Пример 1.

Быстрый старт для K1986BE92QI

Введение

В данном примере последовательно будет показано создание самого элементарного проекта, а также рассмотрены некоторые схемотехнические аспекты проектирования устройств на микроконтроллерах, и не только компании Миландр. Также приведены минимальные настройки среды разработки для работы с МК K1986BE92QI.

На конечном шаге будет произведена загрузка получившегося в результате компиляции .hex файла в микроконтроллер, посредством утилиты 1986UARTWSD.

Таблица 1 – Минимальный набор аппаратного и программного обеспечения для примера 1.

Инструментальная база	Наименование	Вэб-адрес источника
Отладочный комплект	LDM-K1986BE92QI	www.ldm-systems.ru
Драйвер преобразователя USB/RS-232	Silicon Laboratories CP210x VCP Drivers for Windows XP/2003 Server/Vista/7	www.silabs.com
Среда разработки ПО для МК	Keil uVision 4.72a	www.keil.com
Утилита прошивки МК через UART-загрузчик	1986UARTWSD	forum.milandr.ru

Техническое задание.

Разработать программное обеспечение для отображения состояния кнопки SW1 светодиодом VD5 на базе отладочного комплекта LDM-K1986BE92QI.

Схемотехника.

В радиоэлектронике зачастую используется метод эквивалентных схем, позволяющий отобразить основные свойства какого-либо элемента или схемы в более упрощённом или наоборот, более детализированном варианте.

Нас интересует именно упрощение схемотехники, поскольку демонстрационные и отладочные наборы, в силу своего предназначения, как правило, изначально многофункциональны, а стало быть, под нашу задачу избыточны. Полная схема отладочного комплекта (далее ОК) приведена в документации на него и доступна для загрузки на сайте компании-производителя.

Итак, ниже на рисунке 1 приведена эквивалентная схема отладочного комплекта. Здесь убрано всё лишнее и оставлены только принципиально важные вещи, однако позиционные обозначения элементов оставлены такими, как они обозначены в исходной схеме. Далее рассмотрим основные аспекты получившейся принципиальной схемы.

Питание.

На схеме не показаны формирователи напряжения +3.3 и +5 Вольт, а также помехоподавляющие конденсаторы, выключатели и перемычки выбора источника дополнительного и основного питания, это сделано в целях упрощения, но будем иметь ввиду, что на плате всё это есть, хотя для понимания основного принципа работы нам это не критично.

На схеме видно, что питание подключено к выводам микроконтроллера с различными названиями, определяющими его предназначение. Основное напряжение, питающее ядро и большую часть периферийных блоков МК обозначено, как UCC. С точки зрения трассировки печатной платы в нём нет ничего особенного, кроме того, что оно присутствует практически на каждой из 4ёх сторон микросхемы.

Наиболее требовательными, с точки зрения помехоустойчивости, портами питания являются выводы AUCC и AUCC1. Именно с них питается вся аналоговая периферия внутри микроконтроллера, включая физическую часть интерфейса USB, поэтому при разработке схемы, где требуется работа с АЦП, ЦАП, внутренним компаратором или USB схемотехнически желательно предусмотреть дополнительную развязку от основного напряжения UCC.

Таких подходов может быть несколько и самым радикальным из них является питание от отдельного источника, однако для абсолютного большинства случаев вполне хватит развязки через ферритовые бусинки для поверхностного монтажа или ЭМИ-фильтры с конденсаторами, ёмкостью 0.01-0.1 мкФ и 4.7…47мкф возле каждого вывода. Кроме того, если от аналоговой периферии вроде АЦП или ЦАП планируется получить максимально возможные параметры вроде SNR и SFDR то особое внимание следует также уделить правилам разводки «цифровой» и «аналоговой» земель, которые у МК обозначены как GND и AGND соответственно. Впрочем, если к аналоговой периферии не предъявлять высоких требований, то в большинстве случаев с лихвой хватает установки возле каждого вывода конденсаторов, ёмкостью 0.01-0.1 мкФ.

Ещё одна прописная истина, на которую не обращает внимания большая часть радиолюбителей и производителей бытовой аппаратуры (в особенности стиральных машин) без привязки к какому-либо МК – если вы тяните через всю плату питающий проводник, то как минимум через каждые 10-15мм необходимо устанавливать конденсаторы, и в данном случае разных емкостей, например 0.01 и 0.1 мкФ, поскольку в противном случае питающий проводник становится эквивалентен большой антенне, собирающей все окрестные шумы и помехи.

Из этого также следует, что по возможности проводники питания следует делать, как можно более короткими и уменьшать ширину питающего проводника лишь непосредственно при подведении к выводу МК.

Следующий питающий порт, обозначенный как BUCC предназначен для батарейного домена микроконтроллера. В нашем случае, поскольку работа с ним пока не планируется, он просто соединён с UCC, однако на самой отладочной плате возможности для подключения внешней батареи присутствуют.

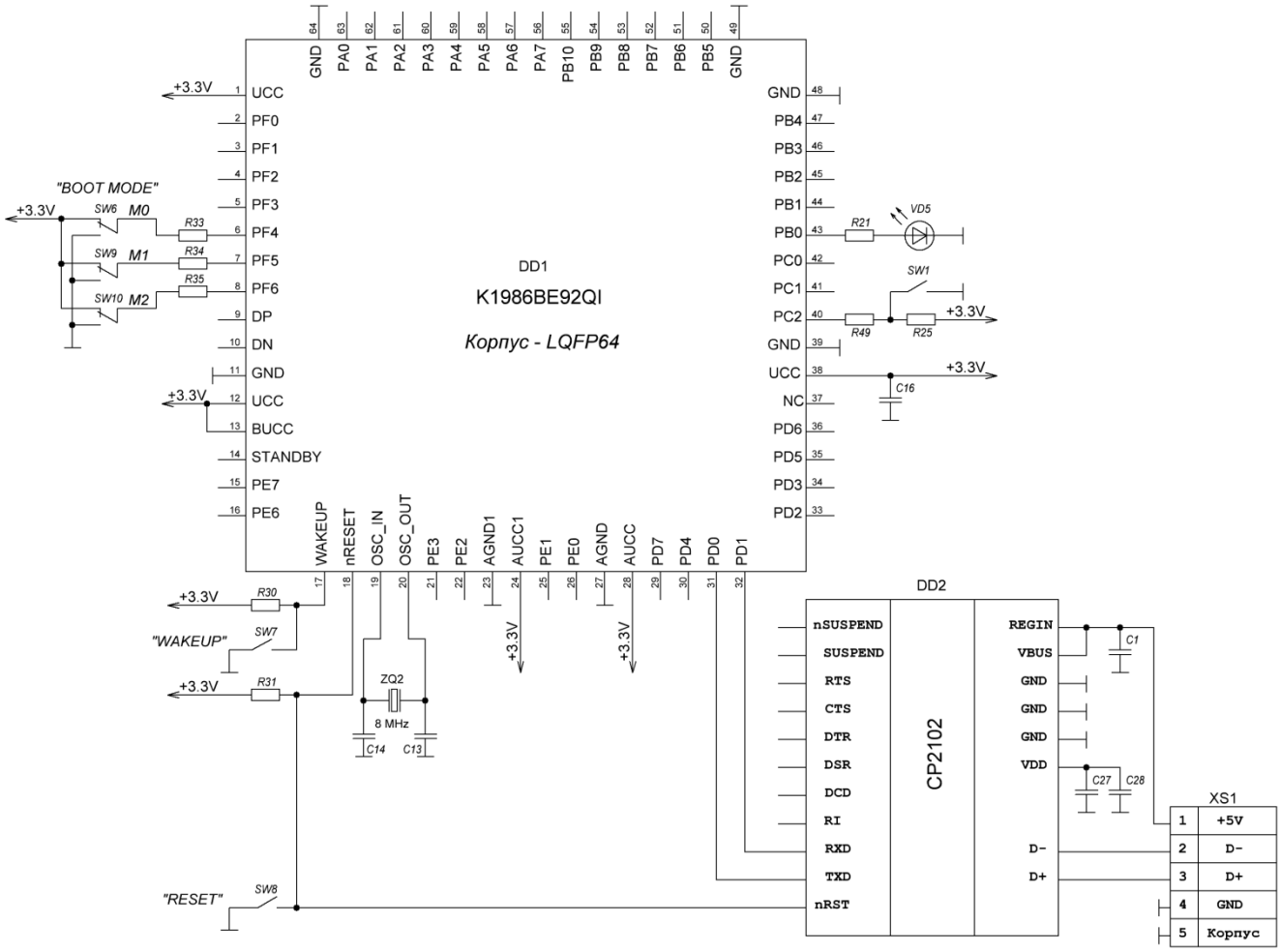


Рисунок 1. Эквивалентная схема отладочного комплекта LDM-K1986BE92QI для простейшего примера индикации состояния кнопки SW1 светодиодом VD5.

Загрузка программного обеспечения.

Вообще, на отладочной плате присутствуют разъёмы для подключения к любому из JTAG-портов микроконтроллера (их у него два, JTAG-A или JTAG-B, любой на выбор, поскольку они равнозначны). Работа с ними, сам процесс программирования или отладки многократно описаны во множестве литературы, если не привязываться к какому-то типу МК конкретно. Нас же интересует другой путь загрузки ПО, а именно – через встроенный в микроконтроллер UART-загрузчик.

Для нашего варианта это будет наиболее простой способ. Во первых, по словам самих разработчиков это «железобетонная конструкция», т.е если кто-то, где-то, что-то не так запрограммировал, то это всегда можно устранить через него. Во-вторых, если рассматривать вопрос серийного производства, то тут есть несомненный плюс в том, что не потребуется приобретать специализированного ПО и программаторов, которых к тому же может потребоваться не один, а сам процесс загрузки файла прошивки значительно упрощается. Ну и в третьих, разработчики часто даже просто для отладки оставляют проверенный временем, простой и надёжный «как топор» интерфейс RS-232.

Итак, поскольку загрузка ПО происходит по UART-интерфейсу, то следовательно необходим способ выбора работы с ним в пользовательской программе МК или загрузки ПО. Если бы этого не было, то микроконтроллер не был бы защищён от несанкционированной или случайной смены/повреждения залитой в него прошивки. А все мы знаем, как при покупке какой-либо электронной техники люди любят сразу же искать «прошивку поновее», после чего немалая

часть устройств нередко «отбрасывает коньки». В данном контроллере всё решено на аппаратном уровне.

Переход в режим загрузки ПО происходит после включения питания и только с предварительно выставленной комбинацией логических уровней на выводах PF4, PF5 и PF6 микроконтроллера.

Различные варианты данной конфигурации описаны в спецификации на МК, но в нашем случае интересны только два состояния:

Таблица 2 – Режимы первоначального запуска МК (из таблицы 9 спецификации на МК)

MODE[2:0] Выводы PF4, PF5, PF6 соответственно	Режим	Стартовый адрес / таблица векторов прерываний	Описание
000	МК в режиме отладки	0x0800_0000	Процессор начинает выполняет программу из внутренней Flash-памяти программ. При этом установлен отладочный интерфейс JTAG_B
001	МК в режиме отладки	0x0800_0000	Процессор начинает выполняет программу из внутренней Flash-памяти программ. При этом установлен отладочный интерфейс JTAG_A
101	UART загрузчик	Определяется пользователем	Микроконтроллер через интерфейс UART2 на выводах PD[1:0] получает код программы в ОЗУ для исполнения
110	UART загрузчик	Определяется пользователем	Микроконтроллер через интерфейс UART2 на выводах PF[1:0] получает код программы в ОЗУ для исполнения

В следующих разделах будет показана работа с утилитой прошивки МК через UART-загрузчик, а сейчас рассмотрим оставшиеся элементы на схеме.

Микросхема DD2 – это интерфейсный преобразователь USB/RS-232 выполненный на базе микросхемы CP2102. Она подключена к портам PD0 и PD1, поэтому, для перевода в режим UART-загрузчика мы в последующем тексте будем режим MODE устанавливать в положение «**101**», перед включением питания. Следует учесть, что перед началом работы с отладочной платы необходимо установить на компьютер драйвер для микросхемы CP2102.

На плате также, опционально может присутствовать классический приёмопередатчик интерфейса RS-232, выполненный на микросхеме ADM3202ARN. Через него также можно работать с аппаратным UART-загрузчиком, но поскольку в полной схеме он подключён к выводам PF0 и PF1 микроконтроллера, то режим MODE должен быть выставлен в положение «**110**».

Поскольку данный интерфейс опционален, в эквивалентной схеме он отсутствует. К плюсам данного интерфейса следует отнести к отсутствию необходимости установки драйверов на компьютер, а к минусам – не на каждом современном ПК он присутствует.

Служебные кнопки и кварцевый резонатор.

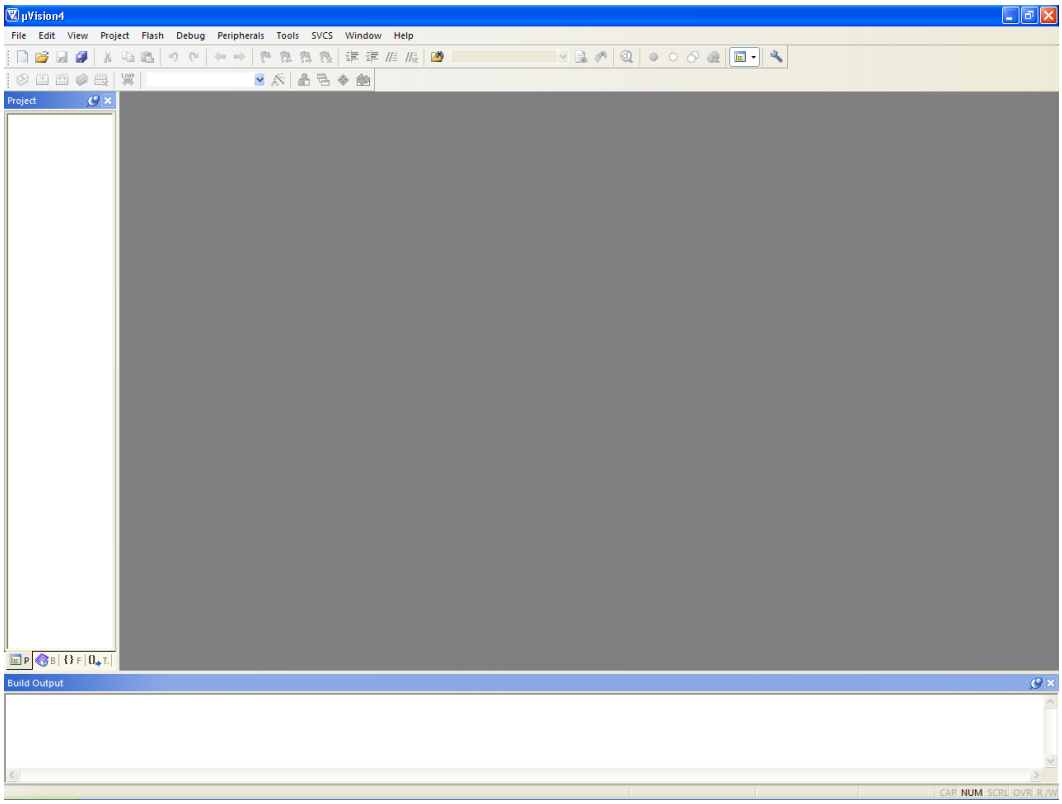
На отладочной плате также дополнительно присутствуют:

- Кнопка для сброса микроконтроллера «RESET», где уже из наименования вывода на микроконтроллере (nRESET) следует, что МК сбрасывается в исходное состояние логическим нулём. Также, при нажатии на кнопку сбросится и микросхема преобразователя USB/RS-232.
- Кнопка сигнала внешнего выхода из режима дежурного режима «WAKEUP». При её нажатии МК должен выйти из дежурного режима, если в таковой был установлен. В принципе её может и не быть, если режим не используется, и тогда вывод «WAKEUP» просто должен быть подтянут к питанию. На эквивалентной схеме она показана лишь для наглядности.
- Кварцевый резонатор ZQ2. В нашем техническом задании не указано, что мы должны работать на максимальной частоте, а поскольку у МК есть ещё внутренний RC-генератор, то для выполнения требований ТЗ его стабильности с лихвой хватит. На эквивалентной схеме он тоже показан «постольку поскольку».

