

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)

Кафедра микроэлектроники, информационных технологий и управляющих
систем (МИТиУС)

DAC, ADC, DMA, TIMER, NVIC

Отчет по лабораторной работе №3

по дисциплине «Аппаратные Средства Телекоммуникационных Систем»

Студенты гр. 735:

_____ Е. Ю. Борисова

_____ И. В. Забатурина

_____ Д. А. Осипов

Принял:

Ст. преподаватель кафедры
МИТУС

_____ С. П. Недяк
(оценка)

Инженер кафедры МИТУС

_____ Ю. Б. Шаропин
(оценка)

(дата)

1 Введение

Цель лабораторной работы: изучить основные принципы организации аналоговой информации в микроконтроллерах. А также, изучение принципов работы DAC, ADC, DMA, NVIC, SysTick.

2 Теория

АЦП, ЦАП, ПДП

Аналого-цифровой преобразователь (АЦП) (ADC - Analog-to-digital converter) - устройство, преобразующее входной аналоговый сигнал в дискретный код (цифровой сигнал).

Цифро-аналоговый преобразователь (ЦАП) (DAC - Digital to analog converter) - устройство для преобразования цифрового (обычно двоичного) кода в аналоговый сигнал.

Прямой доступ к памяти (ПДП) (DMA - Direct memory access) - режим обмена данными между устройствами компьютера или же между устройством и основной памятью, в котором центральный процессор (ЦП) не участвует. Так как данные не пересылаются в ЦП и обратно, скорость передачи увеличивается.

На рисунке 2.1 приведены регистры блоков ДМА, ЦАП, АЦП.

Адрес	Размер	Блок		Примечание
0x4002_8000	80 байт	5	DMA	Регистры контроллера прямого доступа в память
0x4008_8000	48 байт	17	ADC	Регистры управления АЦП
0x4009_0000	12 байт	18	DAC	Регистры управления ЦАП

Рисунок 2.1 – Регистры блоков ЦАП, АЦП, ДМА

АЦП

В микроконтроллере реализовано два 12-разрядных АЦП. С помощью АЦП можно оцифровать сигнал от 16 внешних аналоговых выводов порта D и от двух внутренних каналов, на которые выводятся датчик температуры и источник опорного напряжения. Скорость выборки составляет до 512 тысяч преобразований в секунду для каждого АЦП.

Контроллер АЦП позволяет:

- оцифровать один из 16 внешних каналов;
- оцифровать значение встроенного датчика температуры;
- оцифровать значение встроенного источника опорного напряжения;
- осуществить автоматический опрос заданных каналов;
- выработать прерывание при выходе оцифрованного значения за заданные пределы;
- запускать два АЦП синхронно для увеличения скорости выборки.

В качестве синхросигнала может выступать частота процессора CPU_CLK.

Для работы с портами ввода/вывода используются библиотека MDR32F9Qx_adc.h, которая описывает регистры АЦП с помощью задания структур ADC_InitTypeDef для

настройки преобразователя и ADCx_InitTypeDef для настройки канала преобразователя.

Структура ADC_InitTypeDef имеет следующие поля:

- ADC_SynchronousMode – выбор режима работы двух преобразователей;
- ADC_StartDelay – определяет задержку начала преобразований от старта системы [0:15];
- ADC_TempSensor – включение/выключение температурного датчика;
- ADC_TempSensorAmplifier – включение/выключение усилителя температурного датчика;
- ADC_TempSensorConversion – включение/выключение преобразования показаний от температурного датчика;
- ADC_IntVRefConversion – включение/выключение преобразования показаний опорного напряжения;
- ADC_IntVRefTrimming – определяет интервал считывания значений опорного напряжения;

Структура ADCx_InitTypeDef имеет следующие поля:

- ADC_ClockSource – указывает источник тактирующего сигнала;
- ADC_SamplingMode – задает режим считывания показаний;
- ADC_ChannelSwitching – включение/выключение возможности переключения каналов АЦП;
- ADC_ChannelNumber – номер канала;
- ADC_Channels – маска номеров каналов;
- ADC_LevelControl – включение/выключение слежения за уровнем АЦП;
- ADC_LowLevel – значение нижнего уровня АЦП;
- ADC_HighLevel – значение верхнего уровня АЦП;
- ADC_VRefSource – определяет источник питания АЦП;
- ADC_IntVRefSource – определяет тип напряжения источника питания АЦП;
- ADC_Prescaler – задает параметры предусилителя;
- ADC_DelayGo – задержка начала преобразований в последовательном режиме;

ЦАП

В микроконтроллере реализовано два ЦАП.

Для включения ЦАП необходимо чтобы бит Cfg_ON_DACx был установлен в 1, используемые выводы ЦАП порта E были сконфигурированы как аналоговые и были отключены какие-либо внутренние подтяжки.

Оба ЦАП могут работать независимо или совместно. При независимой работе ЦАП (бит Cfg_SYNC_A=0) после записи данных в регистр данных DACx_DATA на выходе DACx_OUT формируется уровень напряжения, соответствующий записанному значению. При синхронной работе (бит Cfg_SYNC_A=1) данные обоих ЦАП могут быть обновлены одной записью в один из регистров DACx_DATA. ЦАП может работать от внутренней опоры Cfg_M_REFx=0, тогда ЦАП формирует выходной сигнал в диапазоне от 0 до напряжения питания AUCC. В режиме работы с внешней опорой Cfg_M_REFx=1 ЦАП формирует выходное напряжение в диапазоне от 0 до значения DACx_REF. Для работы с аналоговыми портами ввода/вывода используется библиотека MDR32F9Qx_dac.h.

ПДП

Задача контроллера – обеспечить передачу данных по шинам, без участия процессора. Что нужно для этого – указать адрес источника информации, адрес назначения информации, количество информации (сколько это будет), разрядность информации (8, 16, 32).

У него несколько каналов – 32 каналов и не все задействованы. Одновременно может обеспечить передачу по нескольким каналам.

Типы передач которые обеспечивает контроллер:

Память – память

Память – периферия

Периферия – память

Прерывания

При работе с различной периферией, в том числе и таймерами, часто используют прерывания.

