САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ им. ПЕТРА ВЕЛИКОГО

Институт компьютерных наук и технологий
Высшая школа искусственного интеллекта
Направление 3.02.01 Математика и Компьютерные науки

Отчёт по дисциплине Программирование микроконтроллеров.

Лабораторная работа № 8.

Работу выполнила:
Гусева С.А.
студент группы 3530201/10001
Проверила:
Вербова Н. М.

Санкт-Петербург - 2023 г.

Тема:

Использование таймеров STM32F200 для генерирования сложных форм волн.

Цель:

Ознакомиться с основными приемами изучения предметной области программируемой задачи. Ознакомиться с генерированием модулированных колебаний. Закрепить навыки работы с низкоуровневыми библиотеками и промежуточным программным обеспечением микроконтроллера. Закрепить навыки отладки программ.

Постановка задачи:

Используя библиотеки Keil µVision5, разработать программу для генерирования одного из несущих колебаний для частотной манипуляции. Соответствие частот логическим уровням цифровых данных, Гц (вар I):

Логический уровень цифровых данных	I вариант	II вариант	III вариант	IV вариант
"1"	980	1070	1650	2025
"0"	1180	1270	1850	2225

Теоретические данные:

В данной лабораторной работе будет рассмотрена частотная манипуляция. Её примером является техника FSK31, которая используется в радиомодемах для обеспечения цифровой любительской радиосвязи в УКВ диапазоне.

Модем (модулятор/демодулятор), осуществляет транспонирование (перенос) спектра передаваемых электрических сигналов из одного частотного диапазона в другой, что позволяет передавать низкочастотные информационные сигналы по высокочастотным радиоканалам.

Оснащение компьютеров такими устройствами позволяет осуществлять связь между ними.

Следует отметить, что любительская радиосвязь – это *строго регламентированная сфера* общественного пользования и непосредственное подключение к ней должно осуществляться в соответствии с установленными требованиями к любительской радиосвязи.

В настоящее время используются различные режимы связи: данные могут передаваться по радиоканалу только в одном направлении, в двух направлениях, но поочередно в том и другом, или в обоих направлениях одновременно.

Симплекс. В этом режиме передача осуществляется только в одном направлении. Примером может служить радио- и телевещание, пейджинговая связь.

Полудуплекс. В полудуплексном режиме данные могут пересылаться в любом из направлений, но не одновременно в обоих.

Дуплекс. Для представления логических "1" и "0" используется пара тональных сигналов. При частотном уплотнении две пары тональных сигналов разделены между собой по частоте, что дает возможность осуществлять дуплексную связь, т.е. передавать данные в обоих направлениях. Один из модемов системы называют модемом исходящей связи, а именно тот, который начинает передачу данных по линии связи. Модем исходящей связи передает данные с использованием пары исходящих тональных сигналов, которые по принятым соглашениям находятся в нижнем поддиапазоне частот. Модем на входящем конце линии принимает эту пару сигналов, а передачу ведет с помощью тональной пары, находящейся в верхнем частотном поддиапазоне.

Эхоплекс. В большинстве компьютерных систем, обменивающихся данными через радиоканал, используется метод эхоплекса, или эхообразной передачи, когда передаваемая определенная символьная информация посылается обратно отправителю с целью контроля ошибок.

Для того чтобы передать цифровую информацию, аналоговый сигнал модулируется. Существует три основных метода модуляции синусоидального колебания: амплитудная, частотная и фазовая. Используя эти виды модуляции, можно кодировать информацию различным образом. В модемах наиболее широко применяются амплитудная модуляция, частотная манипуляция дифференциальная частотная манипуляция, фазовая манипуляция и комбинированная амплитудно-фазовая модуляция, которая называется квадратурной амплитудной модуляцией.

При **частотной** манипуляции кодирование информации осуществляется двумя частотами: логической единице присваивается одна частота, а логическому нулю – вторая.

Такой вид модуляции можно получить и с помощью ШИМ, правда для получения аналогового сигнала потребуется пропустить ШИМ сигнал через аналоговый фильтр.

