НИУ «МЭИ»
представляет учебный курс:

Проектирование цифровых систем управления на базе отечественного микроконтроллера **НИИЭТ К1921ВК01Т**

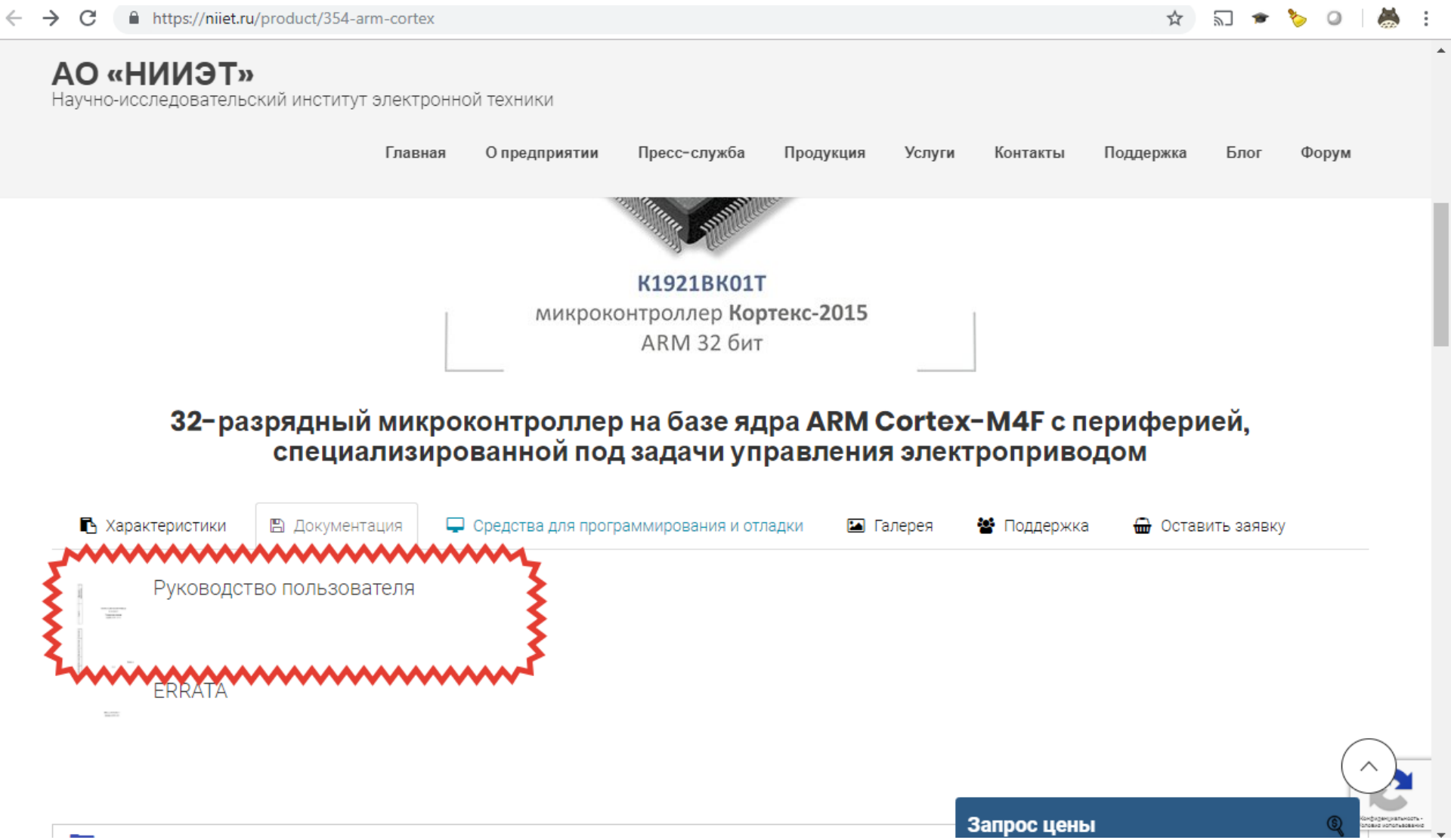
Москва 2019

Лекция 2.1

- Практическая работа по запуску и настройке периферийного модуля широтно-импульсной модуляции (ШИМ) микроконтроллера, предназначенного для управления электродвигателями. Регулирование яркости светового индикатора при помощи ШИМ.

