САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. ПЕТРА ВЕЛИКОГО

Институт компьютерных наук и кибербезопасности

Высшая школа технологий искусственного интеллекта

Отчет по лабораторной работе

Использование цифро-аналогового преобразователя (ЦАП) для формирования микроконтроллером заданной формы волны.

Работу выполнила:

Мартыненко А.П.

студент группы 5130201/30003

Проверила:

Вербова Н.М.

Санкт-Петербург - 2025 г.

1. Цель работы:

Ознакомиться с архитектурой низкоуровневых библиотек и промежуточного программного обеспечения микроконтроллера. Закрепить навыки работы с осциллографом и оценочной платой MCBSTM32F200 в качестве измерительного генератора.

1. Задание на лабораторную работу:

Используя библиотеки Keil μVision5, разработать программу для микроконтроллера (МК) STM32F200, которая выдает на выходе ЦАП заданный уровень напряжения. Изучить и ввести программу, предназначенную для генерирования на выходе ЦАП микроконтроллера (МК) STM32F200 периодической волны напряжения заданной формы. Модифицировать данную программу так, чтобы она выводила сигнал с заданными амплитудными и временными характеристиками (размахом и периодом).

1. Алгоритм программы

А (MyDAC)

#include "stm32f2xx\_hal.h"

//подключаем частные переменные

DAC\_HandleTypeDef DacHandle; // для определения структуры обработчика ЦАП

DAC\_ChannelConfTypeDef sConfig; // для определения постоянной структуры канала конфигурации ЦАП

RCC\_ClkInitTypeDef RCC\_ClkInitStruct; // для определения структуры конфигурации тактирования шин AHB и APB системы RCC

RCC\_OscInitTypeDef RCC\_OscInitStruct; // для определения структуры конфигурации внешнего/внутреннего генератора (HSE, HSI, LSE и LSI) RCC

GPIO\_InitTypeDef GPIO\_InitStruct; // для определения структуры инициализации портов ввода-вывода общего назначения GPIO

//Функция конфигурирования системы тактирования

void SystemClock\_Config(void)

{

/\* Enable HSE Oscillator and activate PLL with HSE as source \*/

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSE;

RCC\_OscInitStruct.HSEState = RCC\_HSE\_ON;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_ON;

RCC\_OscInitStruct.PLL.PLLSource = RCC\_PLLSOURCE\_HSE;

RCC\_OscInitStruct.PLL.PLLM = 25;

RCC\_OscInitStruct.PLL.PLLN = 240;

RCC\_OscInitStruct.PLL.PLLP = RCC\_PLLP\_DIV2;

RCC\_OscInitStruct.PLL.PLLQ = 5;

HAL\_RCC\_OscConfig(& RCC\_OscInitStruct);

/\* Select PLL as system clock source and configure the HCLK, PCLK1 and PCLK2 clocks dividers \*/

RCC\_ClkInitStruct.ClockType = (RCC\_CLOCKTYPE\_SYSCLK | RCC\_CLOCKTYPE\_HCLK | RCC\_CLOCKTYPE\_PCLK1 | RCC\_CLOCKTYPE\_PCLK2);

RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_PLLCLK;

RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV4;

RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV2;

HAL\_RCC\_ClockConfig(& RCC\_ClkInitStruct, FLASH\_LATENCY\_3);

}

//Функция конфигурирования ЦАП

void HAL\_DAC\_MspInit(DAC\_HandleTypeDef \*hdac)

{

/\*##-1- Enable peripherals and GPIO Clocks \*/

/\* Enable GPIO clock \*/

\_\_GPIOA\_CLK\_ENABLE();

/\* DAC Periph clock enable \*/

\_\_DAC\_CLK\_ENABLE();

/\*##-2- Configure peripheral GPIO \*/

/\* DAC Channel1 GPIO pin configuration \*/

GPIO\_InitStruct.Pin = GPIO\_PIN\_4;

GPIO\_InitStruct.Mode = GPIO\_MODE\_ANALOG;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

