Быстрый старт с МК

К1921ВК01Т

Воронеж 2015

Содержание

1 Описание микроконтроллера К1921ВК01Т

Микросхема К1921ВК01Т представляет собой СБИС 32-разрядного микроконтроллера на базе ядра ARM Cortex-M4F предназначенного для промышленных и потребительских приложений, включая системы дистанционного мониторинга, контрольно-измерительные приборы, сетевые устройства, системы автоматизации производственных процессов, автомобильную электронику, системы управления электродвигателями.

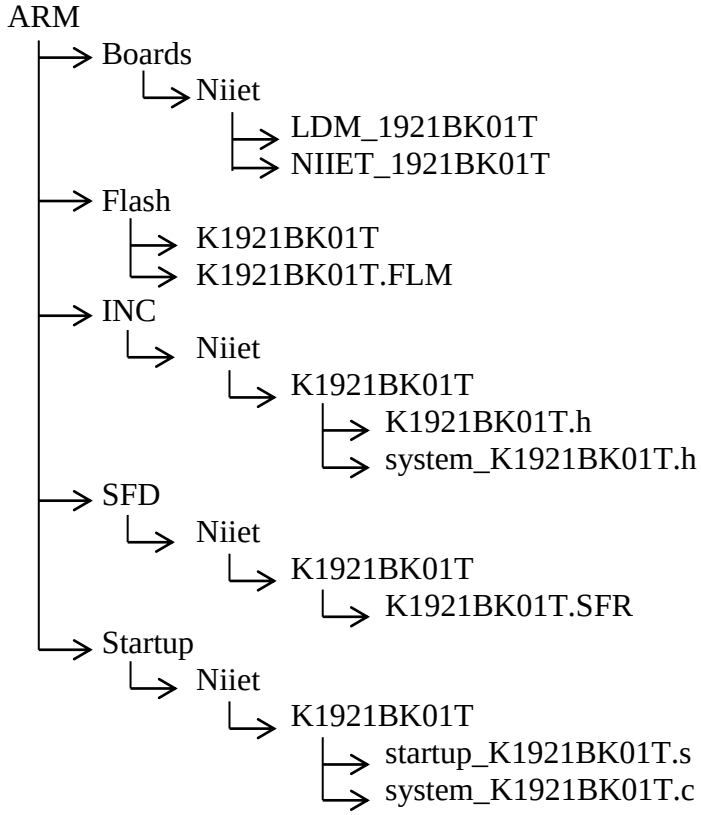
В состав микроконтроллера входят функциональные элементы:

- процессорное ядро ARM Cortex-M4F производительностью не менее 125 миллионов инструкций в секунду с поддержкой набора одноцикловых команд умножения с накоплением (блок MPU), команд централизованного управления потоком данных, арифметических и логических команд и встроенным модулем обработки команд с плавающей запятой с одинарной точностью (блок FPU);
- загрузочная флеш-память (Flash) емкостью 1 Мбайт;
- ОЗУ объемом 192 Кбайт;
- пользовательская флеш-память объемом 64 Кбайта;
- контроллер внешней статической памяти (SRAM, PROM, NOR FLASH);
- 32-канальный контроллер прямого доступа к памяти (DMA);
- схема сброса и сторожевой таймер;
- часы реального времени с батарейным питанием (RTC);
- синтезатор частоты на основе ФАПЧ (PLL);
- двенадцать 2-канальных 12-разрядных АЦП с режимами цифрового компаратора для каждого из каналов (равно или больше, равно или меньше, попадание в диапазон, выход из диапазона) и функцией автоматического запуска модулей ШИМ по событию «окончание преобразования»;
- три аналоговых компаратора с функцией автоматического запуска модулей ШИМ по событиям сравнения «равно или больше» и «равно или меньше»;
- девять модулей ШИМ, шесть из которых поддерживают режим высокого разрешения (возможность изменения длительности импульсов на величину менее периода тактового сигнала);
- два импульсных квадратурных декодера, используемых для обработки сигналов датчиков положения ротора в высокопроизводительных системах для определения положения, направления и скорости вращения;
- шесть модулей захвата/сравнения;
- три 32-разрядных таймера;
- отладочный интерфейс JTAG и ARM SWD (Serial Wire Debug);
- семь 16-разрядных и один 8-разрядный порт ввода/вывода с раздельно программируемыми мультиплексированными выводами общего назначения;
- четыре последовательных порта UART;
- контроллеры интерфейсов:
- CAN (протокол 2.0B) с двумя портами ввода-вывода;
- I2C с поддержкой частоты передачи данных более 1 МГц (два порта);
- SPI (четыре порта);
- USB 2.0 Device/Host с физическим уровнем PHY;
- Ethernet 10/100 Мбит/с с интерфейсом MII.

2 Настройка среды Keil uVision4(5)

2.1 Конфигурация Keil uVision

Для разработки проектов в среде Keil необходимо в каталог с установленным ПО добавить следующие файлы (прилагаются в архиве)



Каталог «Boards» содержит примеры программ для отладочных плат разработанных LDM-SYTEM и NIIET.

В проекте «Blinky» сконфигурированы функции стандартного ввода/вывода printf(), взаимодействующие с UART2, а также осуществляется управление светодиодами соответствующих плат.

Каталог «Flash» содержит алгоритмы программирования внутренней Flash памяти средствами Keil

Каталог «INC» содержит заголовочный файл описания SFR регистров микроконтроллера

Каталог «SFD» содержит описания SFR регистров микроконтроллера для интерфейса отладки (Debug)Keil

Каталог «Startup» содержит файл startup для инициализации векторов прерывания и модуля FPU.

Добавление микроконтроллера в базу данных устройств Keil осуществляется следующим образом:

Открываем «File->Device Database»

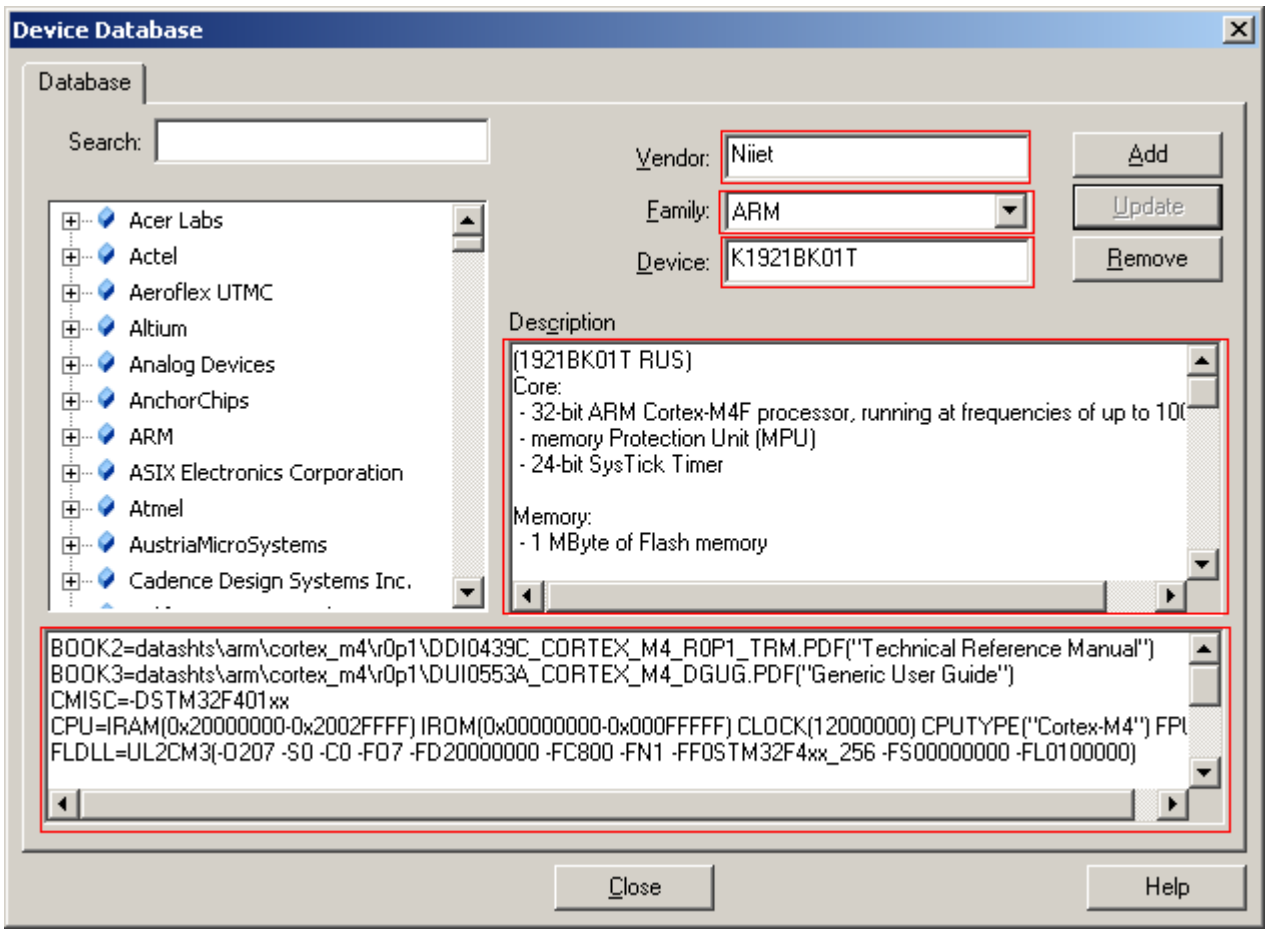


Рисунок 2.1 – Окно добавления МК К1921ВК01Т в базу данных устройств

В появившемся окне вводим:

Vendor: Niiet
Family: ARM
Device: K1921BK01T
Description:
«(1921BK01T RUS)
Core:
- 32-bit ARM Cortex-M4F processor, running at frequencies of up to 100 MHz
- memory Protection Unit (MPU)
- 24-bit SysTick Timer

Memory:
- 1 MByte of Flash memory
- 192 KBytes of SRAM
- external system bus controller (support of SRAM, ROM and NAND Flash)

Supply and Clock rate:
- internal regulator for core voltage supply
- built-in power control system
- internal 5(3.5-6.5) MHz RC clock generator
- external oscillator 12 MHz
- external oscillator 32 KHz
- on-board PLL for clock frequency

Low power consumption modes:
- Sleep, Stop and Standby
- battery domain with RTC and backup registers

Analog features:
- 12 * 12-bit ADC (24 channels) up to 4Ms/sec
- conversion range: 0 to 1.8 V
- internal comparator

Peripheralfeatures:
- DMA controller (24 channel)
- USB 2.0 high-speed OTG controller (Device and Host) with on-chip PHY
- 3 Synch. 16-bit Timers
- 2 * CAN interface (2.0B)
- 4 * UART
- 4 * SPI
- 2 * I2C
- 6 * CAPCOM
- 3 * Analog Comparator
- 9 * PWM with High definition and external synchronization
- 2 * QEP
- 120 GPIO pins

Debug mode:
- 2 * JTAG and SWD»

Ниже поле конфигурации устройства:

```
BOOK2=datashts\arm\cortex_m4\r0p1\DDI0439C_CORTEX_M4_R0P1_TRM.PDF("Technical Reference Manual")
BOOK3=datashts\arm\cortex_m4\r0p1\DUI0553A_CORTEX_M4_DGUG.PDF("Generic User Guide")
CMISC=-DSTM32F401xx
CPU=IRAM(0x20000000-0x2002FFFF) IROM(0x00000000-0x000FFFFF) CLOCK(12000000) CPUTYPE("Cortex-M4")
FPU2
FLDLL=UL2CM3(-O207 -S0 -C0 -FO7 -FD20000000 -FC800 -FN1 -FF0K1921BK01T -FS00000000 -FL0100000)
MON=SARMCM3.DLL("-MPU") TCM.DLL("-pCM4")
REGFILE=K1921BK01T.h("Niiet\K1921BK01T")
SFILE="Startup\Niiet\K1921BK01T\startup_K1921BK01T.s" ("K1921BK01T Startup Code")
SIM=SARMCM3.DLL("-MPU -REMAP") DCM.DLL("-pCM4")
SVD=SFD\Niiet\K1921BK01T\K1921BK01T.SFR
```

И нажимаем «ADD»

После этого микроконтроллер K1921BK01T добавлен в базу данных устройств «LegacyDeviceDatabase».
Теперь при создании проектов выбираем «Project->NewuVisionProject…», указываем директорию для создаваемого проекта, и в появившемся окне «SelectDeviceforTarget ‘Target 1’»выбираем микроконтроллер K1921BK01T (рисунок 2.2)
После этого будет предложено скопировать в новый проект файл startupдля K1921BK01T.

Если Вы используете пробную версию Keil, то в качестве микроконтроллера выберите «ARMCortexM4»-> «ARMCM4_FP». В этом случае придется вручную настроить:

- параметры микроконтроллера (IRAM(0x20000000-0x2002FFFF) IROM(0x00000000-0x000FFFFF)),
- для отладки и просмотра регистров МК дополнительно настраиваем файл System-ViewerFile (*.Sfr) – указываем «ARM\ SFD\Niiet\ K1921BK01T\ K1921BK01T.SFR».
- файл алгоритма программирования загрузочной Flash (указываем путь «ARM\Flash\ K1921BK01T.FLM» выбрать из списка при нажатии на кнопку “Add”, см. рисунок 2.5)
- копировать файлы startup_K1921BK01T.s (расположен в «ARM\Startup\Niiet\ K1921BK01T\ startup_K1921BK01T.s»), K1921BK01T.h (расположен в «ARM\INC\Niiet\ K1921BK01T\ K1921BK01T.h») в директорию нового проекта.

