ЦЕЛЬ РАБОТЫ

Углубление теоретических знаний в области помехоустойчивого кодирования данных в информационных системах, исследование способов построения циклических кодов и реализации кодирующих и декодирующих устройств, приобретение практических навыков моделирования и исследования устройств кодирования информационных сообщений циклическими кодами.

ПРОГРАММА ЛАБОРАТОРНОЙ РАБОТЫ

1. Изучить по рекомендуемой литературе теоретический материал по теме циклического кодирования. Выполняется в процессе домашней подготовки.

2. Рассчитать по выражениям значения проверочных бит для всех возможных 16-ти значений информационных комбинаций.

3. Сравнить полученные экспериментально проверочные комбинации с комбинациями, вычисленными теоретически.

4. Составить в рабочем поле эмулятора схему декодера Хемминга.

5. Подавать последовательно с помощью ключей SW-1…SW-8 все закодированные комбинации, полученные в п.4. и проследить отображаемую информацию на светодиодных индикаторах.

6. Сформулировать выводы по работе и оформить отчет.

ХОД РАБОТЫ

1. Создадим схему в среде Proteus рисунке 1.

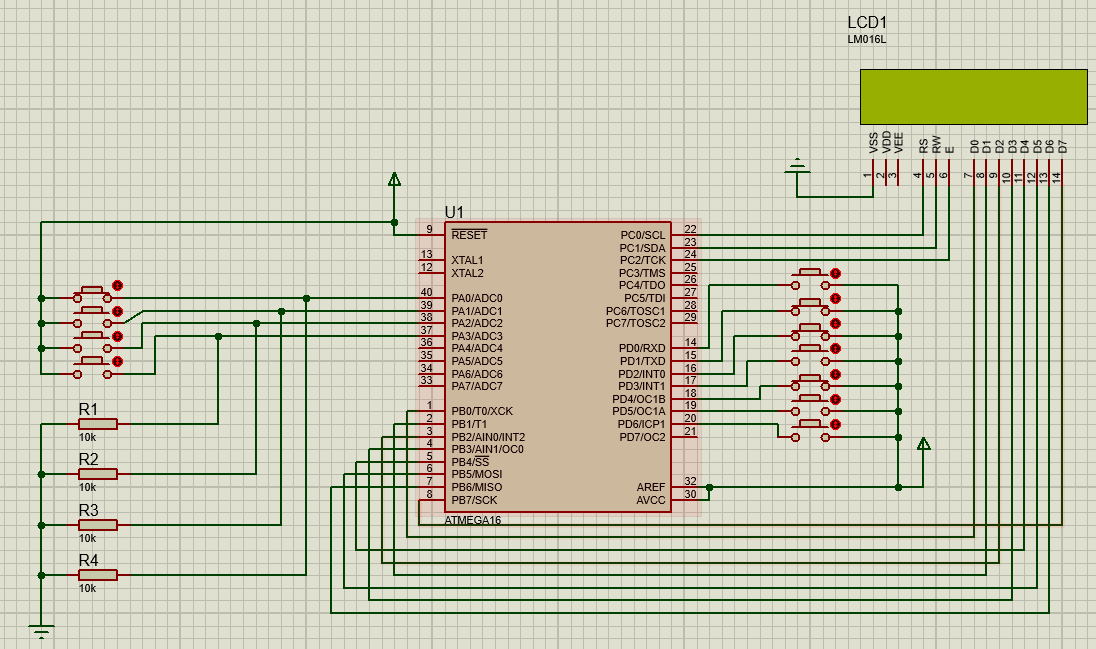


Рисунок 1 — Построение схемы в среде Proteus

2. Напишем код, который будет кодировать и декодировать введенный код.

