codes remain a cornerstone of channel coding, offering an excellent balance of performance and complexity, and serving as a fundamental component in many contemporary communication systems.

LABORATORY WORK
DEVELOPING OF CONVOULTIONAL CODER

The purpose of the work: to understand the basic principles of convolutional code, familiarize yourself with coding and develop a convolutional coder in the VHDL language.

1 THEORETICAL INFORMATION

Convolutional code is used for error correction. The meaning is that if errors appear in the convolutional code when it is transmitted in the communication channels, then when it is decoded, you can get the original correct bit sequence that was encoded. This is achieved due to redundancy in the encoding of the convolutional code.

Convolutional codes are represented by three parameters (n, k, m).

- n is the number of output coded bits.
- k is the number of input bits to be encoded.
- m is number of memory registers to store previous input bits.

In general, these parameters can be selected by the designer.

The convolutional code encoder is essentially a shift register whose outputs are connected to half-adders. **The number of half-adders required to form one bit of code is determined by the number of output bits of the shift register that must be added to obtain the output bit of code.** One half-adder – two output bits of the shift register per one output bit of the code, two half-adders – three output bits of the register per one output bit of the code, etc.

The ratio "bit of data (k)/number of bits of code (n)" is called the coding rate, usually denoted by R. For two bits of code per one bit of data R=1/2, for three bits of code per one bit of data R=1/3 etc.

The length of the shift register is denoted by m and determines the number of data bits that affect the code bits.

In this laboratory work, as an example, the convolutional code (2, 1, 4) will be considered.

With the selected convolutional code encoding method (2, 1, 4), each input bit will be encoded into two output bits. The encoding of these two output bits is done taking into account the new input bit and the previous three input bits that have already been encoded (i.e., 4 memory registers are required). Each of the output encoded bits is not necessarily encoded considering all three previous bits. In this example, the first output bit is encoded by summing modulo 2 (half-adder) of all three stored previous input bits and the input, and the second by the sum of the first, third and input bits.

2 EXAMPLE OF CONVOLUTIONAL CODING (2, 1, 4)

Let's choose an input sequence of bits for encoding – for example, 1011.

In the initial state, the cells of the shift register contain zeros. Further to the entrance, the first one on the left, the first informational symbol arrives. At the same time as writing a symbol to the first cell, its contents are rewritten to the second, the second to the third, and so on. Adders form a sum modulo two of the contents of the encoder cells.

Let's encode the first bit – 1. Since the registers previously contained all zeros, the first bit is formed as follows: 0+0+0+1=1, and the second is formed as follows: 0+0+1=1. Now the registers are already 001 (the input bit is stored in memory).

We encode the second bit - 0. Since the registers already contain 001, the first bit is formed as follows: 0+0+1+0=1, and the second bit is 0+1+0=1. Now the registers already have 010.

Let's code the third bit – 1. Since the registers already have 010, the first bit is formed as follows: 0+1+0+1=0, and the second bit is 0+0+1=1. Now there are already 101 in the registers.

Let's encode the fourth bit - 1. Since there are already 101 in the registers, the first bit is formed as follows: 1+0+1+1=1, and the second bit is 1+1+1=1. Now the registers already have 011.

As a **result**, we get the coded sequence – 11 11 01 11.

Based on the analysis of the above coding rules, it is possible to form a truth table for data coding (Table 1).

Table 1 - Truth table for data encoding

Input Bit	Input State				Output Bits			Output State		
I_1	s_1	s_2	s_3		O_1	O_2		s_1	s_2	s_3
0	0	0	0		0	0		0	0	0
1	0	0	0		1	1		1	0	0
0	0	0	1		1	1		0	0	0
1	0	0	1		0	0		1	0	0
0	0	1	0		1	0		0	0	1
1	0	1	0		0	1		1	0	1
0	0	1	1		0	1		0	0	1
1	0	1	1		1	0		1	0	1
0	1	0	0		1	1		0	1	0
1	1	0	0		0	0		1	1	0
0	1	0	1		0	0		0	1	0
1	1	0	1		1	1		1	1	0
0	1	1	0		0	1		0	1	1
1	1	1	0		1	0		1	1	1
0	1	1	1		1	0		0	1	1
1	1	1	1		0	1		1	1	1

The truth table is necessary to determine the law of operation and further implementation of the coder scheme. Based on this table, the functions of the output signals of the device are formed.