И так, пусть нам требуется сгенерировать синусоидальную волну с некоторой частотой. Наиболее простым способом реализации этого является использование гибкой техники прямого цифрового синтеза. Основной концепцией здесь будет вычисление фазы синусоидальной волны для каждого отсчета. На практике это означает, что нам потребуется независимый генератор тактирования отсчетов, работающий, например, на частоте 100 кГц. Предположим, что нам необходимо генерировать синусоидальную волну с частотой 1 кГц или, что эквивалентно угловой частоте в 2000π рад/сек. Наш период дискретизации равен 10 микросекундам, таким образом, с каждым отсчетом фаза увеличивается на $\varphi = 2000\pi \times 10^{-6} = 20\pi \times 10^{-3}$ рад.

В общем случае для произвольной частоты дискретизации f_s и требуемой частоты f_0 за период дискретизации фаза должна увеличиваться на

$$R = 2\pi \frac{f_0}{f_s}$$

для каждого отсчета. Поскольку мы будем использовать числа с фиксированной запятой, а не действительные (с плавающей запятой), мы поставим в соответствие величине 2π максимально возможное число различных значений, которое можно представить 32 разрядным двоичным словом, т.е. 232. Если бы мы использовали 16 разрядный аккумулятор фазы, то естественно заменить 232 на 216 и т.п. Таким образом, единственным, что нам нужно будет вычислить это только значение

$$R = 2^{32} \frac{f_0}{f_s}.$$

Код программы:

```
#include "math.h"
#include "stm32f2xx_hal.h"            // Keil::Device:STM32Cube HAL:Common

uint16_t sinetable[] = {

127,130,133,136,139,143,146,149,152,155,158,161,164,167,170,173,176,178,181,
 184,187,190,192,195,198,200,203,205,208,210,212,215,217,219,221,223,225,227,
 229,231,233,234,236,238,239,240,242,243,244,245,247,248,249,249,250,251,252,
 252,253,253,253,254,254,254,254,254,254,254,253,253,253,252,252,251,250,249,
 249,248,247,245,244,243,242,240,239,238,236,234,233,231,229,227,225,223,221,
 219,217,215,212,210,208,205,203,200,198,195,192,190,187,184,181,178,176,173,
 170,167,164,161,158,155,152,149,146,143,139,136,133,130,127,124,121,118,115,
 111,108,105,102,99,96,93,90,87,84,81,78,76,73,70,67,64,62,59,56,54,51,49,46,
44,42,39,37,35,33,31,29,25,23,21,20,18,16,15,14,12,11,10,9,7,6,5,5,4,3,2,
 2,1,1,1,0,0,0,0,0,0,0,0,1,1,1,2,2,3,4,5,5,6,7,9,10,11,12,14,15,16,18,20,21,23,
 25,27,29,31,33,35,37,39,42,44,46,49,51,54,56,59,62,64,67,70,73,76,78,81,84,
 87,90,93,96,99,102,105,108,111,115,118,121,124};

#define TIMER_PERIOD 1000
#define WAVE_ZERO 1180
#define WAVE_ONE 980

GPIO_InitTypeDef  GPIO_InitStruct;
TIM_HandleTypeDef htim;

uint32_t R = (256 *(WAVE_ONE/WAVE_ZERO));
```

```
uint16_t pulse_width = 128;
uint32_t phase_accumulator = 0;
uint8_t angle = 0;


void PWM_SetDC(uint16_t channel,uint16_t dutycycle)
{
  if (channel == 1)
  {
    TIM2->CCR1 = dutycycle;
  }
  else if (channel == 2)
  {
    TIM2->CCR2 = dutycycle;
  }
}


void TIM2_IRQHandler(void)
{
 HAL_GPIO_WritePin(GPIOG, GPIO_PIN_7, GPIO_PIN_SET);
 PWM_SetDC(1,pulse_width);
 PWM_SetDC(2,pulse_width);

 // Calculate a new pulse width
 phase_accumulator += R;
 angle = phase_accumulator;
 pulse_width = sinetable[angle];
 TIM2->SR = ~(htim.State);
 HAL_GPIO_WritePin(GPIOG, GPIO_PIN_7, GPIO_PIN_RESET);
}


void InitializeLED()
        {
                //Включение тактирования порта G
                RCC->AHB1ENR |= RCC_AHB1ENR_GPIOGEN;

                        /* GPIO base configuration */
                 GPIO_InitStruct.Pin |= (GPIO_PIN_7);
                 GPIO_InitStruct.Pin |= (GPIO_PIN_8);
                 GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
                GPIO_InitStruct.Speed = GPIO_SPEED_LOW;
                HAL_GPIO_Init(GPIOG, &GPIO_InitStruct);
```

```
		HAL_GPIO_WritePin(GPIOG, GPIO_PIN_7, GPIO_PIN_RESET);
		HAL_GPIO_WritePin(GPIOG, GPIO_PIN_8, GPIO_PIN_RESET);
	}


void InitializeTimer()
 {
		//Включение тактирования таймера
		RCC->APB1ENR |= RCC_APB1ENR_TIM2EN ;

			/* Time base configuration */
			htim.Instance = TIM2;

			htim.Init.Period = TIMER_PERIOD;
			htim.Init.Prescaler = 40000;
			htim.Init.ClockDivision = 0;
			htim.Init.RepetitionCounter = 0;
			htim.Init.CounterMode = TIM_COUNTERMODE_UP;
			HAL_TIM_Base_Init(&htim);

			/* Enable TIM peripheral counter */
			HAL_TIM_Base_Start(& htim);
			HAL_TIM_Base_Start_IT(& htim);
 }

void EnableTimerInterrupt()
	{
   NVIC_SetPriorityGrouping(0);
   NVIC_SetPriority(TIM2_IRQn,1);
   NVIC_EnableIRQ(TIM2_IRQn);
	}

int main()
{
		InitializeLED();
		InitializeTimer();

		for (;;)
		{
			TIM2_IRQHandler();
		}
}
```

