**Федеральное Агентство Связи**

**Ордена Трудового Красного Знамени**

Федеральное Государственное бюджетное образовательное

учреждение высшего образования

**МОСКОВСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**СВЯЗИ И ИНФОРМАТИКИ**

Кафедра «Мультимедийные Сети и Услуги Связи»

Курсовая работа

на тему:

«Создание кодера, декодера и генератора случайных векторов ошибок для циклического кода (7,4)»

Выполнили студенты группы БСУ1401

Ежова Е. А.

Каледина А. В.

Коротыгин А. А.

Краснов К. А.

Петухов Д. А.

Проверил: к.т.н. доцент

Гузеев А.В.

Москва, 2017

**Цель:** реализовать кодер и декодер (7,4) и программу для наложения случайного вектора ошибки на кодовые комбинации.

1. **Блок-схемы алгоритмов:**



Рис. **1.** – Блок-схема модуля «TDMA\_data\_link»

Данный модуль управляет очередностью работы кодера, декодера и генератора шума.



Рис. **2** – Блок-схема модуля «cc\_decoder»



Рис. **3** – Блок-схема модуля «cc\_encoder»



Рис. **4** – Блок-схема модуля «noise\_generator»

В данной реализации завершение работы модуля при успешном выделении общей памяти возможна только прерыванием.



Рис. **5** – Структура блока общей памяти.

TURN – поле активного флага (W, E, N или D)

Описание состояний:

* W – ожидание
* E – работа кодера
* N – работа генератора шума
* D – работа декодера

Data – поле данных (16 кодовых комбинаций по 4 байта)

1. **Метод формирования разрешенных комбинаций, синдромов и векторов ошибок:**

Для формирования разрешенных комбинаций циклического кода (7,4) нам необходимы все информационные комбинации и образующий полином вида 1101 в двоичном виде или x3 + x2 + 1 в форме многочлена.

Условие того, что комбинация является разрешенной

Процесс получения разрешенной комбинации заключается в следующем:

1. Выписываем все информационные комбинации исключая нулевую.
2. Дописываем к каждой информационной комбинации три нуля, увеличивая комбинацию до 7 разрядов.
3. Делим полученную комбинацию на образующий полином и получаем синдром
4. Складываем по модулю 2, полученную в шаге 2 комбинацию и синдром получая разрешенную комбинацию.

Таблица 1. Формирование разрешенных комбинаций.

|  |  |  |
| --- | --- | --- |
| Информационная комбинация | Полученный синдром в результате деления на образующий полином | Разрешенная комбинация |
| 0000 | 000 | **0000000** |
| 0001 | 101 | **0001101** |
| 0010 | 111 | **0010111** |
| 0011 | 010 | **0011010** |
| 0100 | 011 | **0100011** |
| 0101 | 110 | **0101110** |
| 0110 | 100 | **0110100** |
| 0111 | 001 | **0111001** |
| 1000 | 110 | **1000110** |
| 1001 | 011 | **1001011** |
| 1010 | 001 | **1010001** |
| 1011 | 100 | **1011100** |
| 1100 | 101 | **1100101** |
| 1101 | 000 | **1101000** |
| 1110 | 010 | **1110010** |
| 1111 | 111 | **1111111** |

Из теории нам известно, что кодовое расстояние есть минимальное расстояние Хемминга т.е. минимальное количество разрядов, в которых отличаются разрешенные комбинации кода (7,4).

Самым простым способом определения кодового расстояния при условии известности всех кодовых комбинаций, является нахождение минимальной ненулевой разрешенной комбинации, ее вес и будет кодовым расстоянием.

Для нашего кода (7,4) минимальным весом кодовой комбинации будет 3, следовательно, кодовое расстояние d=3.

Зная кодовое расстояние, мы можем определить кратность гарантированно исправляемых кодом ошибок. Данную величину вычисляем из неравенства d≤2t-1 и получаем, для нашего кода кратность t=1, это значит, что код гарантированно исправляет все единичные ошибки и обнаруживает двоичные.

Условие того, что комбинация является разрешенной заключается в делении без остатка этой комбинации на образующий полином, следовательно, если принятая комбинация отличается от всех разрешенных комбинаций, то можно говорить о том, что произошла ошибка.

