НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Кафедра конструювання електронно-обчислювальної апаратури

КУРСОВА РОБОТА

з дисципліни _____Автоматизація проектування цифрових пристроїв_____
на тему: передавач та приймач сповіщень азбукою Морзе

Студента 3 курсу групи ДК-91
Напряму підготовки: Телекомунікації та радіотехніка
_____Геращенко А.Ю._____
(прізвище та ініціали)
Керівник:
_____ст. викладач Антонюк О. І._____
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Національна оцінка: _____
Кількість балів: _____ Оцінка: ECTS _____

Члени комісії: _____ ст. викладач Антонюк О. І._____
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

_____ _____
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

Київ - 2021 рік

ЗМІСТ

СПИСОК УМОВНИХ СКОРОЧЕНЬ

ASCII	American standard code for information interchange
ПЗ	Програмне Забезпечення

ВСТУП

Темою даної курсової роботи є “Створення передавача та приймача сповіщень у коді Морзе”. Азбука Морзе являє собою перелік сигналу, що складаються з ряду цифр, літер алфавіту, розділових знаків та інших символів, що є способом знакового кодування. Сам код складається з точок та тире, що відтворюються за допомогою радіосигналів або переривання постійного електричного струму.

Хоча у сучасному світі існує багато різних способів кодування та передачі інформації, азбука Морзе досі залишається актуальною, зокрема, у військовій справі. Один із інструментів ведення бою сьогодні – придушення різних частот, наприклад, стільникового зв’язку. В таких умовах військовим залишаються лише перевірені часом засоби – телефонний та радіотелеграфний зв'язок, тобто азбука Морзе. Військові зв’язківці постійно працюють над удосконаленням обробки та передачі сигналу, з’явилося багато нового обладнання, але, як і раніше, прийомом сигналу займаються зв’язківці на слух. Крім військових зв’язківців, азбукою Морзе повинні володіти підводники, полярники та метеорологи, щоб отримувати та передавати сигнал із віддалених місць, де не працює жодний інший зв'язок.

Сформулюємо завдання на проектування. Необхідно спроектувати та протестувати передавач та приймач сповіщень азбукою Морзе (використовувати кодування літер для англійського алфавіту). По зовнішньому сигналу «старт» передавач повинен сформувати сповіщення, яке містить прізвище та ім’я студента і номер академічної групи. Сповіщення, що надсилається, зчитується з блоку пам’яті. Приймач повинен прийняти та дешифрувати сповіщення (перекодувати сповіщення у формат ASCII), дешифроване сповіщення необхідно зберегти у блоці пам’яті. Для зупинки приймача використовується знак «кінець передачі». Необхідно витримати наступну умову – тривалість тире втричі перевищує тривалість крапки.

Приклад кодування азбукою Морзе наведено на рис. 1.

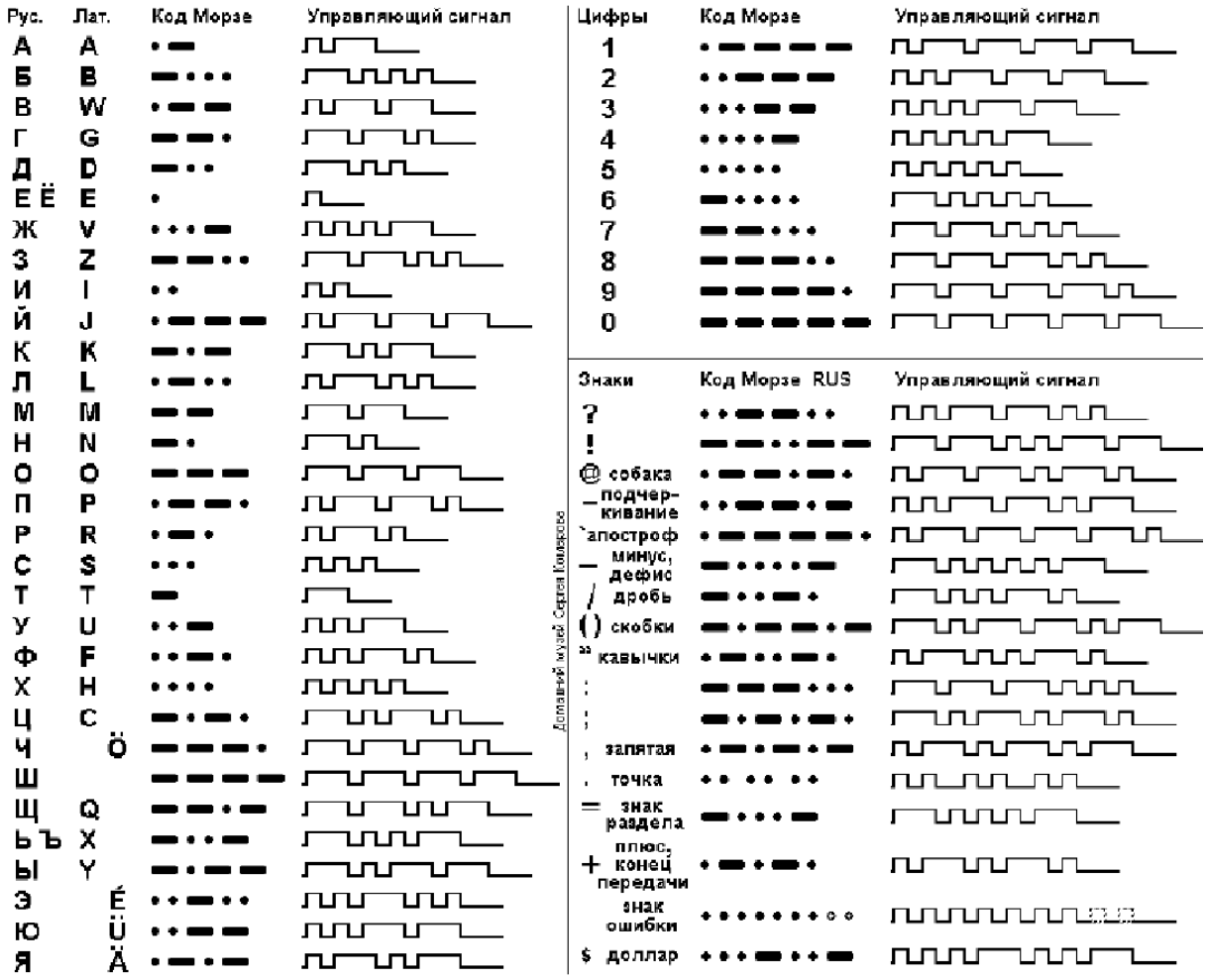

Рис. 1 Кодування символів у коді Морзе

Зважаючи на отримане завдання, я створив структурну схему пристрою, яка складається з блоків передавача та приймача сигналу. Передавач сигналу включає в себе такі елементи:

- лічильник, який потрібен для вказання адреси комірки пам’яті з якої отримують інформацію;
- блок пам’яті в якому записано повідомлення у ASCII форматі;
- шифратор, що перетворює повідомлення з ASCII формату у код Морзе, представлення сигналів показано на рис. 1;
- регістр у який будуть записуватись зашифровані літери у коді Морзе.

Приймач сигналу складається з таких елементів:

- регістр у який записується отриманий код символу з регістру передавача;
- дешифратор, що дешифрує символ у ASCII формат;
- блок пам’яті з лічильником, який потрібний для запису отриманого символу в пам'ять.

Для створення прототипу пристрою буде використано ПЗ Modelsim. Задачею курсової роботи буде створення прототипу приймача та передавача сигналу в коді Морзе для передачі повідомлень з точки «А» у току «Б» та запису повідомлення в блок пам’яті.

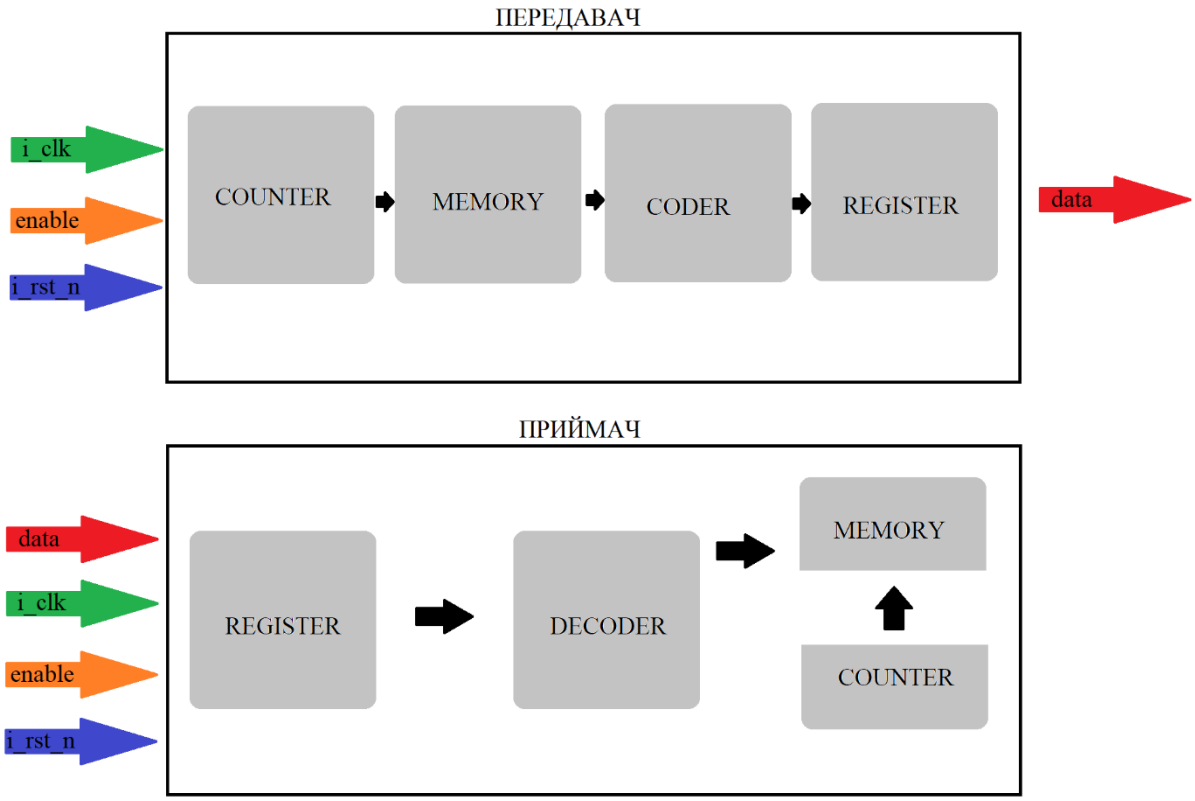

Рис. 2 Структурна схема прототипу пристрою

Оскільки у завданні до курсової роботи вказано, що спроектований пристрій повинен передавати прізвище, ім’я та номер групи студента, я вирішив дещо спростити схему. Спрощення схеми полягає у написанні

функцій для шифратора і дешифратора повідомлень тільки для символів, які будуть зустрічатися у повідомленні “HERASHCHENKO_ARTEM_DK91+”. Таке спрощення дозволить показати логіку роботи шифрування та дешифрування повідомлень і значно зменшить обсяг блоків опису роботи шифратора та дешифратора.

ОСНОВНА ЧАСТИНА

АНАЛІЗ ВИХІДНИХ ДАНИХ ДЛЯ ПРОЕКТУВАННЯ

Зі схеми на рис. 2 видно, що на виході блоку передавача формується 20-ти бітний сигнал, який являє собою представлення даних повідомлення у коді Морзе. Вихідний сигнал має розрядність 20 біт, з рис. 1 можна побачити, що найбільший код мають символи “9” та “1”, щоб усі символи були однакової розрядності я додав до них 0. У таблиці 1 можна ознайомитись з представленням закодованих символів.