Прерывание – событие, требующие немедленной реакции со стороны процессора. Реакция состоит в том, что процессор прерывает обработку текущей программы и переходит к выполнению некоторой другой программы, специально предназначенной для данного события. По завершении этой программы процессор возвращается к выполнению прерванной программы. (Рисунок 2.2)

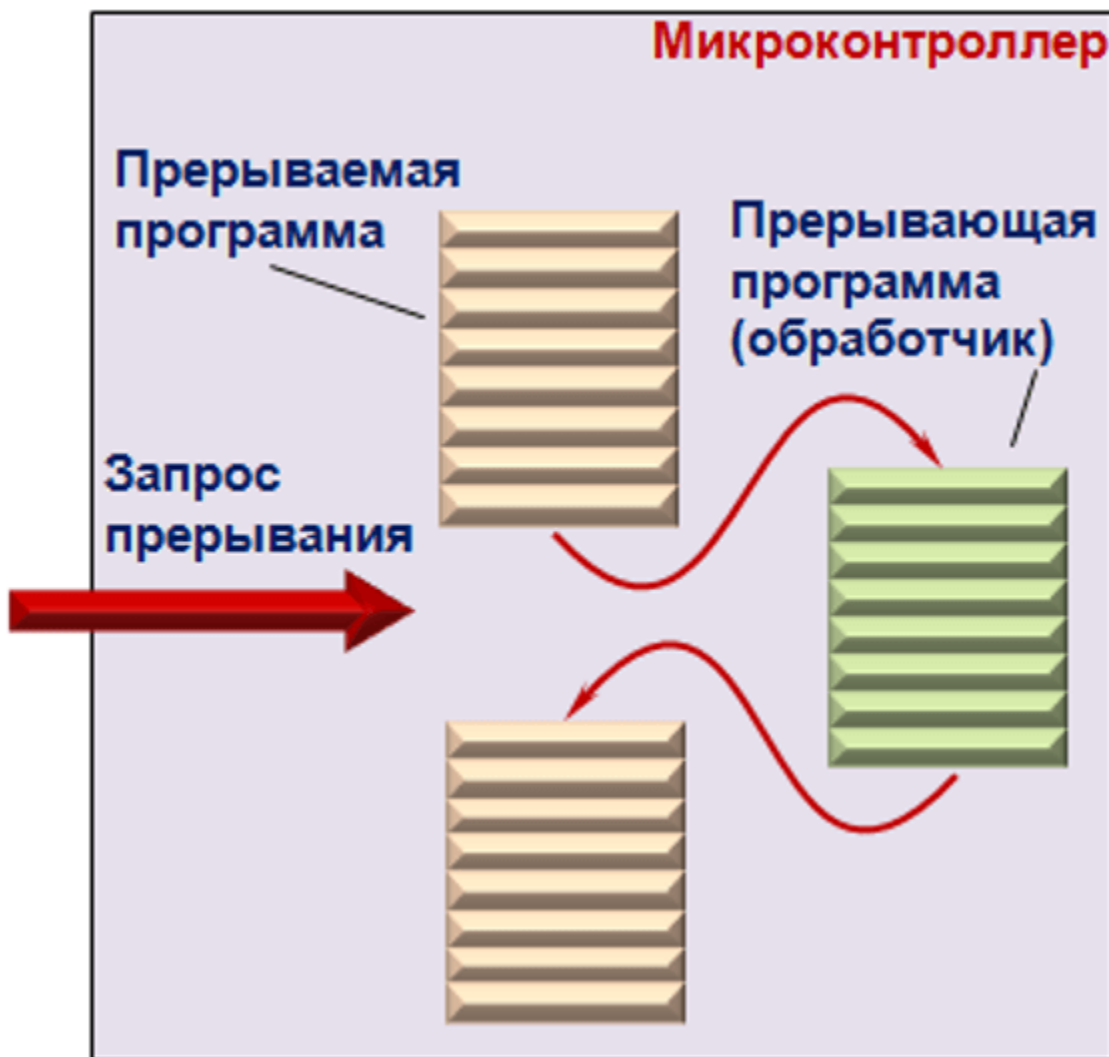


Рисунок 2.2 – Упрощенный алгоритм прерываний

Контроллер вложенных векторных прерываний.

Ключевые особенности контроллера NVIC:

- X каналов для маскируемых прерываний (X - число, разное, в зависимости от ядра; например, у Cortex-M0 32 канала, у Cortex-M4 74 канала);
- Y программируемых уровней приоритетов;
- низкая задержка по времени при обработке исключений и прерываний;
- управление питанием;
- реализация контроллера в виде системных регистров.

NVIC и процессорное ядро тесно связаны, что позволяет обрабатывать прерывания быстро и эффективно.

Все прерывания, включая исключения ядра, находятся под управлением NVIC. Больше подробностей об исключениях и программировании NVIC можно найти в руководстве программиста (для каждой версии ядра своё руководство).»

Далее идёт таблица векторов (вектор = адрес) прерываний. Таким образом, контроллер NVIC можно назвать менеджером прерываний.

Векторный контроллер прерываний с возможностью вложения (NVIC – Nested Vectored Interrupt Controller) обеспечивает:

- программное задание уровня приоритета в диапазоне от 0 до 15 независимо каждому прерыванию. Более высокое значение соответствует меньшему приоритету, таким образом, уровень 0 отвечает наивысшему приоритету прерывания;
- срабатывание сигнала прерывания по импульсу и по уровню;
- динамическое изменение приоритета прерываний;
- разделение исключений по группам с одинаковым приоритетом и по подгруппам внутри одной группы;
- передача управления из одного обработчика исключения на другой без восстановления контекста.

Процессор автоматически сохраняет в стеке свое состояние (контекст) по входу в обработчик прерывания и восстанавливает его по завершению обработчика без необходимости непосредственного программирования этих операций. Это обеспечивает обработку исключительных ситуаций с малой задержкой.

Поскольку прерывание может возникнуть при выполнении любой произвольной команды фона, её адрес запоминается в так называемом программном стеке. После чего выполнение передается на часть программы, специально написанную разработчиком для реакции на событие, вызвавшее данное прерывание. Эта небольшая часть программы называется обработчиком прерывания. Когда обработчик будет выполнен до конца, программа, воспользовавшись адресом, сохранённым в программном стеке, вернётся в то место, откуда была вызвана для обработки данного прерывания

Для использования вектора прерывания необходимо:

- Разрешить использование прерываний в программе;
- Разрешить вызов интересующего нас прерывания специальным битом в соответствующем регистре;
- Создать условия для возникновения прерывания, например, если это переполнение таймера, то запустить его;
- Разместить в программе обработчик прерывания, оформив его в соответствии с требованиями компилятора.

Для разрешения использования прерываний в программе используется функция `void NVIC_EnableIRQ(IRQn_t IRQn)`, в которую передается имя прерывания.

Для разрешения вызовов прерываний у каждого модуля описан свой регистр, например для таймера `TIMER_ITConfig (MDR_TIMER1, TIMER_STATUS_CNT_ARR, ENABLE)`;

Таймеры

Счетчик – делитель частоты, отсчитывает импульсы.

