МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Московский
ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ
(ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ)

Проект по курсу «Микроконтроллеры» 4 семестр

Динамическая индикация с использованием технологии USART

Студенты
Кульбицкий Никита
Соколов Вадим
Лисов Роман
718 группа
Преподаватель
Донов Геннадий Иннокентьевич

22 апреля 2019 г.

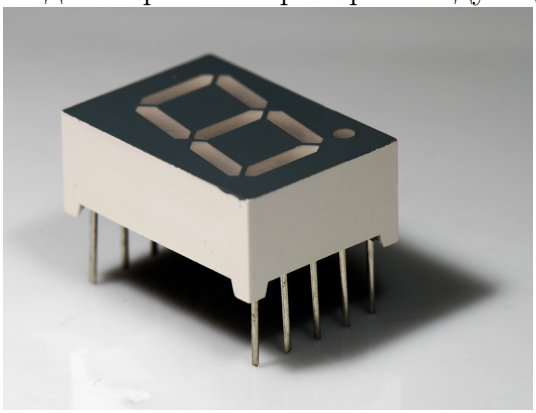
Цели работы

- Получить опыт работы с различными устройствами периферии
- Изучить детали программирования встраиваемых систем

1 Инструкция по работе с модулем

Семисегментный индикатор

Индикатор имеет примерно следующий вид:



Устройство имеет 10 выводов, центральный вывод в каждом ряду это общий анод/катод в зависимости от типа индикатора. Остальные выводы подключаются непосредственно к каждому из сегментов a,b,c,d,e,f,g,dp(точка).

На индикаторе присутствует маркировка, по которой можно найти на него описание, где будет указано какой вывод куда подключен. Для удобства можно составить таблицу с соответствиями выхода с буквой сегмента.

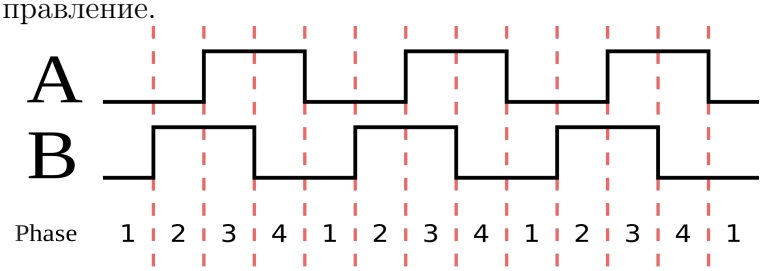
Теперь подключите индикатор к порту микроконтроллера. Для примера возьмите порт GPIOB и подсоедините индикатор по следующей схеме (не забудьте на каждый сегмент подключить последовательно резистор на каждый сегмент номиналом 200-300 Ом):

```
a - GPIOB PIN0
b - GPIOB PIN1
c - GPIOB PIN2
...
g - GPIOB PIN6
```

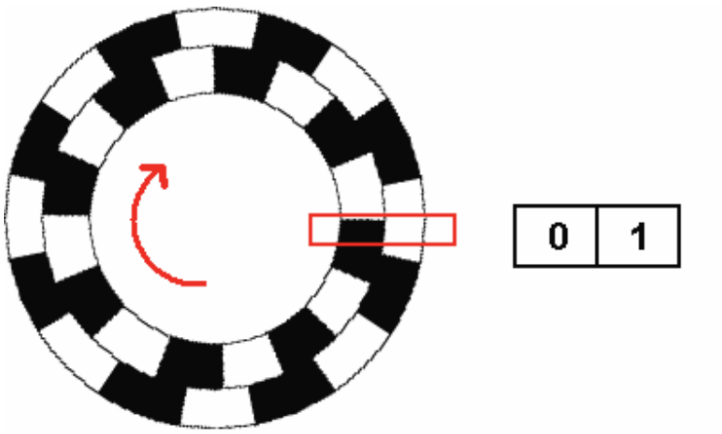
Энкодер

Инкрементальный энкодер это линейное, либо вращательное электромеханическое устройство, имеющее два выхода: A и B, на которых генерируются импульсы при вращении ручки энкодера.

Инкрементальный энкодер не показывает абсолютное положение ручки энкодера, он только показывает изменения положения, учитывая направление.



При вращении сгенерированные импульсы имеют разницу фаз в 90 градусов между выходами A и B. В любой момент времени разница в фазе будет либо положительная, либо отрицательная в зависимости от направления вращения (данное кодирование называется квадратурным). В таком случае, если вращение происходит по часовой стрелке, разница фаз будет +90 градусов. Если вращать энкодер против часовой стрелки, то разница фаз будет -90 градусов. Частота импульсов прямопропорциональна скорости вращения энкодера. На рисунке ниже показана работа роторного инкрементального энкодера.



В механическом инкрементальном энкодере используются скользящие контакты для генерации выходного сигнала. Внутреннее кольцо контактов подключено к выходу A, а внешнее к выходу B. При повороте ручки скользящий контакт по радиусу замыкает контакты на кольцах на землю, либо на питание в зависимости от того, куда подключен средний вывод. Для определенности положим, что средний вывод подключен к питанию, а выходы A и B подтянуты через резистор к земле. При замыкании контактов состояние выводов A и B будет переходить в лог. единицу.