HAL\_GPIO\_Init(GPIOA, & GPIO\_InitStruct);

}

int main(void)

{

/\* STM32F2xx HAL library initialization:

- Configure the Flash prefetch, instruction and Data caches

- Configure the Systick to generate an interrupt each 1 msec

- Set NVIC Group Priority to 4

- Global MSP (MCU Support Package) initialization

\*/

HAL\_Init();

/\* Configure the system clock to have a system clock = 120 MHz \*/

SystemClock\_Config();

/\*##-1- Configure the DAC peripheral \*/

DacHandle.Instance = DAC;

HAL\_DAC\_Init(& DacHandle); //инициализация DAC

/\*##-2- Configure DAC channel1 \*/

sConfig.DAC\_Trigger = DAC\_TRIGGER\_NONE; // Отключение триггера

sConfig.DAC\_OutputBuffer = DAC\_OUTPUTBUFFER\_DISABLE; // Отключение выходного буфера

HAL\_DAC\_ConfigChannel(& DacHandle, & sConfig, DAC\_CHANNEL\_1);

/\*##-3- Set DAC Channel1 DHR register \*/

HAL\_DAC\_SetValue(& DacHandle, DAC\_CHANNEL\_1, DAC\_ALIGN\_12B\_R, 4095); //Установка количества уровней квантования (4095)

/\*##-4- Enable DAC Channel1 \*/

HAL\_DAC\_Start(& DacHandle, DAC\_CHANNEL\_1);

/\* Infinite loop \*/

while (1)

{

}

}

Данная программа генерирует постоянный сигнал на выходе DAC (ЦАП) с максимальным выходным напряжением (3В). На осциллографе это будет представлено как горизонтальная линия (рис. 1). Чтобы изменить значение напряжения, необходимо изменить количество уровней квантования. На рис.2 количество уровней квантования подобрано так, что выходное напряжение сигнала равно 2 В.

B (DACWave)

#include "stm32f2xx\_hal.h"

/\* Private typedef \*/

/\* Private define \*/

/\* Definition for DAC clock resources \*/

#define DACx\_CHANNEL1\_GPIO\_CLK\_ENABLE() \_\_GPIOA\_CLK\_ENABLE()

#define DMAx\_CLK\_ENABLE() \_\_DMA1\_CLK\_ENABLE()

/\* Definition for DACx Channel1 Pin \*/

#define DACx\_CHANNEL1\_PIN GPIO\_PIN\_4

#define DACx\_CHANNEL1\_GPIO\_PORT GPIOA

/\* Definition for DACx's Channel1 \*/

#define DACx\_CHANNEL1 DAC\_CHANNEL\_1

/\* Definition for DACx's DMA Channel1 \*/

#define DACx\_DMA\_CHANNEL1 DMA\_CHANNEL\_7

#define DACx\_DMA\_STREAM1 DMA1\_Stream5

/\* Private macro \*/

/\* Private variables \*/

DAC\_HandleTypeDef DacHandle;

static DAC\_ChannelConfTypeDef sConfig;

const uint8\_t Wave[32] = {0x0, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0xee, 0xdd, 0xcc, 0xbb, 0xaa, 0x99, 0x88, 0x77, 0x66, 0x55, 0x44, 0x33, 0x22, 0x11, 0x00};

/\* Private function prototypes \*/

static void DAC\_Ch1\_WaveConfig(void);

static void TIM6\_Config(void);

static void SystemClock\_Config(void);

/\* Private functions \*/

/\*\*

\* @brief DAC MSP De-Initialization

\* This function frees the hardware resources:

\* - Disable the Peripheral's clock

\* - Revert GPIO to their default state

\* @param hadc: DAC handle pointer

\* @retval None

\*/

void HAL\_DAC\_MspInit(DAC\_HandleTypeDef\* hdac)

{

GPIO\_InitTypeDef GPIO\_InitStruct;

static DMA\_HandleTypeDef hdma\_dac1;