Исправление ошибки в принятой комбинации происходит следующим образом:

1. Принятую комбинацию делим по модулю 2 на образующий полином, получая синдром (остаток от деления);
2. Полученный синдром сравниваем с известными синдромами и находим соответствующий синдрому вектор ошибок.
3. Складываем по модулю 2 принятую комбинацию с вектором ошибки, получая изначально передаваемую комбинацию.

Для формирования вектора таблицы соответствия векторов ошибок и синдромов необходимо: все вектора ошибок разделить по модулю 2 на образующий полином и остаток от деления и будет синдромом для соответствующего вектора ошибки.

Таблица 2. Соответствие синдромов и векторов ошибки.

|  |  |
| --- | --- |
| Вектор ошибки | Синдром |
| 1000000 | 110 |
| 0100000 | 011 |
| 0010000 | 111 |
| 0001000 | 101 |
| 0000100 | 100 |
| 0000010 | 010 |
| 0000001 | 001 |

1. **Исходный код программы:**

* **Include:**

**library.cpp**

#include "library.h"

void cpsleep(int sleep\_ms) {

#ifdef  defined \_\_APPLE\_\_ || defined \_\_linux\_\_

    usleep(sleep\_ms \* 1000);

#elif   defined \_WIN32 || defined \_WIN64

    Sleep(sleep\_ms);

#endif

}

void print\_press\_enter() {

    printf("Press ENTER to return to main menu\n");

    flush;

    char tmp = 0;

    while (tmp != '\r' && tmp != '\n') {

        tmp = getchar();

    }

    flush;

}

**library.h**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | #ifndef library\_h  #define library\_h    #if     defined \_\_APPLE\_\_ || defined \_\_linux\_\_      #include <unistd.h>      #include <stdio.h>  #elif   defined \_WIN32 || defined \_WIN64      #include <Windows.h>      #include <cstdio>  #endif    #define flush fseek(stdin, 0, SEEK\_END)    void cpsleep(int sleep\_ms);  void print\_press\_enter();    #endif |

**shared.cpp**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | // Shared memory realization  #include "shared.h"    void shmem\_alloc\_and\_clean () {      sema.acquire();      if (owed\_shmem.attach()) {          owed\_shmem.detach();      }      if (!shmem.attach()) {          shmem.create(DATA\_OFFSET + DATA\_SIZE);          std::cout << "Shared memory created" << std::endl;      }      sema.release();  } |

**shared.h**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | // Shared memory realization  #ifndef shared\_h  #define shared\_h    #include <iostream>  #include <QtCore/qsharedmemory.h>  #include <QtCore/qsystemsemaphore.h>    static const int n =            7;  static const int k =            4;  static const int f\_polynomial = 0b1011;    static QSystemSemaphore         sema("/sema", 1);  static QSharedMemory            owed\_shmem("/shmem");  static QSharedMemory            shmem("/shmem");    void shmem\_alloc\_and\_clean();    enum turns {E, N, D, W};  static const turns E\_TURN =     E;  static const turns N\_TURN =     N;  static const turns D\_TURN =     D;  static const turns WAIT =       W;    #define TURN\_OFFSET             0  #define DATA\_OFFSET             4  #define DATA\_SIZE               64  #define DATA\_SIZE\_INT           16    #define TURN\_PTR                ((turns \* const) shmem.data() + TURN\_OFFSET)  #define DATA\_PTR                ((int \* const) shmem.data() + DATA\_OFFSET)    #define TURN                    \*TURN\_PTR    #define PASS\_TURN\_E             memcpy(TURN\_PTR, &E\_TURN, sizeof(E\_TURN))  #define PASS\_TURN\_N             memcpy(TURN\_PTR, &N\_TURN, sizeof(N\_TURN))  #define PASS\_TURN\_D             memcpy(TURN\_PTR, &D\_TURN, sizeof(D\_TURN))  #define WAIT\_TURN               memcpy(TURN\_PTR, &WAIT, sizeof(WAIT)) |

* **Кодер:**

**input.cpp**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | // Input parser. Takes source sequence, source data (4-bit codes) and forming polynomial as an input.  #include "input.h"    input :: input() :      default\_s\_sequence({2, 4, 6, 8, 10, 12, 14, 16, 1, 3, 5, 7, 9, 11, 13, 15}),      default\_s\_data({0b0000, 0b0001, 0b0010, 0b0011, 0b0100, 0b0101, 0b0110, 0b0111, 0b1000, 0b1001, 0b1010, 0b1011, 0b1100, 0b1101, 0b1110, 0b1111}),      default\_s\_data\_size(default\_s\_data.size())  {      s\_sequence = default\_s\_sequence.data();      s\_data = default\_s\_data.data();      s\_data\_size = &default\_s\_data\_size;  } |