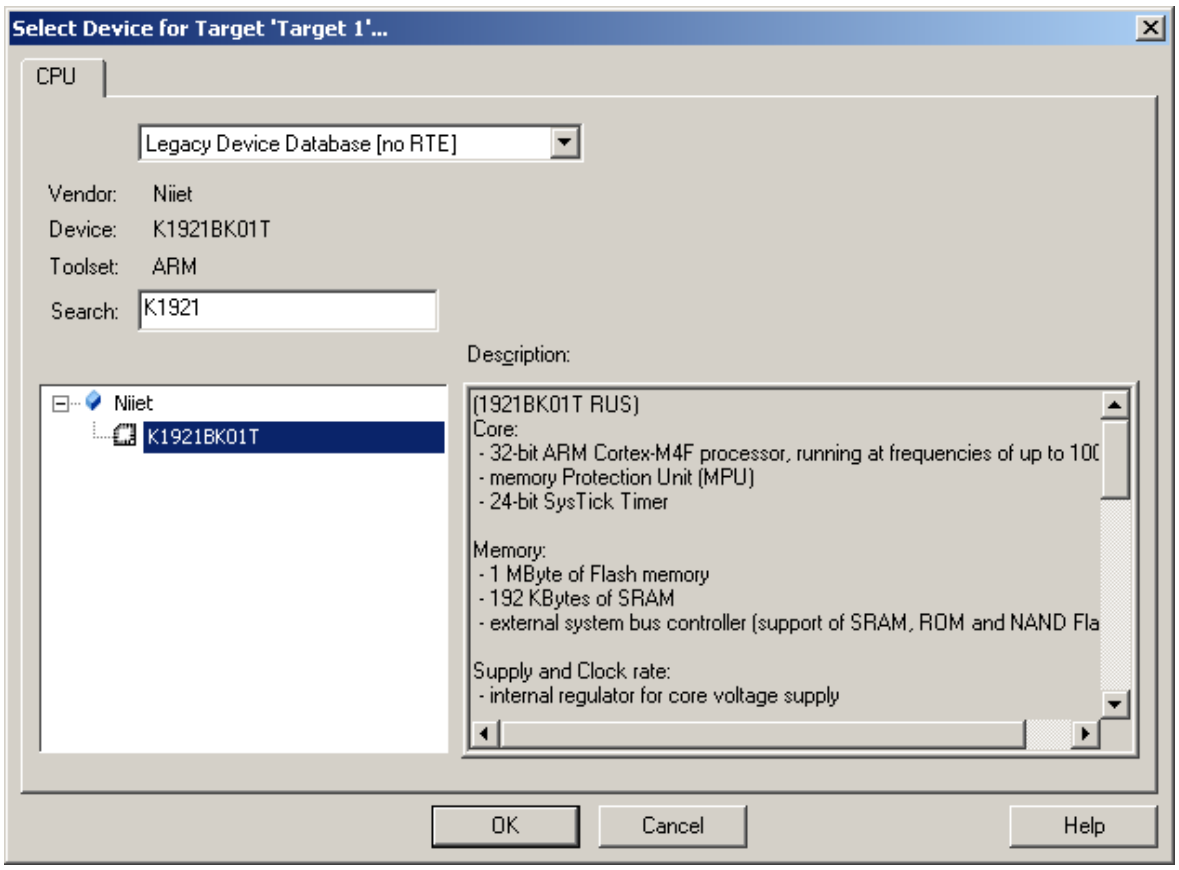


Рисунок 2.2 – Окно выбора МК К1921ВК01Т из базу данных устройств

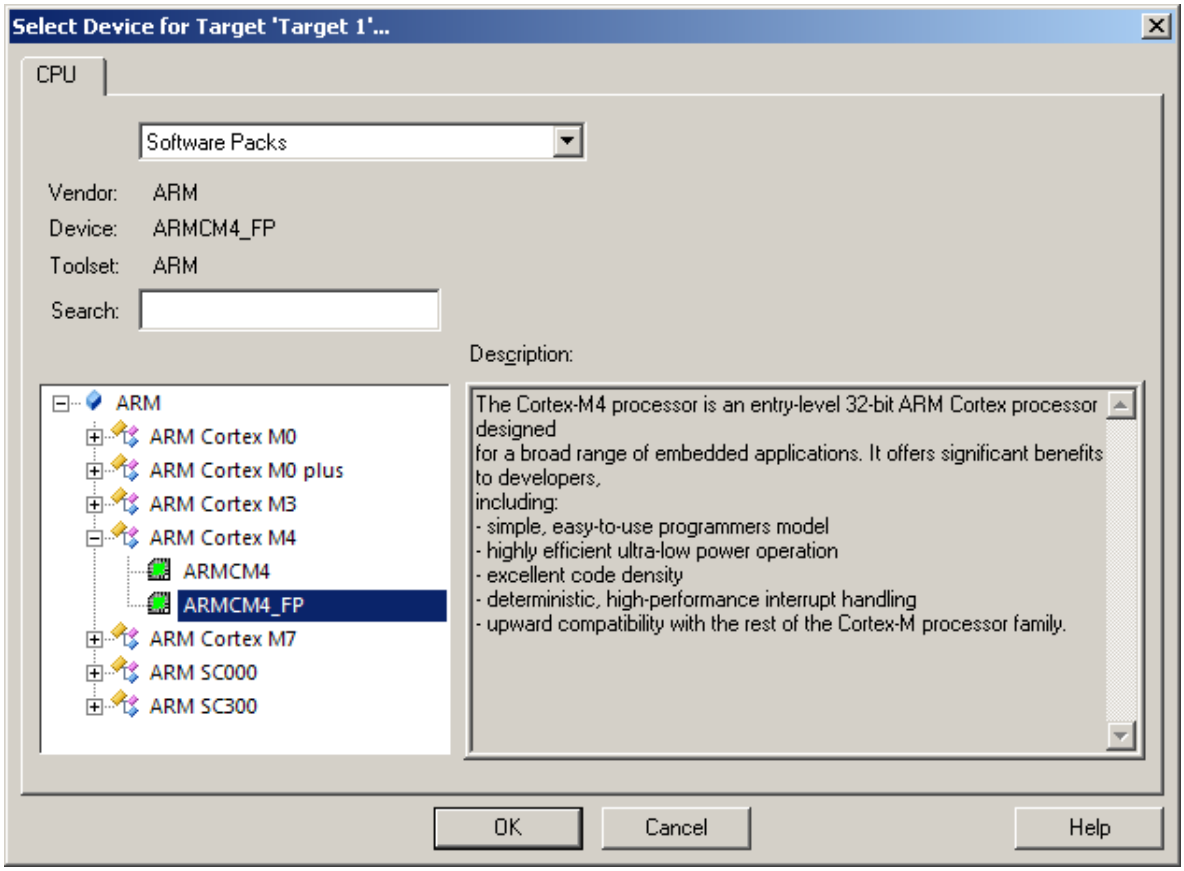


Рисунок 2.3 – Окно выбора типового МК «ARMCM4_FP»

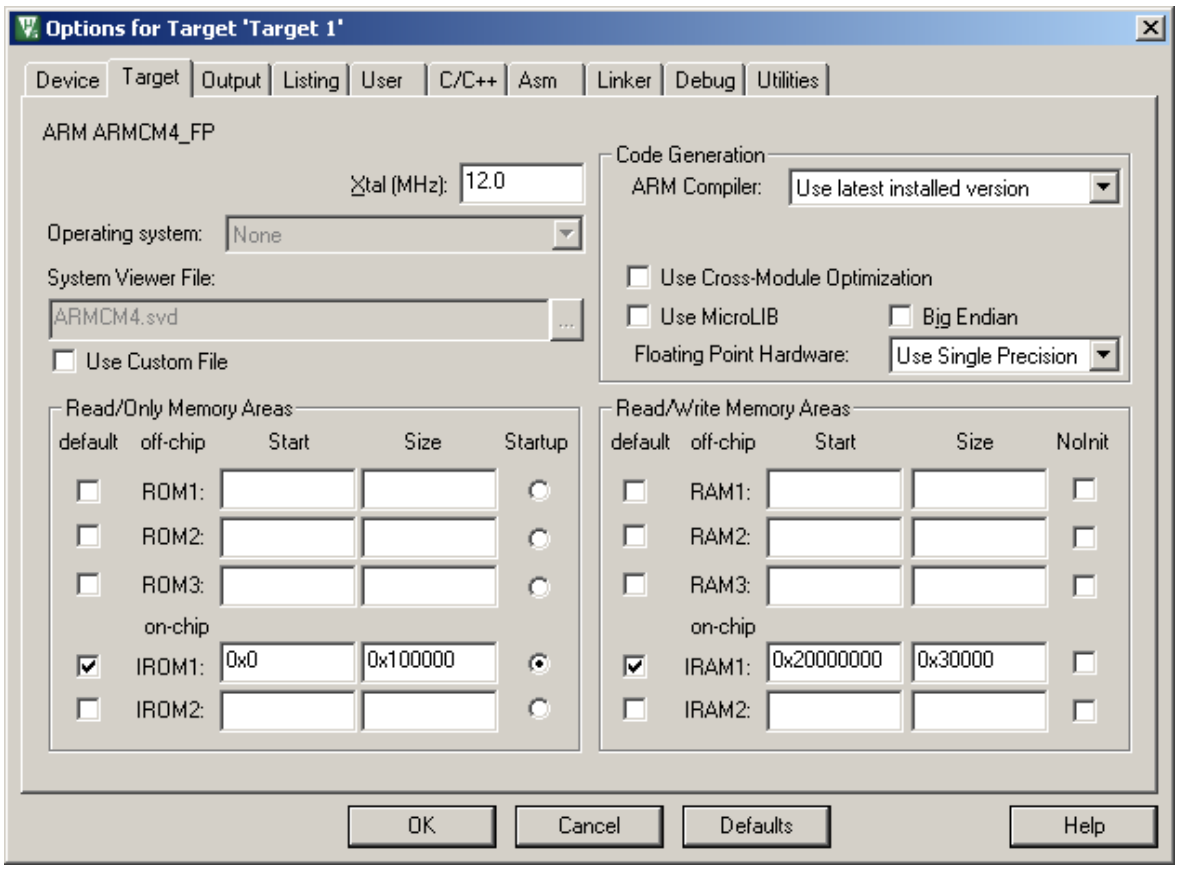


Рисунок 2.4 – Окно конфигурации «OptionsforTarget»

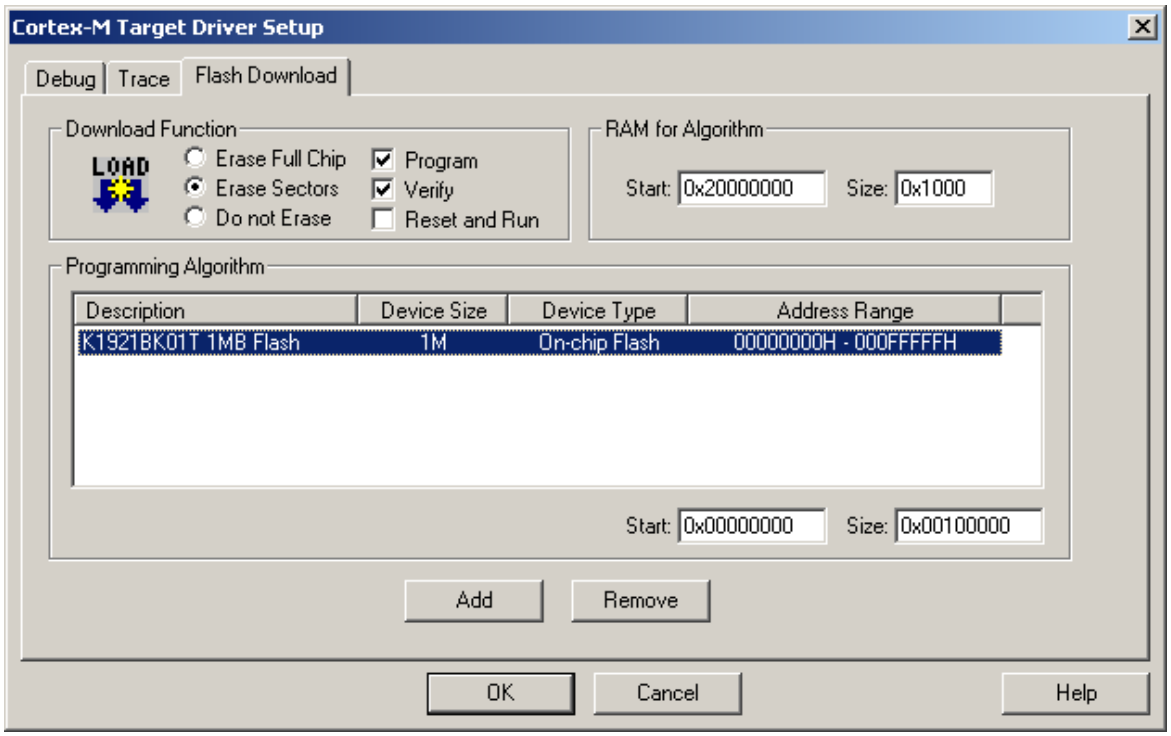


Рисунок 2.5 – Окно конфигурации алгоритма программирования МК К1921ВК01Т

2.2 Настройки проекта

Минимально в проект нужно добавить несколько файлов: «startup_K1921BK01T.s», «system_K1921BK01T.c», «K1921BK01T.h», «main.c».

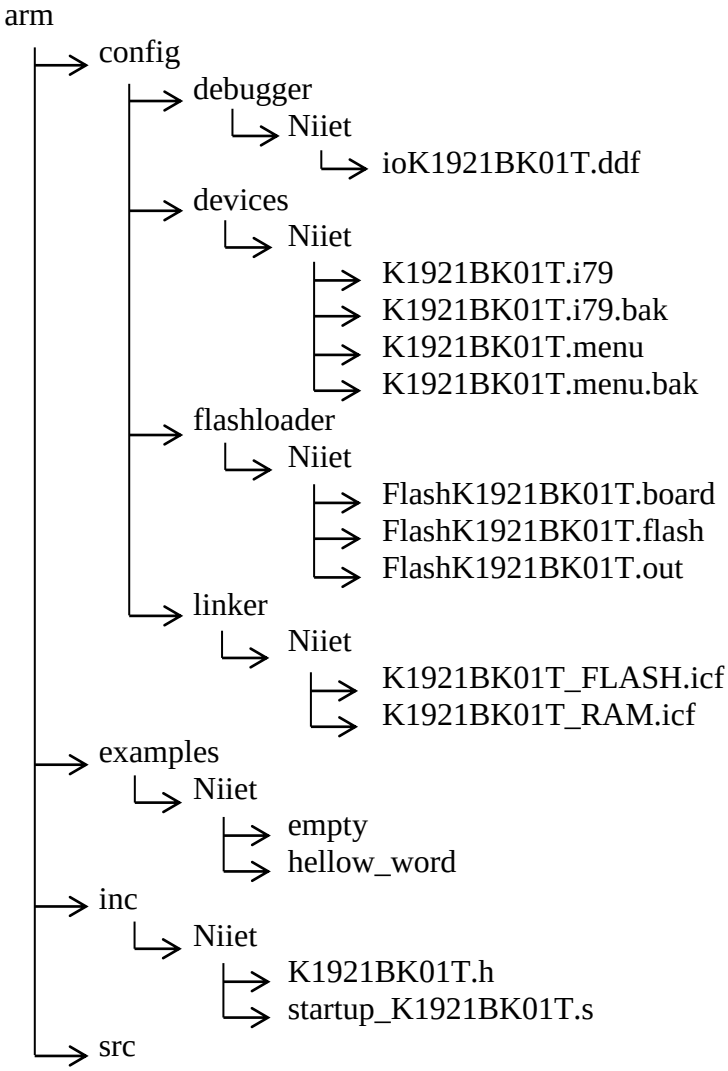
3 Настройки среды IAR

3.1 Введение

Программирование микроконтроллера K1921BK01T рекомендуется производить с использованием среды IAR Embedded Workbench for ARM. Рекомендуемая версия среды – 6.40.2.3992 и более новые.

Настройки создаваемого проекта отличаются в зависимости от того, планируется ли загрузка исполняемой программы во Flash-память микроконтроллера (начиная с адреса 0x00000000), либо в RAM-память (начиная с адреса 0x20000000).

Для разработки проектов в среде IAR необходимо в каталог с установленным ПО добавить следующие файлы каталога arm (прилагаются в архиве)



3.2 Задание настроек проекта

Далее подробно описаны действия по конфигурированию настроек проекта в среде IAREW.

1. Создайте новую рабочую областьWorkspace (File ->New ->Workspace) и внёй создайте новый пустой проект на языкеC (Project ->CreateNewProject).

2. Откройте окно настроек проекта (Project ->Options). Большую часть настроек проекта можно оставить на значениях «по умолчанию» (либо изменить в зависимости от потребностей разработчика). Далее рассматриваются только настройки, требующие каких-либо изменений.

