Представление и отображение информации при работе с микроконтроллерами STM32

А.Е. Близнюк, В.Г. Трубин, Г.В. Саблина

ФГБОУ ВПО НГТУ, г. Новосибирск, Россия

Аннотация: Статья посвящена способам представления и отображения информации при работе с микроконтроллерами семейства *STM32*. Описаны двоичная, шестнадцатеричная системы счисления, переменные разной длины, приведены алгоритмы перевода из одной системы счисления в другую. Рассмотрен вопрос представления отрицательных целых чисел в двоичном виде, объяснена основная причина данного способа представления. Показано, как данные хранятся в памяти микроконтроллера *STM32*. Описаны две программы, в которых происходит передача символов, используя кодировку *ASCII*, с микроконтроллера на персональный компьютер с последующим выводом их на экран. Изложенный материал может представлять интерес не только инженерам и студентам, но и читателям, которые не обладают знаниями в этой сфере.

Ключевые слова: Система счисления, бит, ниббл, тетрада, байт, микроконтроллер, STM32, терминальное приложение, отрицательные числа, дополнительный код, кодировка, ASCII, КОИ-8, CP866, Windows-1251, ASCII-графика, UTF-8, UTF-16, UTF-32, Юникод, CoolTerm, C, таблицы кодировок, IEC 60027-2, ГОСТ 8.417-200.

Способы представления данных и системы счисления

Люди привыкли к десятичной системе счисления, но в цифровых устройствах она не нашла широкого применения. Например, компьютер работает с числами так же, как и человек, но уже в двоичной системе счисления.

В процессорах цифровых устройств находится большое количество транзисторов, которые являются своего рода переключателями с двумя логическими состояниями – «1» или «0». Цифровое устройство использует для управления, хранения данных только нули и единицы.

Английское слово *«bit»* (бит) – сокращение от *«binary digit»*. Здесь *«digit»* – это цифра, а *«binary»* («двоичный») - значит «состоящий из двух частей». Традиционно эти «части» обозначают как «0» и «1», но любые две чёткие альтернативы (горячий/холодный, черный/белый) можно считать вариантами значения бита. Один бит – это один разряд и может быть равен «0» или «1».

Например, с помощью лампочки можно передавать биты информации. Если она включена — это «1», если выключена — «0» (рис. 1).

В микросхемах электрические сигналы «логический ноль» и «логическая единица» имеют характерные уровни напряжения. Напряжение, близкое к напряжению питания, принимается за «1», а близкое к нулевому напряжению за «0».

Логика *TTL* (транзисторно-транзисторная логика) определяет стандарт для уровня напряжения логических сигналов (рис. 2). Диапазон напряжения логического нуля или логической единицы может отличаться для разных микросхем и разных напряжений питания. Для определения реальных уровней напряжения логических сигналов необходимо обратиться к технической документации устройства.

Рис. 1 – Представление значений бита на примере лампочки

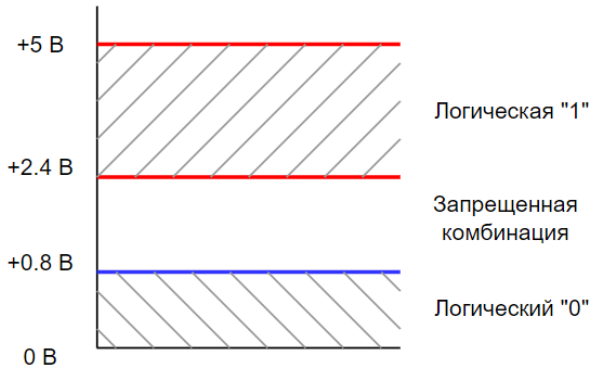

Рис. 2 – Уровни напряжений логических сигналов логики *TTL*

Основной единицей информации является байт, это 8 бит, идущих друг за другом (рис. 3). Байт может принимать значение от «00000000» до «11111111». Также одной из единиц измерения информации является ниббл (англ. *«nibble»*, *«nybble»*), или полубайт — единица измерения информации, равная четырём битам, и удобна тем, что представима одной шестнадцатеричной цифрой, то есть является одним шестнадцатеричным разрядом. Переменная размера ниббл может принимать 16 различных значений. В русском языке используется синоним – тетрада.

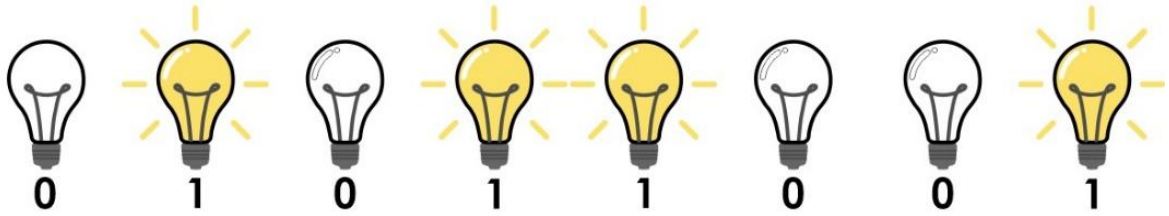

Рис. 3 – Представление байта на примере лампочек

Также биты можно передавать последовательно один за другим, на рис. 4 показан пример передачи 8 бит, где каждому биту информации, в зависимости от его значения (логический ноль или логическая единица), соответствует уровень напряжения согласно логике *TTL*.

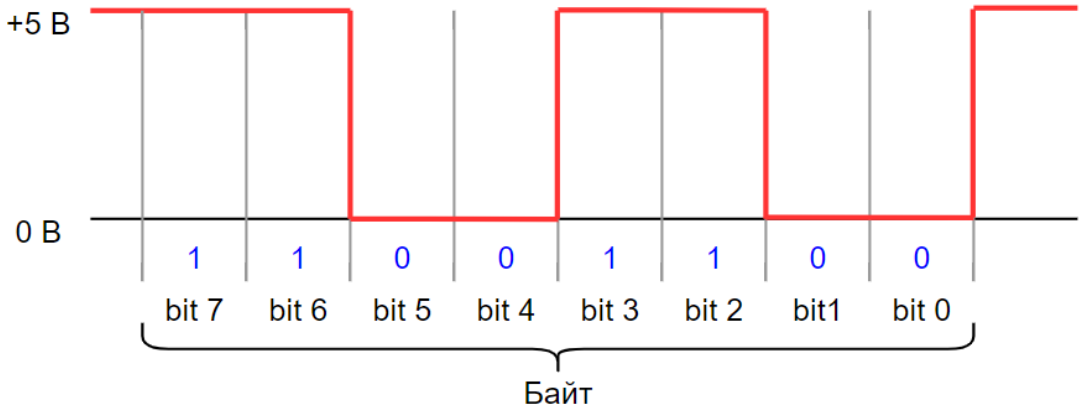

Рис. 4 – Пример последовательной передачи байта

Разряды в любой позиционной системе счисления считаются справа налево. Например, на рис. 5 представлено число в десятичной системе счисления, в котором самый старший разряд крайний левый в числе, а самый младший – крайний правый.

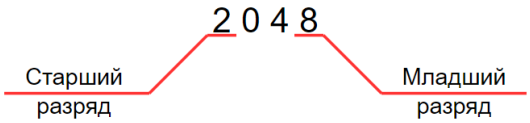

Рис. 5 – Разряды числа в десятичной системе счисления

По такому же принципу распределяется старшинство разрядов в двоичной системе счисления. На рис. 6 представлена структура 32-х битного числа.