Таймер – устанавливается определенное значение и он отсчитывает это время до нуля.

«Процессор имеет 24-х разрядный системный таймер, SysTick, который считает вниз от загруженного в него значения до нуля; перезагрузка (возврат в начало) значения в регистр `LOAD` происходит по следующему фронту синхросигнала, затем счёт продолжается по последующему фронту.». (Рисунок 2.3 – 2.5)

Все блоки таймеров выполнены на основе 16-битного перезагружаемого счетчика, который синхронизируется с выхода 16-битного предделителя. Перезагружаемое значение хранится в отдельном регистре. Счет может быть прямой, обратный или двунаправленный (сначала прямой до определенного значения, а затем обратный).

Каждый из трех таймеров микроконтроллера содержит 16-битный счетчик, 16-битный предделитель частоты и 4-канальный блок захвата/сравнения. Их можно синхронизировать системной синхронизацией, внешними сигналами или другими таймерами.

Помимо составляющего основу таймера счетчика в каждый блок таймера также входит четырехканальный блок захвата/сравнения. Данный блок выполняет как стандартные функции захвата и сравнения, так и ряд специальных функций. Таймеры имеют 4 канала схем захвата и ШИМ с функциями формирования «мертвой зоны» и аппаратной блокировки. Каждый из таймеров может генерировать прерывания и запросы DMA.

Для работы с таймерами используется **структура `TIMER_CntInitTypeDef` с полями:**

- `TIMER_Prescaler` – значение величины предделителя
- `TIMER_Period` – период таймера
- `TIMER_CounterMode` – режим счета
- `TIMER_CounterDirection` – направление счета
- `TIMER_EventSource` – источник событий для таймера
- `TIMER_FilterSampling` – указывает фильтр событий
- `TIMER_ARR_UpdateMode` – режим сброса счетчика
- `TIMER_ETR_FilterConf` – задает параметры выхода ETR
- `TIMER_ETR_Prescaler` – задает параметры предделителя фильтра выхода ETR
- `TIMER_ETR_Polarity` – задает полярность выхода ETR
- `TIMER_BRK_Polarity` – задает полярность выхода BRK

Структура TIMER_ChnInitTypeDef полями:

- TIMER_CH_Mode – задает режим работы таймера
- TIMER_CH_REF_Format – формат выработки сигнала REFв режима ШИМ
- TIMER_CH_Number – номер канала таймера

Структура TIMER_ChnOutInitTypeDef полями:

- TIMER_CH_DirOut_Polarity – полярность выхода CHx таймера
- TIMER_CH_DirOut_Source – задает сигнал на выходе CHx таймера
- TIMER_CH_DirOut_Mode – задает сигнал на выходе CHx таймера
- TIMER_CH_NegOut_Polarity – полярность инверсного выхода CHx таймера
- TIMER_CH_NegOut_Source – задает сигнал на инверсном выходе CHx таймера
- TIMER_CH_NegOut_Mode – задает сигнал на инверсном выходе CHx таймера
- TIMER_CH_Number – номер канала

Таблица 61 – Описание регистров системного таймера SysTick

Адрес	Название	Тип	Доступ	Значение после сброса	Описание
0xE000E010	SysTick				Системный таймер SYSTICK
0xE000E010	CTRL	RW	привилегированный	0x00000004	SysTick->CTRL
0xE000E014	LOAD	RW	привилегированный	0x00000000	SysTick->LOAD
0xE000E018	VAL	RW	привилегированный	0x00000000	SysTick->VAL
0xE000E01C	CALIB	RO	привилегированный	0x00002904 ⁽¹⁾	SysTick->CAL

¹⁾ Калибровочное значение системного таймера.

Рисунок 2.3 – Описание регистров системного таймера

SysTick->CTRL

Регистр CTRL разрешает основные функции системного таймера.

Назначение бит:

Таблица 62 – Регистр контроля и статуса CTRL

Номер	31...17	16	15...3	2	1	0
Доступ						
Сброс						
	-	COUNTFLAG	-	CLKSOURCE	TICKINT	ENABLE

COUNTFLAG

Возвращает 1, если таймер досчитал до нуля с последнего момента чтения.

CLKSOURCE

Указывает источник синхросигнала:

0 - LSI

1 - HCLK

TICKINT

Разрешает запрос на прерывание от системного таймера:

0 - таймер досчитает до нуля и прерывание не возникнет;

1 - таймер досчитывает до нуля и возникает запрос на прерывание.

Программное обеспечение может использовать бит COUNTFLAG, чтобы определить, досчитал таймер до нуля или нет.

ENABLE

Разрешает работу таймера:

0 - работа таймера запрещена;

1 - работа таймера разрешена.

Когда ENABLE установлен в единицу, таймер загружает значение RELOAD из регистра LOAD и затем начинает декрементироваться. По достижению значения 0 таймер устанавливает бит COUNTFLAG и в зависимости от TICKINT генерирует запрос на прерывание. Затем загружается значение RELOAD и продолжается счёт.

Рисунок 2.4 – Описание регистра CTRL

SysTick->LOAD

Регистр LOAD устанавливает стартовое значение, загружаемое в регистр VAL.

Таблица 63 – Регистр перегружаемого значения LOAD

Номер	31...24	23...0
Доступ		
Сброс		
	-	RELOAD

RELOAD

Значение, загружаемое в регистр VAL, когда таймер разрешён и когда достигается значение нуля.

Расчёт значения RELOAD

Значение RELOAD может быть любым в диапазоне 0x00000001–0x00FFFFFF. Значение 0 допустимо, но не оказывает эффекта, потому что запрос на прерывание и активизация бита COUNTFLAG происходит только при переходе таймера из состояния 1 в 0.

Расчёт значения RELOAD происходит в соответствии с использованием таймера:

- Для формирования мультикороткого таймера с периодом N процессорных циклов применяется значение RELOAD, равное N-1. Например, если требуется прерывание каждые 100 циклов, то устанавливается значение RELOAD, равное 99;
- Для формирования одиночного прерывания после задержки в N тактов процессора используется значение N. Например, если требуется прерывание после 400 тактов процессора, то устанавливается RELOAD, равное 400.