Таблиця 1 Представлення ASCII символів у коді Морзе

№	Symbol	HEX	UNSIGNED	MORSE
1	H	48	72	00000000001010101000
2	E	45	69	00000000000000001000
3	R	52	82	00000000001011101000
4	A	41	65	00000000000010111000
5	S	53	83	00000000000010101000
6	H	48	72	00000000001010101000
7	C	43	67	00000011101011101000
8	H	48	72	00000000001010101000
9	E	45	69	00000000000000001000
10	N	4E	78	00000000000011101000
11	K	4B	75	00000000111010111000
12	O	4F	79	00000011101110111000
13	_	5F	95	00001110101010111000
14	A	41	65	00000000000010111000
15	R	52	82	00000000001011101000
16	T	54	84	00000000000000111000
17	E	45	69	00000000000000001000
18	M	4D	77	00000000001110111000
19	_	5F	95	00001110101010111000
20	D	44	68	00000000001110101000
21	K	4B	75	00000000111010111000
22	9	39	57	11101110111011101000
23	1	31	49	10111011101110111000
24	+	2B	43	00001011101011101000

Блок приймача не має на виходів явних сигналів, але проаналізувавши логіку роботи приймача стане зрозуміло, що вихідний сигнал з дешифратора подається на вхід блоку пам’яті у вигляді бінарного представлення ASCII.

ОБГРУНТУВАННЯ ВИБОРУ СКЛАДОВИХ ЧАСТИН ПРИСТРОЮ ТА СПОСОБУ ЇХ ПЕРЕВІРКИ

Для реалізації пристрою, логіка роботи якого була описана в завданні, я вирішив використати такі елементи: два лічильника, два блоки пам’яті, шифратор, два регістри, дешифратор. Пропоную детальніше роздивитись логіку роботи пристрою та його складових частин.

З рис. 3 видно, що на вхід передавача приходять три сигнали: i_clk, enable, i_rst_n.

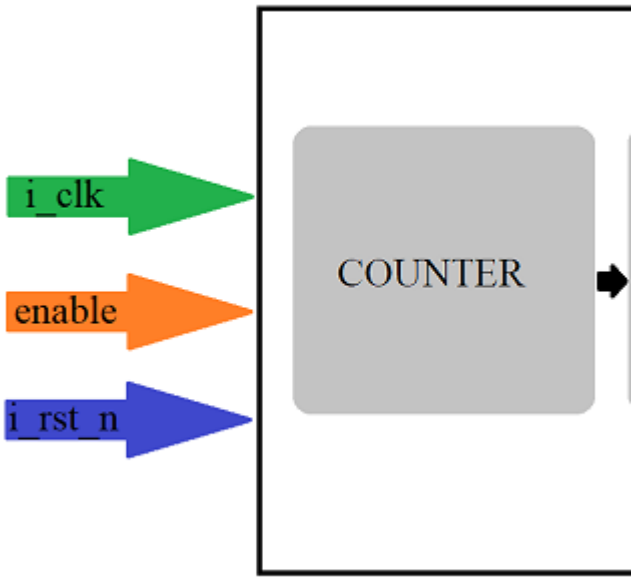

Рис. 3 Фрагмент структурної схеми з вхідними сигналами і блоком лічильника

Сигнал i_clk є тактовим сигналом, він потрібен для тактування схеми. Всі блоки схеми працюють з використанням тактового сигналу. Сигнал enable – це сигнал дозволу, він потрібен для дозволу виконання дій у блоках передавача. Сигнал i_rst_n використовується для виконання збросу лічильника та скидання регістрів.

Лічильник у блоці передавача використовується для формування адреси комірки з блоку пам’яті в якій записано повідомлення. Для початку роботи

лічильника необхідно натиснути кнопку, імітацію натискання кнопки виконує зміна сигналу i_rst_n, що призводить до встановлення лічильника у початковий стан – стан “0”. При активації сигналу enable лічильник почне свій рахунок, рахувати він буде до 24 тому, що кількість символів повідомлення рівна 24.

Наступним у блоці передавача йде блок пам’яті, його позначення на структурній схемі наведено на рис. 4.

Рис. 4 Позначення блоку пам’яті на структурній схемі

Блок пам’яті всередині передавача необхідний для зберігання повідомлення, користувач повинен сформувати своє повідомлення та записати у файл з назвою rom_data.hex, слід зауважити, що запис повідомлення відбувається у шістнадцядковому форматі – кожен символ на окремому рядку. Коди символів для кодування можна взяти з таблиці 1. Для отримання на виході блоку пам’яті даних необхідно подати на його вхід адресу комірки, тактовий сигнал та сигнал дозволу. Дані на виході блоку пам’яті будуть з’являтись по кожному активному фронту тактового сигналу.

З виходу блоку пам’яті дані подаються на вхід шифратора, який перекодовує їх у код Морзе. Окрім входу даних шифратор має вхід тактової частоти та вхід дозволу. Позначення шифратору на структурній схемі можна побачити на рис. 5.

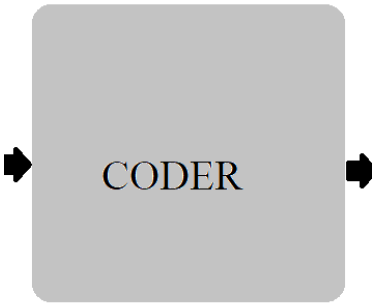

Рис. 5 Позначення шифратора на структурній схемі

Останньою складовою передавача є регістр, в нього записується нове значення закодованого символу по кожному активному фронту. Регістр має розрядність 20 біт, приймає на вхід сигнал даних, тактову частоту, сигнал встановлення та сигнал дозволу запису. Позначення регістру на структурній схемі можна побачити на рис. 6.

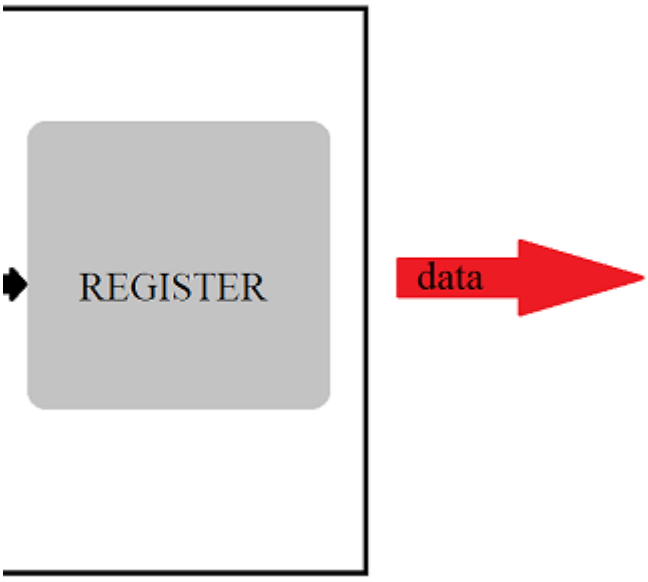

Рис. 6 Позначення регістру на структурній схемі

Блок приймача починається з регістру, у цей регістр працює так само, як регістр на виході передавача. Цей регістр необхідний для запису символу у коді Морзе.