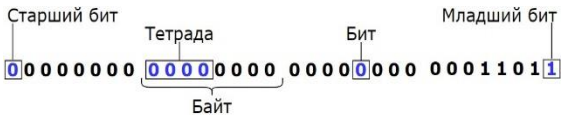

Рис. 6 – Структура 32-битного двоичного числа

Данные в микроконтроллере хранятся в ячейках памяти побайтно. Пример скомпилированного фрагмента кода программы приведён на рис. 7. Также можно прочитать

содержимое памяти микроконтроллера с помощью программы *«STM32 ST-LINK Utility»*. Например, на рис. 8 видно, что байт с адресом «0x8000160 + 3» = «0x8000163» имеет значение «0x08».

```
93  8000160:    0800036c    .word   0x0800036c
94  8000164:    20000000    .word   0x20000000
95  8000168:    20000000    .word   0x20000000
96  800016c:    20000000    .word   0x20000000
```

Рис. 7 – Скомпилированный фрагмент программы

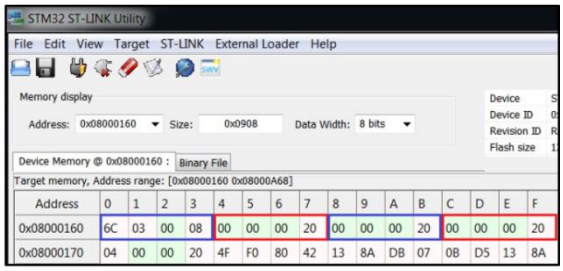

Рис. 8 – Содержимое *Flash* памяти микроконтроллера

В программах часто используются переменные, которые имеют следующие основные свойства:

1. Тип данных;
2. Размер данных.

Например, целочисленный тип данных может содержать как знаковые (*«int»*), так и беззнаковые (*«uint»*) числа. Этот тип данных может иметь размеры 8, 16, 32 и 64 бит (табл. 1).

Табл. 1

Целочисленные типы данных языка *C* для микроконтроллеров *STM32*

Обозначение	Тип данных	Разрядность бит (байт)	Диапазон данных
int8_t (char)	Целочисленный знаковый тип	8 (1)	- 128 … 127
uint8_t (unsigned char)	Целочисленный беззнаковый тип	8 (1)	0 … 255
int16_t (short)	Целочисленный знаковый тип	16 (2)	- 32 768 … 32 768
uint16_t (unsigned short)	Целочисленный беззнаковый тип	16 (2)	0 … 65 535
int32_t (int)	Целочисленный знаковый тип	32 (4)	- 2 147 483 648 … 2 147 483 647
uint32_t (unsigned int)	Целочисленный беззнаковый тип	32 (4)	0 … 4 294 967 295
int32_t (long)	Целочисленный знаковый тип	32 (4)	- 2 147 483 648 … 2 147 483 647
uint32_t (unsigned long)	Целочисленный беззнаковый тип	32 (4)	0 … 4 294 967 295
int64_t (long long)	Целочисленный знаковый тип	64 (8)	- 9 223 372 036 854 775 808 … 9 223 372 036 854 775 807
uint64_t (unsigned long long)	Целочисленный беззнаковый тип	64 (8)	0 … 18 446 744 073 709 551 615

Просматривая значения регистров памяти микроконтроллера, можно подумать, что он управляется не только двоичной, но и десятичной, шестнадцатеричной системой счисления (далее с.с.). На самом деле внутри цифровой части микроконтроллера используются только напряжение, близкое к 0 В (логический «0»), или напряжение, близкое к напряжению питания микроконтроллера (логическая «1»). Т.е. внутри только двоичная логика. Человеку достаточно просто ошибиться при записи, анализе большого количества нулей и единиц. Например, число «1001101001111011» очень похоже на «1001101000111011», однако это разные числа. Поэтому оказалось удобнее при выводе данных на экран, в большинстве случаев, переводить и записывать эти данные в других системах счисления. Приведённые выше числа в 16 с.с. будут соответственно «0x9A7B» и «0x9A3B». Различие в числах видно сразу, запись выглядит короче. Частое использование 16 с.с. объясняется тем, что перевод числа в 2 с.с. и обратно осуществляется намного проще, чем из 10 с.с. в 2 с.с. и обратно.

В языке программирования Си для обозначения шестнадцатеричного числа используется префикс «0x», а для обозначения числа в двоичной системе счисления – «0b» (рис. 9) [1,2].

$$0b11111111_2 = 0xFF_{16}$$

Рис. 9 – Представление числа в двоичной и шестнадцатеричной системах счисления

Далее на рис. 10-12 приведены алгоритмы перевода чисел для разных систем счисления.

Алгоритм перевода десятичного числа в двоичную систему счисления

1. В первой строке справа-налево записать числа 1, 2, 4, …, где каждое следующее число в два раза больше предыдущего.
2. Выбрать минимальное количество чисел из первой строки, которые в сумме давали бы исходное число.
3. Для того, чтобы получить двоичное число необходимо под каждым выбранным числом из первой строки написать «1», в противном случае «0»;
4. Пример перевода числа «45» из десятичной системы счисления в двоичный вид представлен ниже.

$$45_{10} = \begin{array}{cccccccc} 128 & 64 & 32 & 16 & 8 & 4 & 2 & 1 \\ \downarrow & & \downarrow & & \downarrow & \downarrow & & \downarrow \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1_2 \end{array}$$

Рис. 10 – Алгоритм перевода числа из десятичной системы счисления в двоичную

Алгоритм перевода двоичного числа в десятичную систему счисления

1. Справа-налево под каждым битом двоичного числа записать число 1, 2, 4, …, где каждое следующее записанное число в два раза больше предыдущего.
2. Во второй строке выбрать и сложить те числа, которые находятся под битами, которые равны «1».
3. Полученная сумма является исходным числом в десятичной системе счисления;
4. Пример перевода двоичного числа «00101101» в десятичную систему счисления представлен ниже.

$$\begin{array}{cccccccc} 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1_2 \\ 128 & 64 & 32 & 16 & 8 & 4 & 2 & 1 \end{array} = 32+8+4+1 = 45_{10}$$

*Рис.*11 – Алгоритм перевода числа из двоичной системы счисления в десятичную

Соответствие чисел в двоичной и шестнадцатеричной системах счисления представлены в табл. 2.

Алгоритм перевода двоичного числа в шестнадцатеричную систему счисления

1. Дополнить число слева битами «0», пока число разрядов не станет кратно восьми.
2. Разделить получившееся число на группы по 4 бита (тетрады).
3. Опираясь на таблицу 2, заменить тетрады на соответствующие им значения в шестнадцатеричной системе счисления;
4. Пример перевода двоичного числа «101101» в шестнадцатеричную систему счисления приведен ниже.

$$101101_2 = 00101101_2 = \begin{array}{cc} 0010 & 1101_2 \\ \downarrow & \downarrow \\ 2_{16} & D_{16} \end{array} = 2D_{16}$$

Рис. 12 – Алгоритм перевода числа из двоичной системы счисления в шестнадцатеричную

Табл. 2

Соответствие чисел в двоичной и шестнадцатеричной системах счисления

Шестнадцатеричная система счисления	Двоичная система счисления	Шестнадцатеричная система счисления	Двоичная система счисления
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

Необходимо понимать, что система счисления – это метод записи чисел. Арифметические операции справедливы для всех систем счисления, а значит, что полученные результаты после операций над числами в одной системе счисления будут равнозначны результатам, полученным в другой системе счисления.