3 IMPLEMENTATION AND TESTING OF THE CODER (2, 1, 4)

From the above considerations, we can conclude that to implement the encoder, we only need a shift register to store the previous three coded bits, as well as half-adders to form the output bits.

We will describe the coder scheme in VHDL in the Quartus II package. Modeling and testing of the circuit will be performed on the Altera\Intel microcircuits.

So, let's describe a half-adder that sums two input bits (Listing 3.1).

Listing 3.1 – Implementation of the half-adder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity h_adder is
   port(a,b:in std_logic;
        s:out std_logic);
end h_adder;
architecture fh1 of h_adder is
begin
     s<=NOT(a XOR (NOT b));
end fh1;
```

Let's describe the D-flip-flop used in the shift register (Listing 3.2).

Listing 3.2 – Implementation of D-flip-flop

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity DTrigger is
     Port ( d : in std_logic;
         clk : in std_logic;
         q : out std_logic);
end DTrigger;
architecture Behavioral of DTrigger is
```

The end of listing 3.2

```
signal q_temp:std_logic;
begin
 process(clk)
 begin
   if(clk'event and clk='1')then
        q_temp<=d;
   end if;
 end process;
 q<=q_temp;
end Behavioral;
```

Let's describe the shift register in VHDL (Listing 3.3).

Listing 3.3 – Shift register implementation

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity ShiftReg is
    Port( a : in std_logic; -- input bit
          clk : in std_logic; -- according to clk,
     the triggers change their state
          b1: out std_logic; -- register outputs
     (triggers)
          b2: out std_logic;
          b3: out std_logic;
          b4: out std_logic);
end ShiftReg;
architecture gen_shift of ShiftReg is
signal z:std_logic_vector(0 to 3);
component DTrigger   -- Creating a D-trigger
     component
   port (d,clk:in std_logic;
         q:out std_logic);
end component;
```

The end of listing 3.3