З виходу регістру дані приходять на вхід дешифратора, який виконує обернену операцію від шифратора, тобто розшифровує код Морзе і видає на

вихід значення символу в ASCII. Дешифратор містить входи для тактової частоти та вхід даних. Також в середині дешифратора міститься функціонал за допомогою якого виконується перевірка символів повідомлення на рівність коду знаку “+”, при отриманні результату True припиняється декодування.

Після дешифрування коди символів приходять на вхід блоку пам’яті в якій буде зберігатись повідомлення. Позначення блоку пам’яті приймача на структурній схемі наведено на рис. 7.

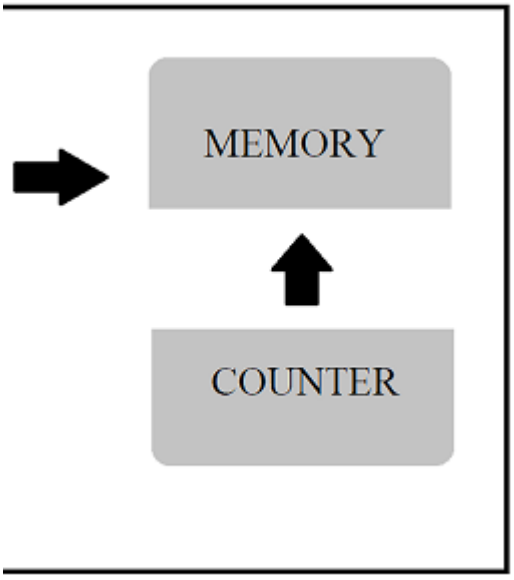

Рис. 7 Позначення блоку пам’яті приймача на структурній схемі

З рис. 7 видно, що блоку пам’яті для запису даних у відповідні комірки необхідно передавати адресу цих комірок. Для формування адреси комірок використовується лічильник, який починає свій рахунок при подачі активного сигналу enable, але цей сигнал є локальним для цього блоку і не має відношення до сигналу, що зображений на структурній схемі. Сигнал дозволу запису встановлюється в активний стан при подачі на вхід i_data блоку пам’яті не 0 даних.

Для перевірки усіх частин приймача та передавача я написав тестові стенди, детальніше про логіку роботи тестових стендів розповідається у наступному пункті.

РОЗРОБКА ФАЙЛІВ ОПИСУ СКЛАДОВИХ ЧАСТИН ПРИСТРОЮ ТА СПОСОБИ ЇХ ПЕРЕВІРКИ

Розглянемо по черзі усі складові частини пристрою, почнемо з лічильника. Лічильник приймає на вхід сигнал тактової частоти, сигнал скидання та дозволу рахунку. Значення лічильника подається на вихідну шину o_counter, яка має розрядність 5 біт – це пов'язано з тим, що для перелічення всіх комірок пам'яті необхідно мінімум 5 розрядів ($2^5 = 32$), а комірок пам'яті 24. При подачі на вхід enable лічильника активується його рахунок. Перед початком рахунку потрібно встановити лічильник у початкове значення за допомогою сигналю i_rst_n тому, що спочатку він знаходиться у невизначеному стані. Також особливістю цього лічильника є те, що він рахує тільки до 24, хоча його розрядність дозволяє рахувати до 32 – це відбувається, бо прописана така умова: else if (o_counter == 24). Окремо стенд тестування лічильника я не робив, було прийнято рішення виконати тестування у звязці з блоком пам'яті. На рис. 8 можна побачити як працює лічильник.

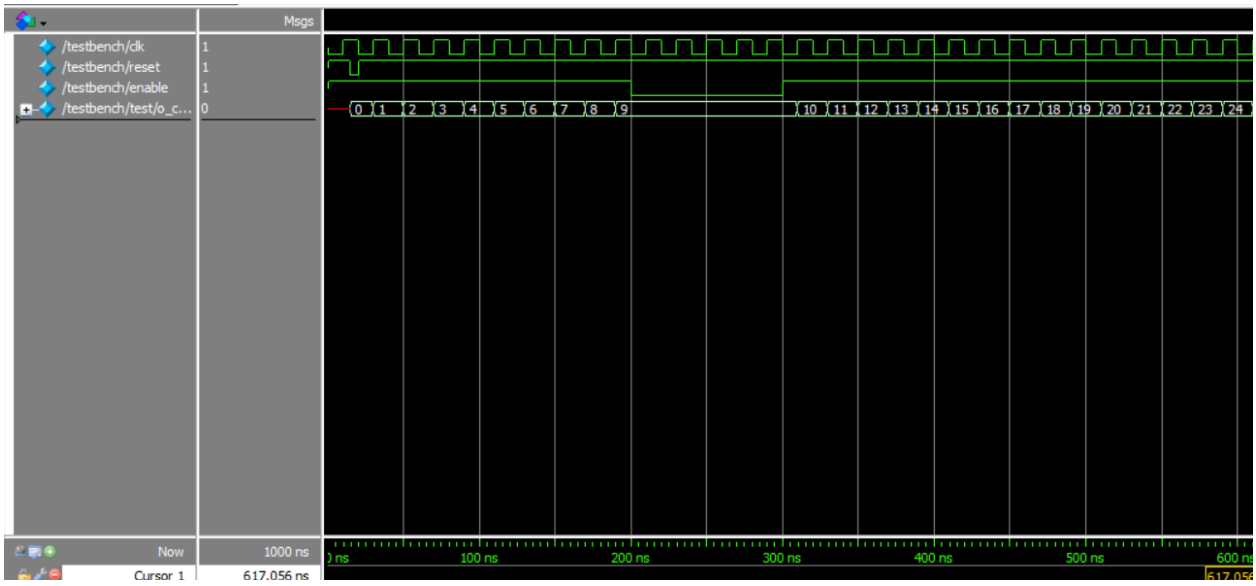

Рис. 8 Приклад тестування лічильника

З рис. 8 видно, що при імітації натискання кнопки сигналом i_rst_n лічильник встановлюється в 0. Також гарно видно, що при неактивному рівню сигналу дозволу рахунку лічильник зберігає попередній стан.