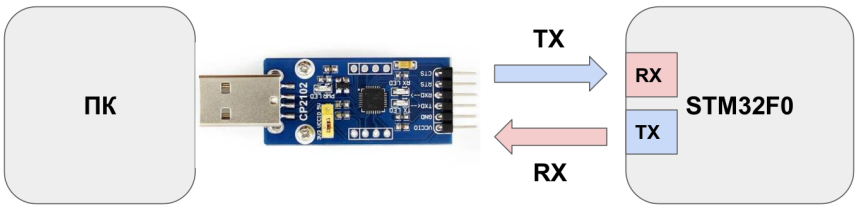
Для декодирования входной последовательности импульсов в направление вращения используется квадратурный декодер. Суть квадратурного декодера состоит в том, чтобы каждый раз опрашивать текущее состояние выходов A и B, далее сохранять его вместе с предыдущем состоянием и предпринимать какие-то действия на основе этих данных. Если посмотреть на выходную диаграмму энкодера, у него есть некоторый набор разрешенных состояний.

USART

Данное устройство должно принимать команду с параметрами с компьютера и выполнять определенное действие.

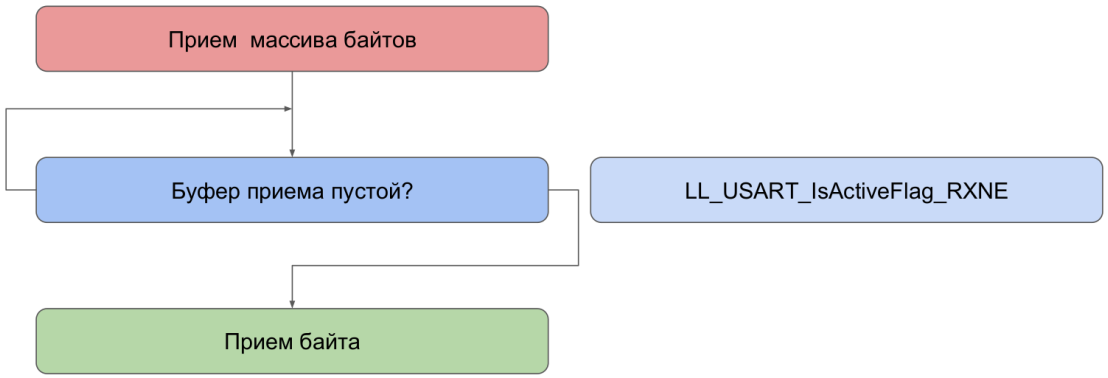
USART в STM32

- Полнодуплексный асинхронный способ связи
- Скорость (baudrate) до 6 Мбит/сек
- Автоматическое определение скорости канала
- Настраиваемая длина пакета (7, 8 or 9 bits)
- Настраиваемый порядок данных (самый старший бит (MSB) или самый младший бит (LSB))
- 1 или 2 бита на стоп-бит
- Возможность полудуплексного способа связи по одной линии
- Синхронный режим работы