/\*##-1- Enable peripherals and GPIO Clocks \*/

/\* DAC Periph clock enable \*/

\_\_DAC\_CLK\_ENABLE();

/\* Enable GPIO clock \*/

DACx\_CHANNEL1\_GPIO\_CLK\_ENABLE();

/\* DMA1 clock enable \*/

DMAx\_CLK\_ENABLE();

/\*##-2- Configure peripheral GPIO \*/

/\* DAC Channel1 GPIO pin configuration \*/

GPIO\_InitStruct.Pin = DACx\_CHANNEL1\_PIN;

GPIO\_InitStruct.Mode = GPIO\_MODE\_ANALOG;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

HAL\_GPIO\_Init(DACx\_CHANNEL1\_GPIO\_PORT, & GPIO\_InitStruct);

/\*##-3- Configure the DMA streams \*/

/\* Set the parameters to be configured for Channel1\*/

hdma\_dac1.Instance = DACx\_DMA\_STREAM1;

hdma\_dac1.Init.Channel = DACx\_DMA\_CHANNEL1;

hdma\_dac1.Init.Direction = DMA\_MEMORY\_TO\_PERIPH;

hdma\_dac1.Init.PeriphInc = DMA\_PINC\_DISABLE;

hdma\_dac1.Init.MemInc = DMA\_MINC\_ENABLE;

hdma\_dac1.Init.PeriphDataAlignment = DMA\_PDATAALIGN\_BYTE;

hdma\_dac1.Init.MemDataAlignment = DMA\_MDATAALIGN\_BYTE;

hdma\_dac1.Init.Mode = DMA\_CIRCULAR;

hdma\_dac1.Init.Priority = DMA\_PRIORITY\_HIGH;

hdma\_dac1.Init.FIFOMode = DMA\_FIFOMODE\_DISABLE;

hdma\_dac1.Init.FIFOThreshold = DMA\_FIFO\_THRESHOLD\_HALFFULL;

hdma\_dac1.Init.MemBurst = DMA\_MBURST\_SINGLE;

hdma\_dac1.Init.PeriphBurst = DMA\_PBURST\_SINGLE;

HAL\_DMA\_Init(& hdma\_dac1);

/\* Associate the initialized DMA handle to the the DAC handle \*/

\_\_HAL\_LINKDMA(hdac, DMA\_Handle1, hdma\_dac1);

}

/\*\*

\* @brief TIM MSP Initialization

\* This function configures the hardware resources:

\* - Peripheral's clock enable

\* - Peripheral's GPIO Configuration

\* @param htim: TIM handle pointer

\* @retval None

\*/

void HAL\_TIM\_Base\_MspInit(TIM\_HandleTypeDef\* htim)

{

/\* TIM6 Periph clock enable \*/

\_\_TIM6\_CLK\_ENABLE();

}

/\*\*

\* @brief Main program.

\* @param None

\* @retval None

\*/

int main(void)

{

/\* STM32F2xx HAL library initialization:

- Configure the Flash prefetch, instruction and Data caches

- Configure the Systick to generate an interrupt each 1 msec

- Set NVIC Group Priority to 4

- Global MSP (MCU Support Package) initialization

\*/

HAL\_Init();

/\* Configure the system clock to have a system clock = 120 MHz \*/

SystemClock\_Config();

/\*##-1- Configure the DAC peripheral \*/

DacHandle.Instance = DAC;

/\*##-2- Configure the TIM peripheral \*/

TIM6\_Config();

/\* Wave generator -------------------------------------------\*/

DAC\_Ch1\_WaveConfig();

/\* Infinite loop \*/

while (1) {}

}

/\*\*

\* @brief System Clock Configuration

\* The system Clock is configured as follow :

\* System Clock source = PLL (HSE)