**input.h**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | // Input parser. Gets source sequence, source data (4-bit codes) and forming polynomial as an input.  #ifndef input\_h  #define input\_h    #include <stdio.h>  #include <fstream>  #include <vector>  using namespace std;      class input {  public:      input();      const int \* s\_sequence;      const int \* s\_data;      const int \* s\_data\_size;    private:      vector<int> default\_s\_sequence;      vector<int> default\_s\_data;      const int default\_s\_data\_size;  };    #endif |

**encoder.h**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | // (7, 4) Hamming code encoder  #ifndef encoder\_h  #define encoder\_h    #include <stdio.h>  #include <iostream>  #include <bitset>  #include <math.h>  #include "../../include/shared.h"    using namespace std;    class encoder {  public:      encoder();      int aj\_data(const int \* \_s\_data);      int get\_syndrome(const int \* \_s\_data);    };    #endif |

**encoder.cpp**

|  |  |  |  |
| --- | --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | // (7, 4) Hamming code encoder  #include "encoder.h"    encoder :: encoder()  {    }    int encoder :: aj\_data(const int \* \_s\_data) {      const int \* s\_data = \_s\_data;        int syndrome = get\_syndrome(\_s\_data);      int aj\_data = \*s\_data << (n - k);        aj\_data += syndrome;        return aj\_data;  }    int encoder :: get\_syndrome(const int \* \_s\_data) {      const int \* s\_data = \_s\_data;      int divider = \*s\_data << (n - k);      int divident = 0;      while (divider >= pow(2, n - k)) {          divident = f\_polynomial << (n - k);          for (int i = n; i > k; i--) {              if (bitset<n>(divider).test(i - 1) == false) {                  divident >>= 1;              } else break;          }          divider ^= divident;      }      return divider;  }  **main.cpp**   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71 | #include <iostream>  #include "../../include/shared.h"  #include "../../include/library.h"  #include "input.h"  #include "encoder.h"    #define UNIVERSITY "MTUCI"  #define STGROUP "BSU1401"  #define DEVELOPERS "Denis Petuhov, Kaledina Anastasija, Korotygin Aleksandr, Ezhova Elena, Krasnov Kirill"    void main(int argc, const char \* argv[]) {      std::cout << "University: " << UNIVERSITY << std::endl << "Student group: " << STGROUP << std::endl << "Developers: " << DEVELOPERS;      std::cout << std::endl << std::endl;        shmem\_alloc\_and\_clean();      shmem.attach();        if (shmem.isAttached()) {          std::cout << "Shared memory size: " << shmem.size() << " " << "Bytes";          std::cout << std::endl << std::endl;            input Input;          encoder Encoder;            while (true) {              switch (TURN) {                  case E:                      std::cout << "Source data codes: " << std::endl;                        for (int i = 0; i < \*Input.s\_data\_size; i++) {                          int \* curr\_dat = new int(Input.s\_data[Input.s\_sequence[i] - 1]);                          std::cout << bitset<k>(\*curr\_dat) << " ";                          delete curr\_dat;                      }                        std::cout << std::endl << std::endl;                        std::cout << "Encoding to (7, 4) anti-jamming cyclic codes..." << std::endl;                        std::cout << "Forming polynomial: " << std::endl;                        std::cout << bitset<k>(f\_polynomial) << std::endl;                        std::cout << "Source anti-jamming codes:" << std::endl;                        for (int i = 0; i < DATA\_SIZE\_INT; i++) {                          int \* curr\_dat = new int(Input.s\_data[Input.s\_sequence[i] - 1]);                          int \* aj\_data = new int(Encoder.aj\_data(curr\_dat));                          memcpy(DATA\_PTR + i, aj\_data, sizeof(\*aj\_data));                          delete aj\_data;                          delete curr\_dat;                      }                        for (int i = 0; i < DATA\_SIZE\_INT; i++) std::cout << bitset<n>(\*(DATA\_PTR + i)) << " ";                      std::cout << std::endl << std::endl;                        std::cout << "Sending...";                      std::cout << std::endl << std::endl;                        WAIT\_TURN;                        break;                    default:                      break;                }              cpsleep(500);          }      } else std::cout << "Shared memory is not attached" << std::endl << std::endl;  } | |