3. В закладке GeneralOptions->Target измените используемый микроконтроллер (Device), из списка выберите «Niiet->K1921BK01T» (рисунок 3.1)

4. Во вкладке Linker->Config задайте icf-файл. Содержимое этого файла зависит от того, загружается ли программа во Flash либо в RAM. После выбора микроконтроллера данных параметр автоматически настраивается на загрузку программы во Flash (указан файл: «$TOOLKIT_DIR$\config\linker\Niiet\K1921BK01T_FLASH.icf»). Если необходима загрузка программы в RAM – необходимо вручную указать путь: «$TOOLKIT_DIR$\config\linker\Niiet\K1921BK01T_RAM.icf» (рисунок 3.2)

5. В случае загрузки программы во Flash-память микроконтроллера необходимо воспользоваться специально написанным для этой цели загрузчиком (Flashloader), который расположен в каталоге «$TOOLKIT_DIR$\config\flashloader\ Niiet\FlashK1921BK01T.flash» . Если загрузка программы предполагается во Flash, во вкладке Debugger ->Download поставьте галочку в окне Useflashloader(s), рисунок 3.3. Если же загрузка программы предполагается в RAM, данные настройки следует оставить без изменения.

6. Выберите используемый отладчик. Во вкладке Debugger->Setupустановите в окне Driver используемый отладчик, например J-Link/J-Trace (рисунок 3.4).

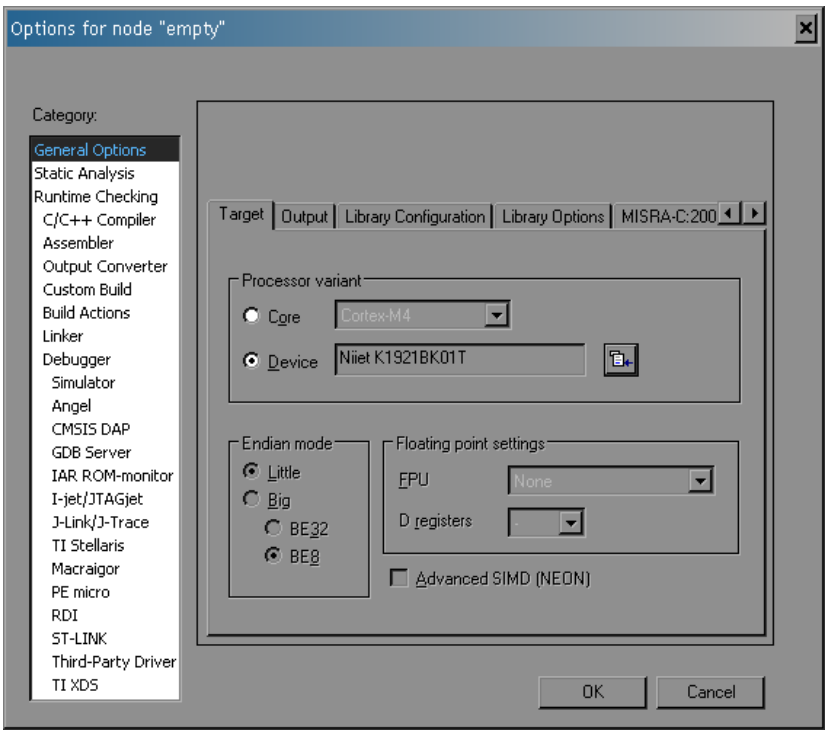


Рисунок 3.1 – Окно настроек проекта. Выбор устройства

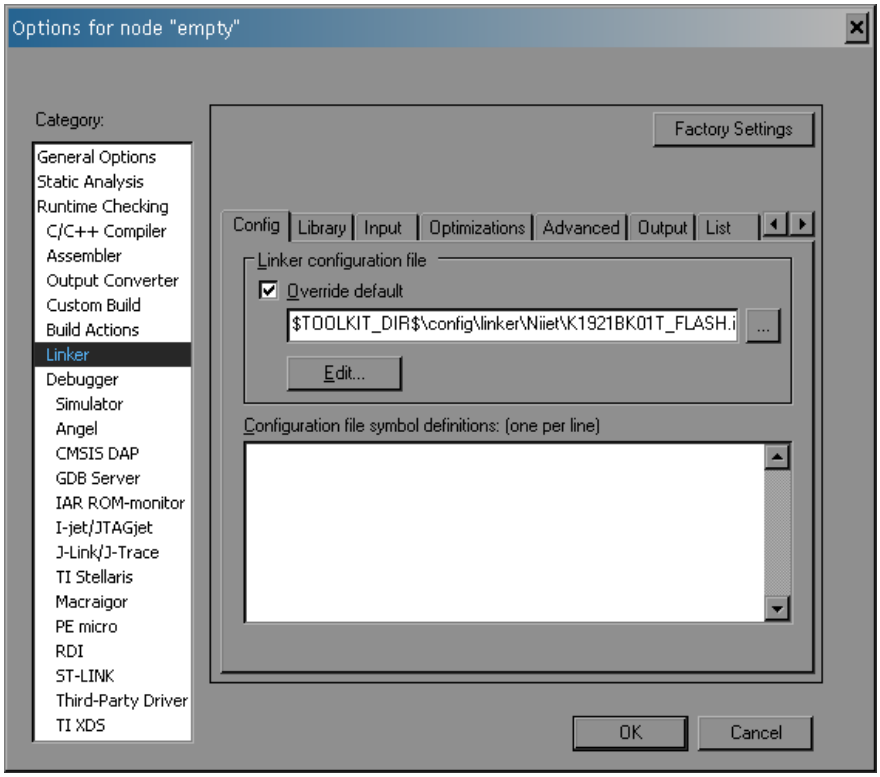


Рисунок 3.2 – Окно настроек проекта. Настройки Linker

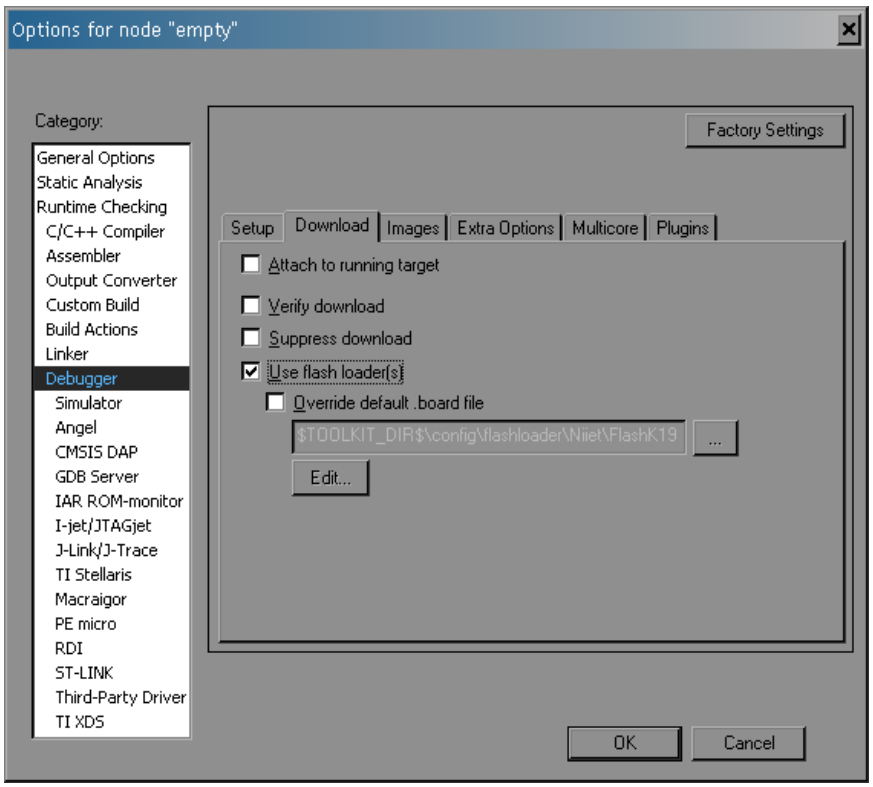


Рисунок 3.3 – Окно настроек проекта. Настройки Debugger (flashloader)

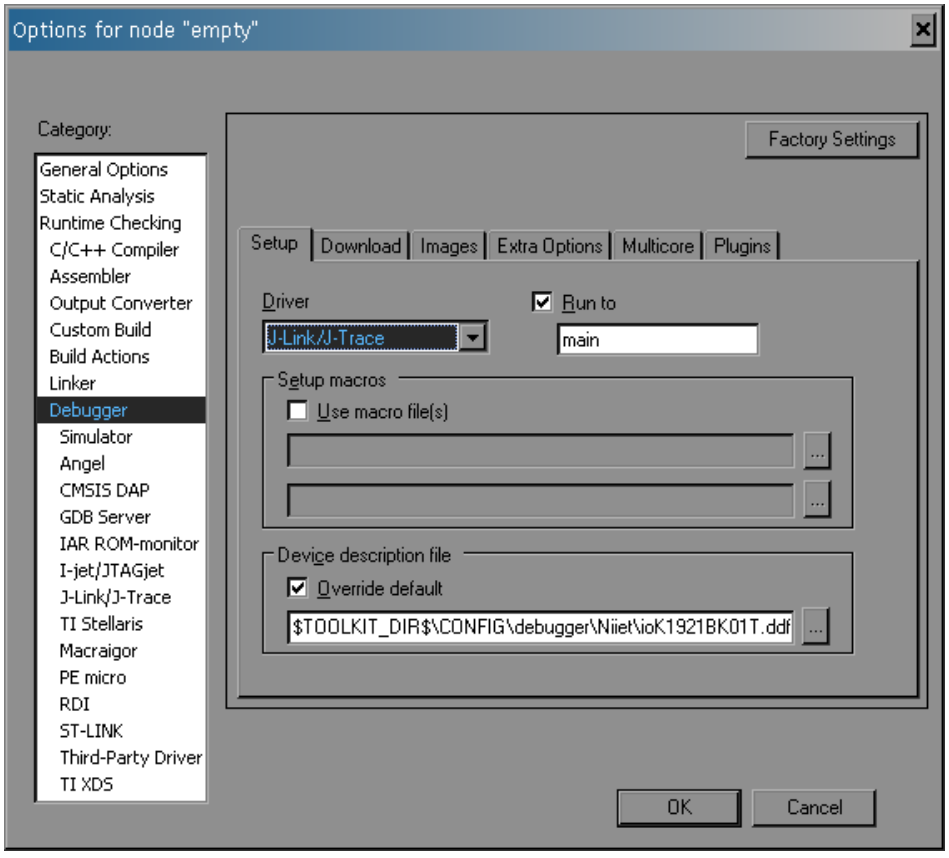


Рисунок 3.4 – Окно настроек проекта. Настройки Debugger

Минимальный проект на Си

```
#include <Niiet\K1921BK01T.h>

int main()

{

   while(1);

}
```

3.3 Описание Flash-загрузчика

Особенностью загрузки программы во Flash-память является то, что доступ на запись в ячейки памяти по адресам в диапазоне 0x00000000 – 0x000FFFFF возможен только с использованием регистров K1921BK01T, что невозможно одними лишь средствами стандартного загрузчика. Регистры управления записью и чтением в загрузочную область памяти расположены по адресу 0xA001C000. С помощью этих регистров программа загрузки выполняет следующие действия:

- FlashInit: Полное стирание Flash-памяти (mass_erase) в начальный момент работы загрузчика
- Далее, для каждой страницы памяти, которая будет занята программой:

1. FlashErase: Постраничное стирание (page_erase)
2. FlashWrite: Запись данных на данную страницу (write)

4 Настройка среды CodeMaster++[ARM]

4.1 Краткое описание среды CodeMaster++[ARM]

Среда CodeMaster++[ARM] изначально разработана для создания и отладки проектов с МК К1921ВК01Т, поэтому особых настроек не требуется.

Среда разработки CodeMaster++[ARM] включает в себя менеджер проектов, редактор исходных кодов, компилятор (C, C++), средства отладки и симуляции микроконтроллера 1921ВК01Т. Среда позволяет осуществлять отладку программ, а также программирование микроконтроллера посредством адаптера JEM-NT-CM4

Адаптер JEM-NT-CM4 обеспечивает взаимодействие между интегрированной средой разработки CodeMaster++[ARM], установленной на персональном компьютере, и отладочными ресурсами, встроенными в микроконтроллер К1921ВК01Т, а также выполнение отладочных функций.

Информационный обмен JEM-NT-CM4 при взаимодействии с целевым микроконтроллером осуществляется по одному из отладочных портов: JTAG или SWD.

Пользовательский интерфейс отладочных функций, реализуемых JEM-NT-CM4, обеспечивается интегрированной средой разработки CodeMaster++[ARM].

Отладка пользовательской программы предполагает 2 основных режима работы: выполнение программы в режиме реального времени (RUN) и останов программы (HALT) на определённом адресе или в определённый момент выполнения. Большинство отладочных функций доступно исключительно в режиме останова. В этом режиме отладчик JEM-NT-CM4 во взаимодействии с CodeMaster++[ARM] позволяет анализировать и изменять ход исполнения пользовательской программы между отдельными участками программы, исполняемыми в RUN.

При работе с пользовательской программой JEM-NT-CM4 обеспечивает выполнение следующих отладочных действий:

сброс МК с остановом пользовательской программы на начальном адресе;
запуск на выполнение программы в режиме реального времени (RUN);
останов программы в произвольный момент времени (STOP);
определение и изменение адреса выполнения программы (чтение и запись PC);
чтение и запись доступных ресурсов МК (Flash, RAM, SFR и т.д.);
установку и снятие точек останова по адресу выполнения программы;
запуск на выполнение программы до определённого места в исходном коде (до курсора, до адреса);
пошаговое исполнение программы: шаги низкого и высокого уровней, с заходом и без захода в подпрограммы.

Среда разработки включает:

- Компилятор C/C++
- Поддержка микроконтроллера К1921ВК01Т
- Симулятор
- Удобный набор отладочных средств: (окна «Дамп памяти», «Анализатор выполнения/доступа», «Распределение памяти», «Дисассмеблер», «Регистры»)

4.2 Задание настроек проекта

Далее подробно описаны действия по конфигурированию настроек проекта в среде CodeMaster++[ARM].

1. Создайте новую рабочую область (Проект->Рабочая область-> Создать) (рисунок 4.1) и в ней создайте новый пустой проект (нажатием правой клавиши мыши в окне рабочей области вызовите контекстное меню и выберите «Добавить->Новый проект») (рисунок 4.2).