#define F\_CPU 10000000UL /\* Define CPU Frequency e.g. here 8MHz \*/

#include <avr/io.h> /\* Include AVR std. library file \*/

#include <math.h>

#include <util/delay.h> /\* Include inbuilt defined Delay header file \*/

#define LCD\_Data\_Dir DDRB /\* Define LCD data port direction \*/

#define LCD\_Command\_Dir DDRC /\* Define LCD command port direction register \*/

#define LCD\_Data\_Port PORTB /\* Define LCD data port \*/

#define LCD\_Command\_Port PORTC /\* Define LCD data port \*/

#define RS PC0 /\* Define Register Select (data/command reg.)pin \*/

#define RW PC1 /\* Define Read/Write signal pin \*/

#define EN PC2 /\* Define Enable signal pin \*/

// Store input bits

int input[4];

// Store cycle code

int code[8];

char Data[4];

char Result[5];

int i;

void LCD\_Command(unsigned char cmnd)

{

LCD\_Data\_Port= cmnd;

LCD\_Command\_Port &= ~(1<<RS); /\* RS=0 command reg. \*/

LCD\_Command\_Port &= ~(1<<RW); /\* RW=0 Write operation \*/

LCD\_Command\_Port |= (1<<EN); /\* Enable pulse \*/

\_delay\_us(1);

LCD\_Command\_Port &= ~(1<<EN);

\_delay\_ms(3);

}

void LCD\_Char (unsigned char char\_data) /\* LCD data write function \*/

{

LCD\_Data\_Port= char\_data;

LCD\_Command\_Port |= (1<<RS); /\* RS=1 Data reg. \*/

LCD\_Command\_Port &= ~(1<<RW); /\* RW=0 write operation \*/

LCD\_Command\_Port |= (1<<EN); /\* Enable Pulse \*/

\_delay\_us(1);

LCD\_Command\_Port &= ~(1<<EN);

\_delay\_ms(1);

}

void LCD\_Init (void) /\* LCD Initialize function \*/

{

LCD\_Command\_Dir = 0xFF; /\* Make LCD command port direction as o/p \*/

LCD\_Data\_Dir = 0xFF; /\* Make LCD data port direction as o/p \*/

\_delay\_ms(20); /\* LCD Power ON delay always >15ms \*/

LCD\_Command (0x38); /\* Initialization of 16X2 LCD in 8bit mode \*/

LCD\_Command (0x0C); /\* Display ON Cursor OFF \*/

LCD\_Command (0x06); /\* Auto Increment cursor \*/

LCD\_Command (0x01); /\* Clear display \*/

LCD\_Command (0x80); /\* Cursor at home position \*/

}

void LCD\_String (char \*str) /\* Send string to LCD function \*/

{

int i;

for(i=0;str[i]!=0;i++) /\* Send each char of string till the NULL \*/

{

LCD\_Char (str[i]);

}

}

void LCD\_String\_xy (char row, char pos, char \*str)/\* Send string to LCD with xy position \*/

{

if (row == 0 && pos<16)

LCD\_Command((pos & 0x0F)|0x80); /\* Command of first row and required position<16 \*/

else if (row == 1 && pos<16)

LCD\_Command((pos & 0x0F)|0xC0); /\* Command of first row and required position<16 \*/

LCD\_String(str); /\* Call LCD string function \*/

}

void LCD\_Clear()

{

LCD\_Command (0x01); /\* clear display \*/

LCD\_Command (0x80); /\* cursor at home position \*/

}

void MakeCRC(char \*BitString)

{

char output[16];

static char Res[5]; // CRC Result

char CRC[4];

int i;

char DoInvert;

for (i=0; i<4; ++i) CRC[i] = 0; // Init before calculation

for (i=0; i<4; ++i)

{

DoInvert = ('1'==BitString[i]) ^ CRC[3]; // XOR required?

CRC[3] = CRC[2];

CRC[2] = CRC[1] ^ DoInvert;

CRC[1] = CRC[0] ^ DoInvert;

CRC[0] = DoInvert;

}

for (i=0; i<4; ++i) Res[3-i] = CRC[i] ? '1' : '0'; // Convert binary to ASCII

Res[4] = 0; // Set string terminator

sprintf(output, " [%s]", Res);

LCD\_String(output);

}

int main()

{

int k = 0;

char firstLine[16];

DDRA = 0x00;

PORTA = 0x00;

DDRD = 0x00;

PORTD = 0x00;

LCD\_Init();

while(1) {

/\* ENCODE \*/

for(k = 0; k<4; k++ ){

if ((PINA & (1<<k))){

input[k] = 1;

Data[k] = '1';

}

else {

input[k] = 0;

Data[k] = '0';