* **Декодер:**

**decoder.h**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | // (7, 4) Hamming code decoder  #ifndef decoder\_h  #define decoder\_h    #include <stdio.h>  #include <iostream>  #include <bitset>  #include <math.h>  #include <vector>  #include "../../include/shared.h"  #include <algorithm>    using namespace std;    class decoder {  public:      decoder();      int d\_data(const int \* \_aj\_data);      int get\_syndrome(const int \* \_aj\_data);    private:      vector<int> error\_vectors;      vector<int> error\_vectors\_syndromes;    };    #endif |

**decoder.cpp**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | // (7, 4) Hamming code decoder  #include "decoder.h"    decoder :: decoder () :  error\_vectors             ({0b0000001, 0b0000010, 0b0000100, 0b0001000, 0b0010000, 0b0100000, 0b1000000}),  error\_vectors\_syndromes   ({0b001, 0b010, 0b100, 0b011, 0b110, 0b111, 0b101})  {    }    int decoder :: get\_syndrome(const int \* \_s\_data) {      const int \* s\_data = \_s\_data;      int divider = \*s\_data;      int divident = 0;      while (divider >= pow(2, n - k)) {          divident = f\_polynomial << (n - k);          for (int i = n; i > k; i--) {              if (bitset<n>(divider).test(i - 1) == false) {                  divident >>= 1;              } else break;          }          divider ^= divident;      }      return divider;  }    int decoder :: d\_data(const int \* \_aj\_data) {      const int \* aj\_data = \_aj\_data;        int syndrome = get\_syndrome(aj\_data);      int error\_vector = 0;      int d\_data = \*aj\_data >> (n - k);        if (syndrome != 0) {          auto it = find(error\_vectors\_syndromes.begin(), error\_vectors\_syndromes.end(), syndrome);          error\_vector = error\_vectors.at(it - error\_vectors\_syndromes.begin());          d\_data = (\*aj\_data ^ error\_vector) >> (n - k);      }        return d\_data;  } |

**main.cpp**

#include <iostream>

#include <math.h>

#include "../../include/shared.h"

#include "../../include/library.h"

#include "decoder.h"

using namespace std;

#define UNIVERSITY "MTUCI"

#define STGROUP "BSU1401"

#define DEVELOPERS "Denis Petuhov, Kaledina Anastasija, Korotygin Aleksandr, Ezhova Elena, Krasnov Kirill"

void main(int argc, const char \* argv[]) {

    cout

    << "University: " <<        UNIVERSITY <<       endl

    << "Student group: " <<     STGROUP <<          endl

    << "Developers: " <<        DEVELOPERS <<       endl

    << endl;

    shmem\_alloc\_and\_clean();

    shmem.attach();

    if (shmem.isAttached()) {

        cout << "Shared memory size: " << shmem.size() << " " << "Bytes";

        cout << endl << endl;

        decoder Decoder;

        while (true) {

            switch (TURN) {

                case D:

                    cout << "Received anti-jamming codes: " << endl;

                    for (int i = 0; i < DATA\_SIZE\_INT; i++) cout << bitset<n>(\*(DATA\_PTR + i)) << " ";

                    cout << endl << endl;

                    cout << "Decoding (7, 3) anti-jamming codes..." << endl;

                    cout << "Received data codes:" << endl;

                    for (int i = 0; i < DATA\_SIZE\_INT; i++) cout << bitset<k>(Decoder.d\_data(DATA\_PTR + i)) << " ";

                    cout << endl << endl;

                    WAIT\_TURN;

                    break;

                default:

                    break;

            }

            cpsleep(500);

        }

        shmem.detach();

    }

    else cout << "Shared memory is not attached" << endl << endl;

}

* **memory\_cleaner:**

**main.cpp**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | // Shared memory cleaner  #include <iostream>  #include "../../include/shared.h"  #include "../../include/library.h"      int main(int argc, const char \* argv[]) {      QSystemSemaphore sema("/sema", 1);        sema.acquire();      if (shmem.attach()) {          std::cout << "Cleaning if unattached shared memory is present..." << std::endl;          shmem.detach();          std::cout << "OK" << std::endl << std::endl;      }      else {          std::cout << "Nothing to clean" << std::endl << std::endl;      }      sema.release();        print\_press\_enter();        return 0;  } |