2. Откройте окно настроек проекта (Проект->Настройки активного проекта) (рисунок 4.2), нажмите «Опции Микроконтроллера». В появившемся окне перейдите на вкладку «Микроконтроллер» (Рисунок 4.3) и выберите микроконтроллер «NIIET K1921BK01T»

3. Переход в режим отладки с помощью JEM-NT-CM4 осуществляется из меню «Отладка->Выбрать отладчик», в появившемся окне выбираем «JTAG-эмулятор JEM-NT-CM4»

4. Программирование микроконтроллера осуществляется из меню «Компиляция->Собрать проект и запустить отладку[Shift+F7]»

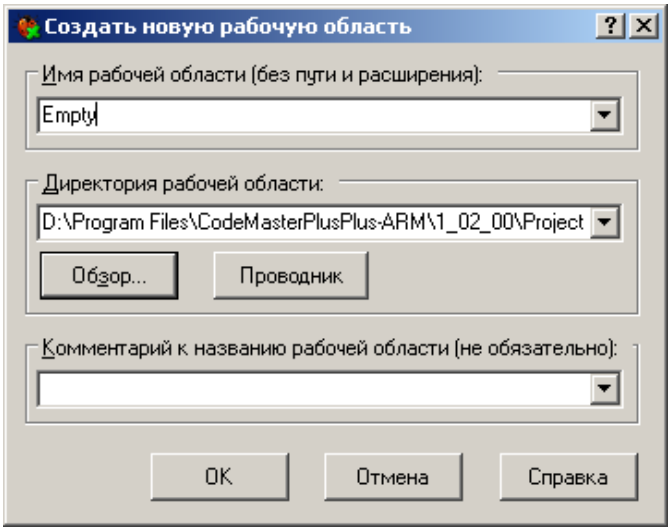


Рисунок 4.1 – Окно создания рабочей области

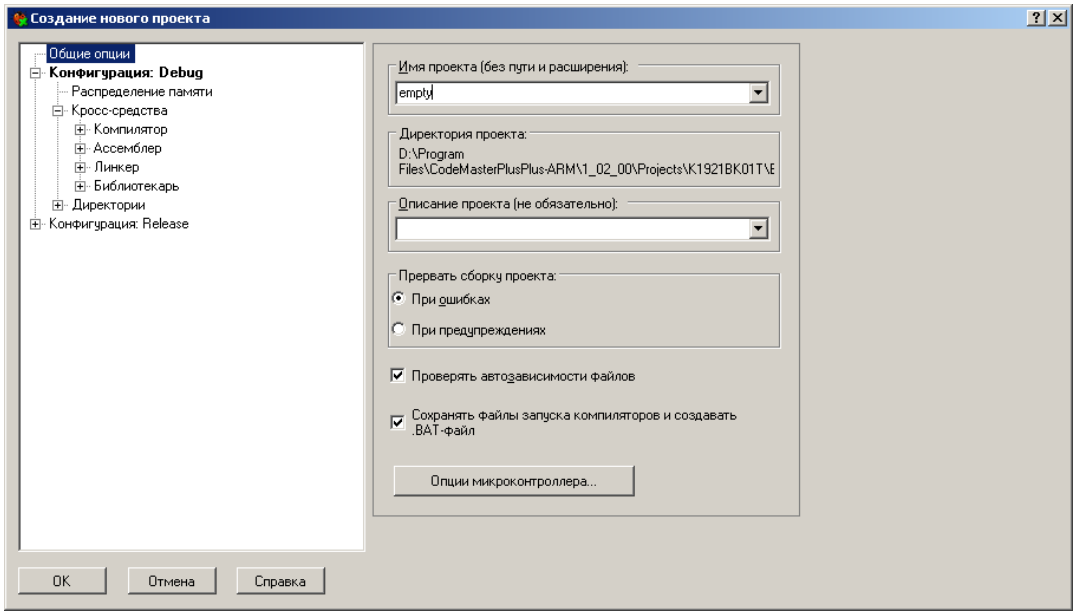


Рисунок 4.2 – Окно свойств проекта

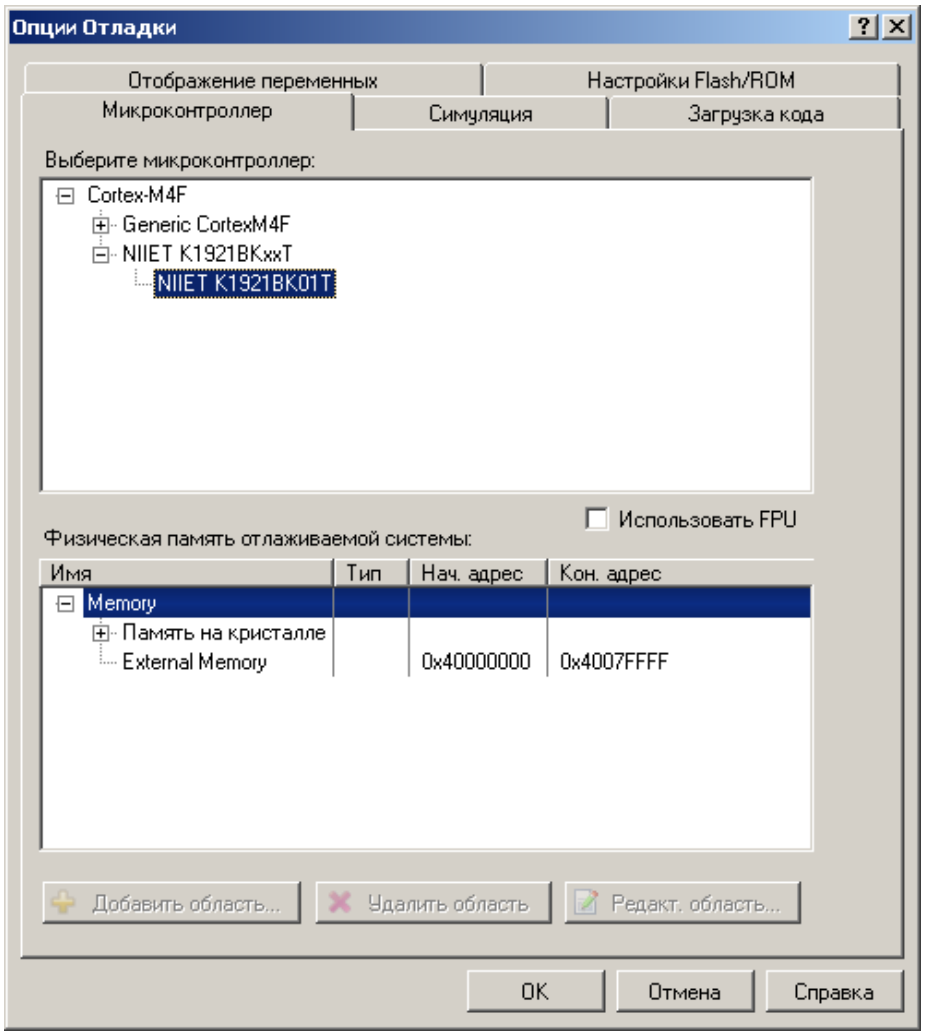


Рисунок 4.3 – Окно настроек микроконтроллера

5 Краткая информация по работе с МК

5.1 Конфигурационное слово

Помимо основной области пользовательская флеш-память содержит информационный блок объемом 512 байт (две страницы), в котором хранятся настройки доступа к страницам загрузочной и пользовательской флеш-памятей, а также параметры загрузки микроконтроллера. Доступ к информационному блоку осуществляется битами READ_IFB, WRITE_IFB и PAGEERASE_IFB регистра UFMC. По адресу 000h располагается информационное слово INFO_WORD, состоящее из четырех байт.

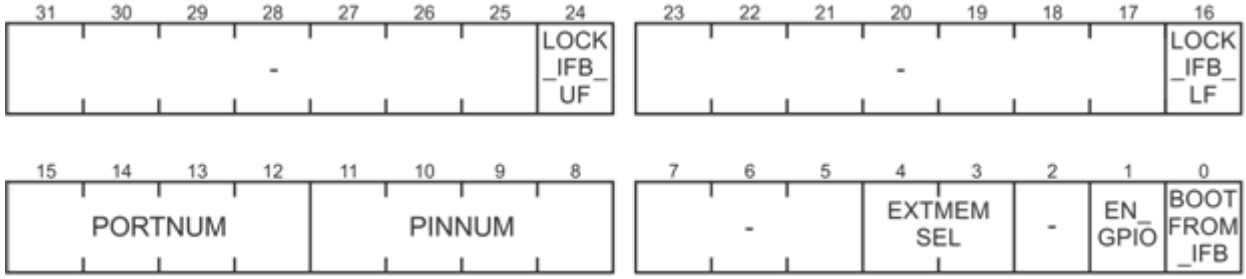
В диапазоне адресов с 040h по 050h расположены биты защиты страниц загрузочной флеш-памяти. Установка или сброс битов позволяет запрещать или разрешать доступ к страницам памяти для записи и стирания.

В диапазоне адресов с 080h по 0A0h расположены биты защиты страниц пользовательской флеш-памяти. Установка или сброс битов позволяет запрещать или разрешать доступ к страницам памяти для записи и стирания.

Остальной объем информационного блока – регистры пользователя.

Таблица 5.1 – Информационное слово пользовательской флеш

INFO_WORD +0000h (смещение относительно начального адреса пользовательской флеш-памяти)



Поле	Биты	Описание	
LOCK_ IFB_UF	24	Бит включения защиты пользовательской флеш	
		0	Запрет записи и стирания информационного блока пользовательской флеш.
		1	Защита выключена (по умолчанию)
LOCK_ IFB_LF	16	Бит включения защиты загрузочной флеш	
		0	Запрет записи и стирания информационного блока загрузочной флеш
		1	Защита выключена (по умолчанию)
PORTNUM, PINNUM	15-12, 11-8	Поле указания вывода микроконтроллера, выбирающего источник загрузки. Поле PORTNUM задает номер порта (от 000b до 111b), поле PINNUM – номер вывода	
EXTMEM SEL	4-3	Выбор номера функции выводов GPIO, на которые подключен интерфейс внешней памяти	
		10	Активна третья функция выводов
		00, 01, 11	Активна первая функция выводов
EN_GPIO	1	Бит включения функции выбора мапируемой памяти	
		0	Включена функция выбора вывода микроконтроллера, которая определяет, что мапируется в глобальный адрес памяти 0000h – флеш-память или внешняя память. Для указания вывода микроконтроллера используются поля PORTNUM и PINNUM
		1	В адрес 0000h мапируется флеш-память (по умолчанию)
BOOT FROM_IFB	0	Бит выбора блока памяти для мапирования в адрес 0000h основного блока загрузочной флеш-памяти	
		0	Информационный блок загрузочной флеш-памяти мапируется в основной блок в область 0000h – 1FFFh. И далее с адреса 0000h стартует программа пользователя. Основной блок в этом случае начинается с 2000h.
		1	Старт программы с адреса 0000h основного блока загрузочной флеш. По умолчанию, бит установлен.

5.2 Очистка внутренней памяти

Алгоритм стирания всей внутренней флеш-памяти

1 Во время сброса микроконтроллера анализируется состояние вывода H2. Если вывод находится в состоянии логической единицы (подтянут к 3,3 В), то загрузочная и пользовательская флеш-памяти (включая защищенный и информационный блоки) переводятся в режим, в котором чтение запрещено (при чтении возвращаются нули).

2 Далее по отладочному интерфейсу (SWD или JTAG) должна быть подана команда записи значения 00000001h в регистр FLASH_FULL_ERASE, после чего будет активировано полное стирание.

Примечание – Если полное стирание не требуется во время сброса, на выводе H2 должен удерживаться логический ноль.

5.3 Порты ввода-вывода

В состав микроконтроллера входят порты ввода-вывода: 8-разрядный Порт H и семь 16-разрядных портов – Порт A, Порт B, Порт C, Порт D, Порт E, Порт F, Порт G. Структуры всех портов и функционирование идентичны.

Полученные данные сохраняются в регистре порта DATA. Данные для передачи записываются в регистр порта DATAOUT.

Каждый цифровой вывод порта микроконтроллера может использоваться как двунаправленный вывод общего назначения (режим GPIO). Помимо этого все выводы имеют альтернативные функции (от одной до трех, а выводы A0 – A7 по четыре).

Таблица 5.2 – Описание альтернативных функций порта A

Вывод	**Функция 1**	**Функция 2**	**Функция 3**	**Аналоговая функция**
A[0]	CLK_USB	SPI_FSS[2]	MEM_DATA[12]	
A[1]	SPI_TxD[0]	PWMxB[8]	-	CMP_C3+
A[2]	PWM_TZ[2]	SPI_TxD[1]	MEM_Oen[0]	CMP_C3-
A[3]	UART_TxD[1]	PWMxA[6]	MEM_Oen[1]	CMP_C2+
A[4]	UART_RxD[1]	PWMxB[6]	-	CMP_C2-
A[5]	SPI_FSS[0]	PWMxA[7]	-	CMP_C1+
A[6]	SPI_CLK[0]	PWMxB[7]	-	CMP_C1-
A[7]	SPI_RxD[0]	PWMxA[8]	-	CMP_DACSUP
A[8]	MEM_ADDR[0]	MII_TXCLK	PWM_TZ[0]	
A[9]	MEM_ADDR[1]	MII_TXD[0]	PWM_TZ[1]	
A[10]	MEM_ADDR[2]	MII_TXD[1]	PWMxB[0]	
A[11]	MEM_ADDR[3]	MII_TXD[2]	PWMxB[1]	
A[12]	MEM_ADDR[4]	MII_TXD[3]	PWMxB[2]	
A[13]	MEM_ADDR[5]	MII_TX_EN	PWMxB[3]	
A[14]	MEM_ADDR[6]	MII_TX_ER	PWMxB[4]	
A[15]	MEM_ADDR[7]	MII_CRS	PWMxB[5]	

Таблица 5.3 – Описание альтернативных функций порта B

Вывод	Функция 1	Функция 2	Функция 3
B[0]	JTAG_TDI	SPI_FSS[1]	MEM_Wen
B[1]	JTAG_TMS/SWDIO	SPI_CLK[1]	MEM_Cen[0]
B[2]	JTAG_TCK/SWCLK	SPI_RxD[1]	MEM_Cen[1]
B[3]	MII_TXCLK	CAN_TX[0]	MEM_ADDR[0]
B[4]	MEM_ADDR[8]	MII_COL	PWMxB[6]
B[5]	MEM_ADDR[9]	MII_MDC	PWMxA[7]
B[6]	MEM_ADDR[10]	MII_MDIO	PWMxB[7]
B[7]	MEM_ADDR[11]	MII_RXCLK	PWMxA[8]
B[8]	MEM_ADDR[12]	MII_RXD[0]	PWMxB[8]
B[9]	MEM_ADDR[13]	MII_RXD[1]	PWM_SYNCI
B[10]	MEM_ADDR[14]	MII_RXD[2]	CMP_OUT[0]
B[11]	MEM_ADDR[15]	MII_RXD[3]	CMP_OUT[1]
B[12]	MEM_ADDR[16]	MII_RX_DV	CMP_OUT[2]
B[13]	MEM_ADDR[17]	MII_RX_ER	SPI_FSS[2]
B[14]	MEM_ADDR[18]	CAN_TX[0]	SPI_CLK[2]
B[15]	MEM_DATA[0]	CAN_RX[0]	SPI_RxD[2]

Таблица 5.4 – Описание альтернативных функций порта C

Вывод	Функция 1	Функция 2	Функция 3
C[0]	MII_TXD[0]		MEM_ADDR[1]
C[1]	MII_TXD[1]	CAN_TX[1]	MEM_ADDR[2]
C[2]	MII_TXD[2]	CAN_RX[1]	MEM_ADDR[3]
C[3]	MII_TXD[3]	UART_TxD[0]	MEM_ADDR[4]
C[4]	MII_TX_EN	UART_RxD[0]	MEM_ADDR[5]
C[5]	MII_TX_ER	UART_TxD[2]	MEM_ADDR[6]
C[6]	MEM_DATA[1]	UART_RxD[1]	SPI_TxD[2]
C[7]	MEM_DATA[2]	UART_TxD[1]	PWM_TZ[2]
C[8]	MEM_DATA[3]	SPI_FSS[0]	PWM_TZ[4]
C[9]	MEM_DATA[4]	SPI_CLK[0]	PWM_TZ[5]
C[10]	MEM_DATA[5]	SPI_RxD[0]	MII_TXCLK
C[11]	MEM_DATA[6]	SPI_TxD[0]	MII_TXD[0]
C[12]	MEM_DATA[7]	Timer_IN[0]	MII_TXD[1]
C[13]	MEM_DATA[8]	Timer_IN[1]	MII_TXD[2]
C[14]	MEM_DATA[9]	Timer_IN[2]	MII_TXD[3]
C[15]	MEM_DATA[10]	NMI	MII_TX_EN