\* SYSCLK(Hz) = 120000000

\* HCLK(Hz) = 120000000

\* AHB Prescaler = 1

\* APB1 Prescaler = 4

\* APB2 Prescaler = 2

\* HSE Frequency(Hz) = 25000000

\* PLL\_M = 25

\* PLL\_N = 240

\* PLL\_P = 2

\* PLL\_Q = 5

\* VDD(V) = 3.3

\* Flash Latency(WS) = 3

\* @param None

\* @retval None

\*/

static void SystemClock\_Config(void)

{

RCC\_ClkInitTypeDef RCC\_ClkInitStruct;

RCC\_OscInitTypeDef RCC\_OscInitStruct;

/\* Enable HSE Oscillator and activate PLL with HSE as source \*/

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSE;

RCC\_OscInitStruct.HSEState = RCC\_HSE\_ON;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_ON;

RCC\_OscInitStruct.PLL.PLLSource = RCC\_PLLSOURCE\_HSE;

RCC\_OscInitStruct.PLL.PLLM = 25;

RCC\_OscInitStruct.PLL.PLLN = 240;

RCC\_OscInitStruct.PLL.PLLP = RCC\_PLLP\_DIV2;

RCC\_OscInitStruct.PLL.PLLQ = 5;

HAL\_RCC\_OscConfig(& RCC\_OscInitStruct);

/\* Select PLL as system clock source and configure the HCLK, PCLK1 and PCLK2 clocks dividers \*/

RCC\_ClkInitStruct.ClockType = (RCC\_CLOCKTYPE\_SYSCLK | RCC\_CLOCKTYPE\_HCLK | RCC\_CLOCKTYPE\_PCLK1 | RCC\_CLOCKTYPE\_PCLK2);

RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_PLLCLK;

RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV4;

RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV2;

HAL\_RCC\_ClockConfig(& RCC\_ClkInitStruct, FLASH\_LATENCY\_3);

}

static void DAC\_Ch1\_WaveConfig(void)

{

/\*##-1- Initialize the DAC peripheral \*/

HAL\_DAC\_Init(& DacHandle);

/\*##-2- DAC channel1 Configuration \*/

sConfig.DAC\_Trigger = DAC\_TRIGGER\_T6\_TRGO;

sConfig.DAC\_OutputBuffer = DAC\_OUTPUTBUFFER\_ENABLE;

HAL\_DAC\_ConfigChannel(& DacHandle, & sConfig, DACx\_CHANNEL1);

/\*##-3- Enable DAC Channel1 and associeted DMA \*/

HAL\_DAC\_Start\_DMA(& DacHandle, DACx\_CHANNEL1, (uint32\_t \*)Wave, 32, DAC\_ALIGN\_12B\_R);

/\*##-4- Enable DAC Channel1 \*/

HAL\_DAC\_Start(& DacHandle, DACx\_CHANNEL1);

/\*##-5- Set DAC channel1 DHR12RD register \*/

HAL\_DAC\_SetValue(& DacHandle, DACx\_CHANNEL1, DAC\_ALIGN\_12B\_R, 0x100);

}

/\*\*

\* @brief TIM6 Configuration

\* @note TIM6 configuration is based on APB1 frequency

\* @note TIM6 Update event occurs each TIM6CLK/256

\* @param None

\* @retval None

\*/

void TIM6\_Config(void)

{

static TIM\_HandleTypeDef htim;

TIM\_MasterConfigTypeDef MasterConfig;

/\*##-1- Configure the TIM peripheral \*/

/\* Time base configuration \*/

htim.Instance = TIM6;

htim.Init.Period = 0x7FF;

htim.Init.Prescaler = 0;

htim.Init.ClockDivision = 0;

htim.Init.CounterMode = TIM\_COUNTERMODE\_UP;

HAL\_TIM\_Base\_Init(& htim);

/\* TIM6 TRGO selection \*/

MasterConfig.MasterOutputTrigger = TIM\_TRGO\_UPDATE;

MasterConfig.MasterSlaveMode = TIM\_MASTERSLAVEMODE\_DISABLE;

HAL\_TIMEx\_MasterConfigSynchronization(& htim, & MasterConfig);