На рис. 13 приведена операция сложения чисел «45» и «14» в разных системах счисления.

$$+\begin{array}{r}45_{10}\\14_{10}\\\hline 59_{10}\end{array} = +\begin{array}{r}00101101_{2}\\00001110_{2}\\\hline 00111011_{2}\end{array} = +\begin{array}{r}2D_{16}\\E_{16}\\\hline 3B_{16}\end{array}$$

Рис. 13 – Операции над числами в разных системах счисления

Для сложения двоичных чисел используется устройство, которое называется «сумматор». Разработчики для выполнения операции «разность», не стали вводить новое устройство – «вычитатель», а предложили способ представления двоичного числа в дополнительном коде. Данный способ даёт возможность представить число в отрицательном виде, что в свою очередь позволяет использовать для суммирования и вычитания чисел одно и то же устройство «сумматор».

Алгоритм получения отрицательного двоичного числа

1. Инвертировать биты двоичного числа;
2. Прибавить к получившемуся числу «0...00001».
3. Пример получения отрицательного двоичного числа представлен ниже.

$$+27_{10} = 00011011_{2}$$
$$\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow$$
$$11100100_{2}$$
$$+\ 00000001_{2}$$
$$-27_{10} = 11100101_{2}$$

Рис. 14 – Алгоритм получения отрицательного двоичного числа

При использовании сумматора проверим правильность операции, отняв число «27» от «28». Результат представлен на рис. 15.

$$+\begin{array}{r}00011100_{2}\\11100101_{2}\\\hline 00000001_{2}\end{array} \qquad 28_{10} + (-27_{10}) = 1_{10}$$

Рис. 15 – Операция разности в двоичной и десятичной системе счисления

В ходе развития информационных технологий начали появляться крупные массивы информации, что требовало ввести новые единицы измерения её объемов. Тогда была введена приставка «К» для обозначения 1024 байт (2^{10} байт). Но, в связи с тем, что люди начали использовать приставки системы измерения СИ, появилась неопределённость в понимании единиц измерения данных [3]. И поэтому в 1999 году Международная электротехническая комиссия (*МЭК*) ввела стандарт *IEC 60027-2* с приставками для обозначения количества информации (табл. 3) [4].

Впоследствии Российской Федерацией был утвержден *ГОСТ 8.417-2002*, который разрешил приставки из системы СИ, а также определил размеры единиц в зависимости от префикса [5].

«Положение о единицах величин, допускаемых к применению в Российской Федерации», утверждённое Правительством РФ 31 октября 2009 года дублирует *ГОСТ 8.417-2002*, а также допускает применение международного обозначения единицы информации с приставками «К», «М», «G» (*KB, MB, GB, Kbyte, Mbyte, Gbyte*) [6]. Но, несмотря на утверждённые ГОСТы и стандарты, на текущее время до сих пор сохраняется «путаница» в использовании терминов.

Табл. 3

Префиксы для двоичных единиц измерения

Множитель	Название приставки	Символ
2^{10}	кибу	Ki(Ku)
2^{20}	мебу	Mi(Mu)
2^{30}	гибу	Gi(Gu)
2^{40}	тебу	Ti(Tu)
2^{50}	пебу	Pi(Pu)
2^{60}	эксбу	Ei(Eu)
2^{70}	зебу	Zi(Zu)
2^{80}	йобу	Yi(Yu)

Кодирование символов

По умолчанию цифровое устройство способно понимать только двоичный код. Как было упомянуто ранее, бит имеет два значения: «1» или «0». И каждому символу, который выводится на экран, соответствует определенная комбинация нулей и единиц – *код символа.* Совокупность таких кодов называется кодировкой. Одной из таких кодировок является *ASCII*, которую можно считать прародителем всех современных кодировок.

Эта аббревиатура расшифровывается как *«American Standard Code for Information Interchange»* – американская стандартная кодировочная таблица для печатных символов и некоторых специальных кодов.

Кодировка *ASCII* была разработана в 1963 году Американской Ассоциацией Стандартов (которая позже стала *«Американским Национальным Институтом Стандартов»* – *ANSI*). Кодировка несколько раз обновлялась – в 1967 и 1986 годах [7].

ASCII – это таблица кодировки символов, в которой каждой букве, цифре или знаку присвоено определённое число. Изначально

ASCII являлась 7-битной кодировкой, включающей в себя 128 символов: 33 непечатных управляющих символа, влияющих на обработку текста и пробелов, и 95 печатных символов, включая цифры, буквы латинского алфавита в строчном и прописном вариантах, ряд пунктуационных символов (табл. 5 – 6).

Первые тридцать два символа в *ASCII*-таблице являются управляющими символами, и не имеют печатного отображения. Например, символ с кодом 10 (столбец *«DEC»*) осуществляет перевод строки, а с помощью символа с кодом 7 (столбец *«DEC»*) происходит вызов звукового сигнала системного динамика.

В 1990 году Андрей Чернов разработал кодировку *КОИ-8* [8]. Разработчики *КОИ-8* присвоили символам русского алфавита коды из диапазона 159 – 255. Таким образом, получилось, что позиции символов кириллицы соответствуют их фонетическим аналогам в английском алфавите из нижней части таблицы. Это означает, что если в тексте, написанном в *КОИ-8*, для каждого символа убрать по одному биту слева, то получится относительно читаемый текст, подобный транслиту. Например, слова «Русский Текст» превратятся в *«rUSSKIJ tEKST»*. Поэтому символы кириллицы в таблице расположены не в алфавитном порядке.

Существует несколько вариантов кодировки *КОИ-8* для различных кириллических алфавитов. Русский алфавит описывается в кодировке *КОИ8-R* (табл. 7 – 8), украинский — в *КОИ8-U*, таджикский — в *КОИ8-T*.

В *КОИ8-R* символы с кодами 0 – 127 совпадают с *ASCII*-таблицей, а символы кириллицы содержатся в диапазоне кодов 163 – 255.

До появления *Windows* для работы в *DOS* применялась кодировка *CP866* (табл. 9 – 10) [9]. Символы кодировки *CP866* в диапазоне от 0 до 127 (столбец *«DEC»*) полностью соответствуют *ASCII* кодировке. Далее, кодировка, в диапазоне от 128 до 255 (столбец *«DEC»*) содержит символы кириллицы и символы псевдографики, благодаря которым можно было выводить на экран (с определёнными ограничениями) различные рисунки, картинки, не переходя в графический режим. Ярким примером качественной визуализации информации с помощью псевдографики является отечественный текстовый редактор *Lexicon* [10]. С кодировкой *CP866* можно встретиться и сейчас, открыв командную строку. Для просмотра используемой кодировки в командной с9троке, необходимо выполнить команду «chcp». (рис. 16). Согласно идентификаторам, приведенным на официальном сайте *Microsoft* [11], выведенное значение «866» соответствует кодировке *CP866*.