Таблица 5.5 – Описание альтернативных функций порта D

Вывод	Функция 1	Функция 2	Функция 3
D[0]	MII_CRS	UART_RxD[2]	MEM_ADDR[7]
D[1]	MII_COL	UART_RxD[3]	MEM_ADDR[8]
D[2]	MII_MDC	UART_TxD[3]	MEM_ADDR[9]
D[3]	MII_MDIO	-	MEM_ADDR[10]
D[4]	MII_RXCLK	-	MEM_ADDR[11]
D[5]	MII_RXD[0]	-	MEM_ADDR[12]
D[6]	MII_RXD[1]	-	MEM_DATA[0]
D[7]	MII_RXD[2]	-	MEM_DATA[1]
D[8]	MII_RXD[3]	-	MEM_DATA[2]
D[9]	MII_RX_DV	scl[1]	MEM_DATA[3]
D[10]	MII_RX_ER	sda[1]	MEM_DATA[4]
D[11]	UART_TxD[0]	CAN_TX[1]	eCMP_OUT[0]
D[12]	MEM_DATA[11]	UART_TxD[2]	MII_TX_ER
D[13]	MEM_DATA[12]		MII_CRS
D[14]	MEM_DATA[13]		MII_COL
D[15]	MEM_DATA[14]		MII_MDC

Таблица 5.6 – Описание альтернативных функций порта E

Вывод	Функция 1	Функция 2	Функция 3
E[0]	UART_RxD[0]	scl[0]	CMP_OUT[1]
E[1]	JTAG_TRST	sda[0]	UART_TxD[2]
E[2]	CAN_RX[0]	PWMxB[5]	MEM_DATA[6]
E[3]	NMI	UART_RTS[0]	MEM_DATA[7]
E[4]	QEPxA/XCLK[0]	CAPxPWM[0]	Timer_IN[0]
E[5]	QEPxB/XDIR[0]	CAPxPWM[1]	MEM_LBn
E[6]	QEPXI[0]	CAPxPWM[2]	MEM_DATA[8]
E[7]	USB_DRVVBUS	QEPXS[0]	PWM_TZ[3]
E[8]	PWM_TZ[0]	UART_RI[0]	SPI_FSS[1]
E[9]	PWM_TZ[1]	UART_DTR[0]	SPI_CLK[1]
E[10]	JTAG_TDO/SWO	PWM_SYNCI	MEM_Ubn
E[11]	CMP_OUT[2]	PWMxB[8]	MEM_DATA[10]
E[12]	MEM_DATA[15]	PWM_TZ[3]	MII_MDIO
E[13]	MEM_Wen	PWM_TZ[4]	MII_RXCLK
E[14]	MEM_Oen[0]	PWM_TZ[5]	MII_RXD[0]
E[15]	MEM_Oen[1]	MII_RXD[1]	

Таблица 5.7 – Описание альтернативных функций порта F

Вывод	Функция 1	Функция 2	Функция 3
F[0]	PWMxB[0]	UART_DSR[0]	SPI_TxD[1]
F[1]	CMP_OUT[1]	UART_CTS[0]	MEM_DATA[11]
F[2]	PWMxB[1]	SPI_CLK[3]	CAN_RX[0]
F[3]	PWMxB[3]	SPI_RxD[2]	MEM_DATA[14]
F[4]	PWMxB[2]	SPI_TxD[3]	UART_RxD[2]
F[5]	scl[0]	PWMxA[7]	MEM_DATA[9]
F[6]	MEM_Cen[0]	CMP_OUT[0]	MII_RXD[2]
F[7]	MEM_Cen[1]	CMP_OUT[1]	MII_RXD[3]
F[8]	MEM_LBn	CMP_OUT[2]	MII_RX_DV
F[9]	MEM_Ubn	UART_CTS[2]	MII_RX_ER
F[10]	UART_TxD[2]		UART_CTS[1]
F[11]	UART_RxD[2]		UART_DCD[1]
F[12]	UART_TxD[3]	UART_RI[2]	UART_DSR[1]
F[13]	UART_RxD[3]	UART_DTR[2]	UART_RI[1]
F[14]	CAN_TX[1]	UART_RTS[2]	UART_DTR[1]
F[15]	CAN_RX[1]	UART_DCD[2]	UART_RTS[1]

Таблица 5.8 – Описание альтернативных функций порта G

Вывод	Функция 1	Функция 2	Функция 3
G[0]	sda[0]	PWMxB[7]	CAN_TX[0]
G[1]	CMP_OUT[0]	PWMxA[8]	QEPXS[0]
G[2]	PWMxA[0]	UART_DCD[0]	SPI_RxD[1]
G[3]	PWMxA[1]	SPI_FSS[3]	CAN_TX[1]
G[4]	PWMxA[2]	SPI_RxD[3]	UART_TxD[2]
G[5]	PWMxA[3]	SPI_CLK[2]	MEM_DATA[13]
G[6]	PWMxA[4]	SPI_TxD[2]	MEM_DATA[15]
G[7]	CAN_TX[0]	PWMxA[5]	MEM_DATA[5]
G[8]	Timer_IN[1]	UART_DSR[2]	CAN_RX[1]
G[9]	PWM_SYNCI	QEPXS[1]	UART_RTS[3]
G[10]	CAPxPWM[3]	QEPxA/XCLK[1]	
G[11]	CAPxPWM[4]	QEPxB/XDIR[1]	Timer_IN[1]
G[12]	CAPxPWM[5]	QEPXI[1]	Timer_IN[2]
G[13]		PWM_TZ[0]	PWMxB[4]
G[14]		PWM_TZ[1]	PWMxA[6]
G[15]		PWM_TZ[2]	PWMxB[6]

Таблица 5.9 – Описание альтернативных функций порта H

Вывод	Функция 1	Функция 2	Функция 3
H[0]		SYSCLK	PWMxA[7]
H[1]			PWMxB[7]
H[2]	PWMxB[5]		
H[3]	PWMxB[4]		UART_CTS[3]
H[4]	PWMxA[6]		UART_DCD[3]
H[5]	PWMxB[6]		UART_DSR[3]
H[6]	PWMxA[7]		UART_RI[3]
H[7]	PWMxB[7]		UART_DTR[3]

Примечание: вывод H[2] также используется для активации режима полного стирания всех блоков внутренней Flash памяти.

Управление выводами осуществляется посредством регистров общего назначения и регистров GPIO.

На рисунке 5.1 приведена структурная схема нулевого вывода цифрового порта A микроконтроллера. Серым цветом отмечены нулевые биты регистров управления. Схемы всех выводов идентичны.

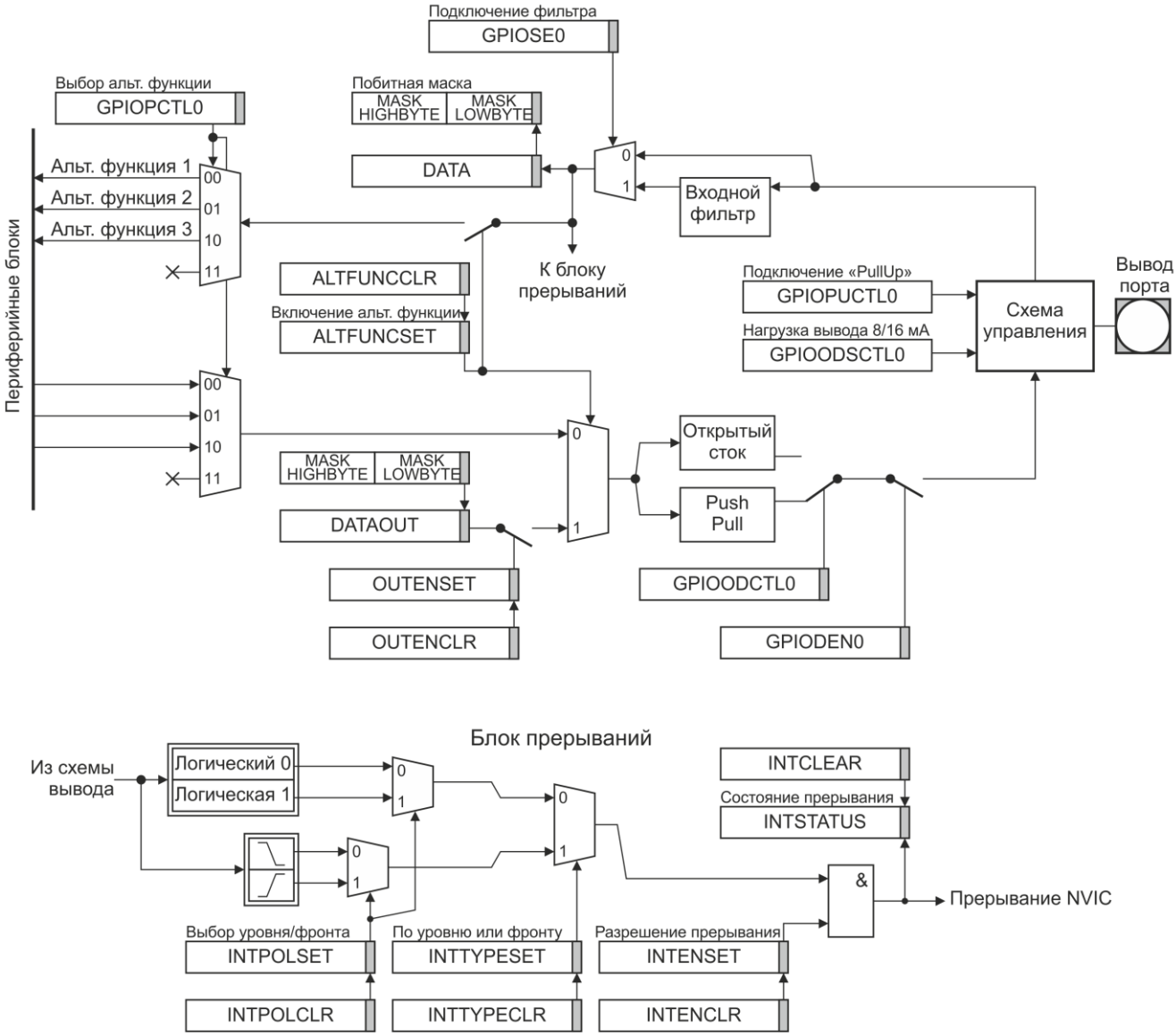


Рисунок 5.1 – Вывод цифрового порта микроконтроллера и управляющие регистры

Схема состоит из двунаправленной площадки вывода, входного фильтра, мультиплексора выбора номера альтернативной функции, мультиплексора выбора режима работы (режим GPIO либо режим альтернативной функции).

Для каждого вывода задается режим работы, номер альтернативной функции, нагрузочная способность вывода, режим подтяжки (PullUp), а также производится настройка порта на работу в режиме с открытым коллектором. Входной сигнал может подаваться для дальнейшей обработки как напрямую (асинхронный вход), так и проходить обработку через фильтр.

После сброса все выводы кроме выводов JTAG, выводов D11 и E0 конфигурируются как выводы общего назначения (режим GPIO). Направление работы выводов определяется состоянием бит регистра OUTENSET (для сброса установленных бит следует записать единицы в регистр OUTENCLR).

Для перевода желаемого вывода порта в режим альтернативной функции необходимо установить соответствующий бит в регистре ALTFUNCSET порта. Для отключения альтернативной функции нужно записать единицу в соответствующий бит регистра ALTFUNCCLR. Выбор номера альтернативной функции осуществляется посредством регистра GPIOPCTLx. Каждому выводу соответствуют два бита регистра.

На рисунке 5.2 показана функциональная схема блока управления альтернативной функцией вывода.

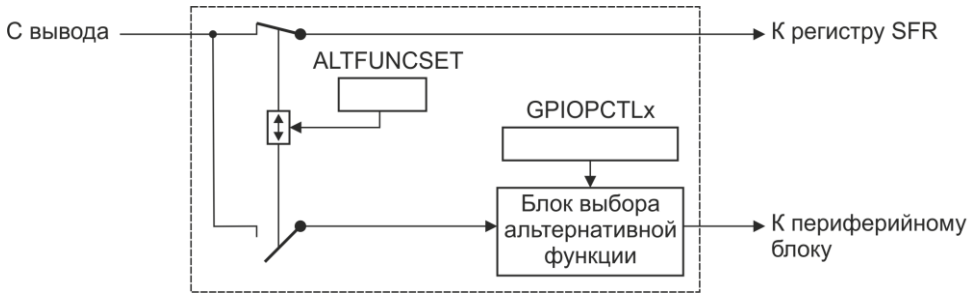


Рисунок 5.2 – Блок управления альтернативной функцией

Входы и выходы периферийных блоков в процессе работы коммутируются с выводами микроконтроллера при условии, что для этих выводов включен режим альтернативной функции. В связи с этим периферийный блок может передавать информацию на несколько выводов одновременно. В то же время прием информации может осуществляться только с одного вывода, во избежание конфликтов уровней сигналов (для этого дополнительно предусмотрена система приоритета альтернативных функций). Количество выводов, сигналы с которых могут быть переданы на периферийный блок, для каждого блока различно (от одного до трех).

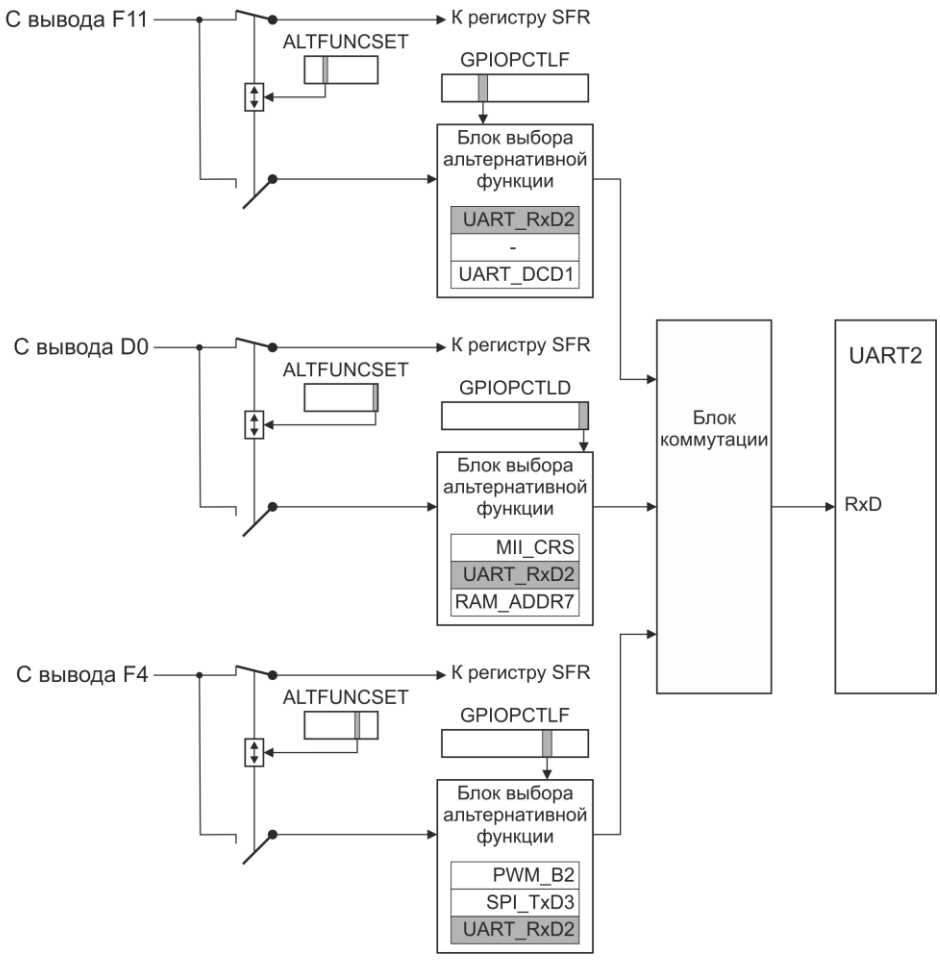


Рисунок 5.3 – Схема соединения блоков управления альтернативными функциями выводов микроконтроллера и блока UART2

Для примера, рассмотрим схему взаимодействия входа RxD блока UART2 и выводов микроконтроллера, запрограммированных на прием внешних сигналов (см. рисунок 5.3).