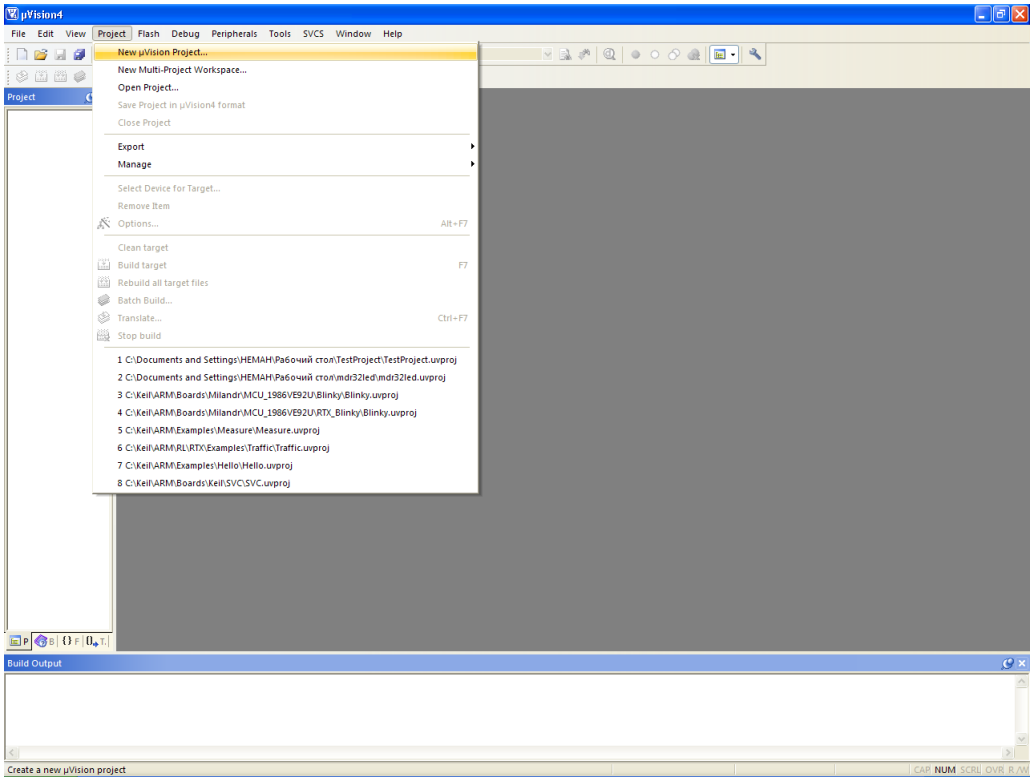
И в общем итоге остались собственно сам светодиод VD5, подключённый к порту PB0 и кнопка SW1, подключённая к порту PC2 через резистор R49. Данный резистор установлен на случай, если пользователь случайно установит порт, подключённый к кнопке на «вывод» и при нажатии на неё замкнёт PC2 на корпус. Конденсатор C16 в данном случае символизирует многочисленное количество фильтрующих емкостей на плате.

Создание простейшего проекта и настройка среды разработки Keil uVision ver. 4.72a

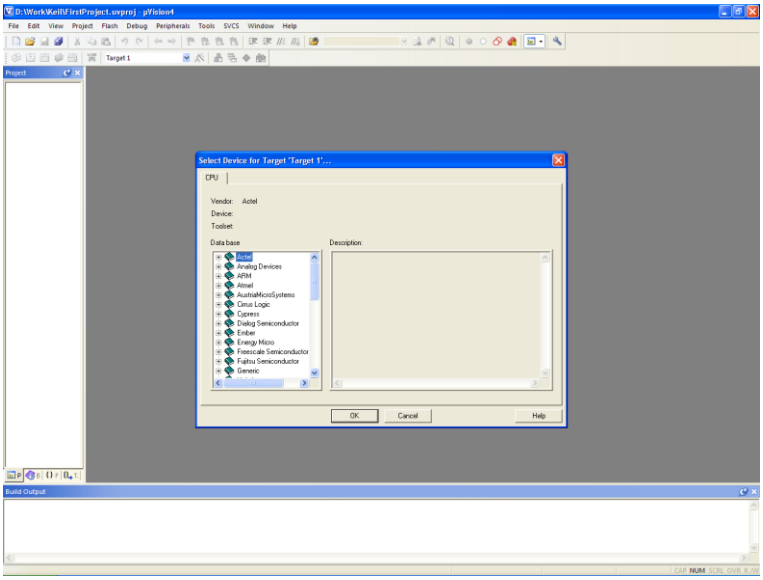
После установки **Keil uVision ver. 4.72a** запускаем и видим следующую картинку:



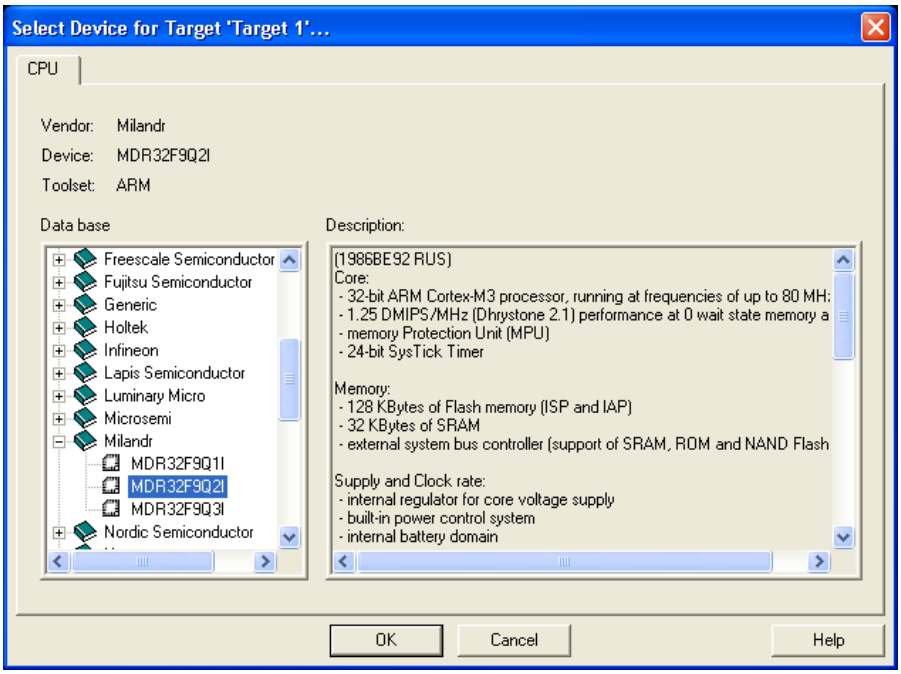
Создаём новый проект.
Для этого на вкладке «**Project**» программы, выбираем «**New uVision Project…**»



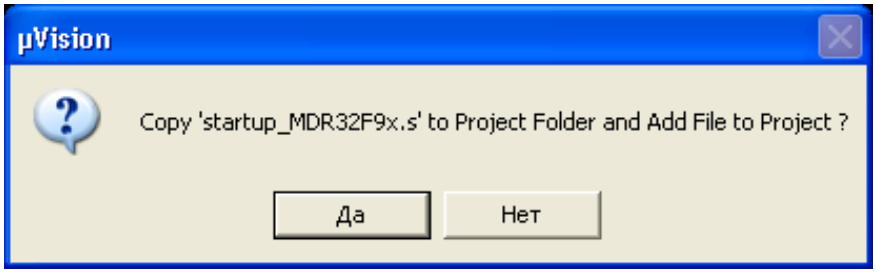
После этого появляется окошко



Где выбираем «**Milandr**», а в нём «**MDR32F9Q2I**»

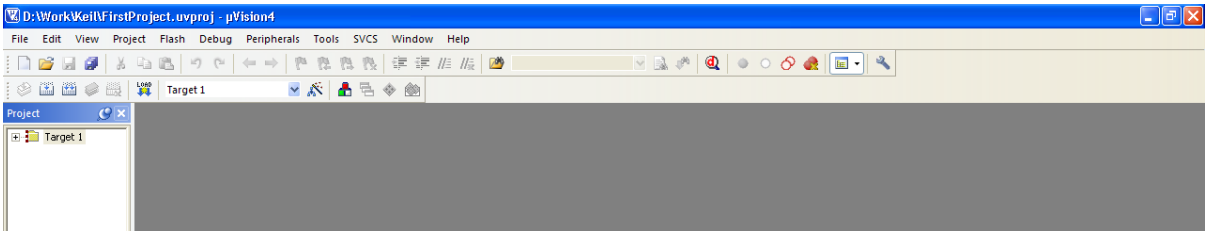


Нажимаем «**OK**» , после чего появляется окошко с вопросом

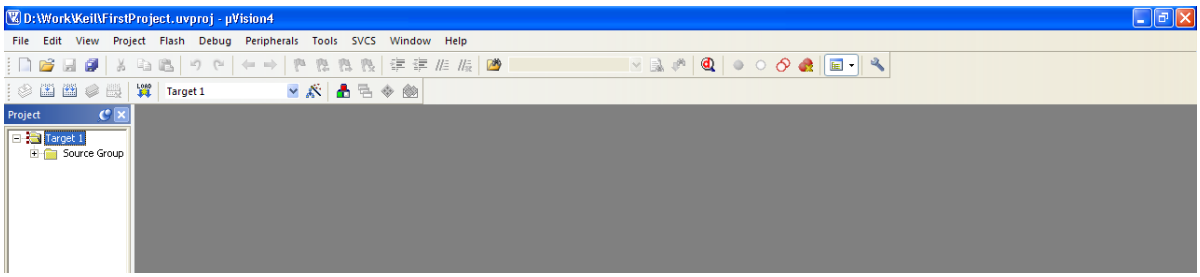


Это файл инициализации контроллера, жмём «**Да**»

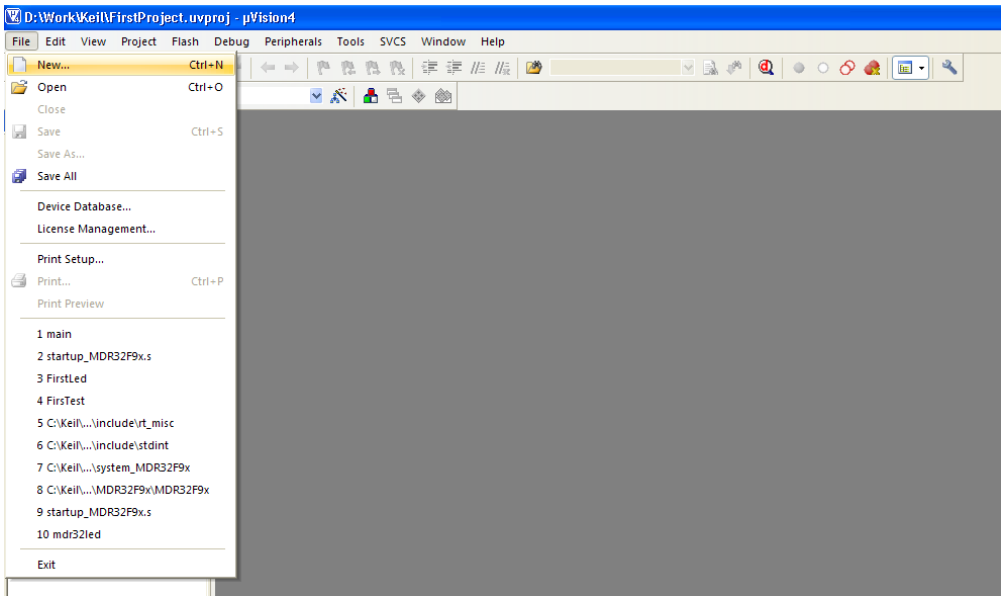
После чего наблюдаем, как слева, в окне программы «**Project**», появилась папка «**Target 1**» .



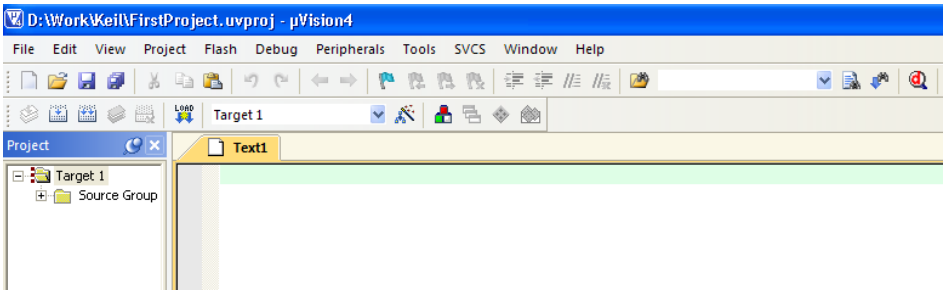
Нажимаем на плюсик, после чего, внутри папки «**Target 1**» появляется папка «**Source Group 1**». В дальнейшем можно будет переименовать папку на своё усмотрение, а пока в ней будут размещены все необходимые нам для этого файлы.