Алгоритм программы:

Сперва подключаем стандартную библиотеку stm, а также дополнительно подключаем математическую библиотеку для использования функции возведения в степень.

В глобальных переменных инициализируем массив sinetable, в котором хранятся значения для модуляции синусоиды.

С помощью #define определяем период таймера, а также уровни логического нуля и единицы и данного нам варианта (I вар), далее инициализируем еще несколько глобальных переменных.

PWM_SetDC: установка DC (dooty cicle) доя PWM (ШИМ).
CCR1 Регистр фиксации/сравнения 1
Регистр является буферизируемым, если установлен бит OC1PE в регистре TIMx_CCMR1.
Address offset: 0x34
Reset value: 0x0000

CCR1 - Capture/Compare 1 value:
Если канал сконфигурирован в режиме выхода
CCR1 содержит значение для загрузки в свой действующий (активный) регистр, т.е. содержит предзагружаемое значение. Значение помещается сразу в активный регистр, если не используется предзагрузка. Настройка буферизации регистра осуществляется с помощью бита OC1PE в регистре TIMx_CCMR1. В случае использования буферизации, предзагружаемое значение копируется в активный регистр по событию обновления. Значение в активном регистре используется для сравнения с текущим значением счётчика TIMx_CNT, результат сравнения используется для формирования выходного сигнала канала.
Если канал сконфигурирован в режиме входа
CCR1 содержит значение счётчика, переданного в этот регистр при возникновении последнего события фиксации в данном канале (IC1, input capture 1 event).

Регистр фиксации/сравнения 2
Регистр является буферизируемым, если установлен бит OC2PE в регистре TIMx_CCMR1.

Address offset: 0x38
Reset value: 0x0000

Регистр фиксации/сравнения канала 2, по функциям аналогичен TIMx_CCR1 (смотрите выше).

Устанавливаем значение dutycicle - рабочего цикла, в зависимости от канала.

TIM2_IRQHandler – функция, настраивающая прерывания для 2го стандартного таймера. Сердцем программы является таймер TIM2 генерирующий прерывания через равные интервалы времени. Светодиод PG7 индицирует попадание программы в обработчик прерывания (это может помочь при отладке программы). Два канала таймера TIM3 работают в режиме альтернативной функции и генерируют импульсы заданной ширины. Значение ширины импульса, например, для первого канала TIM3, загружается в его регистр CCR1 в строке: PWM_SetDC(1,pulse_width). Аккумулятор фазы 8 бит в длину. Находящееся в нем значение используется затем как индекс для поисковой таблицы sinetable.