Документация

http://niiet.ru/product/354-arm-cortex



Заголовочные файлы

Есть три вида заголовочных файлов под разные компиляторы:

1) K1921BK01T.h для Keil. Сгенерирован утилитой Keil SVDConv из файла описания регистров в виде SVD-файла. Обсуждение данного заголовочного файла на bitbucket по ссылке. Отличительная черта – наличие текста «*Generated with SVDConv from CMSIS SVD File 'K1921BK01T.xml»* в шапке заголовочного файла. Имеет немного другие названия регистров по сравнению с упомянутыми ниже файлами.

2) K1921BK01T.h для GCC и CodeMaster. Отличительная черта – наличие текста «*Angioscan Electronics Application Team*».

ОБНОВЛЕНИЕ 2019.06: появился один универсальный заголовочный файл, подходящий для всего.
https://bitbucket.org/niietcm4/niietcm4_pd/src/default/Libs/Device/NIIET/K1921VK01T/Include/
(но в примерах всё еще можно встретить другие версии заголовочников)

Управление оперативной памятью

Для работы встраиваемых систем «жесткого» реального времени не рекомендуется использовать динамическое выделение памяти из кучи (heap), т.е. использовать функции **`malloc`**, **`free`** по следующим причинам:

- Ошибка с утечкой памяти наиболее вероятно приведет к остановке программы, а при управления силовой установкой это опасно.
- Отследить ошибки с «забытыми» вызовами **`free`** очень сложно, они проявляются только на этапе выполнения программы.
- Система управления электроприводом обычно не требует для своих задач динамического выделения памяти, память может быть выделена заранее в виде статических массивов.
- Нехватка памяти при статическом её выделении сразу проявляется на этапе компиляции (линковки) программы.

Распределение памяти

- Под распределением или компоновкой памяти понимается задание пользователем секций оперативной и flash памяти микроконтроллера, в которых будут находиться те или иные данные и программный код, например: секция переменных в ОЗУ, стек, программа пользователя, таблица констант для инициализации переменных в ОЗУ и так далее.

- Разные средства разработки предоставляют разный механизм распределения памяти, начиная от отсутствия каких-либо возможностей повлиять на этот процесс (всё автоматически, задается только размер flash и оперативной памяти) и заканчивая специальным скриптовым языком, позволяющим описать распределение памяти в самых подробностях «вручную».

- В VectorIDE используется пакет утилит GCC, в которых для утилиты линкера (компоновщика памяти) ld используется специальный скриптовый язык, на котором написан файл распределения памяти build.ld.

Пример скриптового языка распределения памяти ld

build.ld

```
29 /* Specify the memory areas */
30 /* Буквы в скобках определяют атрибуты: доступ на чтение,
31  * запись, исполнение, выделение памяти. */
32 MEMORY
33 {
34   FLASH (rx)      : ORIGIN = 0x00000000, LENGTH = 1024K
35   RAM (xrw)       : ORIGIN = 0x20000000, LENGTH = 192K
36 }
37 /* Конец РАМы на НИИЭТЕ (с запасиком) - отсюда внутрь будет расти стек*/
38 _estack = ORIGIN(RAM) + LENGTH(RAM) - 4;
39
40
41 /* Define output sections */
42 SECTIONS
43 {
44   . = ORIGIN(RAM);
45   /* таблица векторов прерываний, обязана быть выровнена по 0x80 */
46   .isr_vector ALIGN(0x80):
47   {
48     __isr_vector_flash_start = LOADADDR (.isr_vector); /* Берем адрес, где на флеше лежит эта таблица */
49     __isr_vector_ram_start = .;
50     KEEP(*(.isr_vector)) /* Startup code - KEEP запрещает удалить секцию сборщику мусора */
51     . = ALIGN(4);
52     __isr_vector_ram_end = .; /* конец оперативки, куда будет скопирован код */
53   } >RAM AT>FLASH
54
55   /* Секция для кода, который исполняется в оперативке, а лежит на флеше.
56    * В начале мейна он копируется из ROM в RAM.*/
57   .fastcode ALIGN(4): {
58      __fastcode_flash_start = LOADADDR (.fastcode); /* Берем адрес, где на флеше лежит код. */
59      /* Точка - это курсор текущего размещения в памяти. Т.е. в
60       * __fastcode_ram_start будет лежать адрес оперативки, где будет исполняться скопированный код.*/
```