Блок пам’яті приймає на вхід тактовий сигнал, адресу комірки, яка подається з виходу лічильника, та сигнал дозволу запису. Для зчитування з блоку пам’яті використовується системна функція в блоці initial. (initial $readmemh ("rom_data.hex", rom);) Об’явлення блоку пам’яті виконується такою командою: reg [7:0] rom [23:0];

Дані, які зчитуються з пам’яті повинні бути записані у файл rom_data.hex, який був використаний в системній функції. Для тестування роботи пам’яті було розроблено тестовий стенд. Результати тестування можна побачити на рис. 9.

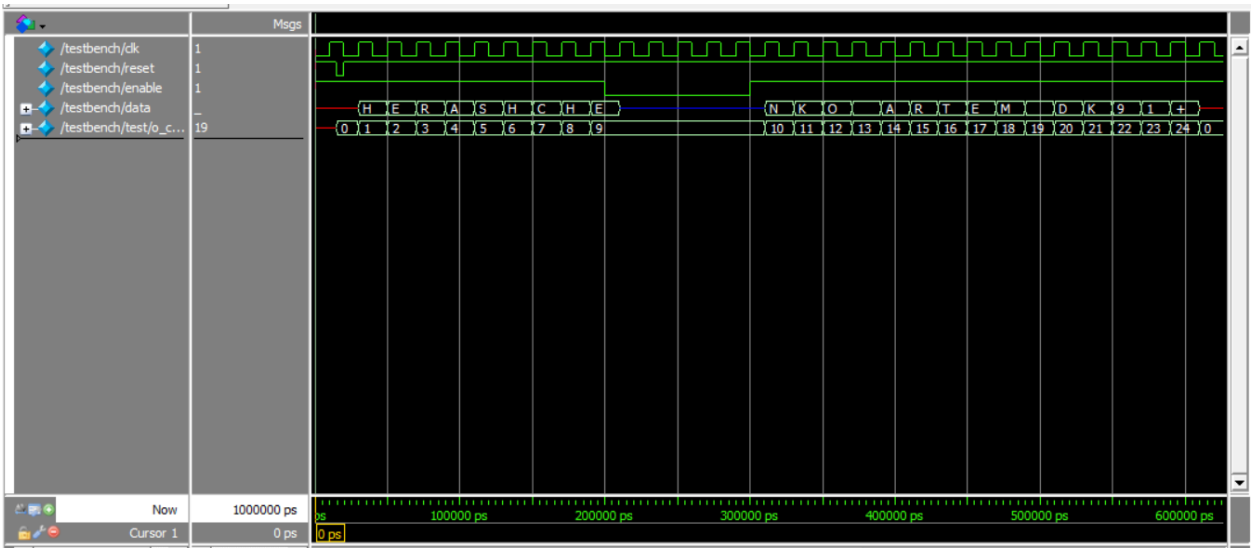

Рис. 9 Результати тестування блоку пам’яті

З рис. 9 видно, що при вимкненні сигналу дозволу зчитування на виході блоку пам’яті встановлюється високоімпедансний стан.

Наступним блоком було описано шифратор. Шифратор приймає на вхід сигнал тактової частоти, дані з виходу пам’яті та сигнал дозволу шифрування. На виході встановлюється вже зашифрований сигнал. Для шифрування було використано case(), для кожного стану i_data (код символа у десятковому форматі) було прописано умову кодування. Фрагмент коду з кодуванням:

```
case (i_data)
            72: o_data_r = 20'b00000000001010101000; //H
            69: o_data_r = 20'b00000000000000001000; //E
            82: o_data_r = 20'b00000000001011101000; //R
            65: o_data_r = 20'b00000000000010111000; //A
            83: o_data_r = 20'b00000000000010101000; //S
            67: o_data_r = 20'b00000011101011101000; //C
            78: o_data_r = 20'b00000000000011101000; //N
            75: o_data_r = 20'b00000000111010111000; //K
            79: o_data_r = 20'b00000011101110111000; //O
            95: o_data_r = 20'b00001110101010111000; //_
            84: o_data_r = 20'b00000000000000111000; //T
            77: o_data_r = 20'b00000000001110111000; //M
            68: o_data_r = 20'b00000000001110101000; //D
            57: o_data_r = 20'b11101110111011101000; //9
            49: o_data_r = 20'b10111011101110111000; //1
            43: o_data_r = 20'b00001011101011101000; //+
            default: o_data_r <= 20'b0;
        endcase
```

Для тестування шифратора я використав тестовий стенд, який поєднує у собі два попередні блоки та блок шифратору – таке рішення було прийнято через те, що так простіше визначити на якому етапі пристрій починає працювати не коректно. Результати тестування можна побачити на рис. 10.