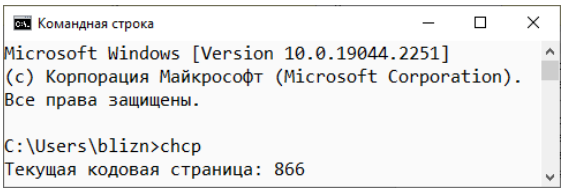

Рис. 16 – Командная строка *Windows*

Следующим шагом в развитии кодировок можно считать появление так называемых *ANSI*-кодировок.

Это были те же расширенные версии *ASCII*, однако из них были удалены различные псевдографические элементы и добавлены символы типографики, для которых ранее не хватало «свободных мест».

Примером такой *ANSI*-кодировки является всем известная *«Windows-1251»* (табл. 11 – 12), которая содержит в себе символы кириллицы [12].

ANSI-кодировка — это собирательное название. В действительности, реальная кодировка при использовании *ANSI* будет определяться тем, что указано в реестре вашей операционной системы *Windows*. В случае с русским языком это будет *«Windows-1251»*, однако, для других языков это будет другая разновидность *ANSI*.

Для того чтобы понять, какая кодировка используется на вашей операционной системе Windows, необходимо:

– вызвать диалоговое окно «Выполнить» с помощью комбинации клавиш *«Win+R»*;

– выполнить команду *«regedit»*, написав ее в текстовое поле «Открыть» (рис. 17).

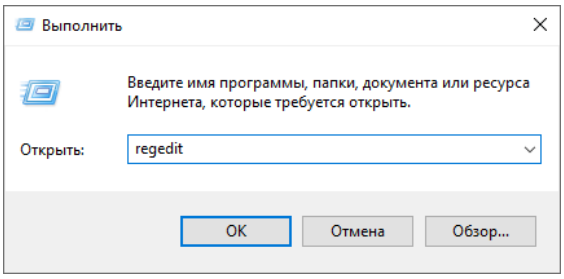

Рис. 17 – Диалоговое окно «Выполнить»

Для того, чтобы узнать используемую кодировку, необходимо открыть в появившемся окне папку *«CodePage»*, расположенную по следующему пути (рис. 18).

Компьютер → HKEY_LOCAL_MACHINE → SYSTEM → → CurrentControlSet → Control → Nls → CodePage

Рис. 18 – Путь к папке для просмотра используемой кодировки

В данной папке в правой стороне окна необходимо найти параметр *«ACP»*, который содержит в себе идентификатор используемой

кодировки (рис. 19). Данные идентификаторы представлены на официальном сайте *Microsoft* [13]. В столбце «Значение» для параметра *ACP* указано «1251», что соответствует кодировке *Windows-1251*.

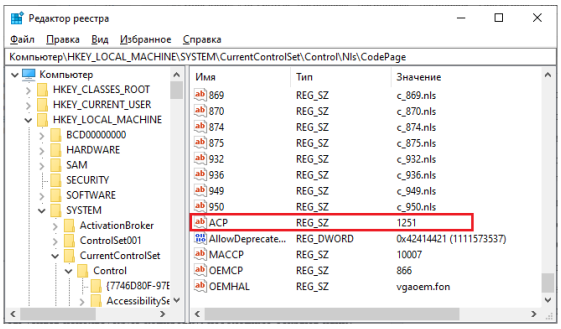

Рис. 19 – Значение параметра *«ACP»* в реестре операционной системы

Стандарт *Unicode* был предложен в 1991 году некоммерческой организацией *«Консорциум Юникода»* (*«Unicode Consortium»*, *«Unicode Inc.»*), и первым результатом его работы стало создание кодировки *UTF-32* [13]. *Unicode* – это таблица соответствия символов и чисел, а различные *UTF*-кодировки определяют, как эти числа переводятся в байты.

В *UTF-32* для кодирования одного символа предполагалось использовать целых 32 бита, т.е. 4 байта информации. На смену ей пришла новая разработка – *UTF-16*.

Из названия следует, что в этой кодировке один символ кодируется уже не 32 битами, а только 16 (т.е. 2 байта). Это позволило сделать любой символ вдвое "легче", чем в *UTF-32*.

Следующим шагом было введение новой кодировки *UTF-8*. Для кодирования одного символа в этой кодировке используется от 1 до 4 байт. Стандарт *Юникод* присваивает каждому символу код в виде неотрицательного целого числа, записываемого обычно в шестнадцатеричной форме с префиксом «*U+*», например, «*U+040F*». Коды в стандарте Юникод разделены на несколько областей. Область с кодами от «*U+0000*» до «*U+007F*» содержит символы набора *ASCII*, и коды этих символов совпадают с их кодами в *ASCII*. Далее расположены области символов других систем письменности, знаки пунктуации и технические символы. Часть кодов зарезервирована для использования в будущем. В зависимости от того, к какой области относится символ, выбирается необходимое количество байт для его кодировки (табл. 4). Так, например, для кодирования символов кириллицы необходимо 2 байта, так как данные символы находятся в промежутке «*U+0400 – U+04FF*».

Табл. 4

Соотношение диапазона кодов и количества байтов необходимого для кодирования

Диапазон кодов	Необходимое количество байтов для кодирования
U+00000000 ... U+0000007F	1
U+00000080 ... U+000007FF	2
U+00000800 ... U+0000FFFF	3
U+00010000 ... U+0010FFFF	4

Таким образом, была получена универсальная кодировка, позволяющая охватить все возможные символы, которые требуется отобразить, не "утяжеляя" без необходимости файлы.

ASCII-ГРАФИКА

Помимо вывода текстовой информации на экран *ASCII* кодировка также применяется в графике. В *ASCII*-графике символы будто представляют собой штрихи на картине, каждый из которых имеет уникальную форму и несёт свою смысловую нагрузку.

Первые поколения компьютеров не имели возможности отображать графические изображения. Поэтому программисты заинтересовались возможностью представления графики с помощью текстовых символов и вывода полученных изображений на экран терминала и устройство печати. Начиная с 1960-х годов, создание программы, которая «рисует» заданное изображение текстовыми символами, было популярным развлечением программистов. Для примера, на рис. 20 представлено изображение в формате *ASCII*-графики.

Сейчас существует большое количество программ и онлайн сервисов, которые предоставляют возможность конвертировать любое заданное изображение в рисунок, состоящий из *ASCII* символов.

Рис. 20 – Изображение, созданное из *ASCII* символов

Вывод данных на экран

На рис. 21 приведена структурная схема системы, позволяющая передать данные от микроконтроллера к компьютеру с выводом информации на экран.

На рис. 22 рассмотрен фрагмент программы для микроконтроллера, который выводит символы «AAAA…AAA» на экран персонального компьютера.