InitializeLED – функция, в которой мы включаем тактирование порта G, настраиваем базовую конфигурацию для PG8: режим работы — выход, скорость — низкая и тд.

*void HAL_GPIO_WritePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)*

Устанавливает состояние вывода.

- *GPIOx* – выбор порта (GPIOA, GPIOB, GPIOC …).
- *Pin* – номер вывода (GPIO_PIN_0 … GPIO_PIN_15).
- *PinState* – состояние вывода:
  - *GPIO_PIN_SET* – высокий уровень;
  - *GPIO_PIN_RESET* - низкий уровень.

Функция работает с регистрами битовых операций портов. Нет опасности, что прерывание может вклиниться в момент выполнения операции. Как это может случиться при использовании регистра вывода данных в режиме чтение, модификация, запись.

Устанавливаем низкий уровень для пинов 7 и 8.

Функция InitializeTimer, в которой мы включаем тактирование таймера TIM2, устанавливаем его конфигурацию, разрешаем периферийное тактирование (счетчик).

Функция EnableTimerInterrupt, в которой мы устанавливаем приоритеты прерываний и разрешаем прерывания TIM2.

В основном теле функции main мы вызываем функции InitializeLED и InitializeTimer, и запускаем бесконечный цикл for, в котором вызываем функцию прерывание таймера TIM2.

Работа с осциллографом:

Период формы волны, частота сигнала представлены на рисунке 1.