Блок UART2 может принимать информацию с одного из выводов F11, D0 или F4. Режим альтернативной функции может быть включен как для одного вывода (на который поступают внешние данные), так и для всех трех. Номер альтернативной функции UART_RxD2 для каждого из выводов разный.

Варианты программирования выводов для приема данных.

1 Прием данных предполагается осуществлять через вывод F11. Альтернативная функция UART_RxD2 является первой и имеет наивысший приоритет. Для выбора этой функции в регистре GPIOPCTLF в поле Pin11 следует записать значение 00h (значение после сброса). Состояния полей Pin0 (регистр GPIOPCTLD) и Pin4 (регистр GPIOPCTLF) не важно. Т.е. даже если одновременно будут включены функции UART_RxD2 для вывода F11 и любого из двух других, на вход блока UART2 будет скоммутирован вывод F11.

2 Прием данных предполагается осуществлять через вывод D0. Альтернативная функция UART_RxD2 является второй и имеет средний приоритет. Для выбора этой функции в регистре GPIOPCTLD в поле Pin0 следует записать значение 01h.

Состояние поля Pin4 (регистр GPIOPCTLF) не важно, а вот состояние поля Pin11 (регистр GPIOPCTLF) не должно быть 00b (т.е. для вывода F11 должна быть выбрана любая альтернативная функция кроме UART_RxD2). Только в этом случае вывод D0 будет скоммутирован на вход блока UART2.

Если для выводов F11 и D0 одновременно выбрана функция UART_RxD2, то согласно приоритету, на вход блока UART2 будет скоммутирован вывод F11 (даже в случае, если режим альтернативной функции этого вывода выключен) и соответственно данные с вывода D0 не будут приняты.

3 Прием данных предполагается осуществлять через вывод F4. Альтернативная функция UART_RxD2 является третьей и имеет низший приоритет. Для выбора этой функции в регистре GPIOPCTLF в поле Pin4 следует записать значение 10h.

При этом состояние поля Pin11 (регистр GPIOPCTLF) не должно быть 00b и состояние поля Pin0 (регистр GPIOPCTLD) не должно быть 01b (т.е. для выводов F11 и D0 должны быть выбраны любые альтернативные функции кроме UART_RxD2). В этом случае вывод F4 будет скоммутирован на вход блока UART2.

Если для выводов F4 и F11 и/или D0 одновременно выбрана функция UART_RxD2, то на вход блока UART2 будет скоммутирован вывод, альтернативная функция которого имеет более высокий приоритет и соответственно данные с вывода F4 не будут приняты.

Указанные правила программирования выводов для приема данных распространяются на все периферийные блоки.

Для разрешения работы вывода необходимо установить соответствующий бит в регистре GPIOODENi (регистр общего назначения). Если бит сброшен, то вывод находится в третьем состоянии.

Выводы A1 – A7 могут также использоваться как входы аналогового компаратора.

Ко всем площадкам выводов подключены входные фильтры. На рисунке 5.4 показана структурная схема фильтра нулевого вывода порта A (в названии регистров присутствует «0», указывая на управляющие регистры порта A)

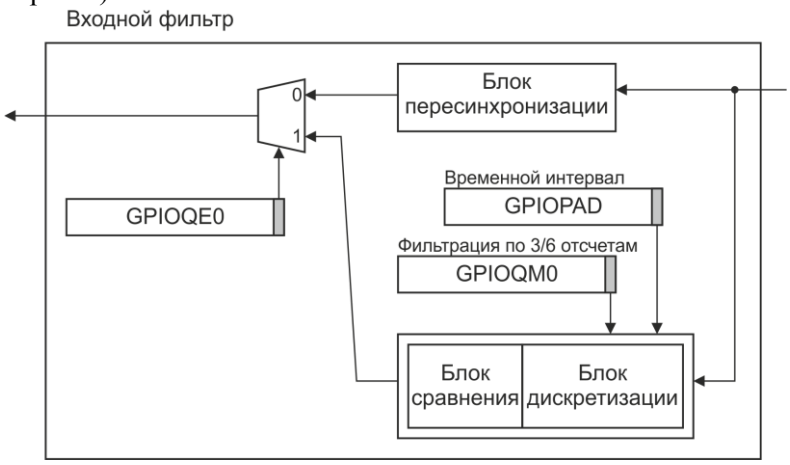


Рисунок 5.4 – Входной фильтр вывода

Входной сигнал с вывода порта может приниматься как напрямую (асинхронный режим), так и пересинхронизироваться (синхронизироваться с тактовой частотой работы микроконтроллера). Управление осуществляется регистром GPIOSEi. Дополнительно есть возможность включения накопления 3 или 6 отсчетов входного сигнала для помехоустойчивости вывода. Если результаты всех отсчетов совпадают, сигнал передается дальше по схеме, в противном случае состояние сигнала не меняется. Временные интервалы между отсчетами задаются в количестве тактов системной частоты посредством регистров GPIOPAD и GPIOEH. Временной интервал задается один для всех выводов порта.

Включение фильтра и задание режима его работы осуществляется посредством регистров GTPIOQEi, GPIOQMi,.

Схема вывода позволяет также осуществлять гибкое управление прерываниями и задавать по какому аппаратному событию генерировать прерывание (по какому фронту или уровню). При возникновении прерывания в регистре INTSTATUS устанавливается соответствующий флаг и выставляется прерывание в контроллере прерываний NVIC. Прерывание может быть сгенерировано программно записью единицы в соответствующий бит регистра INTSTATUS.

Прерывание может быть сброшено программно записью единицы в соответствующий бит регистра INTCLEAR. Для разрешения прерывания вывода порта следует записать единицу в соответствующий выводу бит регистра INTENSET, а для запрета прерывания –единицу в бит регистра INTENCLR.

Для задания события, по которому генерировать прерывание используются регистры INTTYPESET и INTPOLSET, а для сброса настроек – INTTYPECLR и INTPOLCLR, соответственно.

Для управления состоянием (а также чтения) выводов порта дополнительно используется механизм маскирования. Он позволяет устанавливать желаемый уровень сигнала на нужном выводе не затрагивая состояние других выводов. 16-разрядный порт условно разбивается на две части – старший и

младший байты. Для доступа по маске к каждому из двух байт используются регистры MASKLOWBYTE и MASKHIGHBYTE. Для каждого байта имеется массив из 256 регистров, в которых хранятся маски с 00h по FFh. Так, например, для порта A выделены две области памяти с адресами 80010400h – 800107FCh для младшего байта и 8001_0800h – 80010BFCh для старшего. Для того, чтобы изменить состояние выводов порта с использованием маски нужно записать новое значение в ячейку памяти с адресом, по которому расположена маска. Разряды порта, закрытые «нулями» маски останутся неизменными, а остальные примут новые значения. На рисунке 5.5 показан механизм маскирования младшего байта порта A.

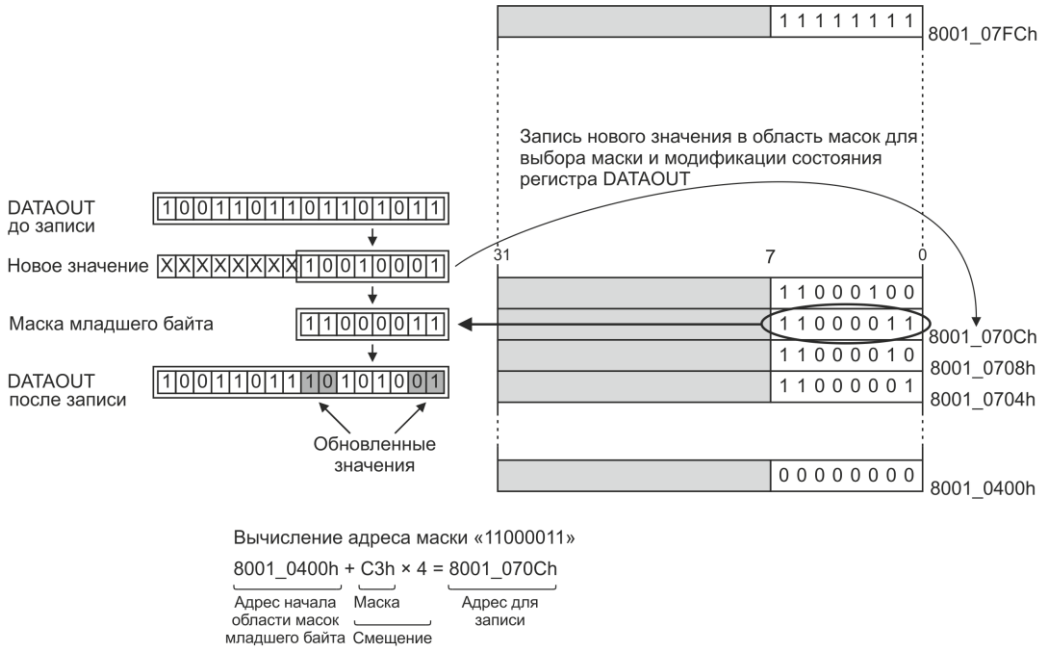


Рисунок 5.5 – Механизм изменения состояния младшего байта порта A с маскированием

Для изменения 0, 1, 6 и 7 битов регистра порта нужно использовать маску 11000011b. Эта маска расположена по адресу 8001070Ch (дополнительно, механизм вычисления маски указан на рисунке 10.3). Новое значение XX90h (старший байт числа не важен) нужно записать в ячейку с адресом 8001070Ch. Далее это значение будет аппаратно маскировано и записано в регистр порта DATAOUT.

Аналогично выполняется маскирование старшего байта (см. рисунок 5.6). Разница лишь в том, что в данном случае берется старший байт нового значения, а младший не важен.

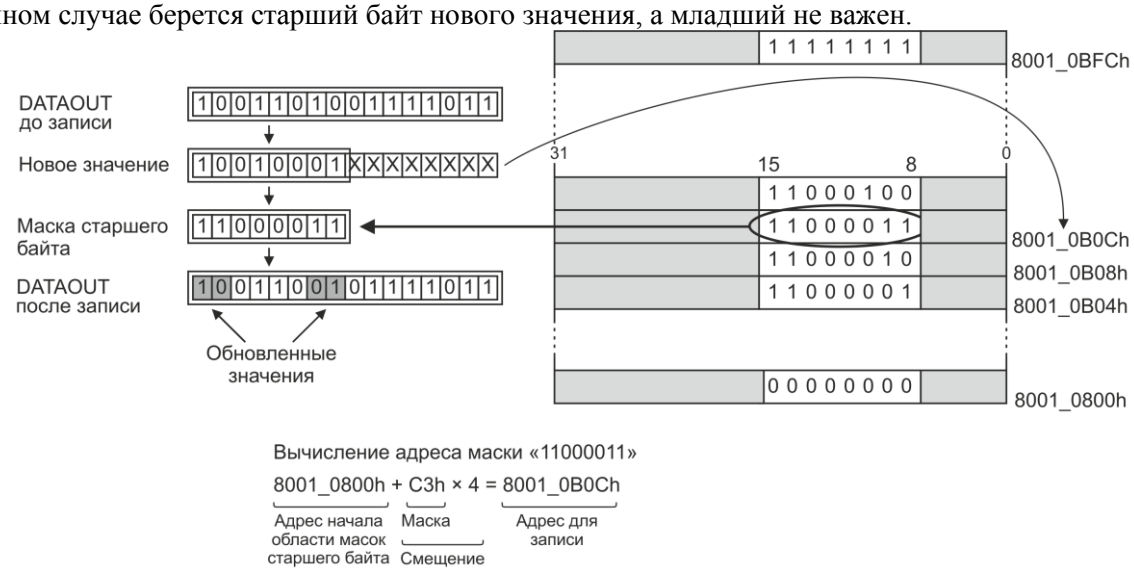


Рисунок 5.6 – Механизм изменения состояния старшего байта порта A с маскированием

5.4 Работа с АЦП

Контроллер АЦП имеет 24 канала – по два (A и B) на каждый модуль АЦП.

Настройка тактирования контроллера АЦП и его модулей осуществляется посредством регистров ADC_CTRL1, ADC_CTRL2 и ADC_CTRL3.
Для правильной работы необходимо обеспечить тактирование модулей АЦП:
- в 10-разрядном режиме частотой 12 МГц (допустимый диапазон от 5 до 14,4 МГц);
- в 12-разрядном режиме частотой 24 МГц (диапазон от 6 до 28,8 МГц).

Опорное напряжение АЦП 1,5В, соответственно, в одиночном режиме измерения проводятся в диапазоне 0-1,5В, в дифференциальном - +/- 0,75В относительно средней точки 0,9В
В одиночном режиме амплитуда входного сигнала - 1,5В, в дифференциальном режиме - 3,0В
Биты OM[4] и OM[6] управляют схемой подключения каналов A и B в дифференциальном режиме (схема подключения на рисунке 5.7)

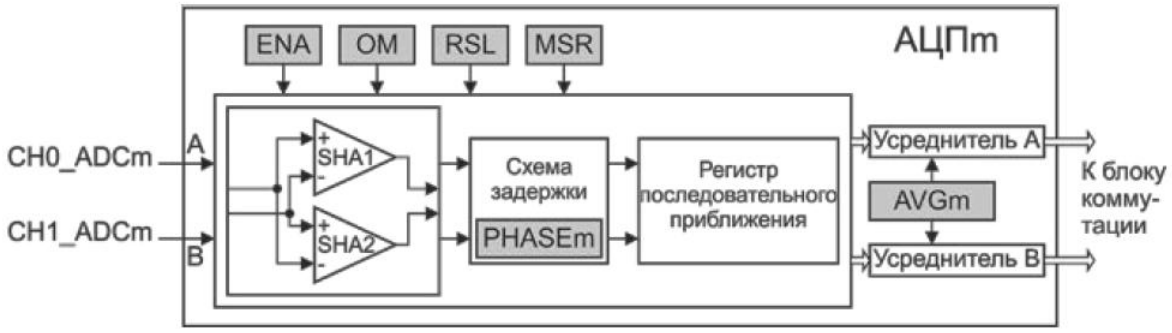


Рисунок 5.7– Модуль АЦП

Т.е. если выбираем блока SH1 в дифференциальном режиме, то результатом преобразования по каналу A будет AIN0-BIN0 (к положительному входу подключен вход AIN0, а к отрицательному - BIN0), а по каналу B (если он включен) - значение BIN0 (в одиночном режиме).
При выборе блока SH2 в дифференциальном режиме - аналогично в соответствии с рисунком выше.

Секвенсор представляет собой управляющий блок, позволяющий разгрузить процессор от управления модулями АЦП. Секвенсор управляет запуском модулей АЦП, обработкой полученных результатов измерений и генерацией прерываний. В состав блока АЦП входят восемь секвенсоров.