```
begin
DTrigger0: DTrigger port map(a,clk,z(0));  -- by
     clk, the value of a is written to z(0)
DTrigger1: DTrigger port map(z(0),clk,z(1)); --
     etc.
DTrigger2: DTrigger port map(z(1),clk,z(2));
DTrigger3: DTrigger port map(z(2),clk,z(3));
b1<=z(0);
b2<=z(1);
b3<=z(2);
b4<=z(3);
end gen_shift;
```

Now it remains only to assemble the encoder, as shown in Figure 3.1. To simplify the process of combining functional blocks, we will use the circuit editor. To do this, we will create a Block Diagram File and add the created VHDL files to the workspace.

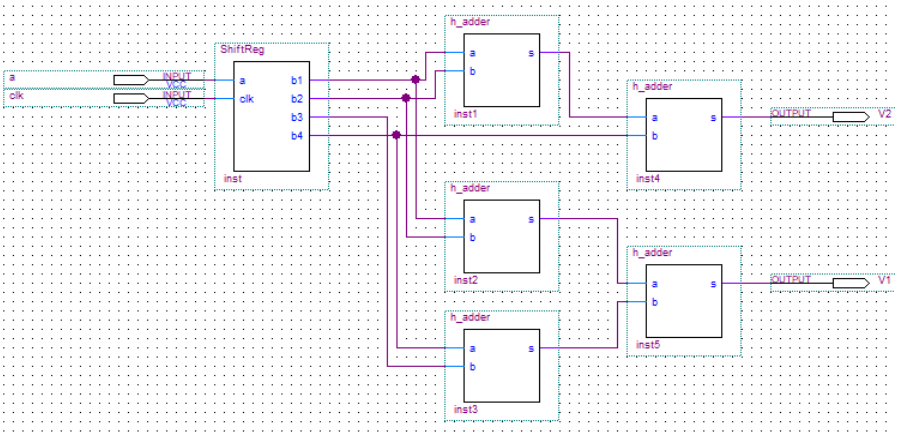


Figure 1 – Functional diagram of the Coder

As can be seen in Figure 3.1, three half-adders are required to calculate the sum of the four bits of the register to obtain the first output bit. Two half-

adders are sufficient to add the three bits of the register to obtain the second output bit.

To test the created circuit, we will create a Vector Waveform File and add circuit signals to it. To set the value of the clk input signal, select it, click Overwrite Clock, enter the data as shown in Figure 3.2 and click OK.

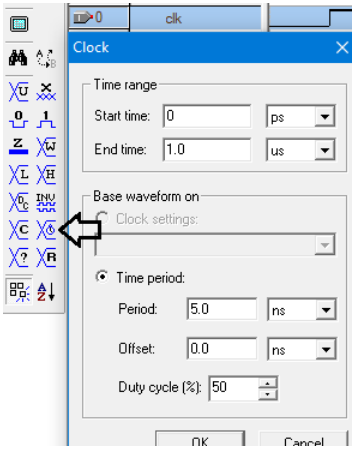


Figure 3.2 – Assignment of the value of the clk input signal

To set the value of the input signal a, select it, click Arbitrary Value, enter the data as shown in Figure 3.3 and click OK.

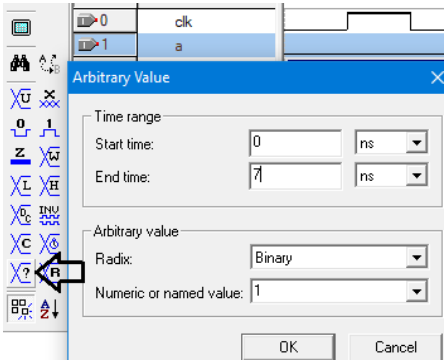


Figure 3.3 – Assignment of the value of the input signal *a*

Click Arbitrary Value again and enter the data as shown in Figure 3.4 and click OK.

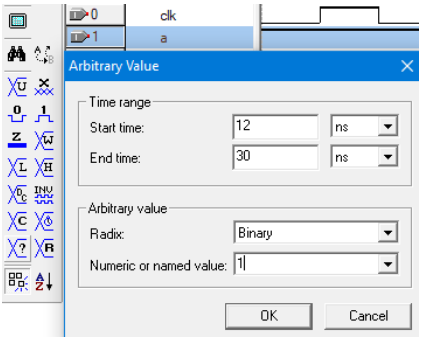


Figure 3.4 – Assignment of the value of the input signal *a*

Figure 3.5 shows the timing diagram of the encoder operation, obtained as a result of testing the scheme.

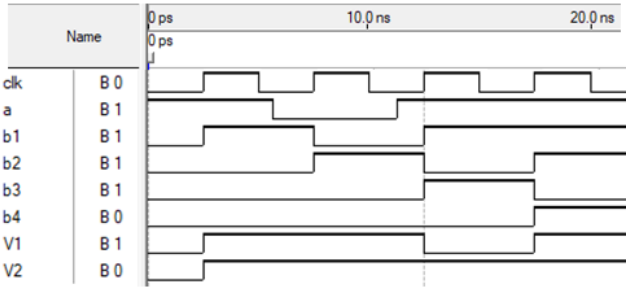
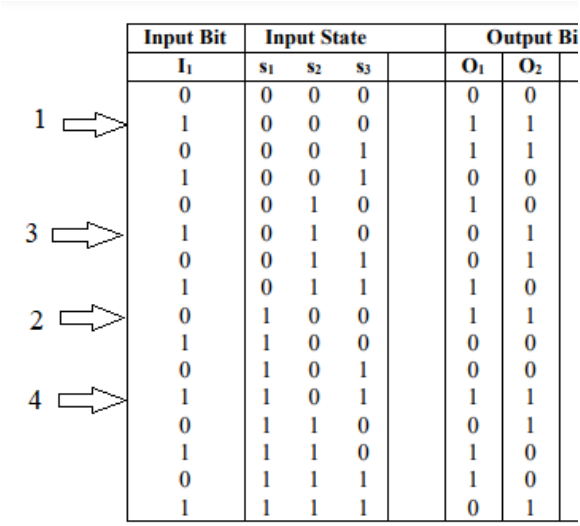


Figure 3.5 – Time diagram of encoder

As you can see, the diagram works according to the truth table. The sequence of coding steps for ease of verification is shown in Figure 3.6.



Input Bit	Input State				Output Bi	
I_1	s_1	s_2	s_3		O_1	O_2
0	0	0	0		0	0
1	0	0	0		1	1
0	0	0	1		1	1
1	0	0	1		0	0
0	0	1	0		1	0
1	0	1	0		0	1
0	0	1	1		0	1
1	0	1	1		1	0
0	1	0	0		1	1
1	1	0	0		0	0
0	1	0	1		0	0
1	1	0	1		1	1
0	1	1	0		0	1
1	1	1	0		1	0
0	1	1	1		1	0
1	1	1	1		0	1

Figure 3.6 – Sequence of coding steps