}

}

sprintf(firstLine, "CRC of [%s] is", Data);

LCD\_String(firstLine);

LCD\_Command(0xC0);

MakeCRC(Data);

\_delay\_ms(500);

LCD\_Clear();

}

return 0;

}

3. Вычислим значения кода Хэмминга для всех 16 комбинаций.

|  |  |
| --- | --- |
| Кодируемое сообщение | Выходное сообщение |
| 0000 | 00000000 |
| 0001 | 00010111 |
| 0010 | 00101110 |
| 0011 | 00111001 |
| 0100 | 01001011 |
| 0101 | 01011100 |
| 0110 | 01100101 |
| 0111 | 01110010 |
| 1000 | 10000001 |
| 1001 | 10010110 |
| 1010 | 10101111 |
| 1011 | 10111000 |
| 1100 | 11001010 |
| 1101 | 11011101 |
| 1110 | 11100100 |
| 1111 | 11110011 |

4. Изменим один бит в передаваемом сообщении и посмотрим, как система реагирует на ошибку.

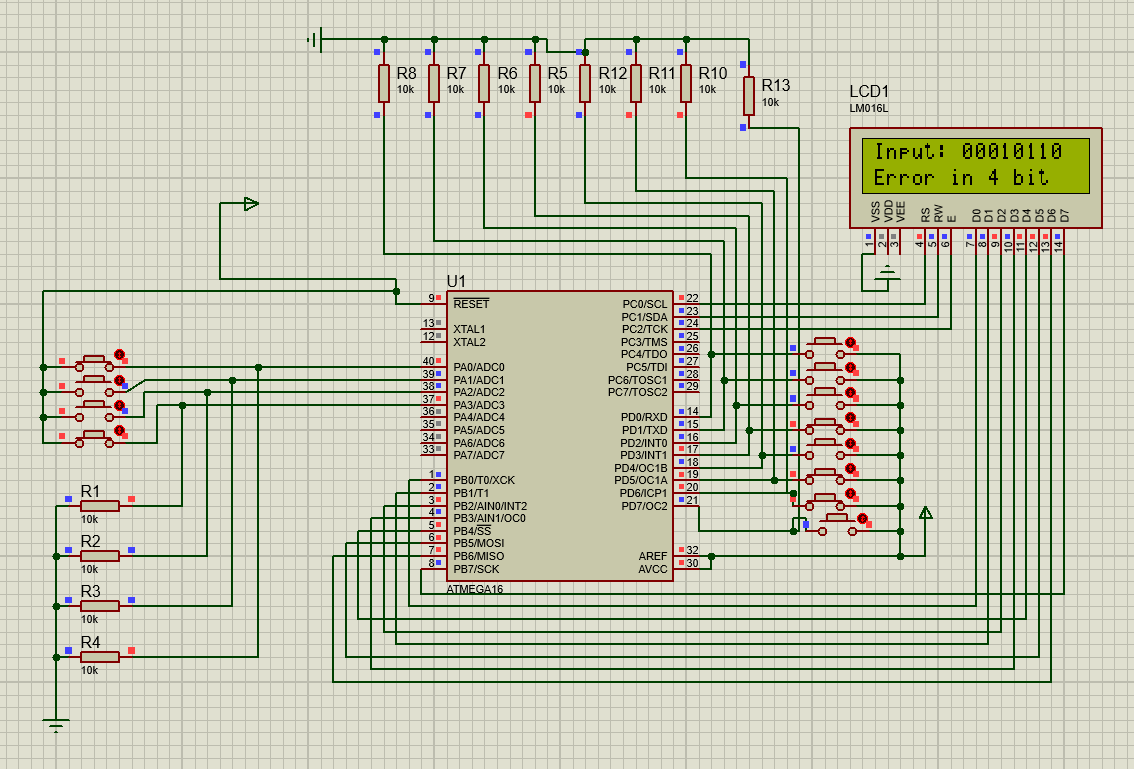


Рисунок 2 — Передача сообщения с ошибкой

ВЫВОДЫ

В ходе лабораторной работы были исследованы методы помехоустойчивого кодирования с помощью кода Хемминга, была построена схема, которая может кодировать и раскодировать сообщения.