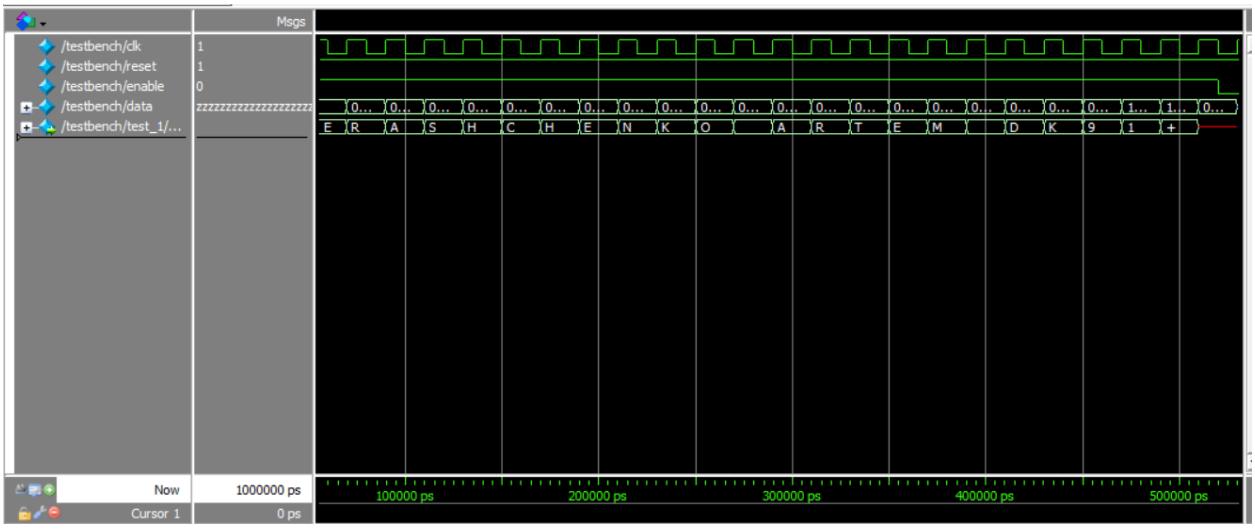

Рис. 10 Приклад роботи шифратору

З рис. 10 видно, що дані на виході шифратора зміщені на один такт вправо – це відбувається тому, що блоки передавача тактуються по передньому фронту тактового сигналу і кожний наступний блок отримує на

своєму виході дані тільки після приходу переднього фронту тактового сигналу.

Регістр отримує на свій вхід сигнал тактової частоти, сигнал скидання, сигнал дозволу запису та данні з виходу шифратора. Під час кожного активного фронту у регістр записується нове значення закодованого символу, якщо сигнал дозволу запису знаходиться у активному рівні. Для тестування роботи регістру я прописав окремий тестовий стенд. Результат тестування можна побачити на рис. 11.

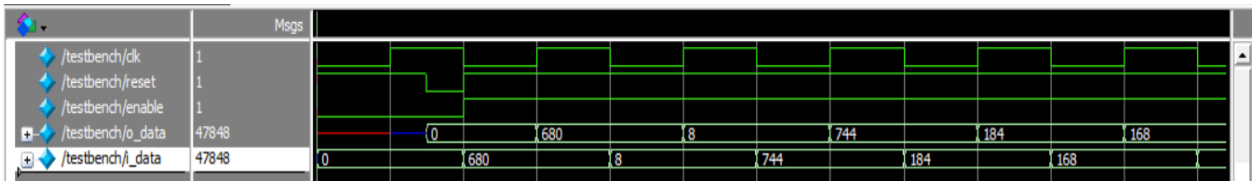

Рис. 11 Результат тестування регістру

З рис. 11 видно, що по активному фронту тактової частоти і за відсутності активного рівня enable на виході регістра встановлюється високоімпедансний стан. Після активації сигналу скидання на виході регістру встановлюється 0, а після подання активного рівня на сигнал дозволу запису в регістр починають записуватись дані. Для зручності відображення результатів закодований сигнал Морзе було представлено, як unsigned.

Після описання всіх окремих складових частин передавача, я написав блок transmitter, який їх об’єднує. Для тестування цього блоку було написано окремий тестовий стенд. Результат тестування можна побачити на рис. 12.

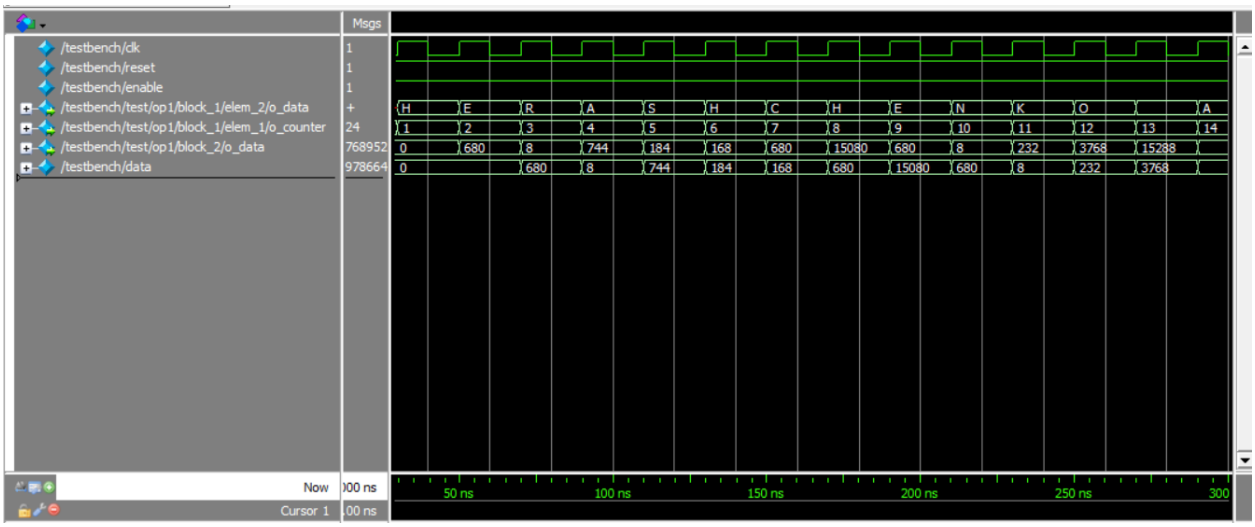

Рис. 12 Результат тестування Передавача

З рис. 12 видно, що логіка роботи передавача відповідає очікуванням, цей тест демонструє сигнали усіх складових частин: виходу лічильника, виходу пам'яті, виходу шифратора та виходу регістра. Також можна помітити, що вихідний сигнал є зміщеним на 2 такти, причиною такого зміщення є використання сигналу тактування – всі складові передавача працюють з використанням сигналу тактування і дані на кожен наступний приходять пізніше на 1 такт. Це стосується регістру та шифратора.