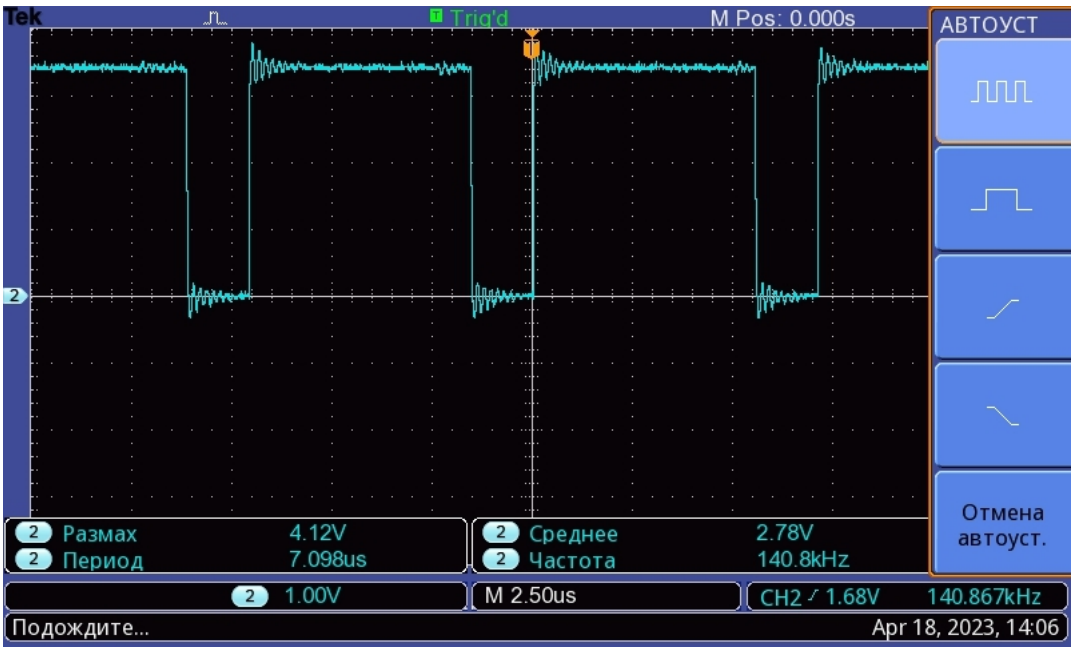


рис.1

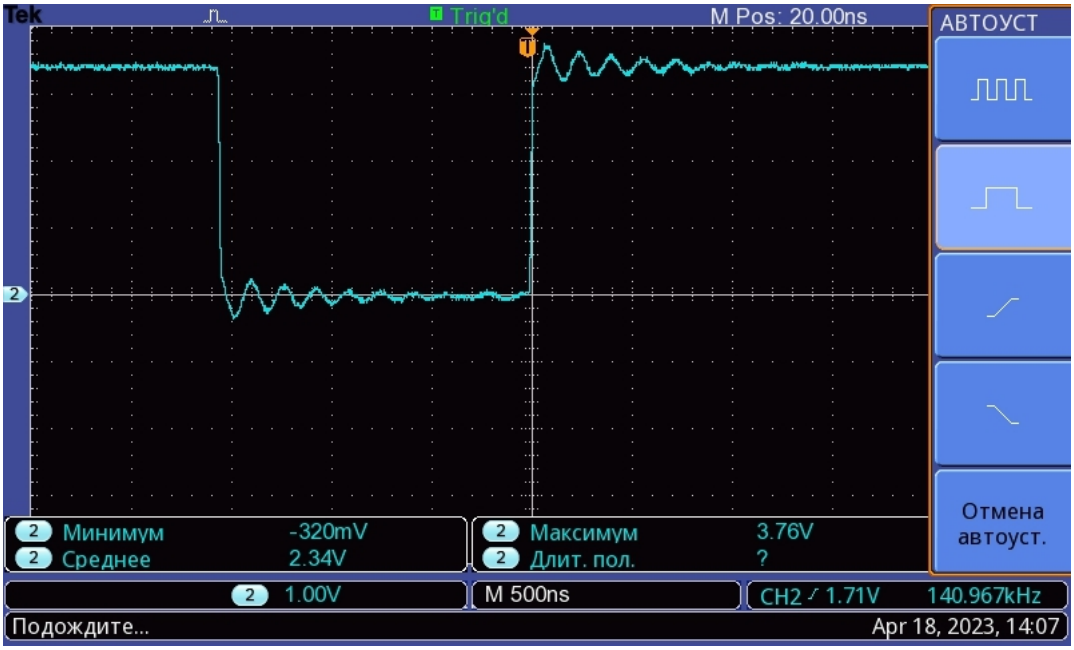
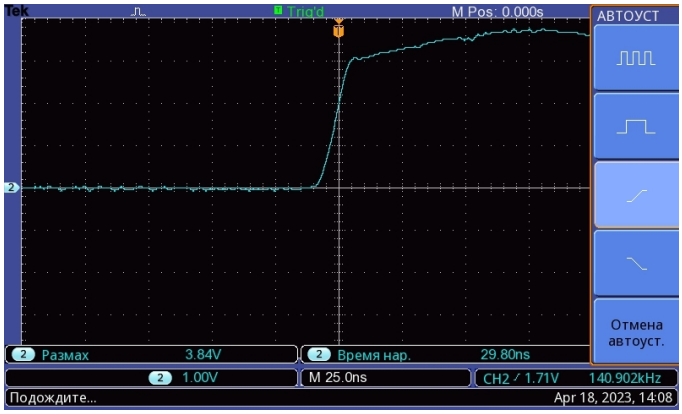


рис.2



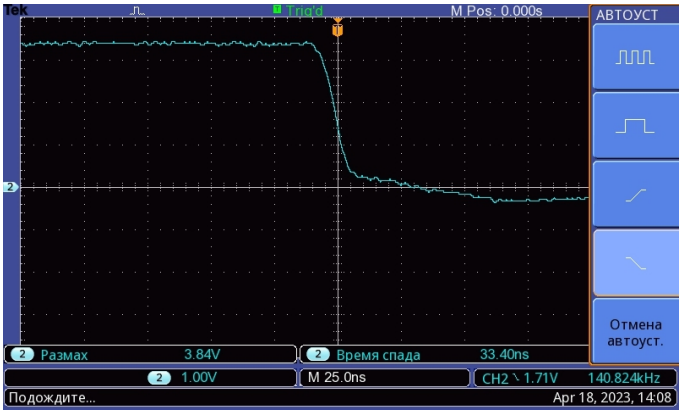


рис.3 **рис.4**

Время нарастания и спада форы волны представлено на рисунках 3 и 4 соответственно.

Вывод:

Мы ознакомились с генерированием модулированных колебаний. Закрепили навыки работы с низкоуровневыми библиотеками и промежуточным программным обеспечением микроконтроллера.