Рисунок 2.5 – Описание регистра LOAD

Поиск вектора прерываний для таймера. (Рисунок 2.6)

56	__Vectors	DCD	__initial_sp	; Top of Stack
57		DCD	Reset_Handler	; Reset Handler
58		DCD	NMI_Handler	; NMI Handler
59		DCD	HardFault_Handler	; Hard Fault Handler
60		DCD	MemManage_Handler	; MPU Fault Handler
61		DCD	BusFault_Handler	; Bus Fault Handler
62		DCD	UsageFault_Handler	; Usage Fault Handler
63		DCD	0	; Reserved
64		DCD	0	; Reserved
65		DCD	0	; Reserved
66		DCD	0	; Reserved
67		DCD	SVC_Handler	; SVC Call Handler
68		DCD	DebugMon_Handler	; Debug Monitor Handler
69		DCD	0	; Reserved
70		DCD	PendSV_Handler	; PendSV Handler
71		DCD	SysTick_Handler	; SysTick Handler

Рисунок 2.6 – Таблица векторов прерываний в файле «startup_1986e9x.s»

Это так называемые «вектора прерываний». Когда появляется какое-то внешнее/внутреннее неотложное событие (прерывание), микроконтроллер прерывает выполнение основной программы, переходит к этой таблице и смотрит, куда ему нужно перейти дальше. Например, когда появляется прерывание от нашего таймера, он переходит к пункту с именем «SysTick_Handler». В случае, если вектор не прописан нами в программе (нет функции с таким именем) – контроллер игнорирует его и продолжает выполнение своей

программы. Но если в программе есть функция с этим именем, то он переходит к ее выполнению.

ШИМ

Широтно-Импульсная Модуляция (ШИМ) – управление средней мощностью нагрузки с помощью серии высокочастотных импульсов. Регулируется усреднённая мощность изменением длительности импульсов и пауз между ними. Чем длиннее импульсы и короче паузы между ними, тем средняя мощность в нагрузке выше.

ШИМ представляет собой импульсный сигнал постоянной частоты и переменной скважности (отношение длительности импульса к периоду его следования). С помощью задания скважности можно менять среднее напряжение на выходе ШИМ. Таким способом, меняя выходную мощность, можно управлять яркостью светодиода.

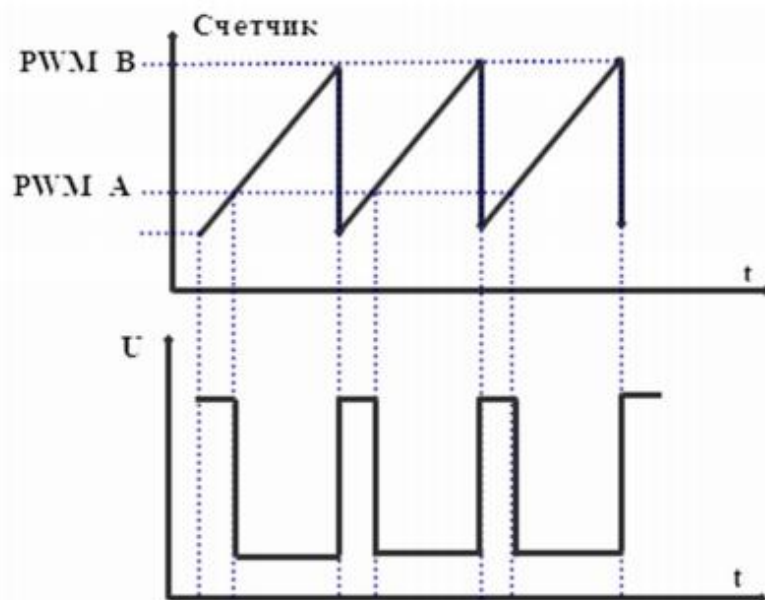
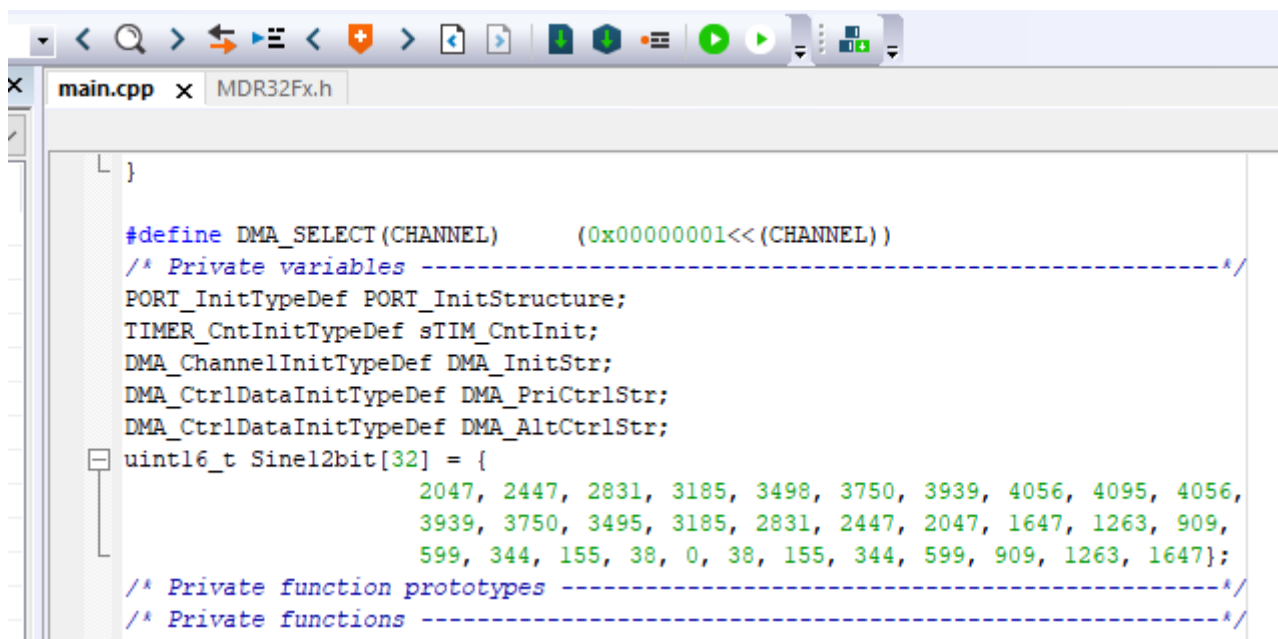


Рисунок 2.7 – Диаграмма генерации ШИМ

3 Ход работы

ЦАП

тактирование является главным требованием при работе с периферией. Для работы с DAC, DMA и таймером был использован пример, предоставленный в библиотеке SPL (Standard Peripherals Library) с названием «DMA_SineWave». В данном примере описана возможность задать значения «точек» с заданным уровнем, которые с помощью DMA подаются на DAC. Сначала объявляем переменные структур порта, таймера и DMA, объявляем и инициализируем массив значений сигнала, которые соответствуют одному периоду синуса. (Рисунок 3.1)



```

}

#define DMA_SELECT(CHANNEL)    (0x00000001<<(CHANNEL))
/* Private variables -----*/
PORT_InitTypeDef PORT_InitStructure;
TIMER_CntInitTypeDef sTIM_CntInit;
DMA_ChannelInitTypeDef DMA_InitStr;
DMA_CtrlDataInitTypeDef DMA_PriCtrlStr;
DMA_CtrlDataInitTypeDef DMA_AltCtrlStr;
uint16_t Sine12bit[32] = {
    2047, 2447, 2831, 3185, 3498, 3750, 3939, 4056, 4095, 4056,
    3939, 3750, 3495, 3185, 2831, 2447, 2047, 1647, 1263, 909,
    599, 344, 155, 38, 0, 38, 155, 344, 599, 909, 1263, 1647};
/* Private function prototypes -----*/
/* Private functions -----*/