Язык программирования: Си или Си++?

- Основные преимущества Си++ проявляются только при использовании динамического выделения памяти, от которого желательно отказаться.

- На Си++ есть сложность с получением адреса функции, когда она находится внутри класса (адрес метода класса). Это по-разному реализовано в разных компиляторах и не всегда совместимо. Получать адрес зачастую надо для реализации функции обратного вызова (callback).

- В Си++ меньше прозрачности при изучении итогового ассемблерного кода, в который иногда приходится заглядывать в сложных ситуациях.

- В Си отсутствуют классы, поэтому для получения модульной программы их необходимо «имитировать» средствами языка Си.

- Компиляция на Си идет обычно в два-три раза быстрее, чем на Си++.

Вывод:
Обоими языками при желании можно успешно пользоваться, но у каждого есть свои нюансы.

Использование операционных систем

- Существует множество операционных систем реального времени для микроконтроллеров: FreeRTOS, KeilRTX, µC/OS и множество других. Есть как платные, так и бесплатные открытые разработки.
- По сути операционная система выполняет функцию планировщика задач: какие функции, как часто и с каким приоритетом вызывать, а также как синхронизировать данные между ними.
- Задачи электропривода чаще всего можно достаточно красиво решить и без операционных систем, используя для разделения задач прерывания таймеров с нужной частотой и приоритетом, а также фоновый цикл.
- Некоторые операционные системы раскрывают свой функционал в полной мере только при использовании динамического выделения памяти.
- Использование любой операционной системы неизбежно снижает общую производительность программы за счет накладных расходов.
- Операционная система вносит новую сущность, с которой приходится разбираться пользователю при её использовании.
- В задачах электропривода зачастую синхронизация процессов должна быть настолько тонкой, что операционная система со своим обобщенным подходом к задачам может больше мешать, чем помогать.

Таблица векторов прерываний

- В ARM Cortex-M4 в ядро встроен контроллер прерываний – NVIC (Nested Vectored Interrupt Controller).

- Если возникает прерывание или исключение, ядро сохраняет контекст и обращается к таблице векторов прерываний, в которой по порядку расположены указатели на функции, которые нужно вызывать для обслуживания того или иного прерывания/исключения.

- По умолчанию эта таблица располагается по нулевому адресу, что соответствует началу flash памяти, но её расположение может быть переопределено при желании на другой адрес через регистр SCB->VTOR.

- Эта таблица заполняется пользователем. Обычно она уже вся заполнена в стартап-файле (startup_MCP_gcc.S или startup.c, startup.h для примеров VectorIDE), где каждому прерыванию ставится в соответствие некоторый обработчик по умолчанию с уникальным именем (например, USART0_RX_IRQHandler). Этот обработчик ничего не делает и останавливает выполнение программы. Но этот обработчик объявлен как «weak», что обозначает, когда в программе будет найдена другая пользовательская функция с именем USART0_RX_IRQHandler, то на этапе компиляции адрес обработчика по умолчанию заменится на адрес функции USART0_RX_IRQHandler.