/\*##-2- Enable TIM peripheral counter \*/

HAL\_TIM\_Base\_Start(& htim);

}

Программа настраивает микроконтроллер для генерации ступенчатого сигнала на выходе DAC с использованием DMA (прямого доступа к памяти) и таймера TIM6. Сначала программа инициализирует системное тактирование, настраивая HSE (высокочастотный внешний генератор) и PLL (фазовая автоподстройка частоты) для работы на частоте 120 МГц. Затем настраивается GPIO для выхода DAC (пин PA4) в аналоговом режиме. Программа конфигурирует DMA для передачи данных из массива Wave в регистр DAC в циклическом режиме, что позволяет автоматически обновлять значение на выходе DAC без участия процессора. Таймер TIM6 настраивается для генерации триггера, который инициирует передачу данных через DMA. В результате на выходе DAC формируется ступенчатый сигнал, соответствующий значениям из массива Wave (рис. 3). Для формирования синусоидального сигнала был увеличен размер массива Wave и уменьшен интервал между его значениями (рис.4).

Работа с осциллографом

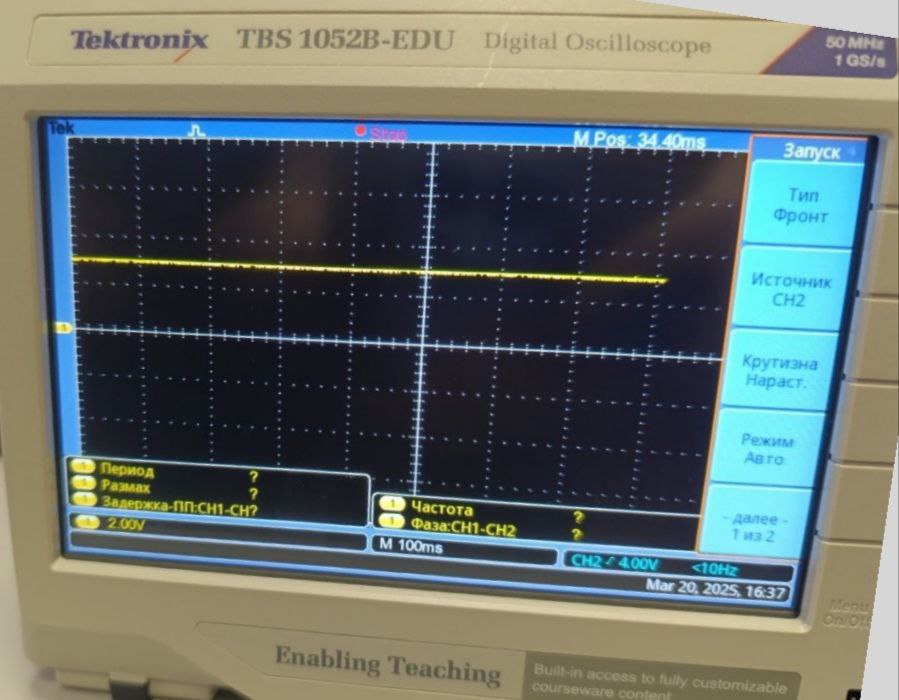


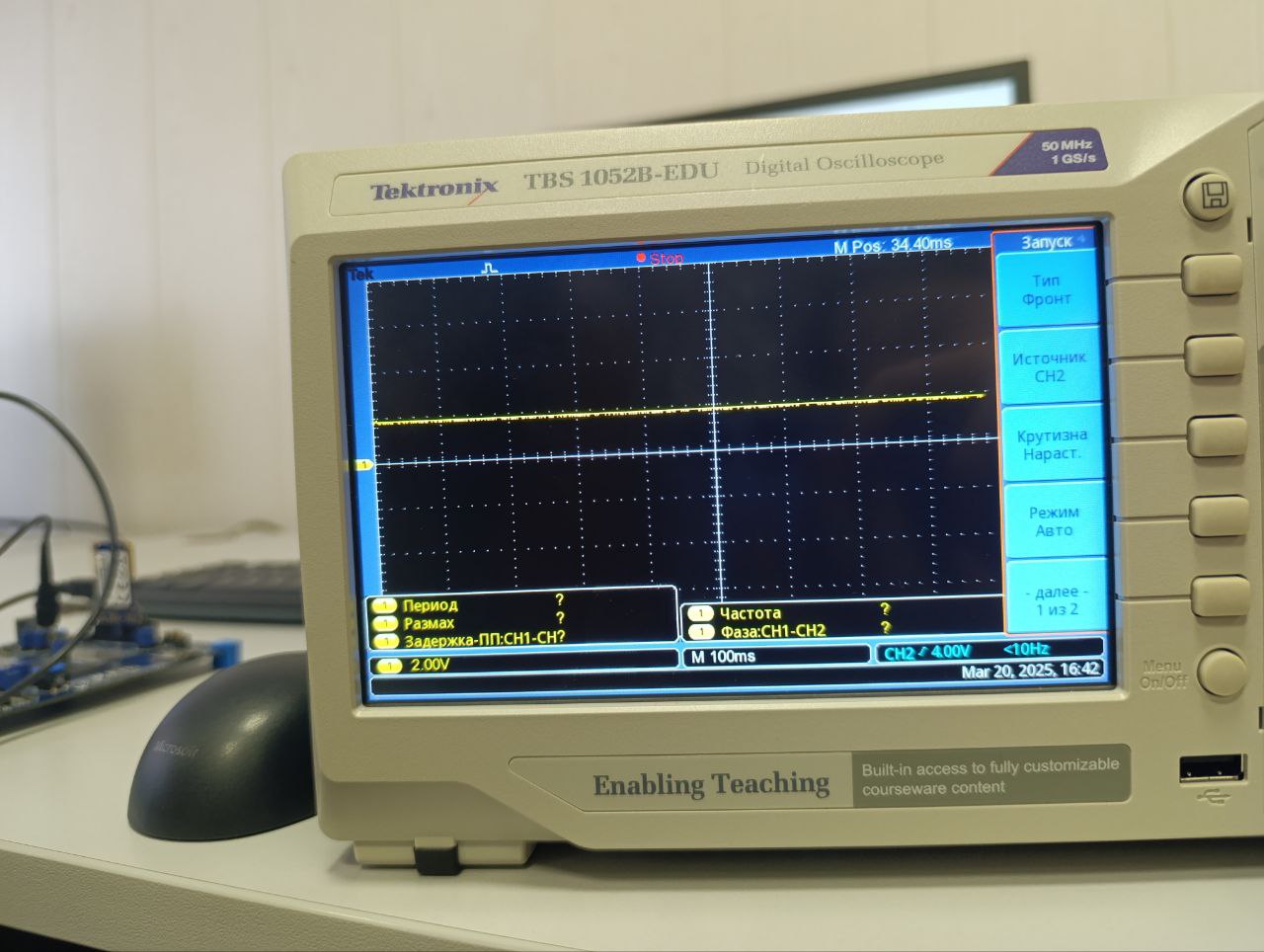
Рисунок 1. Постоянный сигнал с напряжением 3В