```

Рисунок 3.1 – Объявление переменных и переменная «Sine12bit», содержащая синусоиду

Затем производится последовательное включение тактирования всей необходимой периферии (таймер, DAC и т.д.). Далее идет отключение всех прерываний с помощью КВВП (контроллер вложенных векторных прерываний) и конфигурация порта «PORTE_0». (Рисунок 3.2)


```

main.cpp x MDR32Fx.h

/* Enable peripheral clocks ----- */
RST_CLK_PCLKcmd((RST_CLK_PCLK_RST_CLK | RST_CLK_PCLK_DMA | RST_CLK_PCLK_PORTC),ENABLE);
RST_CLK_PCLKcmd((RST_CLK_PCLK_TIMER1 | RST_CLK_PCLK_DAC),ENABLE);
RST_CLK_PCLKcmd((RST_CLK_PCLK_SSP1 | RST_CLK_PCLK_SSP2),ENABLE);

/* Disable all interrupt */
NVIC->ICPR[0] = 0xFFFFFFFF;
NVIC->ICER[0] = 0xFFFFFFFF;

/* Reset PORTC settings */
PORT_DeInit(MDR_PORTC);

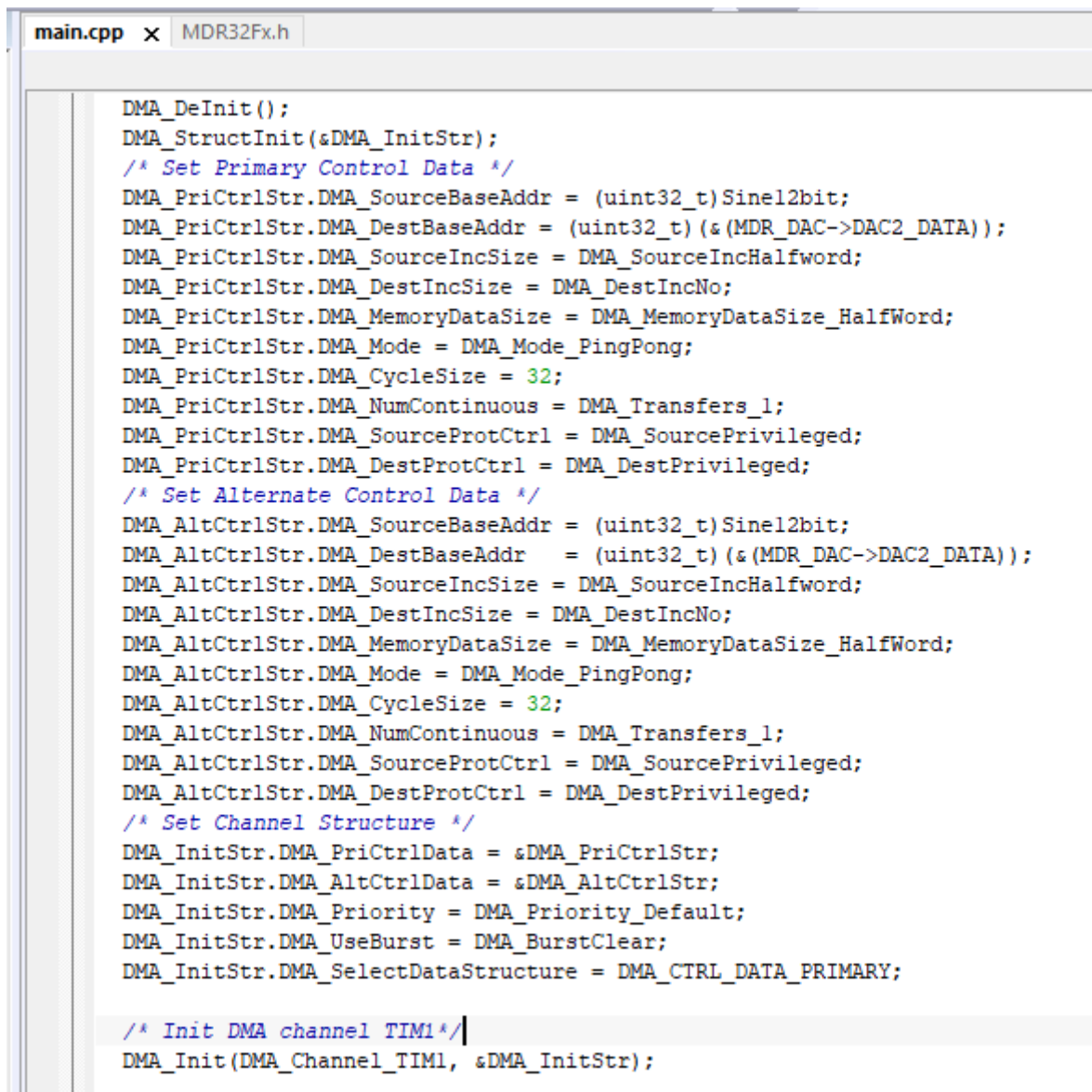
/* Configure DAC pin: DAC2_OUT */

/* Configure PORTC pin 0 */
PORT_InitStructure.PORT_Pin = PORT_Pin_0;
PORT_InitStructure.PORT_OE = PORT_OE_OUT;
PORT_InitStructure.PORT_MODE = PORT_MODE_ANALOG;
PORT_Init(MDR_PORTC, &PORT_InitStructure);