В данной программе, после отправки символа с микроконтроллера, необходимо вставить задержку по времени. Это связано с тем, что команда отправки символа расположена в бесконечном цикле. Отправка данных без задержки приведет к слишком большому потоку отправляемых символов, что может вызвать зависание систем.

```
int main(void){
    uint8_t symbol = 0x41; // 'A'

    while(1){
        USART1->DR = symbol;
        delay_ms(100);
    }
}
```

Рис. 22 – Программа по отправке символов «AAAA…AAA» на компьютер

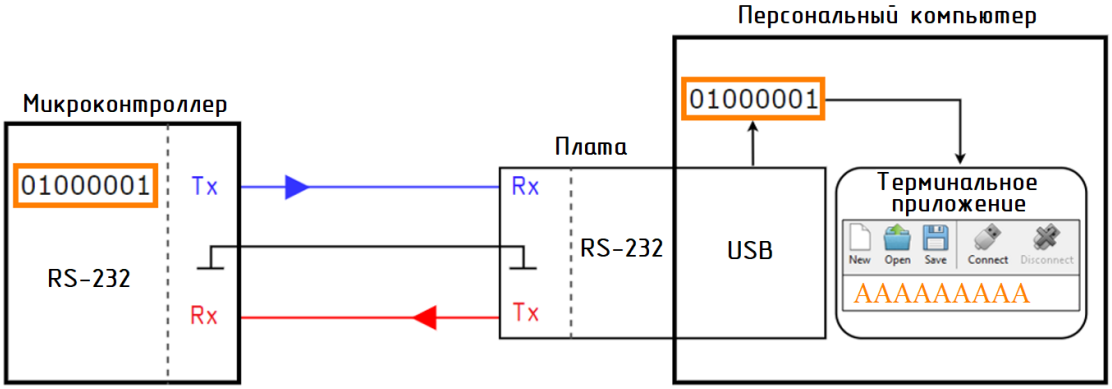

Рис. 21 – Структурная схема системы передачи данных между микроконтроллером *STM32* и персональным компьютером

После загрузки программы в микроконтроллер необходимо установить соединение устройства с терминальным приложением. В данном примере используется терминальное приложение *«CoolTerm»* (рис. 23) [14]. Для этого установим необходимый *COM*-порт. Открыв раздел *«Options»* выберем из выпадающего списка используемый порт (рис. 24).

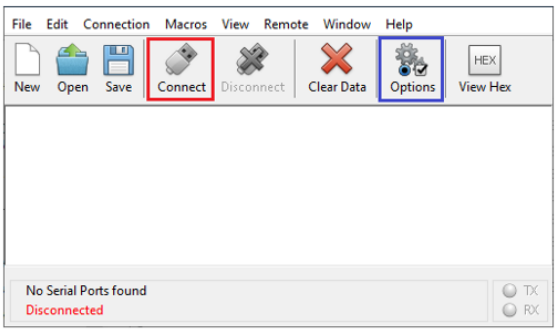

Рис. 23 – Терминальное приложение *«CoolTerm»*

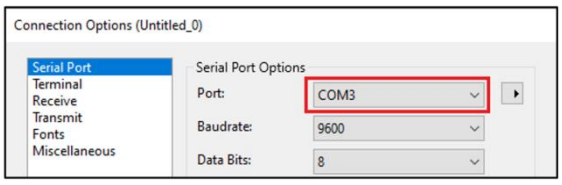

Рис. 24 – Выбор *COM*-порта в настройках терминального приложения

Для того, чтобы узнать какой *COM*-порт использует устройство, необходимо открыть «Диспетчер устройств». Открывается он следующим образом:

1. Нажать на значок «Компьютер», в проводнике или на рабочем столе, правой кнопкой мыши и выбрать «Свойства» (рис. 25).
2. Выбрать пункт «Диспетчер устройств» в открывшемся окне.

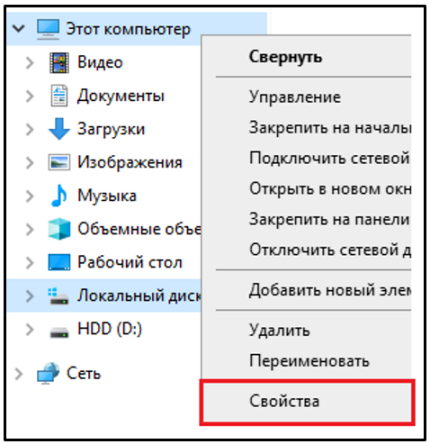

Рис. 25 - Путь к диспетчеру устройств

Для просмотра используемых портов в «Диспетчере устройств» необходимо перейти в

раздел «*COM и LPT*», где и содержится информация о всех используемых *COM*-портах (рис. 26).

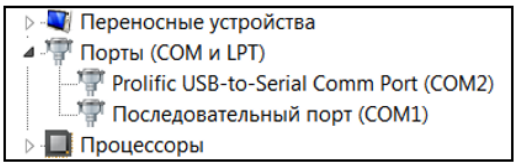

Рис. 26 - Информация о *COM*-портах в «Диспетчере устройств»

После выбора требуемого *COM*-порта, нажав *«Connect»,* установим соединение (рис. 23).

Далее, в разделе *«Options»*, необходимо настроить кодировку в терминальном приложении, выбрав кодировку *«ASCII»* (рис. 27).

После установления соединения и настройки кодировки необходимо запустить выполнение программы на микроконтроллере.

На рис. 28 представлено отображение символов «АААА…ААА» в терминальном приложении.

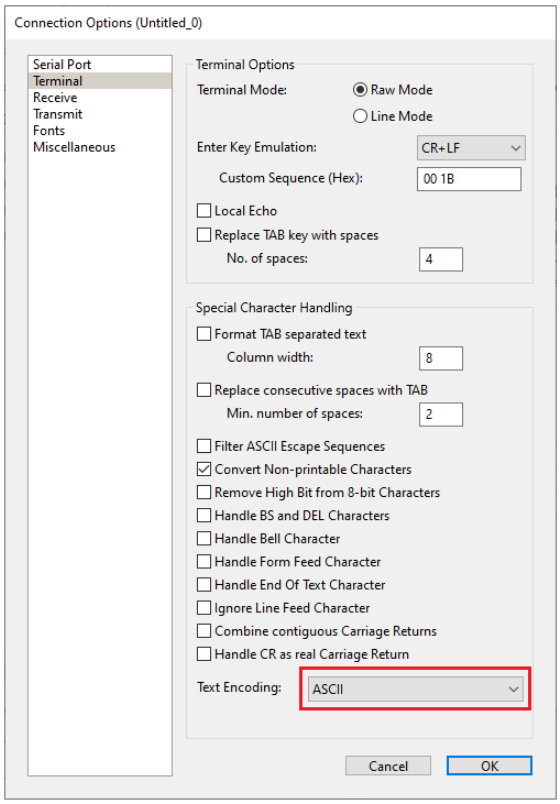

Рис. 27 – Выбор кодировки в настройках терминального приложения

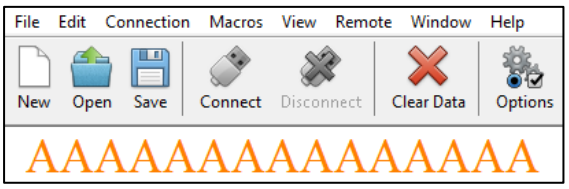

Рис. 28 – Принятые символы в терминальном приложении

Далее реализуем программу, которая бы выводила слово «Бит» с символом «!» (рис. 29). В данной программе отправка осуществляется в главной функции программы. Возможны ситуации, когда в терминальном приложении может отобразиться только часть информации или она вообще не будет выведена. Данная особенность может произойти из-за того, что программа в микроконтроллере уже выполнилась до того, как была установлена связь устройства и терминального приложения.