Каждый секвенсор может независимо запускаться по одному из событий, которое выбирается полем EMs регистра ADCEMUX:
- установка бита GSYNC регистра ADCPSSI;
- сигнал от одного из блоков аналоговых компараторов;
- сигнал от блока таймеров;
- сигнал от одного из блоков ШИМ;
- сигнал прерывания GPIO.

Секвенсор может одновременно запустить измерения на всех 24 каналах блока АЦП. Если выбранный канал соответствует входу A модуля АЦП, то измерение будет запущено только для этого входа. В случае если выбранный канал соответствует входу B, то сначала будет запущено измерение по входу A, а затем по входу B (реализовано на аппаратном уровне и влияет на время измерений).

Выбор каналов для измерений осуществляется посредством регистра ADCSSMUX. Результаты измерений сохраняются в буфере результатов секвенсора. Контроль состояния буфера осуществляется посредством регистров ADCSSFSTAT, ADCOSTAT и ADCUSTAT.

Прерывания, генерируемые секвенсором и передаваемые в контроллер NVIC, могут использоваться блоком DMA для аппаратного копирования содержимого буфера результатов АЦП в ОЗУ.

Проведение измерений

1. Если сигнал запуска получает только один модуль АЦП, то он запускается. Если модуль настроен на проведение нескольких измерений, то по окончании измерения модуль АЦП запускается снова и запуск происходит по истечении времени, определяемого задержкой TMR (незамедлительно, если TMR = 00000h).

Если в течение преобразования возникают сигналы запуска других модулей АЦП, то эти модули переходят в состояние ожидания.

3. Если несколько модулей АЦП получают сигнал запуска одновременно, то сначала запускается модуль с наименьшим порядковым номером. Остальные модули переходят в состояние ожидания (см. рисунок 5.8). Если в течение преобразования какой-либо из незапущенных ранее модулей АЦП получает сигнал запуска, он также переходит в состояние ожидания.

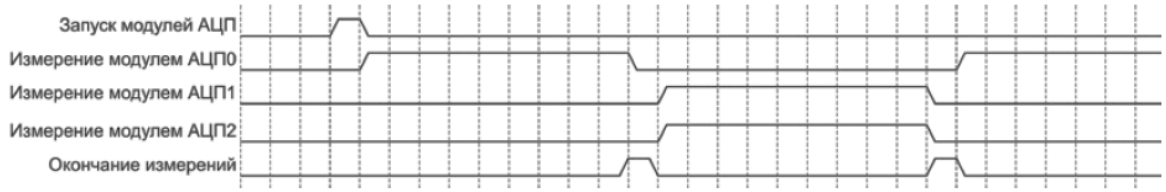


Рисунок 5.8 – Запуск нескольких модулей АЦП

3. На момент окончания измерения происходит проверка состояния всех модулей АЦП и незамедлительный синхронный запуск модулей, которые находятся в состоянии ожидания.

Рисунок 5.9 – Функционирование нескольких модулей АЦП

Расчет времени одного измерения
Временем одного измерения считается время от момента запуска модулей АЦП до момента появления результатов измерения на выходах модуля, имеющего наибольшую фазовую задержку и/или наибольшее количество перезапусков для усреднения результатов.
Время T одного измерения модуля АЦП рассчитывается по формуле:

$$T = R \times A \times (Tadc + Padc), \text{ где}$$

R – количество перезапусков (R = 1 при RCNT = 00h);
A – количество измерений при усреднении (A = 1 при AVGm = 0h);
Tadc – время преобразования для:
- одного канала (A) равняется семи тактам рабочей частоты (например, для 24 МГц параметр Tadc = 292 нс);
- двух каналов (сначала A, затем B) равняется 14 тактам рабочей частоты (для 24 МГц параметр Tadc = 584 нс);
Padc – фазовая задержка (Padc = 0 при PHASEm = 0h).

При расчете частоты перезапуска модулей АЦП необходимо учитывать, что счетчик перезапуска не считает во время проведения измерения, а значит общее время между запусками секвенсоров T_перезапуск=t_измерения+t_счета.

Частота, соответственно, F_перезапуск=1/T_перезапуск. Если время измерений у разных модулей АЦП различны, то ориентироваться нужно на самое длительное, т.к. секвенсор выставляет флаг завершения после проведения измерений по всем необходимым каналам.

Буфер результатов измерений
Каждый секвенсор имеет собственный 12-разрядный буфер результатов измерений, организованный по типу FIFO. В каждой ячейке буфера может храниться результат одного измерения.
Количество ячеек в буфере:
- 32 для секвенсоров 0 и 1;
- 16 для секвенсоров 2 – 5;
- 1 для секвенсоров 6 – 7.
Результаты измерений модулей АЦП записываются в буферы секвенсоров в порядке возрастания номеров каналов, т.е. первым будет записан результат канала A модуля АЦП0, а последним – результат канала B модуля АЦП11. Каналы, на которые не поступили сигналы запуска, пропускаются. При этом порядок записи не изменяется. После записи последнего значения в буфер и/или компаратор, формируется прерывание.

Примечание – Запущенные модули АЦП могут иметь разное время измерения, и, как следствие выдавать результаты измерений не одновременно. Тем не менее, сохранение результатов в буферах секвенсоров выполняется только после того как все запущенные модули закончат измерение.
По мере заполнения буфера секвенсора инкрементируется счетчик количества сохраненных результатов. Состояние счетчика доступно для чтения посредством регистра ADCSSFSTAT.
Как только буфер будет заполнен полностью, установится флаг OVs в регистре ADCOSTAT.
Сохранные в буфере данные доступны для чтения посредством регистра ADCSSFIFO. При считывании данных буфер секвенсора будет опустошаться. Как только буфер будет полностью пуст, установится флаг UVs в регистре ADCUSTAT.

Все компараторы блока АЦП независимы. Каждый компаратор может обрабатывать результат измерения любого канала, поскольку может быть скоммутирован с любым из 24 выходов модулей АЦП.

Правила настройки.

1. Посредством регистра ADCSSMUX выбираются каналы для измерений (установкой битов CHn).

2. Посредством регистра ADCSSDCP выбираются (разрешаются) компараторы для обработки полученных результатов измерений (установкой битов CMPn). Запрещенные компараторы не обрабатывают полученные результаты.

3. Для каждого компаратора n в его регистре ADCDCCTLn в поле CHNL указывается номер канала, результат измерения которого будет передан на компаратор. По умолчанию, значение CHNL = 00h, т.е. все компараторы настроены на работу с нулевым каналом.

Примечание – Для разрешенных компараторов в поле CHNL может быть указан номер только того канала, который выбран в регистре ADCSSMUX. Для запрещенных компараторов в поле CHNL может быть указан номер канала, который выбран в регистре ADCSSMUX или записано любое значение от 18h до 1Fh.

Для корректной работы АЦП необходимо:

1) Включить общее тактирование блоков АЦП (регистр общего назначение APB_CLK[0x800300BC] бит 24 ADCEN)
2) в регистры ADCPP в поле OM записать значение 0x3
3) во все регистры ADCDCCTLx, поле CHNL записать любое "зарезервированное" значение из диапазона 0x18... 0x1F т.к. в логике работы аналогового преобразования заложен алгоритм анализа работы компараторов, и по умолчанию все 24 компаратора анализируют 0- канал АЦП, т.к. в поле CHNL регистров ADCDCCTLx по-умолчанию (после сброса) выбран 0-ой канал АЦП.

5.5 Работа с Аналоговыми компараторами

Некоторые уточнения по работе блока аналоговых компараторов:

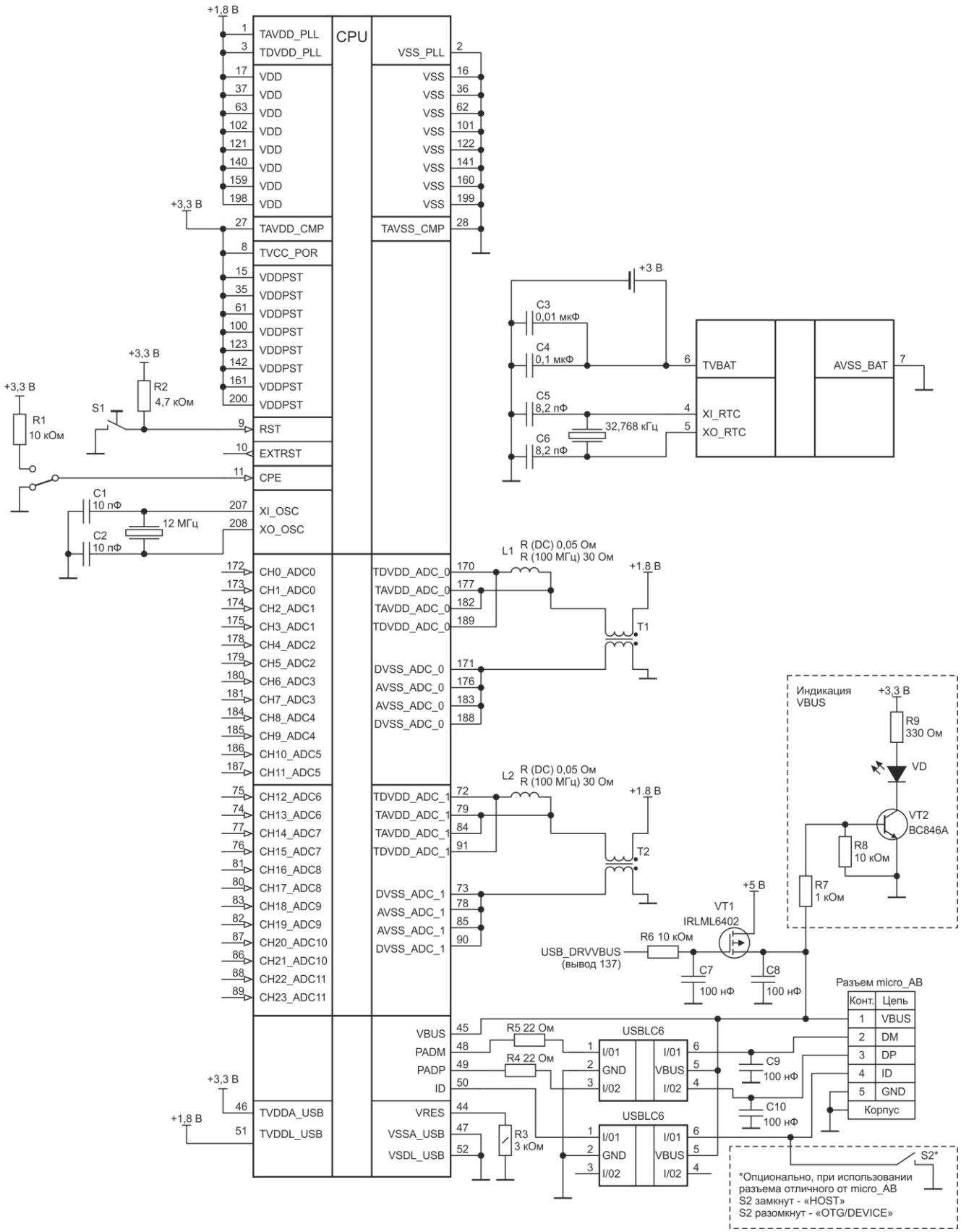
1. Аналоговые входы компараторов (C3+, C3-, C2+, C2-, C1+, C1- и DAC_supply) всегда подключены к соответствующим выводам микроконтроллера A.1 - A.7. Поэтому при работе с блоком аналоговых компараторов нет необходимости кофигурировать порт GPIOA. Однако, можно исключить "возможное" случайное влияние цифрового порта на входы компаратора отключив цифровой порт от выводов микроконтроллера записью в регистр NT_COMMON_REG->GPIODEN0 &= ~(0x00FE);

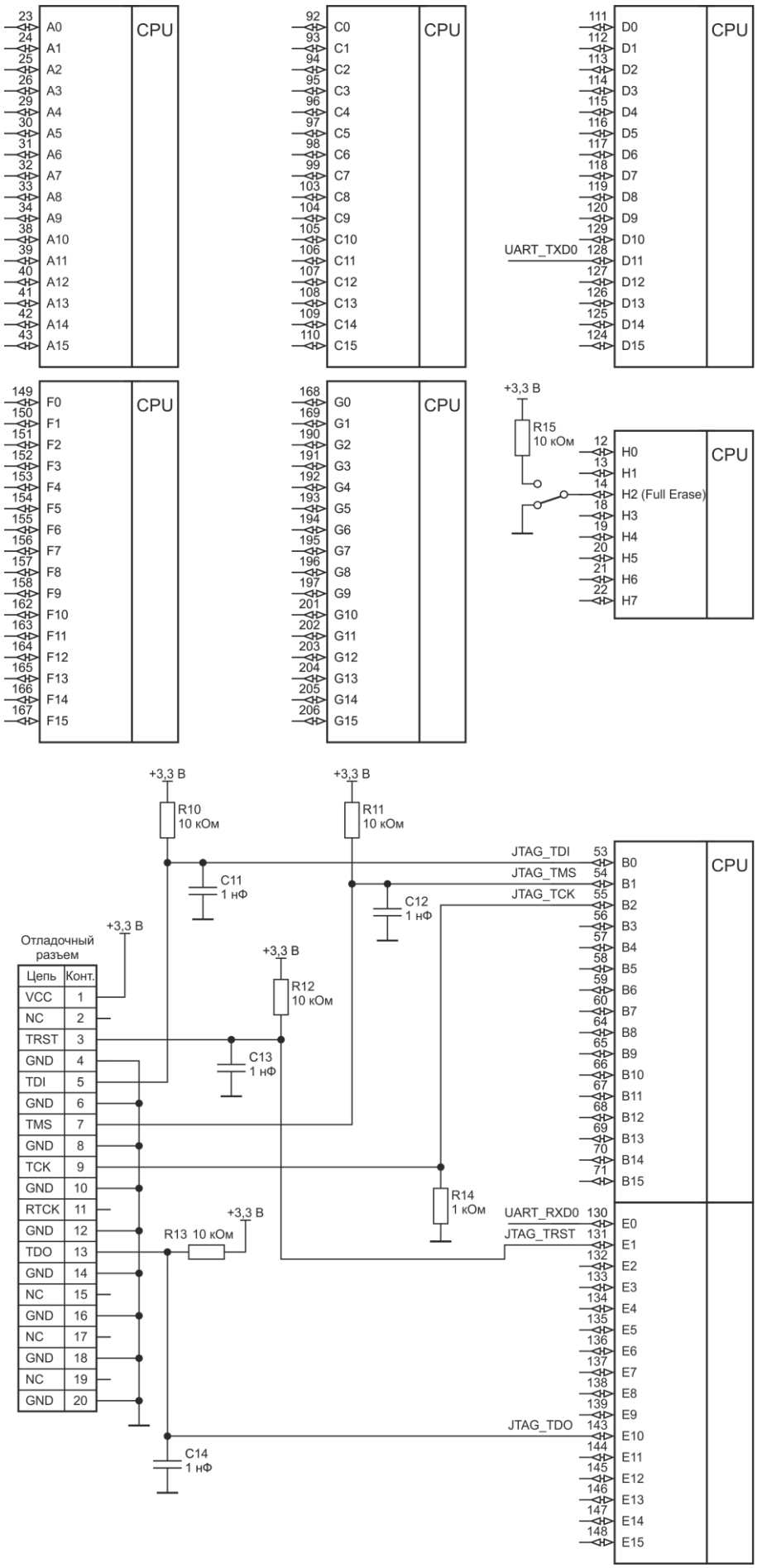
2. Аналоговый вход DAC_supply является входом питания ЦАПов компараторов (на рисунке 3 "Структурная схема модуля ЦАП" обозначен как "eCMPRef"). То есть при подключении одного из ЦАП на входы компараторов необходимо обеспечить их питанием, подключенным к входу DAC_supply (вывод A.7).