The state of the shift register depends only on the transmitted information symbols. At the same time, no matter how many information symbols are transmitted, the number of states of the shift register is limited and is 2k, where k is the length of the shift register.

The sequence of states is also not random. Each subsequent state comes from the previous one by shifting to the right, that is, it "inherits" three bits from the previous state. The fourth bit takes the value "1" or "0", depending on the next input data bit.

4 INDIVIDUAL TASK FOR CONVOLUTIONAL CODER DEVELOPING

1. Independently select and justify the parameters of a convolutional code (n, k, m):
   - n is the number of output coded bits.
   - k is the number of input bits to be encoded.
   - m is number of memory registers to store previous input bits.

2. Develop and test an encoder circuit, and conclude on its correct operation. Provide a detailed report on all development and testing stages.

3. Analyze the implementation report and provide a screenshot of the encoder circuit topology on the selected chip.

4. Answer two control questions of your choice.

5 CONTROL QUESTIONS

1. What is the coding rate for a convolutional code?
2. What are the main parameters for determining the convolutional code?
3. What does the state of the shift register depend on?
4. What are convolutional codes used for?
5. Why does convolutional code allow error correction?

LIST OF REFERENCE SOURCES

1. Sidorenko V. R., Li W., Günlü O., Kramer G. Skew Convolutional Codes. MDPI Entropy. 2020. 22(12):1364. DOI:10.3390/e22121364. URL: https://www.researchgate.net/publication/347325133_Skew_Convolutional_Codes
2. Signal processing & simulation newsletter. : complextoreal.com, 2018. URL: http://complextoreal.com/wp-content/uploads/2018/09/convolutional-codes.pdf
3. Introduction to convolutional codes. URL: https://ocw.mit.edu/courses/6-451-principles-of-digital-communication-ii-spring-2005/43162a4e10d73639903282f4dd58001b_chap9.pdf
4. Convolutional Neural Network (CNN) in Machine Learning. URL: https://www.geeksforgeeks.org/deep-learning/convolutional-neural-network-cnn-in-machine-learning/
5. Design of Convolutional Codes for Varying Constraint Lengths URL:https://www.researchgate.net/publication/384598843_Design_of_Convolutional_Codes_for_Varying_Constraint_Lengths