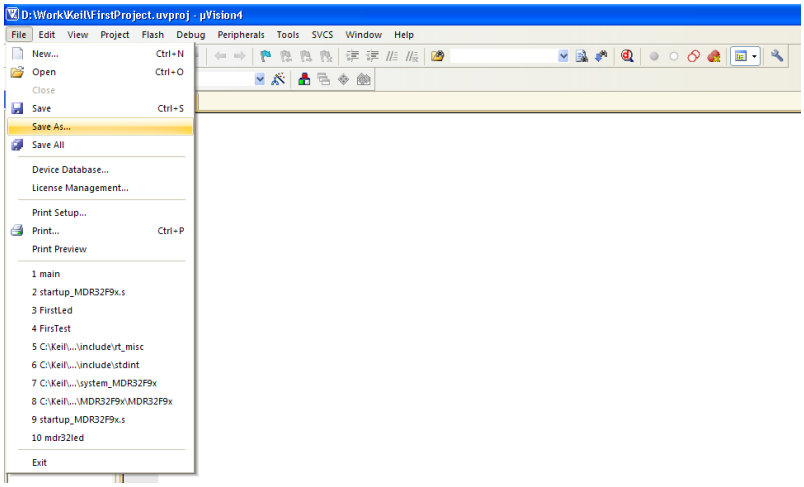
Создадим файл, в котором будет основной код программы. Для этого в меню «**File**» выбираем «**New…**»



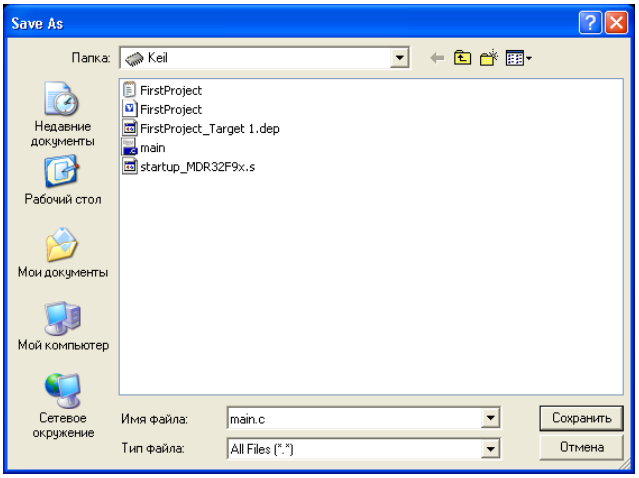
В результате видим, что в окне слева появилось белое поле с именем «**Text1**»



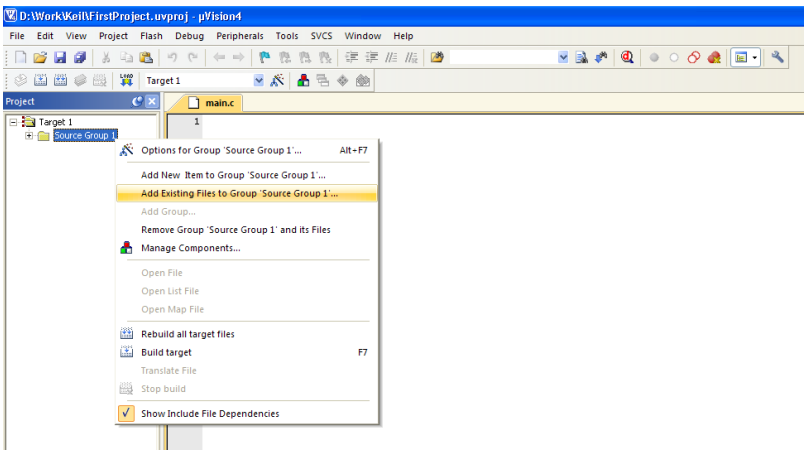
В меню файл выбираем «**Save As…**»



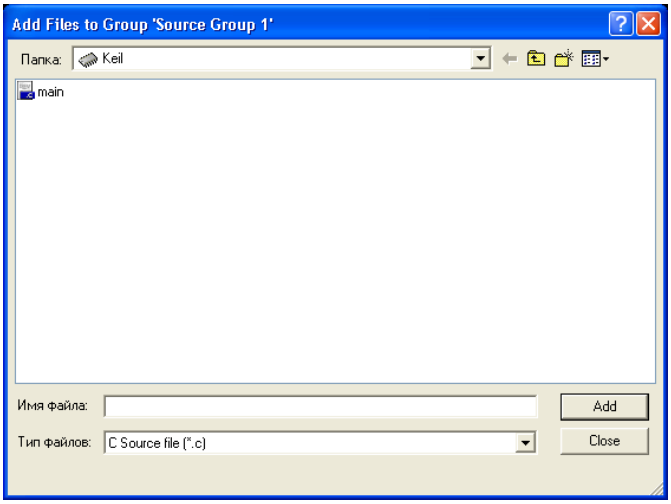
После чего появится окошко, в котором пишем «**имя_нашего_файла. расширение**». В нашем случае это будет «**main.c**», потому что выполнение любой программы всегда начинается с функции «**main**», данная функция обязательно должна присутствовать в любой программе написанной на Си. В принципе, путём некоторых манипуляций, само название главной функции можно изменить, но сейчас нас больше интересуют более приземлённые вещи.



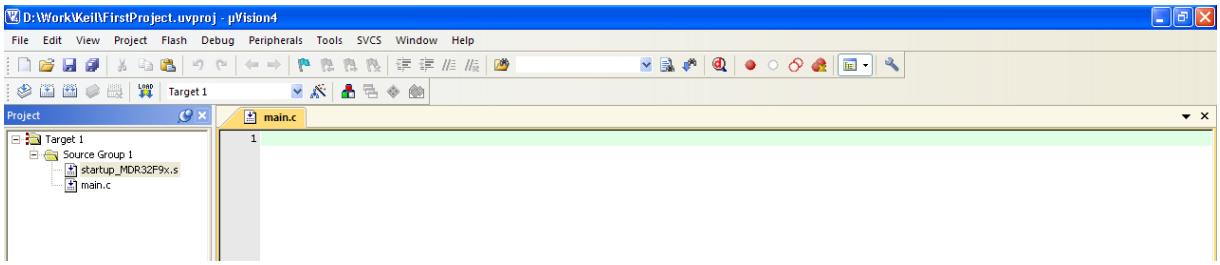
Теперь необходимо файл, сохранённый как «**main.c**» добавить в проект. Для этого жмём правой кнопкой мыши на папку «**Source Group 1**» и в выпадающем меню выбираем «**Add Existing Files to Group 'Source Group 1'**»



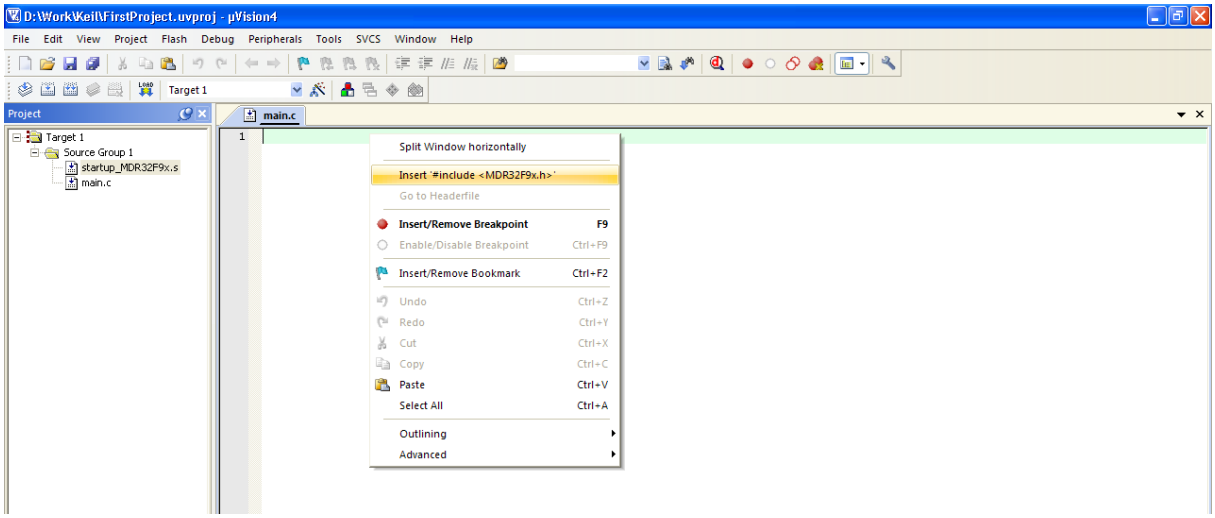
Появилось окошко



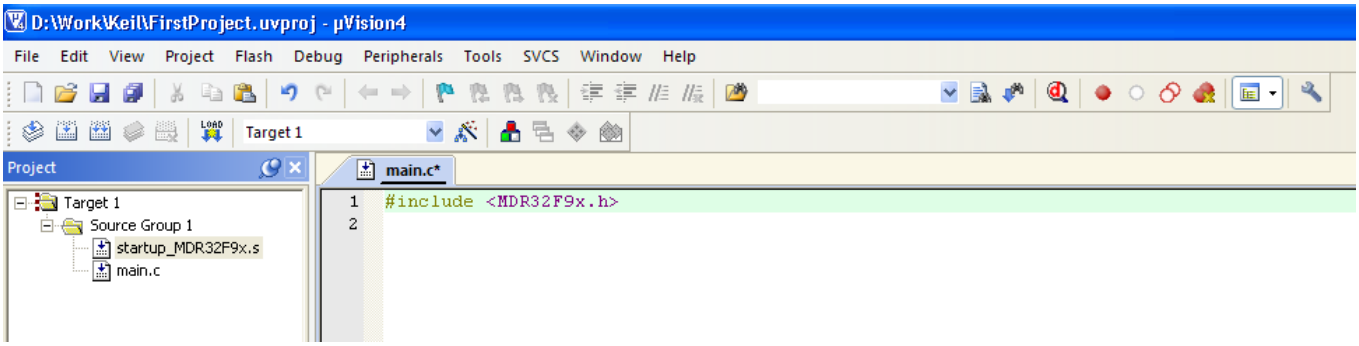
В котором указываем на наш «**main**».
В результате, в папке «**Source Group 1**» видим уже два файла. Файл инициализации контроллера «**startup_MDR32F9x.s**» и созданный нами файл «**main.c**»



Далее необходимо добавить в файл «**main.c**» заголовочный h-файл семейства микроконтроллера для которого мы делаем проект. Для этого жмём правой кнопкой мыши на белое поле файла «**main.c**», где в выпадающем меню выбираем «**Insert '#include <MDR32F9x.h>'**»



В результате появилась строчка



Далее необходимо добавить в проект файл «**system_MDR32F9x.c**» также, как мы добавляли «**main.c**». Его необходимо взять из библиотеки «**MDR32F9x Standart Peripheral Library**». Сама библиотека поставляется в комплекте с платой «**LDM-K1986BE92QI**». Её можно также взять на форуме компании «ЗАО «ПКК Миландр»:

http://forum.milandr.ru

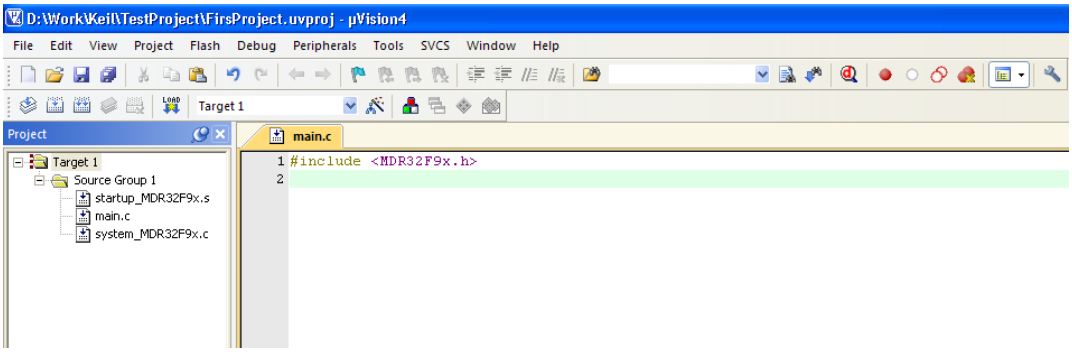
в разделе: *Интегральные микросхемы ЗАО "ПКК Миландр"→32-разрядные микроконтроллеры (1986BE9x, 1986BE1x, 1986BE2x) →32-разрядные микроконтроллеры серии 1986BE9x (ядро ARM Cortex-M3)→MDR32F9x Standart Peripheral Library*

Или на сайте компании

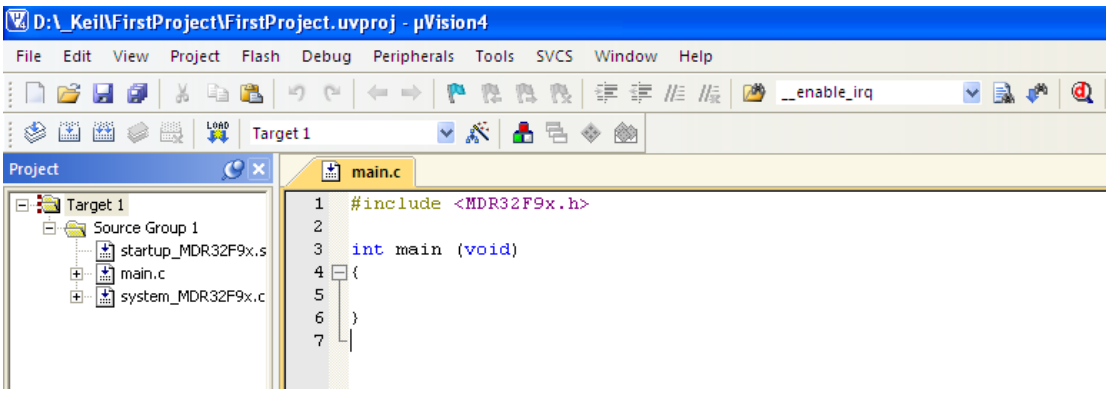
http://milandr.ru

в разделе «Программное обеспечение», или на диске, поставляемом компанией вместе с отладочными комплектами.