Рисунок 2. Постоянный сигнал с напряжением 2В

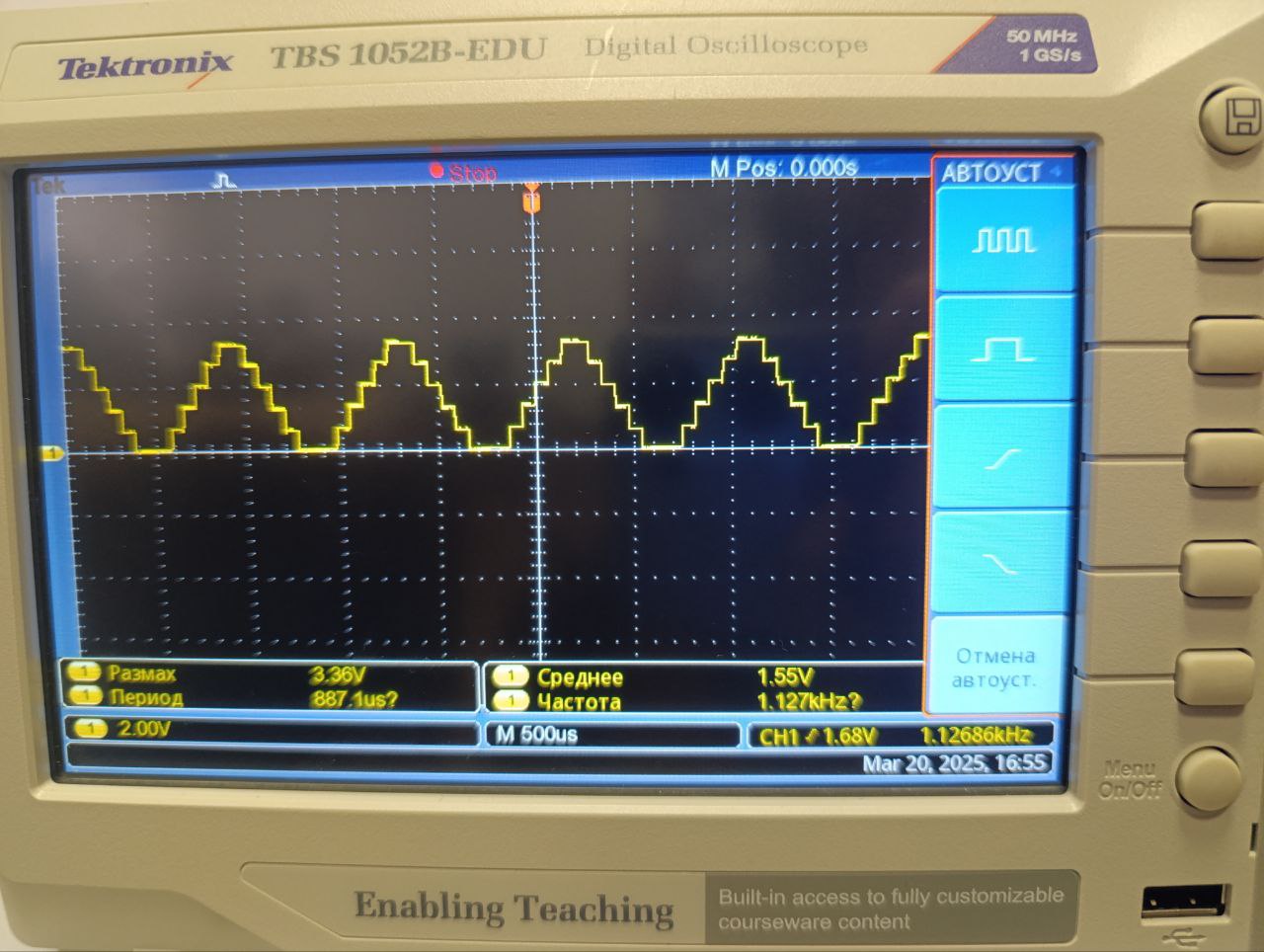


Рисунок 3. Ступенчатый сигнал

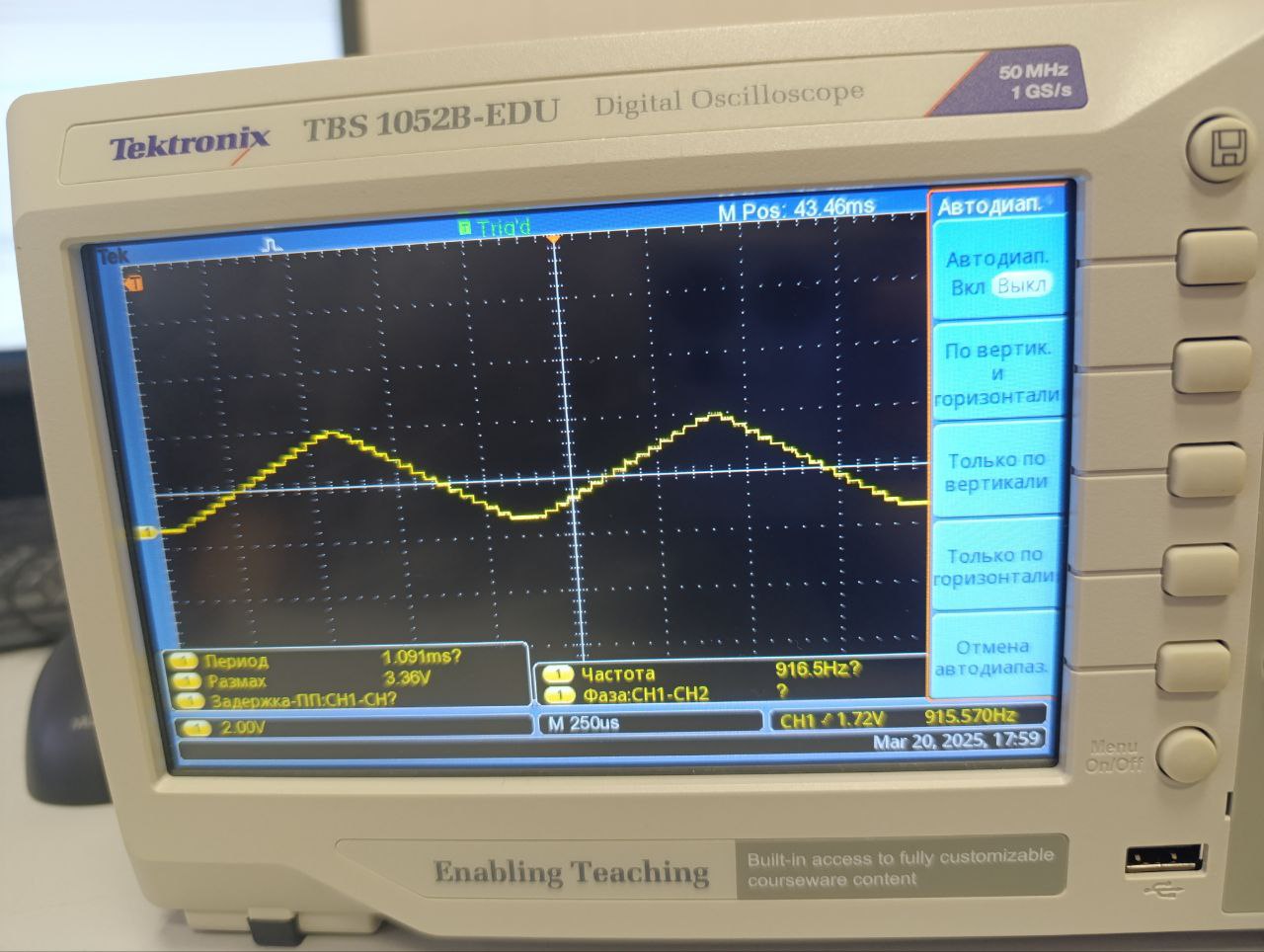


Рисунок 4. Синусоидальный сигнал

1. Полученные результаты

В результате работы программы и измерений мы получили сгенерированные сигналы различных форм заданного напряжения: постоянный и ступенчатый. Модифицируя программу, изменяли выходное напряжение и заданную форму.

1. Выводы

В результате выполнения лабораторной работы была создана программа для микроконтроллера (МК) STM32F200, которая выдает на выходе ЦАП заданный уровень напряжения. Была введена и модифицирована программа, предназначенная для генерирования на выходе ЦАП микроконтроллера (МК) STM32F200 периодической волны напряжения заданной формы. С помощью временных функций осциллографа были измерены временные характеристики синусоидальной формы волны.