Процесс запуска микроконтроллера

- Запуск микроконтроллера взаимосвязан с таблицей векторов прерываний.
- Первые два значения в таблице прерываний – специальные. Первое из них должно содержать начальное значение указателя стека (SP, Stack Pointer), а следующее указатель текущей выполняемой команды (PC, program counter).
- При включении, а также программной перезагрузке микроконтроллер прочитывает первые два значения таблицы векторов прерываний и инициализирует ими свои SP и PC.
- Пользователь должен сконфигурировать таблицу векторов прерываний так, чтобы SP содержал адрес в оперативной памяти, где выделено место под стек (обычно самый конец оперативной памяти, 0x2002FFFC для К1921ВК01Т), а PC содержал адрес функции, с которой нужно начать выполнение (обычно это функция инициализации переменных, еще до вызова main, в примерах VectorIDE это ассемблерная метка Reset_Handler или аналогичная СИшная ф-я).
- До передачи управления функции main должны быть выполнены следующие действия:
- Заполнение переменных в оперативной памяти значениями, которые записаны в соответствующей таблице во flash памяти (инициализируемые переменные).
- Заполнение нулями всех остальных не проинициализированных переменных.
- Включение модуля плавающей точки (если необходимо).
- Передача управления на функцию main.

Лабораторная работа по изучению процесса запуска

1. В настройках отладочной конфигурации отключить переход на main внизу на вкладке Startup
2. Выбрать в build.ld работу проекта из flash памяти
3. Очистить проект (Project – Clean)
4. Создать глобальную переменную int my_test = 123; Скомпилировать и прошить проект
5. Проходя по шагам в startup файле найти, где и когда инициализируется my_test

Работа с периферией микроконтроллера

- Для доступа к периферийным устройствам микроконтроллера используются регистры, расположенные в том же адресном пространстве, что и остальная память: можно обращаться к периферии как к ячейкам оперативной памяти.
- В заголовочном файле микроконтроллера используется структурный подход: можно обращаться к определенным битам периферии как к битовым полям в терминах языка Си.
- Работать с битовыми полями нужно с осторожностью, так как в некоторых случаях можно получить неожиданный результат. Изменение битового поля компилятором превращается в операцию чтение-модификация-запись, что для обычных ячеек памяти абсолютно допустимо, а для некоторых полей периферии – нет.
- Пример: `NT_UART1->DR_bit.DATA = 123;` //так делать нельзя!

Поле	Биты	Описание
OE, BE, PE, FE	11, 10, 9, 8	См. описание бит в регистре RSR_ECR
DATA	7-0	Поле данных. Результатом записи в поле DATA является размещение байта в буфере передатчика, а результатом чтения – считывание байта из буфера приемника
–	31-12	Зарезервировано

Работа с периферией микроконтроллера: GPIO

Можно использовать команды вида

```
NT_GPIOC->DATA &= ~(1 << 2);
```

Но порты физически меняют состояние не сразу, а через 2 такта (это связано с синхронизацией), и, если подряд написать 2 строчки, например

```
NT_GPIOC->DATA &= ~(1 << 2);
NT_GPIOC->DATA &= ~(1 << 3);
```

то первая с большой вероятностью не сработает, так как доступ к биту осуществляется сначала чтением всего слова, изменением нужного бита и записью его назад. Но чтение из GPIO возвращает текущее состояние порта, и к моменту выполнения второй команды (второй строки Си) прочитанное состояние не будет содержать изменений, которые сделала первая команда, поэтому вторая команда затрет тот бит, который хотела выставить первая команда.

При этом проблема проявляется только при включенной высокой оптимизации компилятора. Без оптимизации компилятор генерирует достаточно много ассемблерных команд между обращениями к GPIO и синхронизация успевает выполниться.

Работа с периферией микроконтроллера: GPIO

Другой способ - обращение к GPIO по маске (см. документацию): MASKLOWBYTE - к младшему байту регистра, MASKHIGHBYTE - к старшему.