В программе для микроконтроллера важно использовать бесконечный цикл *«while»*. Данный цикл необходим для того, чтобы выполнение программы оставалось в границах главной функции «*main*». При отсутствии цикла *«while»*, в данной программе, после выполнения написанного кода, микроконтроллер выйдет за рамки главного цикла *«main»*. Далее он начнёт выполнять команды в следующих ячейках памяти, значение которых нам неизвестно, а значит и поведение устройства будет непредсказуемым.

```
int main(void){
    uint8_t symbol[4] = {0xC1,'и',0xF2,'!'};
    USART1->DR = symbol[0]; //'Б'
    delay_ms(10);
    USART1->DR = symbol[1]; //'и'
    delay_ms(10);
    USART1->DR = symbol[2]; //'т'
    delay_ms(10);
    USART1->DR = symbol[3]; //'!'
    delay_ms(10);

    while(1){ }
}
```

Рис. 29 – Программа по отправке слова «Бит!» на компьютер

На рис. 30 представлено отображение слово «Бит!» в терминальном приложении.

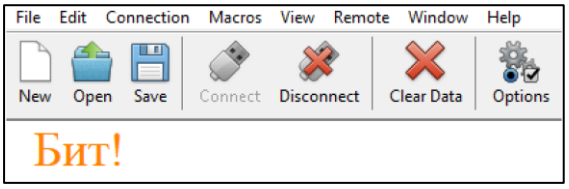

Рис. 30 – Принятое слово «Бит!»

Таблицы кодировок

Табл. 5

Кодировка *ASCII* (символы с кодами 0 - 31, 127)

DEC	HEX	Символ	Описание	DEC	HEX	Символ	Описание
0	0x00	NUL	Нулевой байт	17	0x11	DC1	1-й символ управления устройством
1	0x01	SOH	Начало заголовка	18	0x12	DC2	2-й символ управления устройством
2	0x02	STX	Начало текста	19	0x13	DC3	3-й символ управления устройством
3	0x03	ETX	Конец текста	20	0x14	DC4	4-й символ управления устройством
4	0x04	EOT	Конец передачи	21	0x15	NAK	Не подтверждаю!
5	0x05	ENQ	Прошу подтверждения!	22	0x16	SYN	Символ для синхронизации
6	0x06	ACK	Подтверждаю!	23	0x17	ETB	Конец текстового блока
7	0x07	BEL	Звуковой сигнал — звонок	24	0x18	CAN	Отмена
8	0x08	BS	Возврат на один символ	25	0x19	EM	Конец носителя
9	0x09	HT	Табуляция	26	0x1A	SUB	Подставить
10	0x0A	LF	Перевод строки	27	0x1B	ESC	Escape (расширение)
11	0x0B	VT	Вертикальная табуляция	28	0x1C	FS	Разделитель файлов
12	0x0C	FF	Прогон страницы, новая страница	29	0x1D	GS	Разделитель групп
13	0x0D	CR	Возврат каретки	30	0x1E	RS	Разделитель записей
14	0x0E	SO	Переключиться на другую кодировку	31	0x1F	US	Разделитель юнитов
15	0x0F	SI	Переключиться на исходную кодировку	127	0x7F	Delete	Символ для удаления (на перфолентах)
16	0x10	DLE	Экранирование канала данных				

Табл. 6

Кодировка *ASCII* (символы с кодами 32 - 126)

DEC	HEX	Символ	DEC	HEX	Символ	DEC	HEX	Символ	DEC	HEX	Символ
32	0x20	Пробел	56	0x38	8	80	0x50	P	104	0x68	h
33	0x21	!	57	0x39	9	81	0x51	Q	105	0x69	i
34	0x22	“	58	0x3A	:	82	0x52	R	106	0x6A	j
35	0x23	#	59	0x3B	;	83	0x53	S	107	0x6B	k
36	0x24	$	60	0x3C	<	84	0x54	T	108	0x6C	l
37	0x25	%	61	0x3D	=	85	0x55	U	109	0x6D	m
38	0x26	&	62	0x3E	>	86	0x56	V	110	0x6E	n
39	0x27	‘	63	0x3F	?	87	0x57	W	111	0x6F	o
40	0x28	(	64	0x40	@	88	0x58	X	112	0x70	p
41	0x29	)	65	0x41	A	89	0x59	Y	113	0x71	q
42	0x2A	*	66	0x42	B	90	0x5A	Z	114	0x72	r
43	0x2B	+	67	0x43	C	91	0x5B	[	115	0x73	s
44	0x2C	,	68	0x44	D	92	0x5C	\	116	0x74	t
45	0x2D	–	69	0x45	E	93	0x5D	]	117	0x75	u
46	0x2E	.	70	0x46	F	94	0x5E	^	118	0x76	v
47	0x2F	/	71	0x47	G	95	0x5F	_	119	0x77	w
48	0x30	0	72	0x48	H	96	0x60	`	120	0x78	x
49	0x31	1	73	0x49	I	97	0x61	a	121	0x79	y
50	0x32	2	74	0x4A	J	98	0x62	b	122	0x7A	z
51	0x33	3	75	0x4B	K	99	0x63	c	123	0x7B	{
52	0x34	4	76	0x4C	L	100	0x64	d	124	0x7C	|
53	0x35	5	77	0x4D	M	101	0x65	e	125	0x7D	}
54	0x36	6	78	0x4E	N	102	0x66	f	126	0x7E	~
55	0x37	7	79	0x4F	O	103	0x67	g			

Табл. 7

Кодировка *КОИ8-R* (символы с кодами 128 - 193)

DEC	HEX	Символ	DEC	HEX	Символ	DEC	HEX	Символ
128	0x80	─	150	0x96	·√	172	0xAC	╜
129	0x81	│	151	0x97	≈	173	0xAD	╙
130	0x82	┌	152	0x98	≤	174	0xAE	╘
131	0x83	┐	153	0x99	≥	175	0xAF	╒
132	0x84	└	154	0x9A		176	0xB0	╓
133	0x85	┘	155	0x9B	⌠	177	0xB1	╔
134	0x86	├	156	0x9C	°	178	0xB2	╕
135	0x87	┤	157	0x9D	²	179	0xB3	Ё
136	0x88	┬	158	0x9E	·	180	0xB4	╖
137	0x89	┴	159	0x9F	÷	181	0xB5	╕
138	0x8A	┼	160	0xA0	═	182	0xB6	╤
139	0x8B	▀	161	0xA1	‖	183	0xB7	╥
140	0x8C	▄	162	0xA2	⌡	184	0xB8	╦
141	0x8D	▌	163	0xA3	ё	185	0xB9	╧
142	0x8E	▐	164	0xA4	⌐	186	0xBA	╨
143	0x8F	▄	165	0xA5	⌐	187	0xBB	╩
144	0x90	▒	166	0xA6	¬	188	0xBC	╪
145	0x91	▒	167	0xA7	½	189	0xBD	╫
146	0x92	▒	168	0xA8	¼	190	0xBE	╬
147	0x93	⌠	169	0xA9	⌙	191	0xBF	©
148	0x94	■	170	0xAA	⌊	192	0xC0	ю
149	0x95	▪	171	0xAB	⌊	193	0xC1	а

Табл. 8

Кодировка *КОИ8-R* (символы с кодами 194 - 255)