* **noise\_generator**

**main.cpp**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | // Noise generator    #include <iostream>  #include <math.h>  #include "../../include/library.h"  #include "../../include/shared.h"  using namespace std;    void main(int argc, const char \* argv[]) {      shmem\_alloc\_and\_clean();      shmem.attach();        if (shmem.isAttached()) {          while (true) {              switch (TURN) {                  case N:                      cout << "Generating noise..." << endl << endl;                      for (int i = 0; i < DATA\_SIZE\_INT; i++) {                          const int noise = 0b0000001 << (rand() % 7 + 0);                          int noise\_plus\_data = noise ^ \*(DATA\_PTR + i);                          memcpy(DATA\_PTR + i, &noise\_plus\_data, sizeof(noise\_plus\_data));                      }                        WAIT\_TURN;                        break;                    default:                      break;                }              cpsleep(500);          }      }  } |

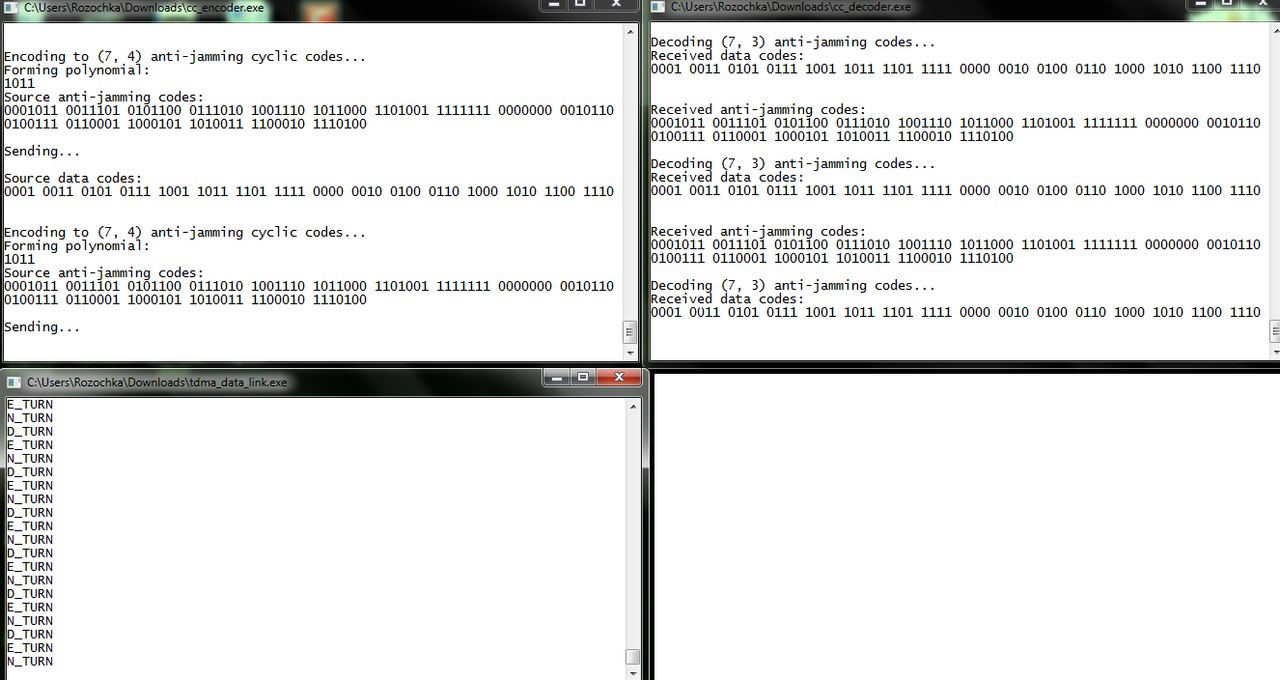
* **tdma\_data\_link**

**main.cpp**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | // TDMA realization    #include <iostream>  #include "../../include/shared.h"  #include "../../include/library.h"    void main(int argc, const char \* argv[]) {      shmem\_alloc\_and\_clean();      shmem.attach();        if (shmem.isAttached()) {          while (true) {              PASS\_TURN\_E;              std::cout << "E\_TURN" << std::endl;              cpsleep(17);              PASS\_TURN\_N;              std::cout << "N\_TURN" << std::endl;              cpsleep(17);              PASS\_TURN\_D;              std::cout << "D\_TURN" << std::endl;              cpsleep(17);          }      }  } |

1. **Скриншоты программы:**

**Результат работы программы: кодер-декодер-**



**Результат работы программы: кодер-декодер-шум**