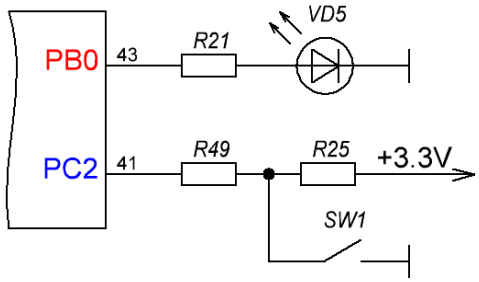
В результате в папке «**Source Group 1**» уже три файла: Файл инициализации контроллера, системный и наш, в котором будем писать код программы.



Далее, добавим в наш файл **main.c** собственно код самой функции main, который пока ничего не будет делать, но внутри которого потом будет размещён основной алгоритм работы программы.



Теперь попробуем представить, с точки зрения технического задания и самой программы саму схему



У нас есть два вывода МК, к одному из них (PB0 – нулевой бит **32ух разрядного регистра «B»**, отвечающего за **16ти разрядный порт «B»**) подключен светодиод. Сам он ничего не даёт контроллеру, а лишь отображает уровень напряжения на его выводе. Значит, наш светодиод принимает данные, а микроконтроллер их передаёт.

К другому выводу подключена кнопка. Она ничего не может отобразить, зато при нажатии на неё, на выводе PC2 (второй бит **32ух разрядного регистра «C»**, отвечающего за **16ти разрядный порт «C»**) формируется логический ноль. Значит, микроконтроллер принимает данные от кнопки SW1.

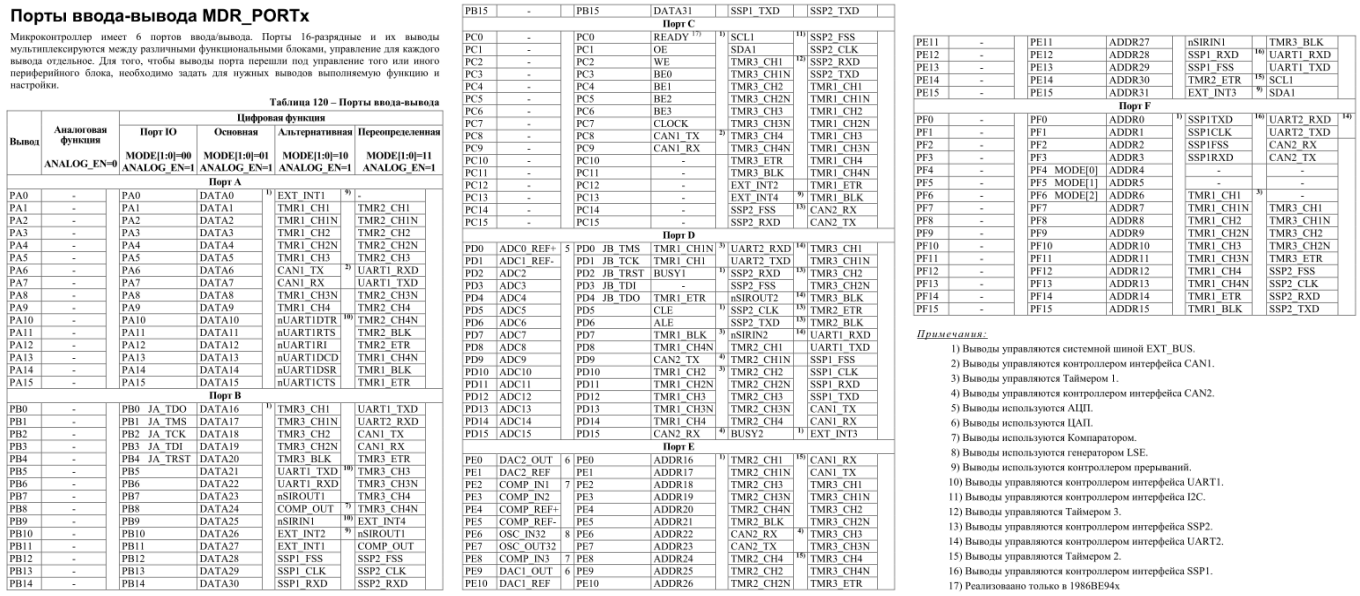
В самом начале необходимо включить тактирование наших периферийных модулей. В нашем случае это порты «B» и «C». Для этого идём в раздел спецификации «Сигналы тактовой частоты MDR_RST_CLK». В нём описывается структура формирования и настройки тактовых частот от внутреннего или внешнего генератора. Если не разрешить тактирование периферии, то соответственно на регистры портов в нашем случае, не поступит тактовая частота и даже наш, самый простой проект работать не будет.

```
MDR_RST_CLK->PER_CLOCK   |= (1 << 22);              // Разрешили тактирование порта B
MDR_RST_CLK->PER_CLOCK   |= (1 << 23);              // Разрешили тактирование порта C
```

Теперь необходимо настроить порты ввода-вывода. Для этого открываем спецификацию (можно взять на сайте компании-производителя) с длинным названием:

«Серия 1986BE9x, K1986BE9x, K1986BE92QI, K1986BE92QC, K1986BE91H4,
высокопроизводительных 32-разрядных микроконтроллеров
на базе процессорного ядра ARM Cortex-M3»

и идём в раздел «Порты ввода-вывода MDR_PORTx». Перед нами открывается таблица:



Порты ввода-вывода MDR_PORTx

Микроконтроллер имеет 6 портов ввода/вывода. Порты 16-разрядные и их выводы мультиплексируются между различными функциональными блоками, управление для каждого вывода отдельное. Для того, чтобы выводы порта перешли под управление того или иного периферийного блока, необходимо задать для нужных выводов выполняемую функцию и настройки.

Таблица 120 – Порты ввода-вывода

Вывод	Аналоговая функция	Цифровая функция			
		Порт IO	Основная	Альтернативная	Переопределенная
	ANALOG_EN=0	MODE[1:0]=00 ANALOG_EN=1	MODE[1:0]=01 ANALOG_EN=1	MODE[1:0]=10 ANALOG_EN=1	MODE[1:0]=11 ANALOG_EN=1
Порт A					
PA0	-	PA0	DATA0	EXT_INT1 [9]	-
PA1	-	PA1	DATA1	TMR1_CH1	TMR2_CH1
PA2	-	PA2	DATA2	TMR1_CH1N	TMR2_CH1N
PA3	-	PA3	DATA3	TMR1_CH2	TMR2_CH2
PA4	-	PA4	DATA4	TMR1_CH2N	TMR2_CH2N
PA5	-	PA5	DATA5	TMR1_CH3	TMR2_CH3
PA6	-	PA6	DATA6	CAN1_TX [2]	UART1_RXD
PA7	-	PA7	DATA7	CAN1_RX	UART1_TXD
PA8	-	PA8	DATA8	TMR1_CH3N	TMR2_CH3N
PA9	-	PA9	DATA9	TMR1_CH4	TMR2_CH4
PA10	-	PA10	DATA10	nUART1DTR [10]	TMR2_CH4N
PA11	-	PA11	DATA11	nUART1RTS	TMR2_BLK
PA12	-	PA12	DATA12	nUART1RI	TMR2_ETR
PA13	-	PA13	DATA13	nUART1DCD	TMR1_CH4N
PA14	-	PA14	DATA14	nUART1DSR	TMR1_BLK
PA15	-	PA15	DATA15	nUARTCTS	TMR1_ETR
Порт B					
PB0	-	PB0 JA_TDO	DATA16 [1]	TMR3_CH1	UART1_TXD
PB1	-	PB1 JA_TMS	DATA17	TMR3_CH1N	UART2_RXD
PB2	-	PB2 JA_TCK	DATA18	TMR3_CH2	CAN1_TX
PB3	-	PB3 JA_TDI	DATA19	TMR3_CH2N	CAN1_RX
PB4	-	PB4 JA_TRST	DATA20	TMR3_BLK	TMR3_ETR
PB5	-	PB5	DATA21	UART1_TXD [10]	TMR3_CH3
PB6	-	PB6	DATA22	UART1_RXD	TMR3_CH3N
PB7	-	PB7	DATA23	nSIROUT1	TMR3_CH4
PB8	-	PB8	DATA24	COMP_OUT [7]	TMR3_CH4N
PB9	-	PB9	DATA25	nSIRIN1 [10]	EXT_INT4
PB10	-	PB10	DATA26	EXT_INT2 [9]	nSIROUT1
PB11	-	PB11	DATA27	EXT_INT1	COMP_OUT
PB12	-	PB12	DATA28	SSP1_FSS	SSP2_FSS
PB13	-	PB13	DATA29	SSP1_CLK	SSP2_CLK
PB14	-	PB14	DATA30	SSP1_RXD	SSP2_RXD
PB15	-	PB15	DATA31	SSP1_TXD	SSP2_TXD
Порт C					
PC0	-	PC0	READY [17]	SCL1 [11]	SSP2_FSS
PC1	-	PC1	OE	SDA1	SSP2_CLK
PC2	-	PC2	WE	TMR3_CH1 [12]	SSP2_RXD
PC3	-	PC3	BE0	TMR3_CH1N	SSP2_TXD
PC4	-	PC4	BE1	TMR3_CH2	TMR1_CH1
PC5	-	PC5	BE2	TMR3_CH2N	TMR1_CH1N
PC6	-	PC6	BE3	TMR3_CH3	TMR1_CH2
PC7	-	PC7	CLOCK	TMR3_CH3N	TMR1_CH2N
PC8	-	PC8	CAN1_TX [2]	TMR3_CH4	TMR1_CH3
PC9	-	PC9	CAN1_RX	TMR3_CH4N	TMR1_CH3N
PC10	-	PC10	-	TMR3_ETR	TMR1_CH4
PC11	-	PC11	-	TMR3_BLK	TMR1_CH4N
PC12	-	PC12	-	EXT_INT2	TMR1_ETR
PC13	-	PC13	-	EXT_INT4 [9]	TMR1_BLK
PC14	-	PC14	-	SSP2_FSS [13]	CAN2_RX
PC15	-	PC15	-	SSP2_RXD	CAN2_TX
Порт D					
PD0	ADC0_REF+ [5]	PD0 JB_TMS	TMR1_CH1N [3]	UART2_RXD [14]	TMR3_CH1
PD1	ADC1_REF-	PD1 JB_TCK	TMR1_CH1	UART2_TXD	TMR3_CH1N
PD2	ADC2	PD2 JB_TRST	BUSY1 [1]	SSP2_RXD [13]	TMR3_CH2
PD3	ADC3	PD3 JB_TDI	-	SSP2_FSS	TMR3_CH2N
PD4	ADC4	PD4 JB_TDO	TMR1_ETR	nSIROUT2 [14]	TMR3_BLK
PD5	ADC5	PD5	CLE [1]	SSP2_CLK [13]	TMR2_ETR
PD6	ADC6	PD6	ALE	SSP2_TXD [13]	TMR2_BLK
PD7	ADC7	PD7	TMR1_BLK [3]	nSIRIN2 [16]	UART1_RXD
PD8	ADC8	PD8	TMR1_CH4N	TMR2_CH1	UART1_TXD
PD9	ADC9	PD9	CAN2_TX [4]	TMR2_CH1N	SSP1_FSS
PD10	ADC10	PD10	TMR1_CH2 [3]	TMR2_CH2	SSP1_CLK
PD11	ADC11	PD11	TMR1_CH2N	TMR2_CH2N	SSP1_RXD
PD12	ADC12	PD12	TMR1_CH3	TMR2_CH3	SSP1_TXD
PD13	ADC13	PD13	TMR1_CH3N	TMR2_CH3N	CAN1_TX
PD14	ADC14	PD14	TMR1_CH4	TMR2_CH4	CAN1_RX
PD15	ADC15	PD15	CAN2_RX [4]	BUSY2 [1]	EXT_INT3
Порт E					
PE0	DAC2_OUT [6]	PE0	ADDR16 [1]	TMR2_CH1 [15]	CAN1_RX
PE1	DAC2_REF	PE1	ADDR17	TMR2_CH1N	CAN1_TX
PE2	COMP_IN1 [7]	PE2	ADDR18	TMR2_CH3	TMR3_CH1
PE3	COMP_IN2	PE3	ADDR19	TMR2_CH3N	TMR3_CH1N
PE4	COMP_REF+	PE4	ADDR20	TMR2_CH4N	TMR3_CH2
PE5	COMP_REF-	PE5	ADDR21	TMR2_BLK	TMR3_CH2N
PE6	OSC_IN32 [8]	PE6	ADDR22	CAN2_RX [4]	TMR3_CH3
PE7	OSC_OUT32	PE7	ADDR23	CAN2_TX	TMR3_CH3N
PE8	COMP_IN3 [7]	PE8	ADDR24	TMR2_CH4 [15]	TMR3_CH4
PE9	DAC1_OUT [6]	PE9	ADDR25	TMR2_CH2	TMR3_CH4N
PE10	DAC1_REF	PE10	ADDR26	TMR2_CH2N	TMR3_ETR
PE11	-	PE11	ADDR27	nSIRIN1	TMR3_BLK
PE12	-	PE12	ADDR28	SSP1_RXD [16]	UART1_RXD
PE13	-	PE13	ADDR29	SSP1_FSS	UART1_TXD
PE14	-	PE14	ADDR30	TMR2_ETR [15]	SCL1
PE15	-	PE15	ADDR31	EXT_INT3 [9]	SDA1
Порт F					
PF0	-	PF0	ADDR0 [1]	SSP1TXD [16]	UART2_RXD [14]
PF1	-	PF1	ADDR1	SSP1CLK	UART2_TXD
PF2	-	PF2	ADDR2	SSP1FSS	CAN2_RX
PF3	-	PF3	ADDR3	SSP1RXD	CAN2_TX
PF4	-	PF4 MODE[0]	ADDR4	-	-
PF5	-	PF5 MODE[1]	ADDR5	-	-
PF6	-	PF6 MODE[2]	ADDR6	TMR1_CH1 [3]	-
PF7	-	PF7	ADDR7	TMR1_CH1N	TMR3_CH1
PF8	-	PF8	ADDR8	TMR1_CH2	TMR3_CH1N
PF9	-	PF9	ADDR9	TMR1_CH2N	TMR3_CH2
PF10	-	PF10	ADDR10	TMR1_CH3	TMR3_CH2N
PF11	-	PF11	ADDR11	TMR1_CH3N	TMR3_ETR
PF12	-	PF12	ADDR12	TMR1_CH4	SSP2_FSS
PF13	-	PF13	ADDR13	TMR1_CH4N	SSP2_CLK
PF14	-	PF14	ADDR14	TMR1_ETR	SSP2_RXD
PF15	-	PF15	ADDR15	TMR1_BLK	SSP2_TXD

Примечания:

1) Выводы управляются системной шиной EXT_BUS.
2) Выводы управляются контроллером интерфейса CAN1.
3) Выводы управляются Таймером 1.
4) Выводы управляются контроллером интерфейса CAN2.
5) Выводы используются АЦП.
6) Выводы используются ЦАП.
7) Выводы используются Компаратором.
8) Выводы используются генератором LSE.
9) Выводы используются контроллером прерываний.
10) Выводы управляются контроллером интерфейса UART1.
11) Выводы управляются контроллером интерфейса I2C.
12) Выводы управляются Таймером 3.
13) Выводы управляются контроллером интерфейса SSP2.
14) Выводы управляются контроллером интерфейса UART2.
15) Выводы управляются Таймером 2.
16) Выводы управляются контроллером интерфейса SSP1.
17) Реализоваано только в 1986BE94x

Из которой видно, что у портов микроконтроллера есть аналоговая и цифровая функции. Аналоговая отвечает за АЦП, ЦАПы и компаратор. Нам это не требуется для реализации ТЗ, поэтому не обращаем на неё пока внимание.