```

Рисунок 3.2 - Включение тактирования и конфигурация порта

Чтобы заполнить поля структур DMA нужно указать адрес источника для DMA, который соответствует первому элементу массива Sine12bit. В адресе назначения задается адрес регистра данных DAC2. Размер инкрементации источника задается значением в половину слова, так как изначально тип данных массива соответствует размеру половины слова. Количество итераций в цикле было задано 32, так как это значение соответствует размеру массива. (Рисунок 3.3)



```

main.cpp x MDR32Fx.h

DMA_DeInit();
DMA_StructInit(&DMA_InitStr);
/* Set Primary Control Data */
DMA_PriCtrlStr.DMA_SourceBaseAddr = (uint32_t)Sine12bit;
DMA_PriCtrlStr.DMA_DestBaseAddr = (uint32_t)(&(MDR_DAC->DAC2_DATA));
DMA_PriCtrlStr.DMA_SourceIncSize = DMA_SourceIncHalfword;
DMA_PriCtrlStr.DMA_DestIncSize = DMA_DestIncNo;
DMA_PriCtrlStr.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
DMA_PriCtrlStr.DMA_Mode = DMA_Mode_PingPong;
DMA_PriCtrlStr.DMA_CycleSize = 32;
DMA_PriCtrlStr.DMA_NumContinuous = DMA_Transfers_1;
DMA_PriCtrlStr.DMA_SourceProtCtrl = DMA_SourcePrivileged;
DMA_PriCtrlStr.DMA_DestProtCtrl = DMA_DestPrivileged;
/* Set Alternate Control Data */
DMA_AltCtrlStr.DMA_SourceBaseAddr = (uint32_t)Sine12bit;
DMA_AltCtrlStr.DMA_DestBaseAddr = (uint32_t)(&(MDR_DAC->DAC2_DATA));
DMA_AltCtrlStr.DMA_SourceIncSize = DMA_SourceIncHalfword;
DMA_AltCtrlStr.DMA_DestIncSize = DMA_DestIncNo;
DMA_AltCtrlStr.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
DMA_AltCtrlStr.DMA_Mode = DMA_Mode_PingPong;
DMA_AltCtrlStr.DMA_CycleSize = 32;
DMA_AltCtrlStr.DMA_NumContinuous = DMA_Transfers_1;
DMA_AltCtrlStr.DMA_SourceProtCtrl = DMA_SourcePrivileged;
DMA_AltCtrlStr.DMA_DestProtCtrl = DMA_DestPrivileged;
/* Set Channel Structure */
DMA_InitStr.DMA_PriCtrlData = &DMA_PriCtrlStr;
DMA_InitStr.DMA_AltCtrlData = &DMA_AltCtrlStr;
DMA_InitStr.DMA_Priority = DMA_Priority_Default;
DMA_InitStr.DMA_UseBurst = DMA_BurstClear;
DMA_InitStr.DMA_SelectDataStructure = DMA_CTRL_DATA_PRIMARY;

/* Init DMA channel TIM1 */
DMA_Init(DMA_Channel_TIM1, &DMA_InitStr);

```

Рисунок 3.3 - Настройка DMA

Далее происходит заполнение структуры таймера, для чего задаются такие параметры, как значение предделителя частоты, основание счета таймера, направление и вид счёта (однонаправленный), частота семплирования данных, режим обновления регистра основания счёта и т.д. Затем осуществляется процедура инициализация таймера. (Рисунок 3.4)

```

/* TIMER1 Configuration */
/* Time base configuration */
TIMER_DeInit(MDR_TIMER1);
TIMER_BRGInit(MDR_TIMER1,TIMER_HCLKdiv1);
sTIM_CntInit.TIMER_Prescaler           = 0;
sTIM_CntInit.TIMER_Period               = 0xFF;
sTIM_CntInit.TIMER_CounterMode          = TIMER_CntMode_ClkFixedDir;
sTIM_CntInit.TIMER_CounterDirection     = TIMER_CntDir_Up;
sTIM_CntInit.TIMER_EventSource          = TIMER_EvSrc_None;
sTIM_CntInit.TIMER_FilterSampling       = TIMER_FDTS_TIMER_CLK_div_1;
sTIM_CntInit.TIMER_ARR_UpdateMode       = TIMER_ARR_Update_Immediately;
sTIM_CntInit.TIMER_ETR_FilterConf       = TIMER_Filter_1FF_at_TIMER_CLK;
sTIM_CntInit.TIMER_ETR_Prescaler        = TIMER_ETR_Prescaler_None;
sTIM_CntInit.TIMER_ETR_Polarity         = TIMER_ETRPolarity_NonInverted;
sTIM_CntInit.TIMER_BRK_Polarity         = TIMER_BRKPolarity_NonInverted;
TIMER_CntInit (MDR_TIMER1,&sTIM_CntInit);

/* Enable DMA for TIMER1 */
TIMER_DMACmd(MDR_TIMER1, (TIMER_STATUS_CNT_ARR), ENABLE);

/* TIMER1 enable counter */
TIMER_Cmd(MDR_TIMER1,ENABLE);

/* Enable DMA IRQ */
NVIC_EnableIRQ(DMA_IRQn);

```

Рисунок 3.4 - Настройка таймера и включение прерывания

Форма сигнала имеет ступенчатый вид, так как аналоговый сигнал на выходе ЦАПa был построен на дискретных значениях. (Рисунок 3.5 – 3.6)