Universal Synchronous/Asynchronous Receiver/Transmitter

USART. Передача и прием данных



Code

Представим фрагменты кода для прошивки микроконтроллера.

```
#include <stdio.h>
#include <stdlib.h>
#include "stm32f0xx_ll_rcc.h"
#include "stm32f0xx_ll_system.h"
#include "stm32f0xx_ll_bus.h"
#include "stm32f0xx_ll_gpio.h"
#include "stm32f0xx_ll_exti.h"
#include "stm32f0xx_ll_utils.h"
#include "stm32f0xx_ll_cortex.h"

#include "stm32f0xx_ll_usart.h"

#include "xprintf.h"

enum Const
{
        SYSCLK = 48000000,
        MY_FREQ = 200,
        DEBOUNCE = 50,
        NUM = 9990
};

int number = NUM;
int save_number_value = NUM;

/**
  * System Clock Configuration
  * The system Clock is configured as follow :
  *    System Clock source            = PLL (HSI/2)
  *    SYSCLK(Hz)                     = 48000000
  *    HCLK(Hz)                       = 48000000
  *    AHB Prescaler                  = 1
  *    APB1 Prescaler                 = 1
  *    HSI Frequency(Hz)              = 8000000
  *    PLLMUL                         = 12
  *    Flash Latency(WS)              = 1
  */
```

```c
static void rcc_config(void)
{
    /* Set FLASH latency */
    LL_FLASH_SetLatency(LL_FLASH_LATENCY_1);

    /* Enable HSI and wait for activation*/
    LL_RCC_HSI_Enable();
    while (LL_RCC_HSI_IsReady() != 1);

    /* Main PLL configuration and activation */
    LL_RCC_PLL_ConfigDomain_SYS(LL_RCC_PLLSOURCE_HSI_DIV_2,
                                LL_RCC_PLL_MUL_12);

    LL_RCC_PLL_Enable();
    while (LL_RCC_PLL_IsReady() != 1);

    /* Sysclk activation on the main PLL */
    LL_RCC_SetAHBPrescaler(LL_RCC_SYSCLK_DIV_1);
    LL_RCC_SetSysClkSource(LL_RCC_SYS_CLKSOURCE_PLL);
    while (LL_RCC_GetSysClkSource() != LL_RCC_SYS_CLKSOURCE_STATUS_PLL);

    /* Set APB1 prescaler */
    LL_RCC_SetAPB1Prescaler(LL_RCC_APB1_DIV_1);

    /* Update CMSIS variable (which can be updated also
     * through SystemCoreClockUpdate function) */
    SystemCoreClock = 48000000;
}

static void gpio_config(void)
{
        LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOC);
        LL_GPIO_SetPinMode(GPIOC, LL_GPIO_PIN_8, LL_GPIO_MODE_OUTPUT);
        LL_GPIO_SetPinMode(GPIOC, LL_GPIO_PIN_9, LL_GPIO_MODE_OUTPUT);

        LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOA);
        LL_GPIO_SetPinMode(GPIOA, LL_GPIO_PIN_8, LL_GPIO_MODE_INPUT);
        LL_GPIO_SetPinPull(GPIOA, LL_GPIO_PIN_0, LL_GPIO_PULL_DOWN);
        LL_GPIO_SetPinPull(GPIOA, LL_GPIO_PIN_1, LL_GPIO_PULL_DOWN); //
        LL_GPIO_SetPinMode(GPIOA, LL_GPIO_PIN_11, LL_GPIO_MODE_OUTPUT);
        LL_GPIO_SetPinMode(GPIOA, LL_GPIO_PIN_12, LL_GPIO_MODE_OUTPUT);
        LL_GPIO_SetPinMode(GPIOA, LL_GPIO_PIN_13, LL_GPIO_MODE_OUTPUT);
        LL_GPIO_SetPinMode(GPIOA, LL_GPIO_PIN_14, LL_GPIO_MODE_OUTPUT);

        LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOB);
        LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_0, LL_GPIO_MODE_OUTPUT);
        LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_1, LL_GPIO_MODE_OUTPUT);
        LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_2, LL_GPIO_MODE_OUTPUT);
        LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_3, LL_GPIO_MODE_OUTPUT);
        LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_4, LL_GPIO_MODE_OUTPUT);
        LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_5, LL_GPIO_MODE_OUTPUT);
        LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_6, LL_GPIO_MODE_OUTPUT);
        LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_7, LL_GPIO_MODE_OUTPUT);
```

```c
static void usart_config(void)
{
    /*
     * Setting USART pins
     */
    LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOA);
    //USART1_TX
    LL_GPIO_SetPinMode(GPIOA, LL_GPIO_PIN_9, LL_GPIO_MODE_ALTERNATE);
    LL_GPIO_SetAFPin_8_15(GPIOA, LL_GPIO_PIN_9, LL_GPIO_AF_1);
    LL_GPIO_SetPinSpeed(GPIOA, LL_GPIO_PIN_9, LL_GPIO_SPEED_FREQ_HIGH);
    //USART1_RX
    LL_GPIO_SetPinMode(GPIOA, LL_GPIO_PIN_10, LL_GPIO_MODE_ALTERNATE);
    LL_GPIO_SetAFPin_8_15(GPIOA, LL_GPIO_PIN_10, LL_GPIO_AF_1);
    LL_GPIO_SetPinSpeed(GPIOA, LL_GPIO_PIN_10, LL_GPIO_SPEED_FREQ_HIGH);
    /*
     * USART Set clock source
     */
    LL_APB1_GRP2_EnableClock(LL_APB1_GRP2_PERIPH_USART1);
    LL_RCC_SetUSARTClockSource(LL_RCC_USART1_CLKSOURCE_PCLK1);
    /*
     * USART Setting
     */
    LL_USART_SetTransferDirection(USART1, LL_USART_DIRECTION_TX_RX);
    LL_USART_SetParity(USART1, LL_USART_PARITY_NONE);
    LL_USART_SetDataWidth(USART1, LL_USART_DATAWIDTH_8B);
    LL_USART_SetStopBitsLength(USART1, LL_USART_STOPBITS_1);
    LL_USART_SetTransferBitOrder(USART1, LL_USART_BITORDER_LSBFIRST);
    LL_USART_SetBaudRate(USART1, SystemCoreClock,
                         LL_USART_OVERSAMPLING_16, 115200);
    /*
     * USART turn on
     */
    LL_USART_Enable(USART1);
    while (!(LL_USART_IsActiveFlag_TEACK(USART1) &&
             LL_USART_IsActiveFlag_REACK(USART1)));
    return;
}

static char usart_getc(void)
{
    while (!(LL_USART_IsActiveFlag_RXNE(USART1)));
    return LL_USART_ReceiveData8(USART1);
}

static void usart_putc(char symbol)
{
    while (!LL_USART_IsActiveFlag_TXE(USART1));
    LL_USART_TransmitData8(USART1, symbol);
    while (!LL_USART_IsActiveFlag_TC(USART1));
}
```