Цифровая функция порта в свою очередь разделена на несколько возможных видов, причём три из них, основной, альтернативный и переопределённый, отвечают за взаимодействие внутренних периферийных компонентов с выводами МК.

Именно через эти значения к выводам МК подключаются такие модули как UART, SPI, I2C, внешняя системная шина или блок захвата/сравнения таймеров общего назначения. В нашем случае это тоже не требуется, поэтому в итоге мы работаем лишь с той колонкой таблицы, что отвечает за использование портов как «Порт IO».

Дальше нас интересует таблица с общим описанием портов ввода-вывода. Она весьма небольшая и на основании её мы уже сразу можем сделать заготовку в нашем **main.c**

Базовый Адрес	Название	Описание	
0x400A_8000	MDR_PORTA	Порт A	
0x400B_0000	MDR_PORTB	Порт B	
0x400B_8000	MDR_PORTC	Порт C	
0x400C_0000	MDR_PORTD	Порт D	
0x400C_8000	MDR_PORTE	Порт E	
0x400E_8000	MDR_PORTF	Порт F	
Смещение			
0x00	RXTX[15:0]	MDR_PORTx->RXTX	Данные порта
0x04	OE[15:0]	MDR_PORTx->OE	Направление порта
0x08	FUNC[31:0]	MDR_PORTx->FUNC	Режим работы порта
0x0C	ANALOG[15:0]	MDR_PORTx->ANALOG	Аналоговый режим работы порта
0x10	PULL[31:0]	MDR_PORTx->PULL	Подтяжка порта
0x14	PD[31:0]	MDR_PORTx->PD	Режим работы выходного драйвера
0x18	PWR[31:0]	MDR_PORTx->PWR	Режим мощности передатчика
0x1C	GFEN[15:0]	MDR_PORTx->GFEN	Режим работы входного фильтра

Следует тут же заметить, что в самой спецификации на каждое описание группы регистров, соответствующих настройке и управлению каким-либо аппаратным компонентом есть подобная «общая» таблица, из которой видна их структура, а также общее количество.

Светодиод в нашем случае подключён к выводам PB0, поэтому настройка порта для него будет выглядеть так:

```
  // Настраиваем  порт PB
MDR_PORTB    -> OE       = (1 << 0);          // направление передачи данных = Выход
MDR_PORTB    -> ANALOG   = (1 << 0);          // режим работы контроллера = Цифровой
MDR_PORTB    -> FUNC     = (0 << 0*2);        // режим работы вывода порта = Порт
MDR_PORTB    -> PULL     = (1 << 0) << 16;    // разрешение подтяжки к VCC
MDR_PORTB    -> PD       = (0 << 0);          // режим работы выхода = управляемый драйвер
MDR_PORTB    -> PWR      = (1 << 0*2);        // скорость фронта вывода = медленный
```

Теперь рассмотрим значения, которые необходимо выставить для каждого из смещений регистра.

Запись вида (1<<0) интерпретируется как «единицу (в десятичной системе) сдвинуть влево на ноль бит». Благодаря такой маске программный код становится более читаем, потому что из этой записи виден как номер порта, так и его состояние. Таким вот незатейливым способом мы, например, присвоили смещению «OE» регистра «MDR_PORTB» логическую единичку.

MDR_PORTx->OE

Таблица 124 – Регистр OE

Номер	31...16	15...0
Доступ	U	R/W
Сброс	0	0
	-	**PORT OE[15:0]**

Таблица 125 – Описание бит регистра OE

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31...16	-	Зарезервировано
15…0	PORT OE[15:0]	Режим работы контроллера. Направление передачи данных на выводах порта: 1 – выход; 0 – вход

Регистр «FUNC» отвечает за режим работы порта. Ранее мы уже пришли к выводу, что нас интересует цифровая функция, в режиме «Порт IO», поэтому для нулевого бита (потому что мы PB0 настраиваем) необходимо выставить нулевое значение. Можно записать как (0<<0), но мы поступим проще, более универсально, записав следующим образом:

```
MDR_PORTB->FUNC = (0 << 0*2);
```

Выражение (0 << 0*2) говорит нам о том, что биты, настраиваются попарно. Это можно увидеть из спецификации на МК или в таблице, приведённой ниже. Если бы мы настраивали к примеру, вывод PB1, а не PB0 и не в «Порт IO», а в «Альтернативный режим» работы, то запись стала бы выглядеть следующим образом:

```
MDR_PORTB->FUNC = (2 << 1*2);
```

Из неё видно, что после операции «1*2» наша запись преобразуется в следующий вид:

```
MDR_PORTB->FUNC = (2 << 2);
```

Что можно описать как – *Два (в десятичной системе) сдвинуть влево на два бита*

MDR_PORTx->FUNC

Таблица 126 – Регистр FUNC

Номер	31	30	…	3	2	1	0
Доступ	R/W	R/W	…	R/W	R/W	R/W	R/W
Сброс	0	0	…	0	0	0	0
	MODE15[1:0]		**…**	**MODE1[1:0]**		**MODE0[1:0]**	

Таблица 127 – Описание бит регистра FUNC

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31...2	MODEx	Аналогично MODE0 для остальных бит порта
1…0	MODE0[1:0]	Режим работы вывода порта: 00 – порт; 01 – основная функция; 10 – альтернативная функция 11 – переопределенная функция

Регистр «ANALOG» отвечает собственно за функцию порта - «Аналоговую» или «Цифровую». Выше мы уже определили, что наш PB0 является цифровым портом, поэтому исходя из описания регистров в спецификации

```
MDR_PORTB->ANALOG   = (1 << 0);
```

MDR_PORTx->ANALOG

Таблица 128 – Регистр ANALOG

Номер	31…16	15...0
Доступ	U	R/W
Сброс	0	0
	-	**ANALOG EN[15:0]**

Таблица 129 – Описание бит регистра ANALOG

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…16	-	Зарезервировано
15…0	ANALOG EN[15:0]	Режим работы контроллера: 0 – аналоговый; 1 – цифровой

Регистр «PULL» отвечает за подключение к выводам подтягивающих резисторов. Если в электрической схеме они предусмотрены, можно их не включать, тем более что их сопротивление весьма велико (порядка 50-ти кОм), поэтому существенного влияния они не оказывают и скорее являются страховочным вариантом. В нашем случае подключим подтягивающий резистор вывода PB0 к питанию, а поскольку за это отвечают старшие 16 разрядов регистра «PULL», то запись будет выглядеть так:

```
MDR_PORTB->PULL   = (1<<0) << 16;
```

Этой записью мы сдвинули все, что находится в скобках влево, на 16 бит. А поскольку в скобках происходит ещё сдвиг единицы (в десятичной системе) на ноль бит, то получается, что 17-ый бит регистра «PULL» мы выставили в высокое логическое состояние.

При записи в шестнадцатеричной форме, это выглядело бы так:

```
MDR_PORTB->PULL   = 0x1000;
```

Регистр «PD» отвечает за режим работы выхода/входа. В нашем случае вывод PB0 является выходным портом, и светодиод, подключённый к нему, питается от него, без дополнительных подтягивающих резисторов на схеме, поэтому необходимо выставить его в положение «управляемый драйвер». Тогда, согласно техническим характеристикам на МК, его вывод сможет обеспечить управляемое питание нагрузкой с током минимум от 6ти и максимум до 10ти миллиампер. Наш светодиод работает при токе, обеспеченном ограничительным резистором, порядка 3ёх миллиампер, поэтому в данном случае всё согласуется, а запись этого выглядит следующим образом:

```
MDR_PORTB->PD   = (0 << 0);
```

Регистр «PWR» отвечает за скорость спада/нарастания фронта сигнала. Нам светодиод с мегагерцовой частотой формировать не надо, поэтому устанавливаем в положение «медленный фронт» наш нулевой бит порта «B». Опять же с точки зрения схемотехники – чем «медленнее»

фронт, тем меньше вероятность того, что наш вывод PB0 создаст помехи какому-нибудь аналоговому устройству.

Далее, аналогично порту PB0 настраиваем вывод PC2. В результате получаем запись следующего вида:

```
// Настраиваем  порт PC
MDR_PORTC    -> OE       = (0 << 2);          // направление передачи данных = Вход
MDR_PORTC    -> ANALOG   = (1 << 2);          // режим работы контроллера = Цифровой
MDR_PORTC    -> FUNC     = (0 << 2*2);        // режим работы вывода порта = Порт
MDR_PORTC    -> PULL     = (1 << (2 << 16));  // разрешение подтяжки к VCC
MDR_PORTC    -> PD       = (1 << (2 << 16));  // режим работы входа = триггер Шмидта вкл.
MDR_PORTC    -> PWR      = (1 << 2*2);        // скорость фронта вывода = медленный
MDR_PORTC    -> GFEN     = (1 << 2);          // входной фильтр включен
```

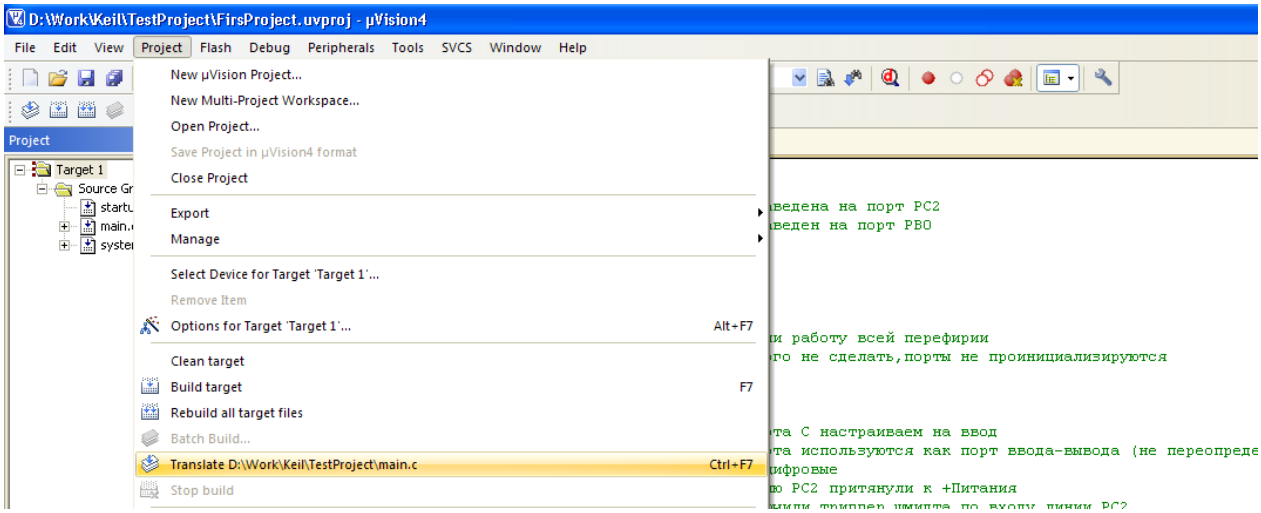
Для того, чтобы наша программа приняла более читаемый вид, запись (1<<2) заменим на «SW1», а (1<<0) на VD5 через макрос #define. Ниже приведён получившийся код, из которого видна следующая последовательность действий:

- Включаем тактирование периферии
- Настраиваем порты ввода-вывода
- Начинаем выполнять основной код программы

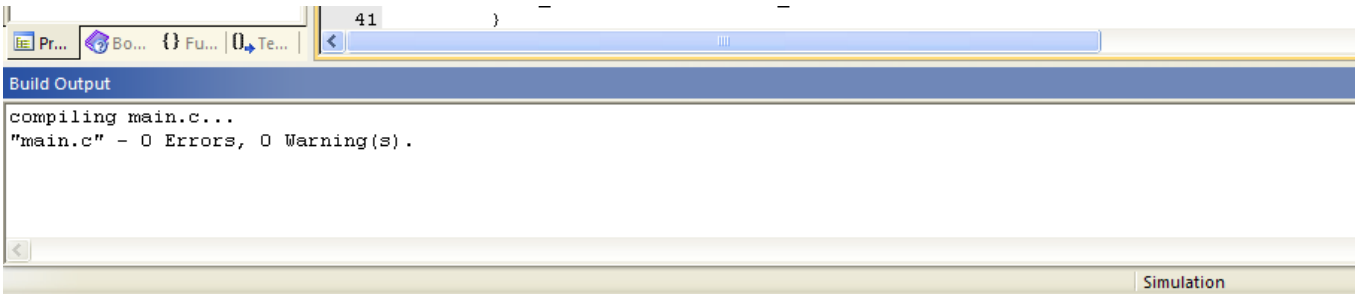
```
//================================================================================
#include <MDR32F9x.h>

#define SW1    (1<<2)                        // Кнопка SW1 подключена к порту PC2
#define VD5   (1<<0)                         // Светодиод VD5 подключена к порт PB0


int main (void)
{
  // Настраиваем  тактирование периферии
  MDR_RST_CLK->PER_CLOCK |= (1 << 22);       // Разрешили тактирование порта B
  MDR_RST_CLK->PER_CLOCK |= (1 << 23);       // Разрешили тактирование порта C
  // Настраиваем  порт PB
  MDR_PORTB    -> OE       = (1 << 0);       // направление передачи данных = Выход
  MDR_PORTB    -> ANALOG   = (1 << 0);       // режим работы контроллера = Цифровой
  MDR_PORTB    -> FUNC     = (0 << 0*2);     // режим работы вывода порта = Порт
  MDR_PORTB    -> PULL     = (1 << 0) << 16; // разрешение подтяжки к VCC
  MDR_PORTB    -> PD       = (0 << 0);       // режим работы выхода = управляемый драйвер
  MDR_PORTB    -> PWR      = (1 << 0*2);     // скорость фронта вывода = медленный
  // Настраиваем  порт PC
  MDR_PORTC    -> OE       = (0 << 2);       // направление передачи данных = Вход
  MDR_PORTC    -> ANALOG   = (1 << 2);       // режим работы контроллера = Цифровой
  MDR_PORTC    -> FUNC     = (0 << 2*2);     // режим работы вывода порта = Порт
  MDR_PORTC    -> PULL     = (1 << 2) << 16; // разрешение подтяжки к VCC
  MDR_PORTC    -> PD       = (1 << 2) << 16; // режим работы входа = триггер Шмитта вкл.
  MDR_PORTC    -> PWR      = (1 << 2*2);     // скорость фронта вывода = медленный
  MDR_PORTC    -> GFEN     = (1 << 2);       // входной фильтр включен

while(1)                                     // Основной  цикл  работы  программы
  {
   if (MDR_PORTC->RXTX & SW1)                // Если  бит  установлен (кнопка  отпущена)
    {
      MDR_PORTB->RXTX &= ~ VD5;              // Установили  порт PB0 в 0 (выкл. светодиод)
    }
else
    {
       MDR_PORTB->RXTX |= VD5;               // Установили  порт PB0 в 1 (вкл. светодиод)
    }
  }
}
//================================================================================
```

Теперь, по нажатию кнопки «**SW1**» на плате **LDM-K1986BE92QI**, подключённой к порту **PC2**, должен зажигаться светодиод **VD5**, подключённый к порту **PB0**.