3. Включение питания блока компараторов (бит PWU регистра NT_CMP->POWER) необходимо осуществлять после отключения сигнала сброса блока и подаче тактового сигнала на блок компараторов .
   Рекомендуемый порядок включения аналоговых компараторов:

```
NT_COMMON_REG->PER_RST2_bit.ECMPRST = 1; //Clear reset signal
NT_COMMON_REG->APB_CLK_bit.CMPEN = 1; //Clock enable
NT_CMP->POWER_bit.PWU = 1; //Power ON
```

5.6 Схема включения микроконтроллера




CPU
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15
C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15
D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 D10 D11 D12 D13 D14 D15
UART_TXD0
F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15
G0 G1 G2 G3 G4 G5 G6 G7 G8 G9 G10 G11 G12 G13 G14 G15
+3,3 В
R15 10 кОм
H0 H1 H2 (Full Erase) H3 H4 H5 H6 H7
R10 10 кОм
R11 10 кОм
C11 1 нФ
C12 1 нФ
C13 1 нФ
C14 1 нФ
R12 10 кОм
R13 10 кОм
R14 1 кОм
Отладочный разъем
Цепь Конт.
VCC 1
NC 2
TRST 3
GND 4
TDI 5
GND 6
TMS 7
GND 8
TCK 9
GND 10
RTCK 11
GND 12
TDO 13
GND 14
NC 15
GND 16
NC 17
GND 18
NC 19
GND 20
JTAG_TDI
JTAG_TMS
JTAG_TCK
JTAG_TRST
JTAG_TDO
UART_RXD0
B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12 B13 B14 B15
E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 E10 E11 E12 E13 E14 E15

6 Фрагменты кода для программирования К1921ВК01Т

6.1 Инициализация GPIO для альтернативной функции

```
#define GPIOPIN_1      (1 << 1)
#define GPIOPIN_NUM1
void GPIO_init_altf(){
//if alt function is only  IN (example UART_Rx), then you may  comment next line (do not necessarily)
   NT_COMMON_REG->GPIODEND |= GPIOPIN_1; // enable output on pins
   NT_COMMON_REG->GPIOPCTLD &= ~ (3 << (GPIOPIN_NUM << 1)) //clear field num alt.function
   NT_COMMON_REG->GPIOPCTLD |= (1 << (GPIOPIN_NUM << 1) ) // set num alt.function
   NT_GPIOD->ALTFUNCSET = GPIOPIN_1; // enable alt.function on pins
}
```

6.2 Инициализация GPIO на вход

```
#define GPIOPIN_1      (1 << 1)
#define GPIOPIN_NUM1
void GPIO_init_in(){
   NT_COMMON_REG->GPIODEND &= ~(GPIOPIN_1);
   NT_GPIOD->ALTFUNCCLR = GPIOPIN_1;
}
Int main(){
unsigned short a;
GPIO_init_in();
while (1){
          a = NT_GPIOD->MASKLOWBYTE[GPIOPIN_1]; // read pin PD.1 by MASK
}
}
```

6.3 Инициализация GPIO на выход

```
#define GPIOPIN_1      (1 << 1)
#define GPIOPIN_NUM 1
void GPIO_init_out()
{
   NT_COMMON_REG->GPIODEND |= GPIOPIN_1;
   NT_GPIOD->OUTENSET = GPIOPIN_1;
   NT_GPIOD->ALTFUNCCLR = GPIOPIN_1;
}
Int main(){
unsigned short a;
GPIO_init_out();
a = 1;
while (1){
          NT_GPIOD->MASKLOWBYTE[GPIOPIN_1] = a; // write pin PD.1 by MASK
}
}
```

6.4 Использование прерываний на примере АЦП

```
#define IRQ_EN(NUM) *((volatile unsigned int *)(0xE000E100 + (NUM>>5)*4)) = (1<<(NUM%32));
#define IRQ_DIS(NUM) *((volatile unsigned int *)(0xE000E180 + (NUM>>5)*4)) = (1<<(NUM%32));
#ifdef __cplusplus
extern "C" {
#endif  /* __cplusplus */
void ADC_SEQ0_IRQHandler() {
  NT_GPIOA->MASKLOWBYTE[0x3] = ~NT_GPIOA->MASKLOWBYTE[0x3];
  NT_ADC->ADCISC = 0x01; // clear flag
}
#ifdef __cplusplus
}
#endif  /* __cplusplus */

void adc_init(){
   uint8_t i;
   for(i=0;i<24;i++) NT_ADC->ADCDCCTL[i] = 0x18<<16; //disable all comparators

    //Common ADC Clock
    NT_COMMON_REG->APB_CLK |= (1<<24); // ADC Clock Enable

    //CH_ADC[0] ADC0(A)
    NT_COMMON_REG->ADC_CTRL1 = 1; //enable clk
    NT_ADC->ADCPP[0] = (1<<31)|(3<<16); //ENABLE, OM[2:0]

    //ADC_SEQ0
    NT_ADC->ADCEMUX = 0xF; //ADC_SEQ0 start at ADCPSSI
    //NT_ADC->ADCSAC |= (6<<0); // AVERAGES for seq0 eq 64 samples
    NT_ADC->ADSSEQ[0].ADCSSMUX = 1;//(<<14)|(1<<18); // CH14,CH18 enable
    NT_ADC->ADCACTSS |= 1; //ADC_SEQ0 enable

    NT_ADC->ADCIM = 0x01; // set mask flag
}

int main()
{
 IRQ_EN(ADC_SEQ0_IRQn); //Enable interrupt ADC_SEQ0 in NVIC
 adc_init();
   __enable_irq();
   while (1) ;
 return 0;
}
```

6.5 Инициализация Аналоговых компараторов

```
void cmp_init()
{
  NT_COMMON_REG->PER_RST2_bit.ECMPRST = 1; //Clear reset signal
  NT_COMMON_REG->APB_CLK_bit.CMPEN = 1; //Clock enable
  NT_CMP->POWER_bit.PWU = 1; //Power ON
}
```

6.6 Инициализация АЦП

```
void adc_init()
{
    uint32_t i;
   for(i=0;i<24;i++) NT_ADC->ADCDCCTL[i] = 0x18<<16; //disable all comparators

    //Common ADC Clock
    NT_COMMON_REG->APB_CLK |= (1<<24); // ADC Clock Enable

    //CH_ADC[0] ADC0(A)
    NT_COMMON_REG->ADC_CTRL1 |= ((1 | 1<<1 | 2<<2)); // freq/(2*(1+(2)))= freq /8
    NT_ADC->ADCPP[0] = (1<<31)|(3<<16); //ENABLE, OM[2:0]
    //CH_ADC[2] ADC1(A)
    NT_COMMON_REG->ADC_CTRL1 |= ((1 | 1<<1 | 2<<2)<<8); // freq/(2*(1+(2)))= freq /8    NT_ADC-
>ADCPP[1] = (1<<31)|(3<<16); //ENABLE, OM[2:0]
    //CH_ADC[4] ADC2(A)
    NT_COMMON_REG->ADC_CTRL1 |= ((1 | 1<<1 | 2<<2)<<16); // freq/(2*(1+(2)))= freq /8
    NT_ADC->ADCPP[2] = (1<<31)|(3<<16); //ENABLE, OM[2:0]
    //CH_ADC[6] ADC3(A)
    NT_COMMON_REG->ADC_CTRL1 |= ((1 | 1<<1 | 2<<2)<<24); // freq/(2*(1+(2)))= freq /8
    NT_ADC->ADCPP[3] = (1<<31)|(3<<16); //ENABLE, OM[2:0]
    //CH_ADC[8] ADC4(A)
    NT_COMMON_REG->ADC_CTRL2 |= ((1 | 1<<1 | 2<<2)); // freq/(2*(1+(2)))= freq /8
    NT_ADC->ADCPP[4] = (1<<31)|(3<<16); //ENABLE, OM[2:0]
    //CH_ADC[10] ADC5(A)
    NT_COMMON_REG->ADC_CTRL2 |= ((1 | 1<<1 | 2<<2)<<8); // freq/(2*(1+(2)))= freq /8
    NT_ADC->ADCPP[5] = (1<<31)|(3<<16); //ENABLE, OM[2:0]
    //CH_ADC[12] ADC6(A)
    NT_COMMON_REG->ADC_CTRL2 |= ((1 | 1<<1 | 2<<2)<<16); // freq/(2*(1+(2)))= freq /8
    NT_ADC->ADCPP[6] = (1<<31)|(3<<16); //ENABLE, OM[2:0]
    //CH_ADC[14] ADC7(A)
    NT_COMMON_REG->ADC_CTRL2 |= ((1 | 1<<1 | 2<<2) << 24); // freq/(2*(1+(2)))= freq /8
    NT_ADC->ADCPP[7] = (1<<31)|(3<<16); //ENABLE, OM[2:0]
    //CH_ADC[18] ADC9(A)
    NT_COMMON_REG->ADC_CTRL3 |= ((1 | 1<<1 | 2<<2) << 8); // freq/(2*(1+(2)))= freq /8
    NT_ADC->ADCPP[9] = (1<<31)|(3<<16); //ENABLE, OM[2:0]
 //CH_ADC[16] ADC8(A)
  NT_COMMON_REG->ADC_CTRL3 |= ((1 | 1<<1 | 2<<2)); // freq/(2*(1+(2)))= freq /8
  NT_ADC->ADCPP[8] = (1<<31)|(3<<16); //ENABLE, OM[2:0]

//ADC_SEQ0
    NT_ADC->ADCEMUX |= (0x0<<0); //ADC_SEQ0 start at ADCPSSI
    NT_ADC->ADCSAC |= (6<<0); // AVERAGES for seq0 eq 64 samples
    NT_ADC->ADSSEQ[0].ADCSSMUX = (1<<14)|(1<<18); // CH14,CH18 enable
    NT_ADC->ADCACTSS |= 1; //ADC_SEQ0 enable
}

int main()
{
    float data[2];

    adc_init();
    NT_ADC->ADCPSSI = (1<<31)|1; //START SEQ0
    while ((NT_ADC->ADCRIS & 0x01) == 0); //WAIT COMPLETE
    data[0] = NT_ADC->ADSSEQ[0].FIFO*1.5/4096; // data channel 14
    data[1] = NT_ADC->ADSSEQ[0].FIFO*1.5/4096; // data channel 18
    NT_ADC->ADCISC = 0x01; // clear flag
    return 0;
}
```

6.7 Инициализация USB_DEVICE

```
#define IRQ_EN(NUM) *((volatile unsigned int *)(0xE000E100 + (NUM>>5)*4)) = (1<<(NUM%32));
#define IRQ_DIS(NUM) *((volatile unsigned int *)(0xE000E180 + (NUM>>5)*4)) = (1<<(NUM%32));
#define USB_DEVICE_MAX_EP0_SIZE 64

#ifdef __cplusplus
extern "C" {
#endif  /* __cplusplus */
void USB_IRQHandler() {
   //здесь следует смотреть прерывания от самого USB_DEVICE
   if(USB_DEVICE->IRQ_ENB_L & USB_DEVICE->IRQ_STAT_L & 0x1)
   {
      //здесь мы смотрим прерывания, которые мы разрешили в USB_IRQ_ENB
      if(USB_DEVICE->USB_IRQ_STAT & USB_DEVICE-> USB_IRQ_ENB & (1<<1))
      {
         //сюда мы попадем по прерыванию сброса линии USB
         //следует сбрасывать флаг (записью "1"), для того, чтобы снова не вызвать прерывание на
вектор USB
         USB_DEVICE->USB_IRQ_STAT |= (1<<1);

         //настроить внутренний буфер конечной точки (существует внутренняя ОЗУ на 2 кб,
которая должна поделиться между контрольной точкой и остальными конечными точками)
         USB_DEVICE->CEP_START_ADDR = 0x0;
          USB_DEVICE->CEP_END_ADDR = CEP_START_ADDR + USB_DEVICE_MAX_EP0_SIZE;
      }
      ///.....if(другие прерывания){};
   }
   //либо от КОНТРОЛЬНОЙ ТОЧКИ
   if(USB_DEVICE->IRQ_ENB_L & USB_DEVICE-> IRQ_STAT_L & (1<<1))
   {
                  //обработка прерывания от контрольной точки.
                  //смотрим бит SETUPPKTINT (бит 1) в регистре CEP_IRQ_STAT (0x80090034)
         if(USB_DEVICE->CEP_IRQ_STAT & USB_DEVICE->CEP_IRQ_ENB & (1<<1))
         {
         //если мы здесь, то пришел пакет сетап на контрольную точку. следует сбросить флаг
этого прерывания
         USB_DEVICE->CEP_IRQ_STAT |= (1<<1);
         //далее производится обработка принятого пакета. Полностью принятый пакет сетап
отображается в регистрах SETUP1_0, SETUP3_2, SETUP5_4, SETUP7_6
         USBD_DCD_Setup_Stage();
         }
   }
   //либо от одной из конечных точек
   if(USB_DEVICE->IRQ_ENB_L & USB_DEVICE-> USB_DEVICE->IRQ_STAT_L & (0xF<<2))
   {
   ...
   }

   //здесь следует смотреть прерывания от OTG
   {
         //если включено прерывание session fail
      if(USB_DEVICE->OTGIRQEN & (1<<4))
         {
            //здесь смотрим прерывание session fail (возникает в случае физического отключения
device от хоста при пропадании VBUS)
         if(USB_DEVICE->OTGIRQSTS & (1<<14))
            {
                  //сбрасываем флаг прерывания
                  USB_DEVICE->OTGIRQSTS |= (1<<14);
                  //выключаем прерывания от контрольной и конечной точек
                  USB_DEVICE->IRQ_ENB_L &= 0x01;
```

```
                    USB_DEVICE->CEP_IRQ_ENB = 0;
                    USB_DEVICE->USB_EP[0]->IRQ_ENB = 0;
                    USB_DEVICE->USB_EP[1]->IRQ_ENB = 0;
                //переводим наше устройство в состояние USB_POWERED
                 }
            }
    }
}
#ifdef __cplusplus
}
#endif  /* __cplusplus */


void usb_dev_init(){
        //configurate clk USB internal from Xtall 12MHz or external GPIO_A0-12/24 MHz
        NT_COMMON_REG->GPIOPCTLA &= ~(0x3); //clear field num alt.function
        NT_COMMON_REG->GPIOPCTLA |= 0; // set num alt.function
        NT_GPIOA->ALTFUNCSET = 0x1; // enable alt.function on pins

        NT_COMMON_REG->PER_RST1 |= (1 << 3); //disable reset USB
        USB_DEVICE->USB_IRQ_ENB |= (1 << 1); //enable interrupt by reset line of USB
        USB_DEVICE->IRQ_ENB_L |= (1 << 0);  //enable interrupt USB_DEVICE
//      USB_DEVICE->USB_OPER &= ~(1 << 1); //reset bit HIGHSPEED, uncommeted this line for
activate FULL_SPEED mode, default set HIGH_SPEED
        USB_DEVICE->OTGIRQEN |= (1<<4); //enable interrupt OTG (detach)

}

int main(){
        usb_dev_init();
        IRQ_EN(USB_IRQn); //Enable interrupt USB in NVIC
}
```