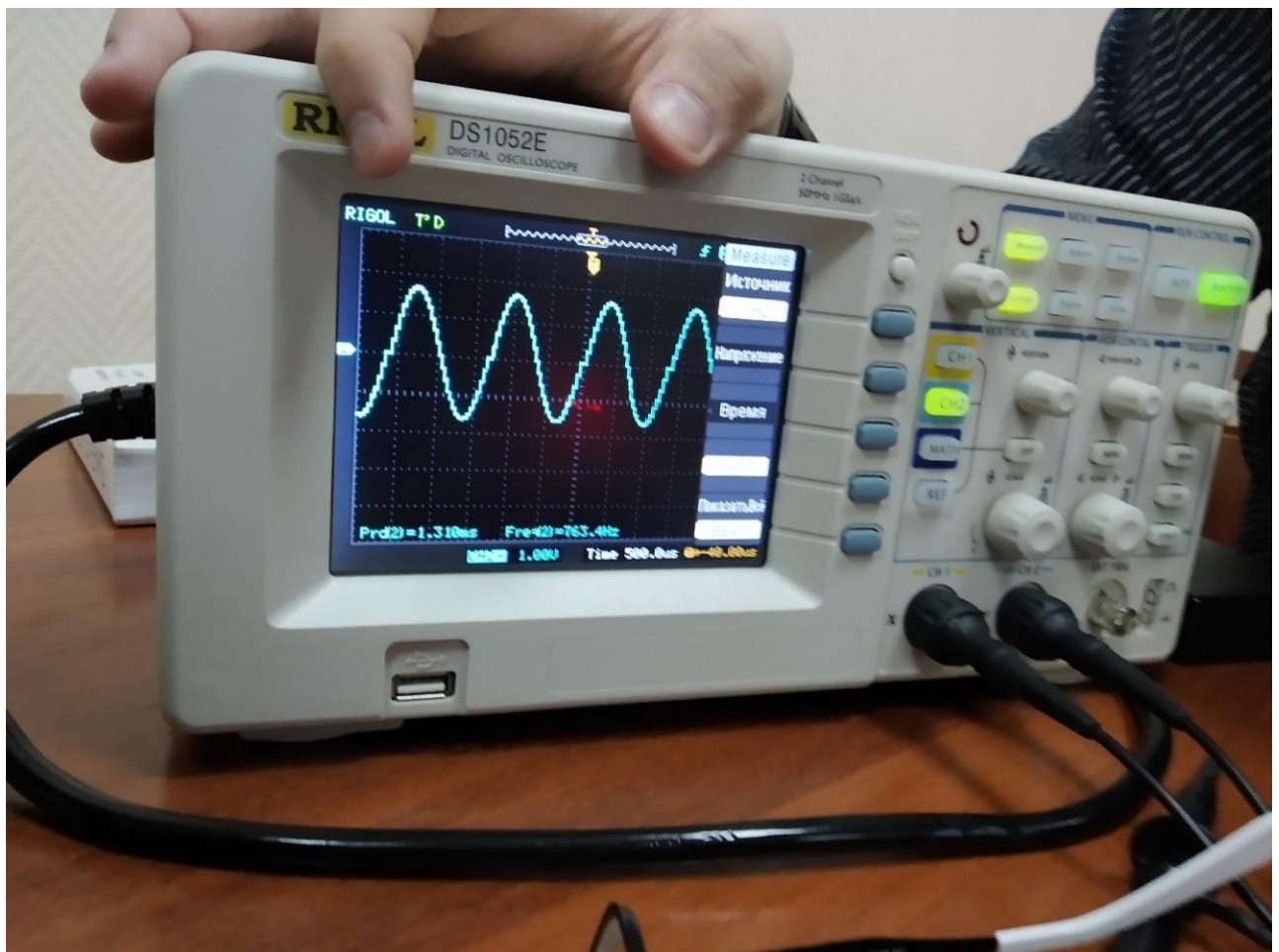


Рисунок 3.5 – Получение сигнала осциллографом – фото 1

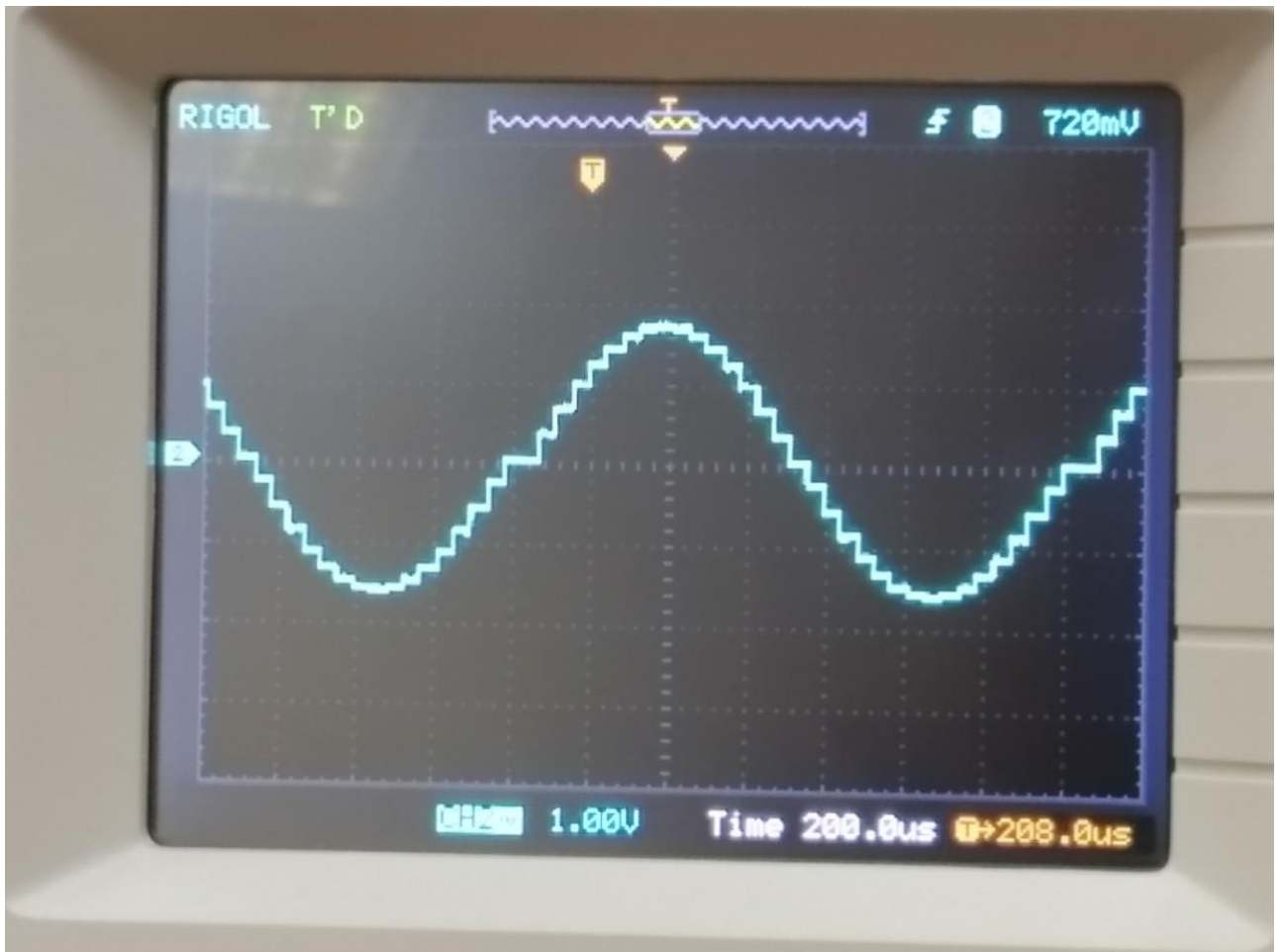


Рисунок 3.6 - Получение сигнала осциллографом – фото 2

АЦП

Для работы с АЦП был использован пример, предоставленный в библиотеке SPL (Standard Peripherals Library) с названием «ADC1_DMA». ADC1 и DMA использовались для передачи непрерывно преобразованных данных из ADC1 в память устройства. АЦП31 - датчик температуры. АЦП1 настроен на непрерывное преобразование канала АЦП31. После окончания преобразования DMA передает в режиме пинг-понга преобразованные данные из регистра ADC1 RESULT в массив ADC_ConvertedValue [10]. Настройка DMA производилась аналогично примеру с ЦАП, отличие заключается в адресах источника и назначения. (Рисунок 3.7)

```

/* DMA Configuration */
/* Reset all settings */
DMA_DeInit();
DMA_StructInit(&DMA_InitStr);
/* Set Primary Control Data */
DMA_PriCtrlStr.DMA_SourceBaseAddr = (uint32_t) (&(MDR_ADC->ADC1_RESULT));
DMA_PriCtrlStr.DMA_DestBaseAddr = (uint32_t) ADCConvertedValue;
DMA_PriCtrlStr.DMA_SourceIncSize = DMA_SourceIncNo;
DMA_PriCtrlStr.DMA_DestIncSize = DMA_DestIncHalfword;
DMA_PriCtrlStr.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
DMA_PriCtrlStr.DMA_Mode = DMA_Mode_PingPong;
DMA_PriCtrlStr.DMA_CycleSize = 10;
DMA_PriCtrlStr.DMA_NumContinuous = DMA_Transfers_1;
DMA_PriCtrlStr.DMA_SourceProtCtrl = DMA_SourcePrivileged;
DMA_PriCtrlStr.DMA_DestProtCtrl = DMA_DestPrivileged;

/* Set Alternate Control Data */
DMA_AltCtrlStr.DMA_SourceBaseAddr = (uint32_t) (&(MDR_ADC->ADC1_RESULT));
DMA_AltCtrlStr.DMA_DestBaseAddr = (uint32_t) ADCConvertedValue;
DMA_AltCtrlStr.DMA_SourceIncSize = DMA_SourceIncNo;
DMA_AltCtrlStr.DMA_DestIncSize = DMA_DestIncHalfword;
DMA_AltCtrlStr.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
DMA_AltCtrlStr.DMA_Mode = DMA_Mode_PingPong;
DMA_AltCtrlStr.DMA_CycleSize = 10;
DMA_AltCtrlStr.DMA_NumContinuous = DMA_Transfers_1;
DMA_AltCtrlStr.DMA_SourceProtCtrl = DMA_SourcePrivileged;
DMA_AltCtrlStr.DMA_DestProtCtrl = DMA_DestPrivileged;

```

Рисунок 3.7 - Настройка DMA

Настройки работы температурного датчика: независимый запуск двух АЦП, включение температурного датчика (бит TS_EN), разрешение работы выходного усилителя (бит TS_BUF), выбор для оцифровки датчика температуры (бит SEL_TS), запрет выбора источника опорного напряжения для оцифровки (бит SEL_VREF значение 1.23 В).

Для настройки АЦП были выбраны источник тактирования, режим многократного преобразования, отключение режима автоматического переключения каналов, выбран канал с датчиком температуры, отключен контроль входного сигнала, выбран внутренний источник опорного напряжения, выбран делитель тактовой частоты, установлено значение задержки перед началом следующего преобразования. (Рисунок 3.8)