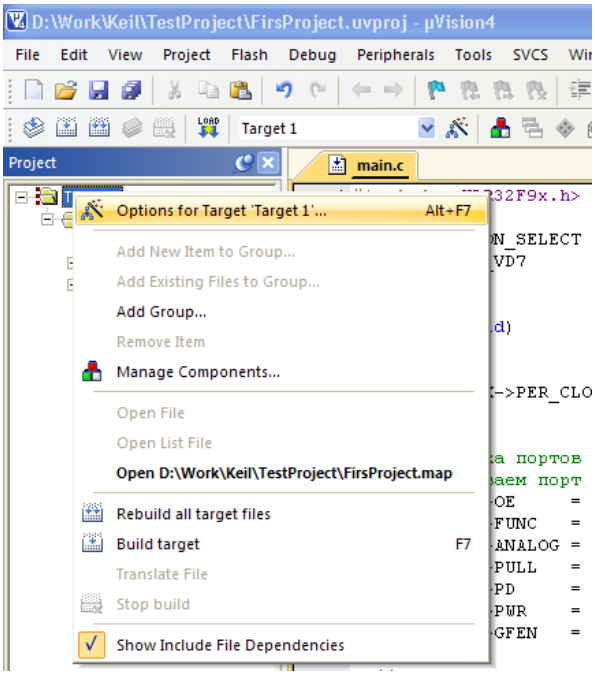
Для того, чтобы проверить, не возникло-ли ошибок, в меню «**Project**» выбираем «**Translate**» или жмём на соответствующую иконку на интерфейсе программы.



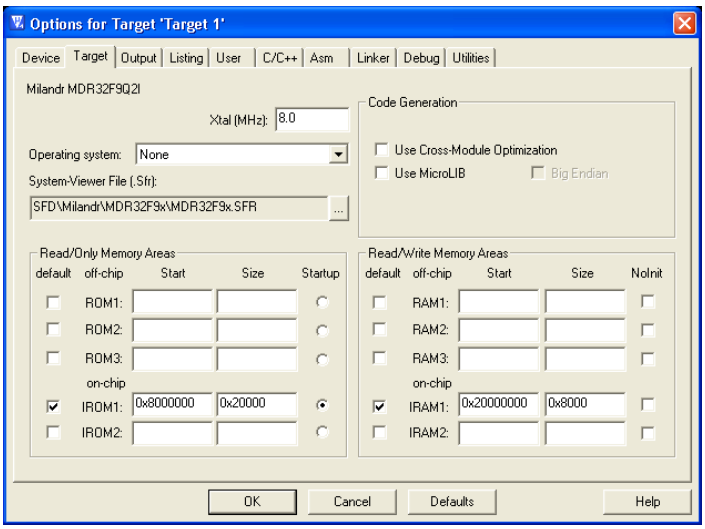
Ошибки или предупреждения, будут выведены в нижнем окне программы «**Build Output**»



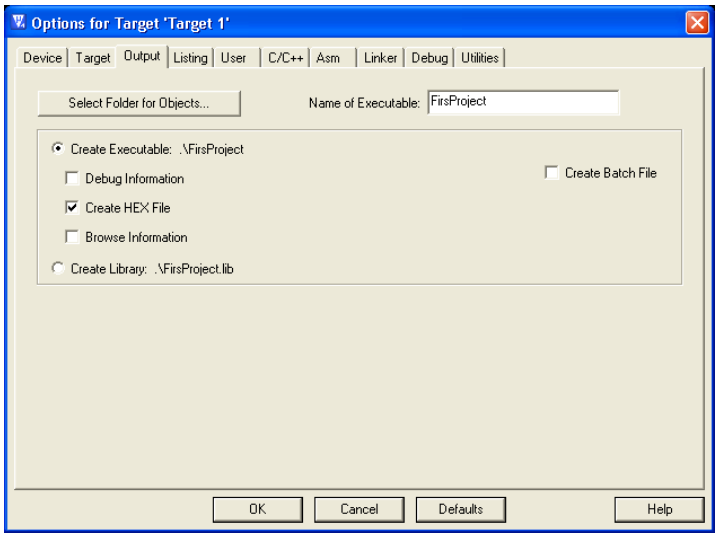
Далее необходимо указать среде разработки необходимость формирования hex-файла. Для этого жмём правой кнопкой мыши на папку «**Target 1**» в окне программы «**Project**» и в выпадающем меню выбираем «**Option for Target 'Target 1'**»



И после появления окошка «**Option for Target 'Target 1'**»

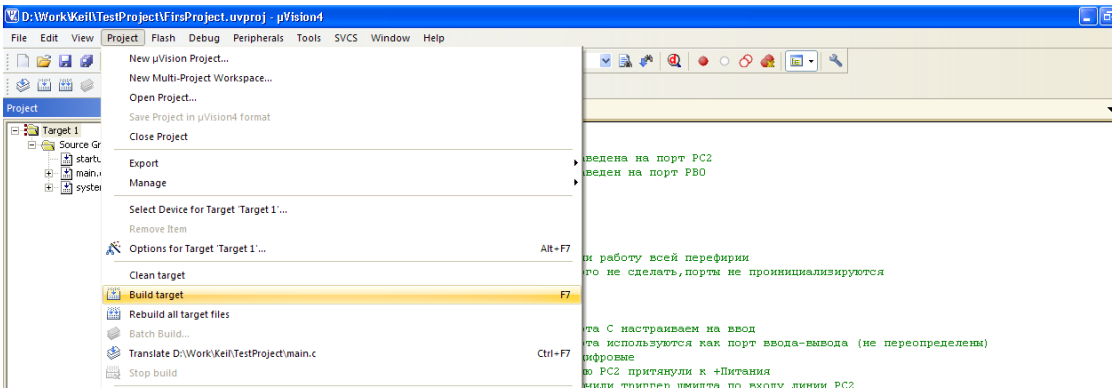


Переходим на вкладку «**Output**», где ставим галочку «**Create HEX File**»



После чего жмём «**OK**»

Теперь, необходимо полностью откомпилировать проект. Для этого в меню «**Project**» выбираем «**Build Target**»



Можно увидеть, что в папке проекта, в случае отсутствия ошибок или предупреждений в окне «**Build Output**», в папке проекта появился файл с расширением **.hex**.

Загрузка ПО в микроконтроллер,
через встроенный UART-загрузчик.

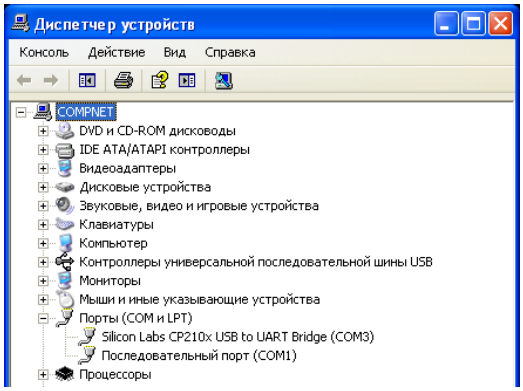
Осталось загрузить наш проект в микроконтроллер. Для этого необходимо загрузить программу прошивки микроконтроллера через внутренний UART-загрузчик, предоставленную пользователем «vasili» на forum.milandr.ru. в разделе:

Интегральные микросхемы ЗАО "ПКК Миландр" → 32-разрядные микроконтроллеры (1986BE9x, 1986BE1x, 1986BE2x) → 32-разрядные микроконтроллеры серии 1986BE9x (ядро ARM Cortex-M3) → AppNotes или примеры кода

Перед тем, как запустить программу, необходимо установить на компьютер драйвер для микросхемы CP2102. Скачать драйвера можно непосредственно с сайта производителя, компании «Silicon Labs»

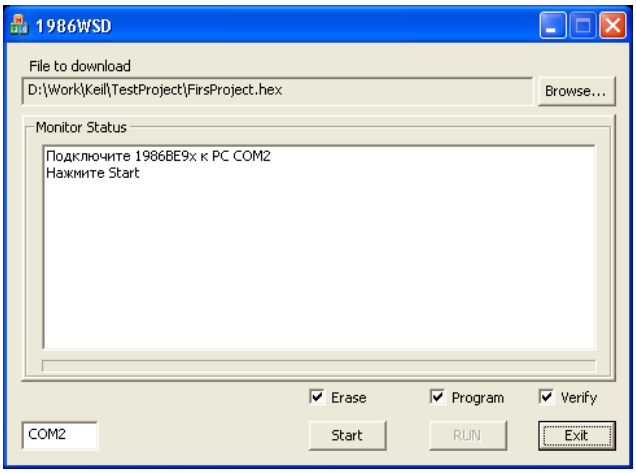
http://www.silabs.com

После установки драйвера, в системе, при подключении платы кабелем к разъёму USB-A, в диспетчере устройств появится COM-порт. В данном примере например это COM3.

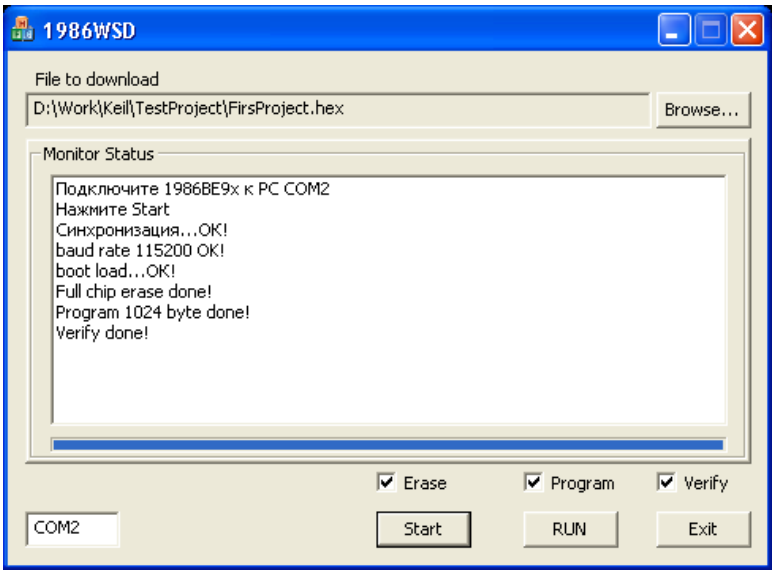


Необходимо учитывать, что программа UART-загрузчик корректно работает с портами 1-9. Далее, перед тем, как включить питание платы, необходимо выставить режим загрузки микроконтроллера MODE. Для этого переводим переключатели SW10 и SW6 в положение «1», а переключатель SW9 в положение «0». Таким образом мы получаем режим MODE «101» согласно которому при включении питания мы переходим в режим загрузки по UART-у, расположенному на выводах PD0 и PD1. Подробнее о режимах загрузки микроконтроллера написано в спецификации на микросхему.

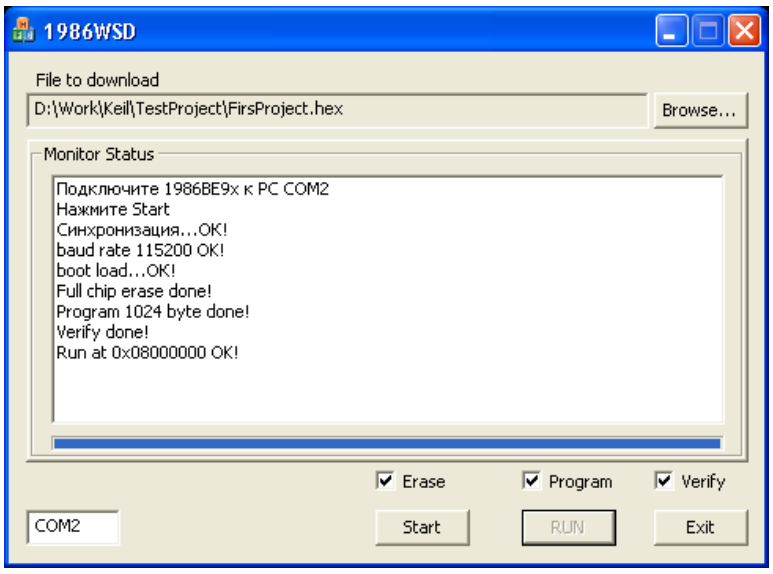
Включаем питание и запускаем программу UART-загрузчик «**1986WSD.exe**»



Жмём «**Start**» после чего в окне «**Monitor Status**» увидим соответствующую надпись об успешном выполнении загрузки программы во внутреннюю flash-память микроконтроллера.



Можно непосредственно произвести запуск программы, нажатием кнопки «**RUN**» после окончании операции загрузки.

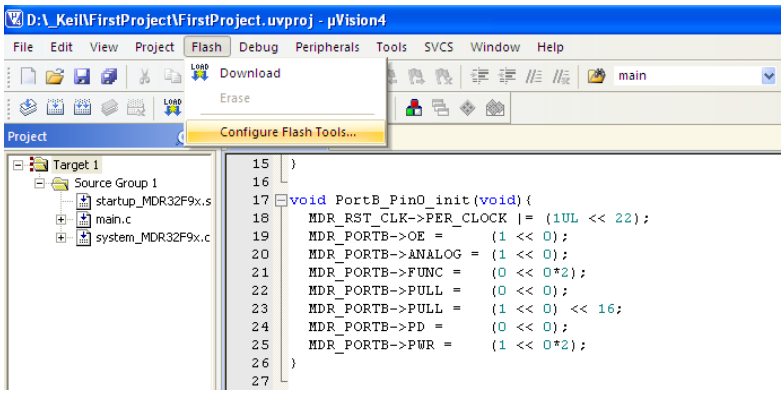


Об успешном выполнении программа также сообщит.