Після опису передавача я виконав тестування блоків приймача. Наводити роботу регістру приймача я не буду тому, що вона аналогічна регістру передавача. Почнемо з дешифратора, він має 2 входи: тактової частоти та вхід даних. Дешифратор працює також з використанням case(), але інверсно від шифратора. Фрагмент коду дешифратора:

```
        case (i_data)
            20'b00000000001010101000: o_data_r = 72; //H
            20'b00000000000000001000: o_data_r = 69; //E
            20'b00000000001011101000: o_data_r = 82; //R
            20'b00000000000010111000: o_data_r = 65; //A
            20'b00000000000010101000: o_data_r = 83; //S
            20'b00000011101011101000: o_data_r = 67; //C
            20'b00000000000011101000: o_data_r = 78; //N
            20'b00000000111010111000: o_data_r = 75; //K
            20'b00000011101110111000: o_data_r = 79; //O
            20'b00001110101010111000: o_data_r = 95; //_
            20'b00000000000000111000: o_data_r = 84; //T
```

```
            20'b00000000001110111000: o_data_r = 77; //M
            20'b00000000001110101000: o_data_r = 68; //D
            20'b11101110111011101000: o_data_r = 57; //9
            20'b10111011101110111000: o_data_r = 49; //1
            20'b00001011101011101000: begin
               o_data_r = 43;                        //+
               enable = 0;
            end
```

У фрагменті коду можна побачити сигнал дозволу, про нього я згадував раніше. Цей сигнал дозволу є локальним і використовується в середині дешифратора для перевірки символів повідомлення на рівність коду знаку “+”, при отриманні результату True припиняється декодування. Для тестування дешифратора було написано тестовий стенд, який поєднує у собі регістр приймача та використовує сигнал з виходу передавача для декодування. Результати тестування наведено на рис. 13.

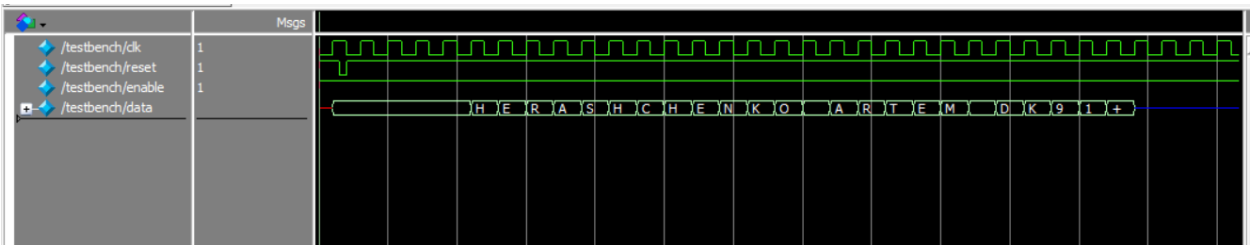

Рис. 13 Результати тестування дешифратора

З рис. 13 видно, що дані починають декодуватись з затримкою на декілька тактів, причина такої затримки була пояснена раніше. Також можна побачити, що при подачі знаку “+” декодування завершується.

Після декодування дані приходять на вхід пам’яті. Блок пам’яті приймача дещо відрізняється від блоку пам’яті передавача. Об’являється блок пам’яті аналогічно, як і в передавачі, але він містить у собі лічильник, який вказує адресу комірки для запису. Також блок пам’яті містить у собі локальний сигнал enable, який активує рахунок лічильника, коли на вхід даних приходить не нульове значення, після досягнення числа 24 лічильник припиняє рахунок. Запис у пам'ять відбувається за допомогою системної функції ($writememh

("rom_res.hex", rom_1);) в середині always блока. Дані записуються у файл з назвою rom_res.hex в шістнадцятковому форматі. Результат тестування блоку пам’яті приймача можна побачити на рис. 14. Для тестування блоку пам’яті було написано окремий тестовий стенд.

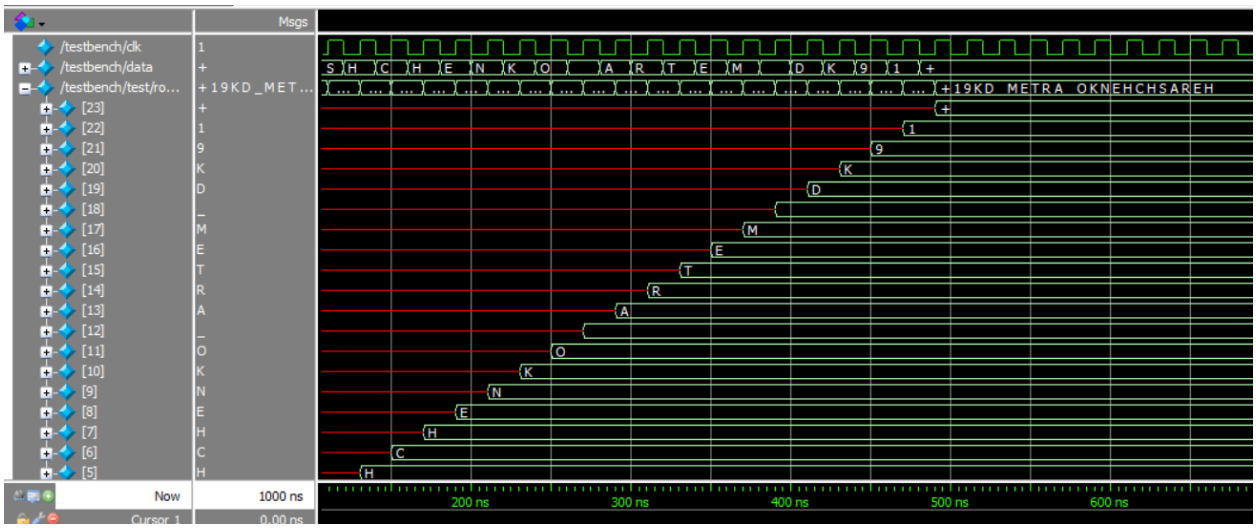

Рис. 14 Результат тестування блоку пам’яті приймача

На рис. 14 гарно видо, як відбувається запис у пам'ять. Повідомлення відображене реверсивно через те, що заповнення починається з молодших бітів, але у файлі з результатами повідомлення буде виглядати як завжди.

РОЗРОБКА ФАЙЛУ ВЕРХНЬОГО РІВНЯ ІЄРАРХІЇ ПРОЕКТУ ТА ФАЙЛУ ТЕСТУВАННЯ ПРИСТРОЮ

Для створення файлу верхнього рівня ієрархії я поєднав файл передавача та приймача у окремий блок. Цей файл містить всі складові передавача та приймача. Для тестування цього блоку я написав окремий тестовий стенд. Приклад роботи блоку зображено на рис. 15.