DEC	HEX	Символ	DEC	HEX	Символ	DEC	HEX	Символ
194	0xC2	б	215	0xD7	в	236	0xEC	Л
195	0xC3	ц	216	0xD8	ь	237	0xED	М
196	0xC4	д	217	0xD9	ы	238	0xEE	Н
197	0xC5	е	218	0xDA	з	239	0xEF	О
198	0xC6	ф	219	0xDB	ш	240	0xF0	П
199	0xC7	г	220	0xDC	э	241	0xF1	Я
200	0xC8	х	221	0xDD	щ	242	0xF2	Р
201	0xC9	и	222	0xDE	ч	243	0xF3	С
202	0xCA	й	223	0xDF	ъ	244	0xF4	Т
203	0xCB	к	224	0xE0	Ю	245	0xF5	У
204	0xCC	л	225	0xE1	А	246	0xF6	Ж
205	0xCD	м	226	0xE2	Б	247	0xF7	В
206	0xCE	н	227	0xE3	Ц	248	0xF8	Ь
207	0xCF	о	228	0xE4	Д	249	0xF9	Ы
208	0xD0	п	229	0xE5	Е	250	0xFA	З
209	0xD1	я	230	0xE6	Ф	251	0xFB	Ш
210	0xD2	р	231	0xE7	Г	252	0xFC	Э
211	0xD3	с	232	0xE8	Х	253	0xFD	Щ
212	0xD4	т	233	0xE9	И	254	0xFE	Ч
213	0xD5	у	234	0xEA	Й	255	0xFF	Ъ
214	0xD6	ж	235	0xEB	К			

Табл. 9

Кодировка *CP866* (символы с кодами 128 - 193)

DEC	HEX	Символ	DEC	HEX	Символ	DEC	HEX	Символ
128	0x80	А	150	0x96	Ц	172	0xAC	м
129	0x81	Б	151	0x97	Ч	173	0xAD	н
130	0x82	В	152	0x98	Ш	174	0xAE	о
131	0x83	Д	153	0x99	Щ	175	0xAF	п
132	0x84	Е	154	0x9A	Ъ	176	0xB0	▒
133	0x85	Ж	155	0x9B	Ы	177	0xB1	▒
134	0x86	З	156	0x9C	Ь	178	0xB2	▓
135	0x87	И	157	0x9D	Э	179	0xB3	│
136	0x88	Й	158	0x9E	Ю	180	0xB4	┤
137	0x89	К	159	0x9F	Я	181	0xB5	╡
138	0x8A	К	160	0xA0	а	182	0xB6	╢
139	0x8B	Л	161	0xA1	б	183	0xB7	╖
140	0x8C	М	162	0xA2	в	184	0xB8	╕
141	0x8D	Н	163	0xA3	г	185	0xB9	╣
142	0x8E	О	164	0xA4	д	186	0xBA	║
143	0x8F	П	165	0xA5	е	187	0xBB	╗
144	0x90	Р	166	0xA6	ж	188	0xBC	╝
145	0x91	С	167	0xA7	з	189	0xBD	╜
146	0x92	Т	168	0xA8	и	190	0xBE	╛
147	0x93	У	169	0xA9	й	191	0xBF	┐
148	0x94	Ф	170	0xAA	к	192	0xC0	└
149	0x95	Х	171	0xAB	л	193	0xC1	┴

Табл. 10

Кодировка *CP866* (символы с кодами 194 - 255)

DEC	HEX	Символ	DEC	HEX	Символ	DEC	HEX	Символ
194	0xC2	┬	215	0xD7	╫	236	0xEC	ь
195	0xC3	├	216	0xD8	╪	237	0xED	э
196	0xC4	─	217	0xD9	┘	238	0xEE	ю
197	0xC5	┼	218	0xDA	┌	239	0xEF	я
198	0xC6	╞	219	0xDB	█	240	0xF0	Ё
199	0xC7	╟	220	0xDC	▄	241	0xF1	ё
200	0xC8	╚	221	0xDD	▌	242	0xF2	Є
201	0xC9	╔	222	0xDE	▐	243	0xF3	є
202	0xCA	╩	223	0xDF	▀	244	0xF4	Ї
203	0xCB	╦	224	0xE0	р	245	0xF5	ї
204	0xCC	╠	225	0xE1	с	246	0xF6	Ў
205	0xCD	═	226	0xE2	т	247	0xF7	ў
206	0xCE	╬	227	0xE3	у	248	0xF8	°
207	0xCF	╧	228	0xE4	ф	249	0xF9	∙
208	0xD0	╨	229	0xE5	х	250	0xFA	·
209	0xD1	╤	230	0xE6	ц	251	0xFB	√
210	0xD2	╥	231	0xE7	ч	252	0xFC	№
211	0xD3	╙	232	0xE8	ш	253	0xFD	¤
212	0xD4	╘	233	0xE9	щ	254	0xFE	■
213	0xD5	╒	234	0xEA	ъ	255	0xFF	Неразрывный пробел
214	0xD6	╓	235	0xEB	ы			

Табл. 11

Кодировка *Windows-1251* (символы с кодами 128 - 193)

DEC	HEX	Символ	DEC	HEX	Символ	DEC	HEX	Символ
128	0x80	Ђ	150	0x96	–	172	0xAC	¬
129	0x81	Ѓ	151	0x97	—	173	0xAD	Мягкий перенос
130	0x82	‚	152	0x98	Начало строки	174	0xAE	®
131	0x83	ѓ	153	0x99	™	175	0xAF	Ї
132	0x84	„	154	0x9A	љ	176	0xB0	°
133	0x85	…	155	0x9B	›	177	0xB1	±
134	0x86	†	156	0x9C	Њ	178	0xB2	І
135	0x87	‡	157	0x9D	Ќ	179	0xB3	і
136	0x88	€	158	0x9E	ћ	180	0xB4	ґ
137	0x89	‰	159	0x9F	џ	181	0xB5	µ
138	0x8A	Љ	160	0xA0	Неразрывный пробел	182	0xB6	¶
139	0x8B	‹	161	0xA1	Ў	183	0xB7	·
140	0x8C	Њ	162	0xA2	ў	184	0xB8	ё
141	0x8D	Ќ	163	0xA3	Ј	185	0xB9	№
142	0x8E	Ћ	164	0xA4	¤	186	0xBA	є
143	0x8F	Џ	165	0xA5	Ґ	187	0xBB	»
144	0x90	ђ	166	0xA6	¦	188	0xBC	ј
145	0x91	‘	167	0xA7	§	189	0xBD	Ѕ
146	0x92	’	168	0xA8	Ё	190	0xBE	ѕ
147	0x93	“	169	0xA9	©	191	0xBF	ї
148	0x94	”	170	0xAA	Є	192	0xC0	А
149	0x95	•	171	0xAB	«	193	0xC1	Б

Табл. 12

Кодировка *Windows-1251* (символы с кодами 194 - 255)