Допустим, мы хотим выдать "1" на GPIO2, GPIO3 и GPIO4 порта C:
0001 1100 маска, является индексом в массиве MASKLOWBYTE
xxx1 11xx записываемое значение: в нужных битах - единицы, в остальных - не важно, т.к. они не попадают в маску и не будут изменены

```
NT_GPIOC->MASKLOWBYTE_bit[28].MASKLB = 28;   //28 = 11100b
```

так выглядит код команды, как вариант:

```
NT_GPIOC->MASKLOWBYTE_bit[28].MASKLB = 0xFF;
```

если надо включить GPIO2 и GPIO4 и выключить GPIO3, то:

```
NT_GPIOC->MASKLOWBYTE_bit[28].MASKLB = 20;   //20 = 0001 0100b
```

Доступ к битам порта по маске **гарантирует**, что **другие биты**, которые не попадают в маску, будут **не затронуты**.

Также доступ по маске позволяет **одновременно одной командой** переключить два бита порта, если это требуется.

Инициализация тактирования

Смотрим файл K1921VK01T_init.c из проекта VectorIDE

K1921VK01T_init.c

```
171
172 void InitCLK(void)
173 {
174     volatile int i;
175     NT_BOOTFLASH->T_ACC = 6;        //Задержка от установки адреса, до считывания данных из флеш-памяти (в транзакциях чтения)
176
177     //Настройка частоты тактирования
178     //выходная частота равна FOUT = (FIN * NF) / ( NR * NO), где FIN - частота кварца
179     NT_COMMON_REG->PLL_OD = 2;      //Выходной делитель PLL NO=2
180     NT_COMMON_REG->PLL_NR = 1;      //Опорный делитель PLL NR=R_PLL+2=3
181 #ifdef QUARTZ_10MHZ
182     NT_COMMON_REG->PLL_NF = 58;     //Делитель обратной связи PLL NF=F_PLL+2=60
183 #endif
184 #ifdef QUARTZ_12MHZ
185     NT_COMMON_REG->PLL_NF = 48;     //Делитель обратной связи PLL NF=F_PLL+2=50
186 #endif
187     //FOUT = 12*50/(3*2) = 100 МГц, вроде сходится
188     do
189     {
190         //Выбор источника синхросигнала
191         NT_COMMON_REG->SYS_CLK = SYSCLK_REFCLK;     //Выбор источника зависит от состояния сигнала на входе микроконтроллера CPE_pad: 0 - Блок POR,
192         NT_COMMON_REG->SYS_CLK = SYSCLK_PLLCLK;     //Блок PLL
193         for(i = 0; i < 50; i++);
194     }
195     while (NT_COMMON_REG->SYS_CLK_bit.CURR_SRC != SYSCLK_PLLCLK);   //Текущий источник тактирования должен совпадать с выбранным
196
197     // Разрешение работы периферии
198     NT_COMMON_REG->APB_CLK = 0x7FFFF | 0x1000000 | 0x80000;
199
200     for (int i = 0; i < 100; i++){//чтобы типа прошло время... пускай там порезетится
201         NT_COMMON_REG->PER_RST0=0;
202         NT_COMMON_REG->PER_RST1=0;
203     }
204
205     NT_COMMON_REG->PER_RST0 = 0xFFFFFFFF;
206     NT_COMMON_REG->PER_RST1 = 0xFFFFFFFF;
207     NT_COMMON_REG->GPIODEN0=0xFFFFFFFF;
208     NT_COMMON_REG->GPIODEN1=0xFFFFFFFF;
209     NT_COMMON_REG->GPIODEN2=0xFFFFFFFF;
210     NT_COMMON_REG->GPIODEN3=0xFFFFFFFF;
```

Лабораторная работа управлению ШИМ

Смотрим ТО на TO_K1921VK01T.pdf, глава «14 Блоки ШИМ».

Смотрим на схемотехнику VectorCARD.

Изучаем пример Example_LED_blinking в VectorIDE

Изучаем пример Example_PWM в VectorIDE

Изучаем пример Example_Timer в VectorIDE

Практическая работа: совместить примеры с управлением ШИМ и пример с организацией прерываний: сделать программу, изменяющую яркость светодиода по закону синуса частотой 1Гц.