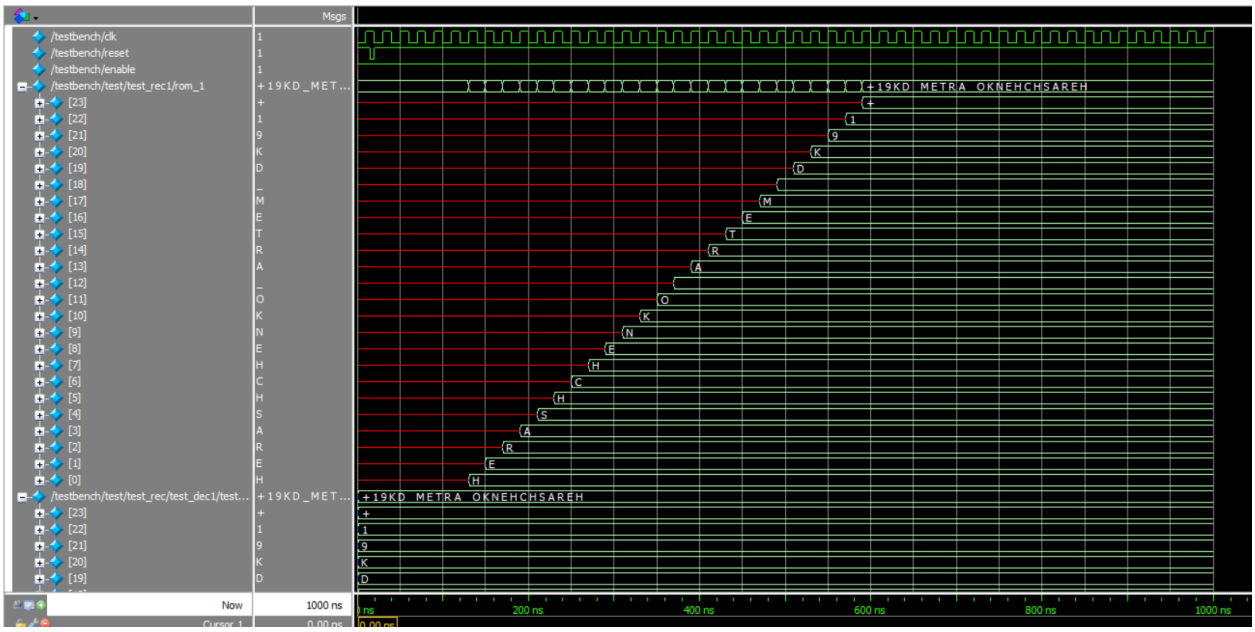

Рис. 15 Результат тестування блоку приймача - передавача

На рис. 15 я не виводив проміжні сигнали з виходів усіх складових елементів, але і так добре видно, що запис у пам'ять відбувся і також сталося зміщення вихідного результату на декілька тактів.

Результати записані у пам’ять:

rom_data.hex

```
48
45
52
41
53
48
43
48
45
4E
4B
4F
5F
41
52
54
45
4D
5F
44
4B
39
31
2B
```

Рис. 16.1 Дані записані у блок пам’яті передавача

rom_res.hex

```
// memory data file (do not edit the following line - required for mem load use)
// instance=/testbench/test/test_rec1/rom_1
// format=hex addressradix=h dataradix=h version=1.0 wordsperline=1 noaddress
48
45
52
41
53
48
43
48
45
4e
4b
4f
5f
41
52
54
45
4d
5f
44
4b
39
31
2b
```

Рис. 16.2 Дані записані у блок пам’яті приймача

Для зручності відтворення моделювання опишу файли та складові частини кожного елементу у вигляді таблиці. Переглянути файли, які наведені в таблиці можна за посиланням link.

Таблиця 2 Назви файлів складових частин пристрою

Пам'ять	Шифратор	Регістр	Передавач
counter.v	coder.v	register.v	transmitter.v
mem.v	coder_mem.v	register_tb.v	transmitter_tb.v
counter_mem.v	coder_mem_tb.v	sim3.do	sim4.do
counter_mem_tb.v	sim2.do		
sim1.do			
Регістр передавача	Дешифратор	Запис у пам'ять	Приймач - Передавач
reg_rec.v	reg_dec.v	mem_rec.v	rec.v
reg_rec_tb.v	reg_dec_tb.v	mem_rec_tb.v	rec_tb.v
sim_regrec.do	sim_regdec.do	mem_rec.do	sim_rec.do

ВИСНОВОК

Підсумовуючи виконану роботу, можна стверджувати, що завдання курсової було виконано. Результати тестування складових частин та всього пристрою в цілому відповідають очікуванням. Умова початку роботи пристрою по зовнішньому сигналу enable виконується, також виконується умова завершення запису при подачі символу кінця передачі “+”, форма сигналу у коді Морзе відповідає ілюстраціям наведеним на рис.1.

Виконавши курсову роботу, я удосконалив свої навички опису пристроїв на мові Verilog, використав на практиці системні функції для запису та зчитування з пам’яті та розібрався з логікою їх роботи, детальніше розглянув логіку написання тестових стендів та файлів для автоматичного запуску симуляції «.do».

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Verilog [Електронний ресурс] – Режим доступу до ресурсу: https://classroom.google.com/u/1/c/ODYyODc3NjAwOTVa/m/MTU4NzA1MzM2Njg2/details .

ДОДАТКИ

1. coder.v
2. coder_mem.v
3. coder_mem_tb.v
4. counter.v
5. counter_mem.v
6. counter_mem_tb.v
7. decoder.v
8. mem.v
9. mem_rec.do
10. mem_rec.v
11. mem_rec_tb.v
12. rec.v
13. rec_tb.v
14. reg_dec.v
15. reg_dec_tb.v
16. reg_rec.v
17. reg_rec_tb.v
18. register.v
19. register_tb.v
20. rom_data.hex
21. rom_res.hex
22. sim1.do
23. sim2.do
24. sim3.do
25. sim4.do
26. sim_rec.do
27. sim_regdec.do
28. sim_regrec.do

29. transmitter.v
30. transmitter_tb.v