DEC	HEX	Символ	DEC	HEX	Символ	DEC	HEX	Символ
194	0xC2	В	215	0xD7	Ч	236	0xEC	м
195	0xC3	Г	216	0xD8	Ш	237	0xED	н
196	0xC4	Д	217	0xD9	Щ	238	0xEE	о
197	0xC5	Е	218	0xDA	Ъ	239	0xEF	п
198	0xC6	Ж	219	0xDB	Ы	240	0xF0	р
199	0xC7	З	220	0xDC	Ь	241	0xF1	с
200	0xC8	И	221	0xDD	Э	242	0xF2	т
201	0xC9	Й	222	0xDE	Ю	243	0xF3	у
202	0xCA	К	223	0xDF	Я	244	0xF4	ф
203	0xCB	Л	224	0xE0	а	245	0xF5	х
204	0xCC	М	225	0xE1	б	246	0xF6	ц
205	0xCD	Н	226	0xE2	в	247	0xF7	ч
206	0xCE	О	227	0xE3	г	248	0xF8	ш
207	0xCF	П	228	0xE4	д	249	0xF9	щ
208	0xD0	Р	229	0xE5	е	250	0xFA	ъ
209	0xD1	С	230	0xE6	ж	251	0xFB	ы
210	0xD2	Т	231	0xE7	з	252	0xFC	ь
211	0xD3	У	232	0xE8	и	253	0xFD	э
212	0xD4	Ф	233	0xE9	й	254	0xFE	ю
213	0xD5	Х	234	0xEA	к	255	0xFF	я
214	0xD6	Ц	235	0xEB	л			

Заключение

В данной статье рассмотрены способы представления и отображения информации при работе с микроконтроллерами семейства STM32. По результатам работы можно сделать следующие основные выводы:

Цифровые устройства для управления, при хранении данных используют исключительно нули и единицы. Другие системы счисления (8 с.с., 10 с.с., 16 с.с.) используются при отображении для облегчения процесса восприятия данной информации.

Система счисления – это метод записи чисел, поэтому результаты арифметических операций в одной системе счисления будут равнозначны результатам в другой системе счисления.

Для отображения информации на экране монитора и т.п. существуют перекодировочные таблицы, где приведены соответствия между двоичными данными и символами, цифрами, буквами. При получении таких двоичных данных устройство отображения выводит соответствующие символы, цифры, буквы, которые и воспринимает человек.

Литература

[1] Шестнадцатеричная система счисления. [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/Шестнадцатеричная_система_счисления

[2] Двоичная система счисления. [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/Двоичная_система_счисления

[3] Сколько байтов в килобайте? [Электронный ресурс]. Режим доступа: https://www.artlebedev.ru/kovodstvo/sections/84

[4] ГОСТ IEC 60027-2— 2015. [Электронный ресурс]. Режим доступа: https://files.stroyinf.ru/Data2/1/4293758/4293758971.pdf

[5] ГОСТ 8.417-2002. [Электронный ресурс]. Режим доступа: https://astro.insma.urfu.ru/sites/default/files/upload_files/doc/gost_8.417-2002.pdf

[6] "Об утверждении Положения о единицах величин, допускаемых к применению в Российской Федерации". [Электронный ресурс]. Режим доступа: https://base.garant.ru/196573/?ysclid=ladx35a073471909821

[7] ASCII. [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/ASCII

[8] КОИ-8. [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/КОИ-8

[9] CP866. [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/CP866

[10] Лексикон (программа). [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/Лексикон_(программа)

[11] Идентификаторы кодовой страницы. [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/intl/code-page-identifiers

[12] Windows-1251. [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/Windows-1251

[13] Юникод. [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/Юникод

[14] Терминальное приложение «CoolTerm». [Электронный ресурс]. Режим доступа: https://coolterm.en.lo4d.com/windows

Александр Евгеньевич Близнюк – магистрант группы ААМ-21 кафедры Автоматики НГТУ. Автор трёх научных работ.
E-mail: bliznyuksaha@mail.ru

Виталий Геннадьевич Трубин – старший преподаватель кафедры Автоматики НГТУ, директор ООО «КБ Автоматика». Автор более 50 научных и учебных работ.
E-mail: trubin@ngs.ru

Галина Владимировна Саблина – доцент кафедры Автоматики НГТУ, кандидат технических наук. Автор более 40 научных и учебных работ.
E-mail: sablina@corp.nstu.ru

Статья поступила 10.11.2022.

Presentation and Display of Information when Working with STM32 Microcontrollers

A.E. Bliznyuk, V.G. Trubin, G.V. Sablina
Novosibirsk State Technical University, Novosibirsk, Russia

Abstract: The article is devoted to the ways of presenting and displaying information when working with microcontrollers of the *STM32* family. Binary, hexadecimal number systems, variables of different lengths are described and translation algorithms from one number system to another are given. The question of the representation of negative integers in binary form is considered, the main reason for this method of representation is explained. It is shown how data is stored in the memory of the *STM32* microcontroller, two programs in which characters are transferred from the microcontroller to a personal computer

and then displayed on the screen when using *ASCII* encoding are presented. The presented material may be interesting not only to engineers and students, but also to readers who do not have knowledge in this field.

Key words: Number system, bit, nibble, tetrad, byte, microcontroller, *STM32*, terminal application, negative numbers, additional code, encoding, *ASCII*, *KOI-8*, *CP866*, *Windows-1251*, *ASCII* graphics, *UTF-8*, *UTF-16*, *UTF-32*, *Unicode*, *CoolTerm*, *C*, encoding tables, *IEC 60027-2*, *GOST 8.417-200*.

References

[1] Hexadecimal number system. [electronic resource]. Access mode: https://ru.wikipedia.org/wiki/Шестнадцатеричная_система_счисления

[2] Binary number system. [electronic resource]. Access mode: https://ru.wikipedia.org/wiki/Двоичная система счисления

[3] How many bytes are there in a kilobyte? [electronic resource]. Access mode: https://www.artlebedev.ru/kovodstvo/sections/84

[4] GOST IEC 60027-2— 2015. [electronic resource]. Access mode: https://files.stroyinf.ru//Data2/1/4293758/4293758971.pdf

[5] GOST 8.417-2002. [electronic resource]. Access mode: https://astro.insma.urfu.ru/sites/default/files/upload_files/doc/gost_8.417-2002.pdf

[6] "On approval of the Regulations on Units of Quantities Allowed for Use in the Russian Federation". [electronic resource]. Access mode: https://base.garant.ru/196573/ysclid=ladx35a073471909821

[7] ASCII. [electronic resource]. Access mode: https://ru.wikipedia.org/wiki/ASCII

[8] KOI-8. [Electronic resource]. Access mode: https://ru.wikipedia.org/wiki/КОИ-8

[9] CP866. [Electronic resource]. Access mode: https://ru.wikipedia.org/wiki/CP866

[10] Lexicon (program). [electronic resource]. Access mode: https://ru.wikipedia.org/wiki/Lexicon_(program)

[11] Code page identifiers. [electronic resource]. Access mode: https://learn.microsoft.com/ru-ru/windows/win32/intl/code-page-identifiers

[12] Windows-1251. [electronic resource]. Access mode: https://ru.wikipedia.org/wiki/Windows-1251

[13] Unicode. [electronic resource]. Access mode: https://ru.wikipedia.org/wiki/Юникод

[14] Terminal application "CoolTerm". [electronic resource]. Access mode: https://coolterm.en.lo4d.com/windows

Alexander E. Bliznyuk, student of the 2nd year of the master's program of the Automation Department of NSTU. Author of three scientific papers.
E-mail: bliznyuksaha@mail.ru

Vitaly G. Trubin, senior lecturer at the Department of Automation of NSTU, Director of KB «Automatika LLC». Author of more than 50 scientific and educational works.
E-mail: trubin@ngs.ru

Galina V. Sablina, Associate Professor of the Department of Automation of NSTU, Candidate of Technical Sciences. Author of more than 40 scientific and educational works.
E-mail: sablina@corp.nstu.ru

The paper has been recieved 10.11.2022.