```

/* ADC Configuration */
/* Reset all ADC settings */
ADC_DeInit();
ADC_StructInit(&sADC);

sADC.ADC_SynchronousMode      = ADC_SyncMode_Independent;
sADC.ADC_StartDelay           = 0;
sADC.ADC_TempSensor           = ADC_TEMP_SENSOR_Enable;
sADC.ADC_TempSensorAmplifier  = ADC_TEMP_SENSOR_AMPLIFIER_Enable;
sADC.ADC_TempSensorConversion = ADC_TEMP_SENSOR_CONVERSION_Enable;
sADC.ADC_IntVRefConversion    = ADC_VREF_CONVERSION_Disable;
sADC.ADC_IntVRefTrimming      = 1;
ADC_Init (&sADC);

/* ADC1 Configuration */
ADCx_StructInit (&sADCx);
sADCx.ADC_ClockSource      = ADC_CLOCK_SOURCE_CPU;
sADCx.ADC_SamplingMode     = ADC_SAMPLING_MODE_CICLIC_CONV;
sADCx.ADC_ChannelSwitching = ADC_CH_SWITCHING_Disable;
sADCx.ADC_ChannelNumber    = ADC_CH_TEMP_SENSOR;
sADCx.ADC_Channels         = 0;
sADCx.ADC_LevelControl     = ADC_LEVEL_CONTROL_Disable;
sADCx.ADC_LowLevel         = 0;
sADCx.ADC_HighLevel        = 0;
sADCx.ADC_VRefSource       = ADC_VREF_SOURCE_INTERNAL;
sADCx.ADC_IntVRefSource    = ADC_INT_VREF_SOURCE_INEXACT;
sADCx.ADC_Prescaler        = ADC_CLK_div_512;
sADCx.ADC_DelayGo         = 7;
ADC1_Init (&sADCx);

/* Enable ADC1 EOCIF and AWOIFEN interrupts */
ADC1_ITConfig((ADCx_IT_END_OF_CONVERSION | ADCx_IT_OUT_OF_RANGE), DISABLE);

```

Рисунок 3.8 - Настройка АЦП

В примерах с ЦАП и АЦП используются прерывания. На рисунке 3.9 представлен обработчик прерывания DMA_IRQHandler. С помощью него и реализован режим «пинг-понг», в котором контроллер выполняет цикл DMA, используя одну из структур управляющих данных, а затем выполняет еще один цикл DMA, используя другую структуру управляющих данных. Контроллер выполняет циклы DMA с переключением структур до тех пор, пока не считает «неправильную» структуру данных или пока процессор не запретит работу канала. (Рисунок 3.9)

Само прерывание возникает при установке контроллером DMA флага DMA_DONE в состояние 1, то есть после завершения работы одной из структур DMA.

```

void DMA_IRQHandler(void)
{
    /* Reconfigure the inactive DMA data structure*/
    if((MDR_DMA->CHNL_PRI_ALT_SET & (1<<DMA_Channel_ADC1)) == (0<<DMA_Channel_ADC1))
    {
        DMA_AltCtrlStr.DMA_CycleSize = 10;
        DMA_Init(DMA_Channel_ADC1, &DMA_InitStr);
    }
    else if((MDR_DMA->CHNL_PRI_ALT_SET & (1<<DMA_Channel_ADC1)) == (1<<DMA_Channel_ADC1))
    {
        DMA_PriCtrlStr.DMA_CycleSize = 10;
        DMA_Init(DMA_Channel_ADC1, &DMA_InitStr);
    }
}

```

Рисунок 3.9 - Обработчик прерывания DMA_IRQHandler

3 Заключение

В ходе работы были изучены и описаны теоретические сведения о ЦАП, АЦП, NVIC, DMA и таймерах, а также отлажены и запущены тестовые программы примеров.

Был написан отчёт согласно требованиям ОС ТУСУР 01-2013.