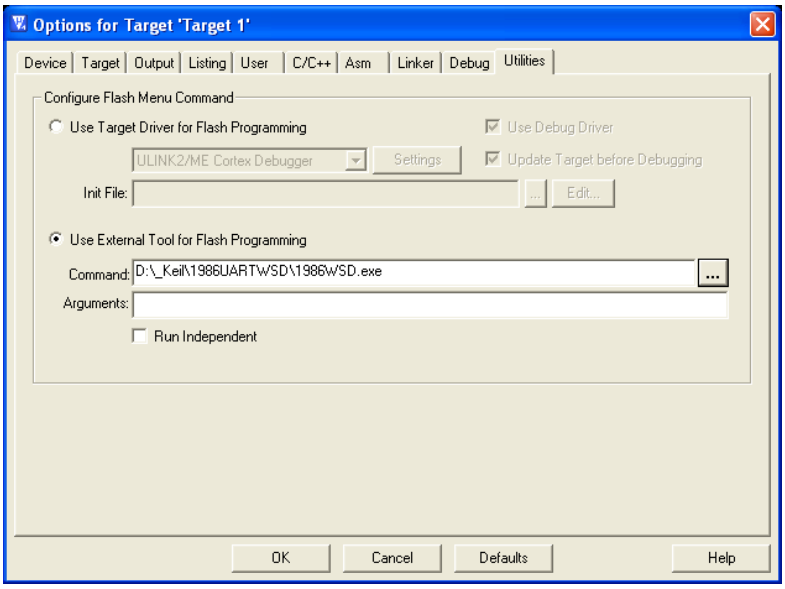
В общем виде последовательность действий для загрузки, созданного нами .hex-файла выглядит так:

- ✓ Устанавливаем «режим загрузки МК» MODE логическими уровнями на выводах PF4, PF5, PF6 (в нашем случае это «101»).
- ✓ Включаем питание
- ✓ Запускаем программу 1986UARTWSD.exe, указываем в ней тот COM-порт, с которым ассоциируется наше устройство, указываем путь к .hex-файлу, обычно находящемуся в папке проекта.
- ✓ Нажимаем «Start»
- ✓ После успешной загрузки выключаем питание
- ✓ Устанавливаем «режим загрузки МК» MODE логическими уровнями на выводах PF4, PF5, PF6 (в нашем случае это «000» или «001»).
- ✓ Включаем питание, работаем.

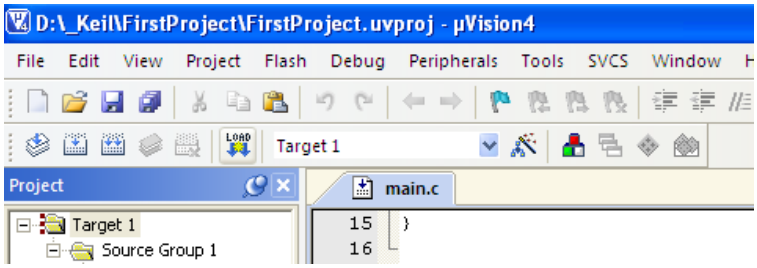
Кроме того можно значительно упростить вызов программы 1986UARTWSD. Для этого в среде разработки переходим в меню «**Flash**» где выбираем «**Configure Flash Tools…**»



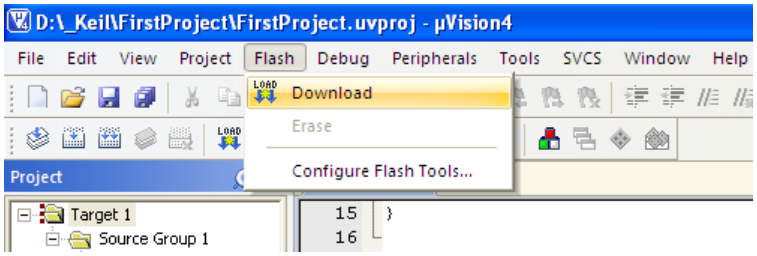
Далее, в появившемся окне переходим на вкладку «**Utilities**», выбираем «**Use External Tool for Flash Programming**» и в строке «**Command**» указываем путь к программе 1986UARTWSD.exe



Теперь саму программу можно вызывать из главного окна среды разработки, нажатием на кнопку «**Load**»



Или через меню «**Flash**»



Пример 2.
Использование в работе утилиты
«Milandr PLL» версии 2.2

Некоторое время назад на форуме компании появилась утилита для расчёта частот тактирования МК К1986ВЕ92QI. Написана она была в инициативном порядке пользователем **AntonAS**, за что ему большое человеческое спасибо.

Попробуем показать на нашем простейшем примере как её использовать для практических целей. Согласно нашему ТЗ из первого примера мы должны:

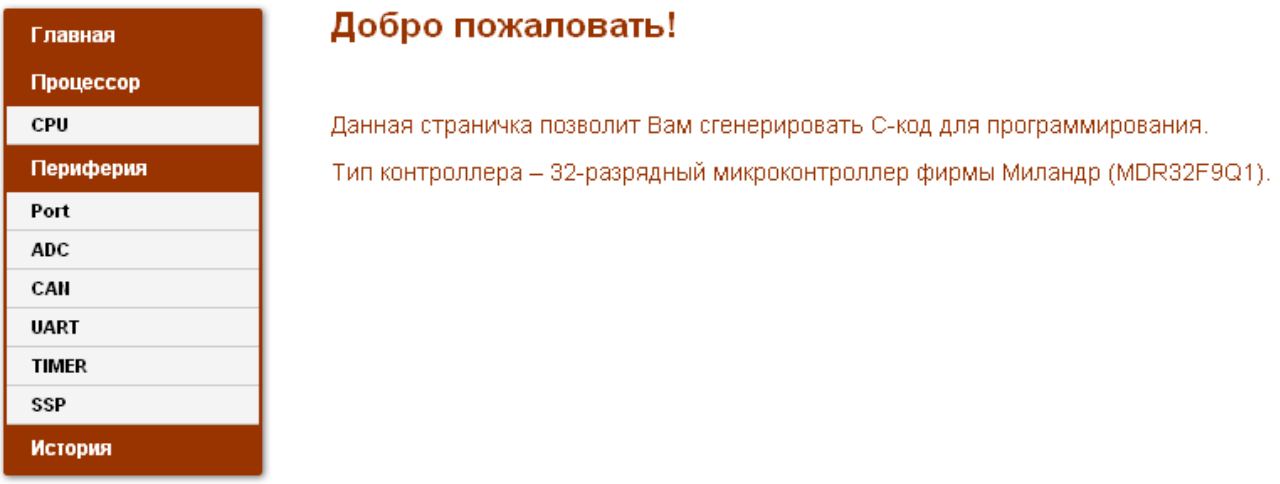
Разработать программное обеспечение для отображения состояния кнопки SW1 светодиодом VD5 на базе отладочного комплекта LDM-K1986BE92QI.

И там же, по ходу реализации, мы пришли к выводу, что нам не требуется максимальная тактовая частота 80МГц. Теперь попробуем представить, что такая потребность есть и нам просто жуть как необходимо работать на максимальной тактовой. Так же, при помощи данной утилиты покажем настройку портов ввода-вывода.

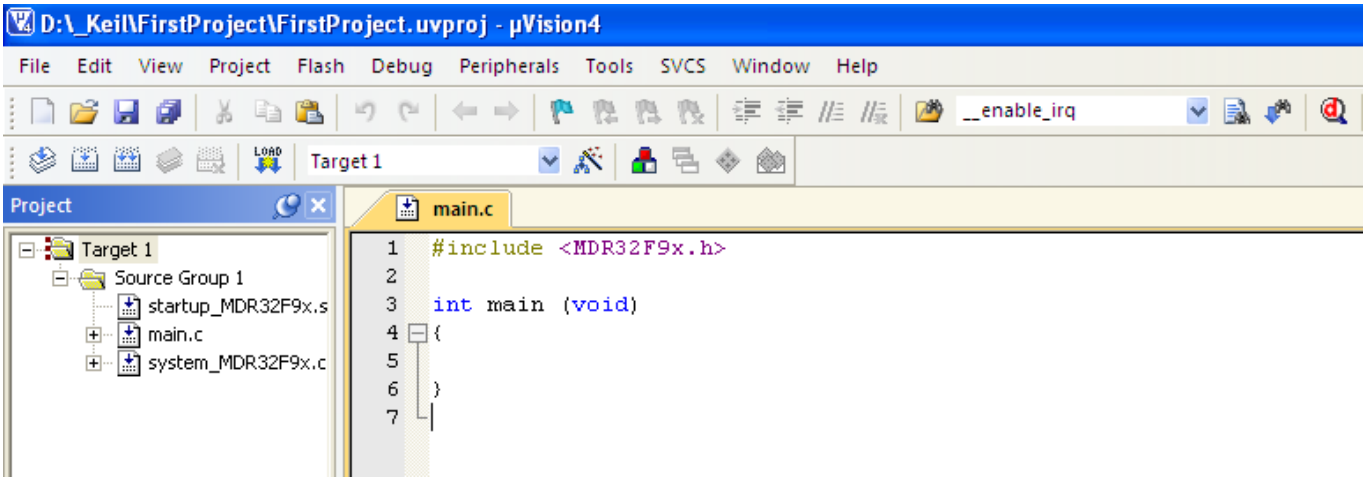
Таблица 3 – Минимальный набор аппаратного и программного обеспечения для примера 2

Инструментальная база	Наименование	Вэб-адрес источника
Отладочный комплект	LDM-K1986BE92QI	www.ldm-systems.ru
Драйвер преобразователя USB/RS-232	Silicon Laboratories CP210x VCP Drivers for Windows XP/2003 Server/Vista/7	www.silabs.com
Среда разработки ПО для МК	Keil uVision 4.72a	www.keil.com
Утилита прошивки МК через UART-загрузчик	1986UARTWSD	forum.milandr.ru
Утилита расчёта частот тактирования для МК 1986ВЕ92	Milandr_PLLv2.2	forum.milandr.ru

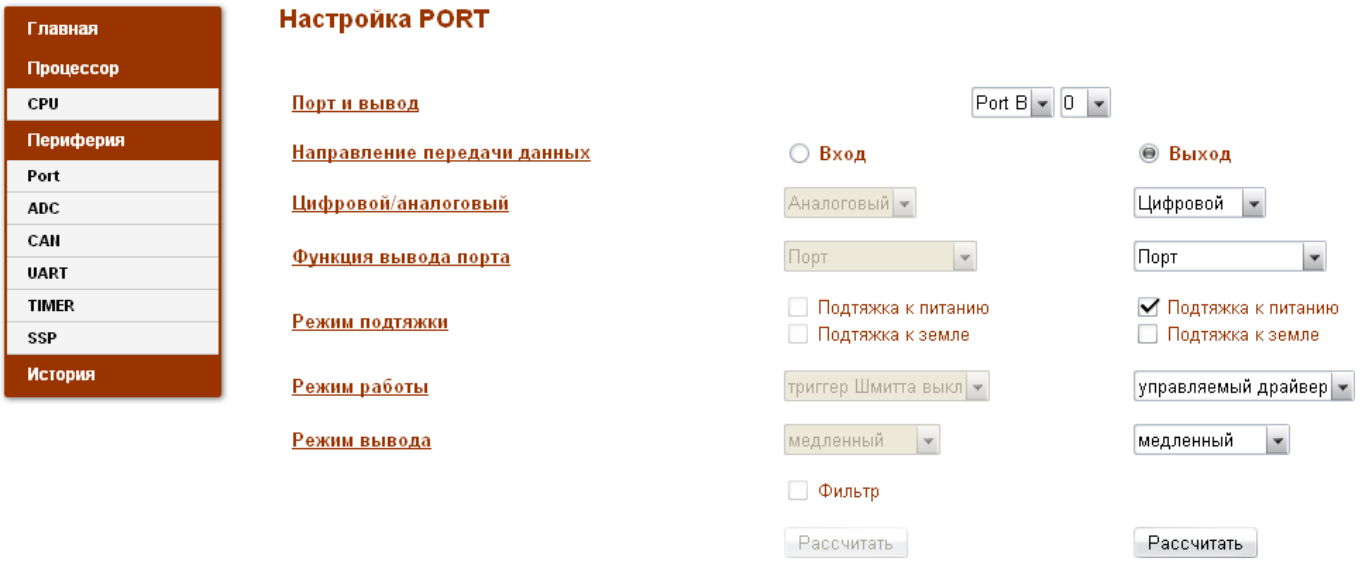
После загрузки утилита сразу готова к работе. Для этого на компьютере должен быть установлен браузер, хотя бы Internet Explorer начиная с 6-ой версии. Для запуска необходимо кликнуть два раза на файл **index.HTML** находящийся внутри папки с программой, в результате в браузере отобразится следующее окно:



Далее, у нас есть пустой проект, созданный в среде Keil uVision. В нём мы добавили код, который в общем-то ничего не делает:



Теперь, в утилите «**Milandr_PLLv2.2**» выбираем меню «**Port**», и устанавливаем в появившемся окне необходимые нам параметры, вначале для порта «B»:



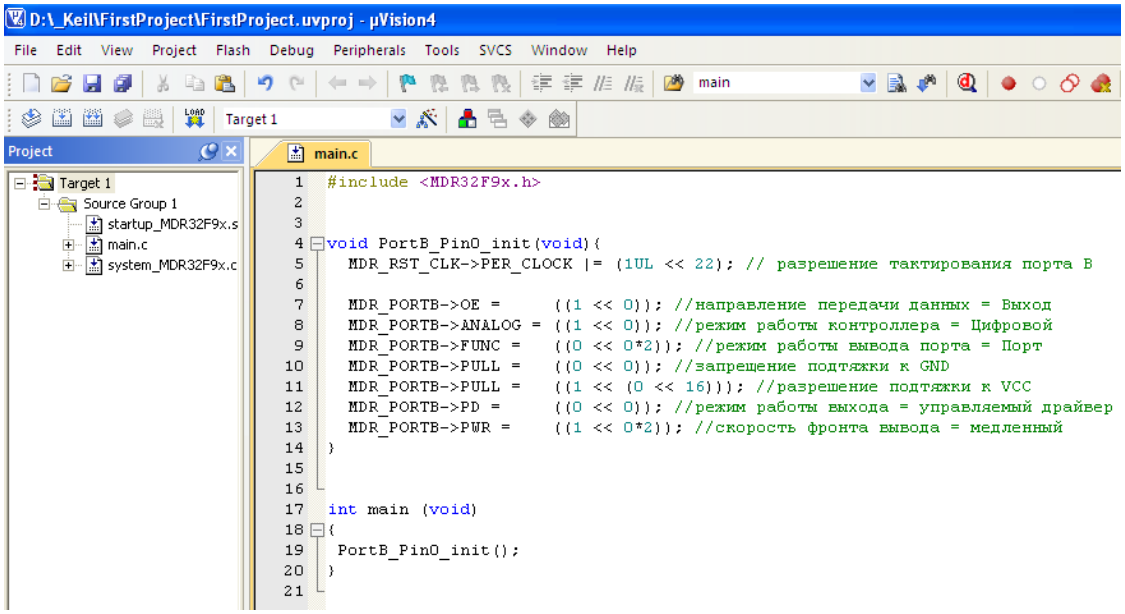
Код для вставки:

```
void PortB_Pin0_init(void){
  MDR_RST_CLK->PER_CLOCK |= (1UL << 22); // разрешение тактирования порта B

  MDR_PORTB->OE =     ((1 << 0)); //направление передачи данных = Выход
  MDR_PORTB->ANALOG = ((1 << 0)); //режим работы контроллера = Цифровой
  MDR_PORTB->FUNC =   ((0 << 0*2)); //режим работы вывода порта = Порт
  MDR_PORTB->PULL =   ((0 << 0)); //запрещение подтяжки к GND
  MDR_PORTB->PULL =   ((1 << (0 << 16))); //разрешение подтяжки к VCC
  MDR_PORTB->PD =     ((0 << 0)); //режим работы выхода = управляемый драйвер
  MDR_PORTB->PWR =    ((1 << 0*2)); //скорость фронта вывода = медленный
}
```

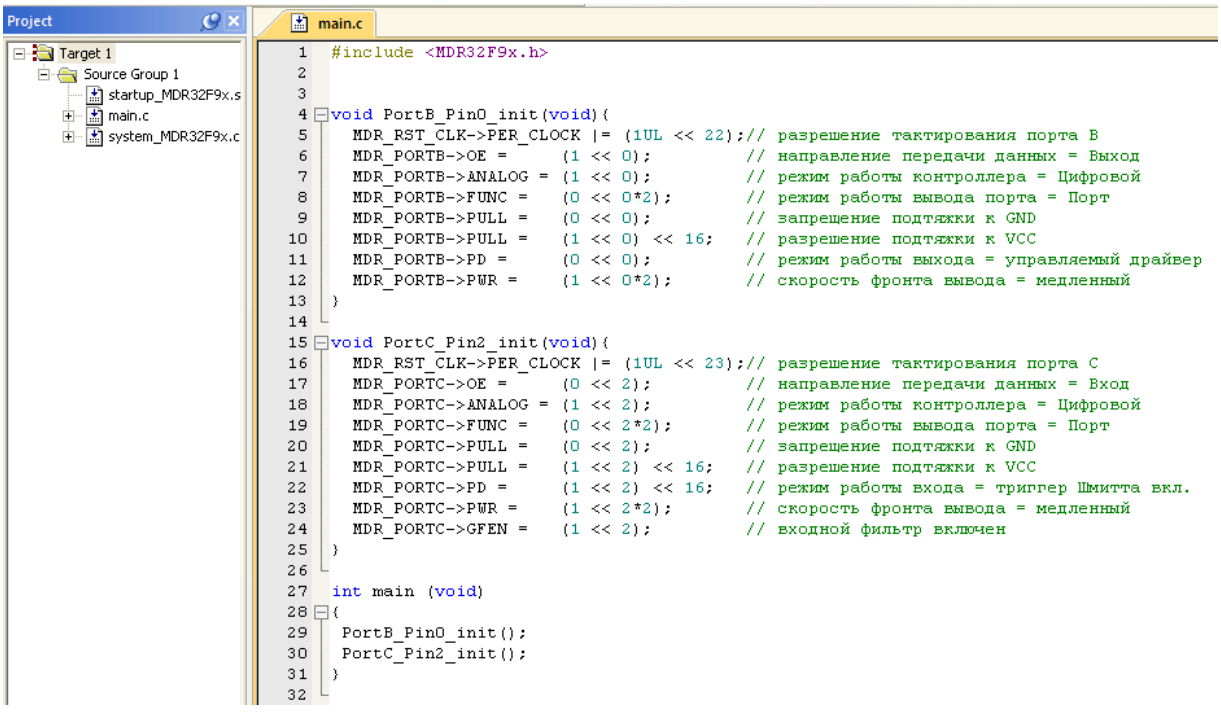
В результате, после нажатия на кнопку «**Рассчитать**», появился код, создающий нам отдельную функцию «**PortB_Pin0_init**» настройки порта «**B**» который вставляем к себе, выше функции «**main**». В самом «**main**» пишем:

```
PortB_Pin0_init();
```

И получаем следующую конструкцию:

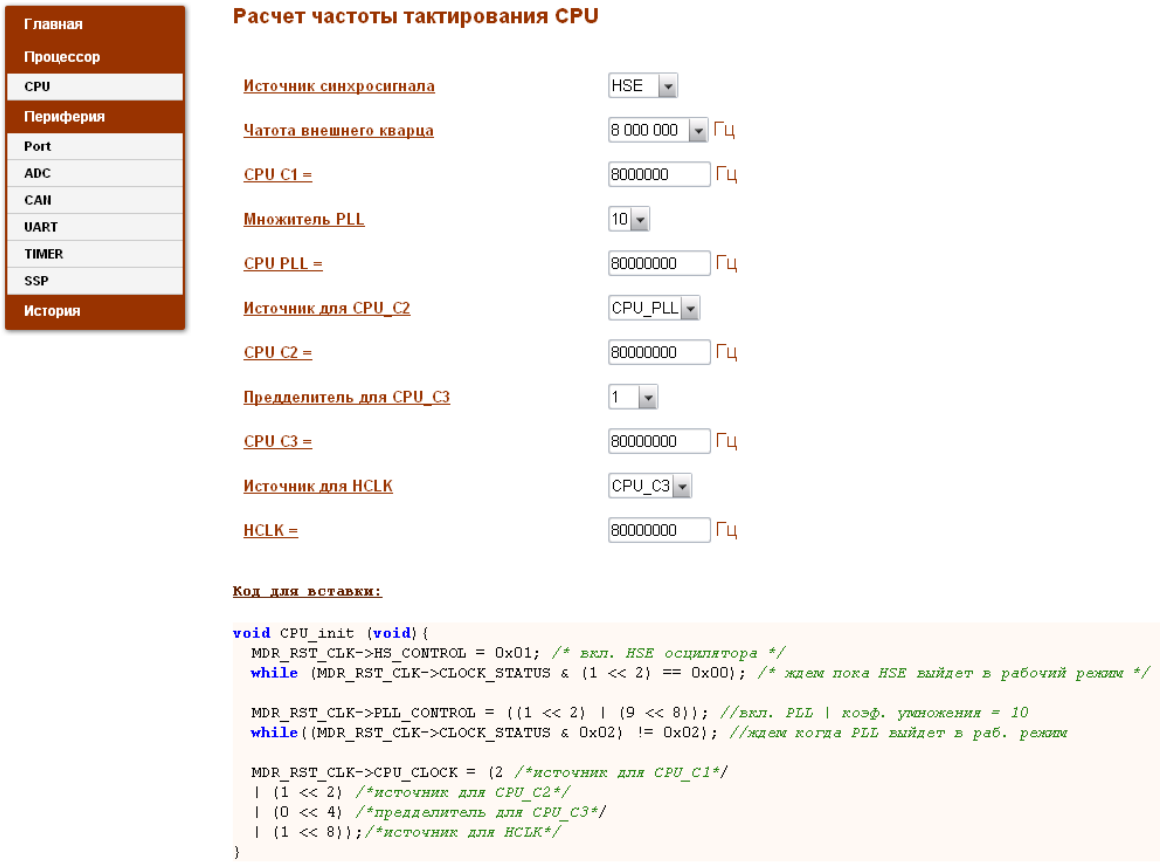


Тоже самое проделываем для настройки порта «C». В итоге, предварительно подчистив лишние скобки, сделанные в утилите для универсальности, должна получиться следующая конструкция:

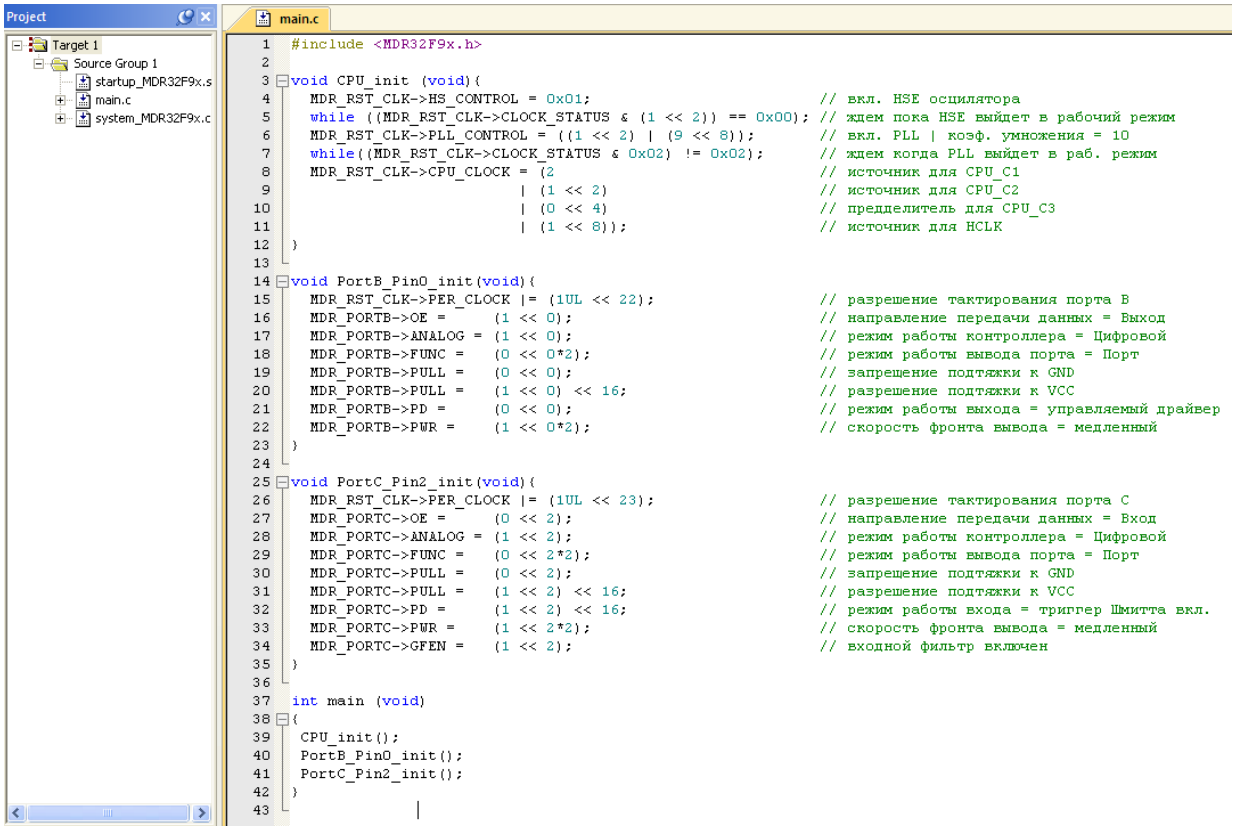


Если скомпилировать данный код, то ошибок или предупреждений не появится. Теперь перейдём на вкладку «**CPU**» где можно произвести настройку параметров, относящихся к разделу «**Сигналы тактовой частоты MDR_RST_CLK**» спецификации. В спецификации на МК данный раздел описан достаточно подробно, поэтому в нашем случае просто покажем, как при помощи утилиты «**Milandr_PLLv2.2**» выставить максимальную тактовую частоту, равную 80-ти МГц.

Итак, установив параметры формирования тактовой частоты на вкладке «CPU» мы получаем следующую конструкцию:



Мы получили функцию «**CPU_init**», которую также вставляем к себе в «**main**». Следует заметить, что МК должен сначала провести операции с тактовой, и лишь затем включить и настроить периферию. В итоге предварительно почистив лишние скобки и приведя форматирование к более удобочитаемому виду получаем следующий результат:



Теперь добавляем основной цикл работы программы, в результате получив такую вот законченную и рабочую конструкцию.

```
//================================================================================
#include <MDR32F9x.h>

#define SW1   (1<<2)                                        // Кнопка SW1 подключена к порту PC2
#define VD5   (1<<0)                                        // Светодиод VD5 подключена к порт PB0

void CPU_init (void){
  MDR_RST_CLK->HS_CONTROL = 0x01;                           // вкл. HSE осцилятора
  while ((MDR_RST_CLK->CLOCK_STATUS & (1 << 2)) == 0x00); // ждем пока HSE выйдет в рабочий режим
  MDR_RST_CLK->PLL_CONTROL = ((1 << 2) | (9 << 8));         // вкл. PLL | коэф. умножения = 10
  while((MDR_RST_CLK->CLOCK_STATUS & 0x02) != 0x02);        // ждем когда PLL выйдет в раб. режим
  MDR_RST_CLK->CPU_CLOCK = (2                               // источник для CPU_C1
                          | (1 << 2)                        // источник для CPU_C2
                          | (0 << 4)                        // предделитель для CPU_C3
                          | (1 << 8));                      // источник для HCLK
}

void PortB_Pin0_init(void){
  MDR_RST_CLK->PER_CLOCK |= (1UL << 22);                    // разрешение тактирования порта B
  MDR_PORTB->OE =      (1 << 0);                            // направление передачи данных = Выход
  MDR_PORTB->ANALOG = (1 << 0);                             // режим работы контроллера = Цифровой
  MDR_PORTB->FUNC =    (0 << 0*2);                          // режим работы вывода порта = Порт
  MDR_PORTB->PULL =    (0 << 0);                            // запрещение подтяжки к GND
  MDR_PORTB->PULL =    (1 << 0) << 16;                      // разрешение подтяжки к VCC
  MDR_PORTB->PD =      (0 << 0);                            // режим работы выхода = управляемый драйвер
  MDR_PORTB->PWR =     (1 << 0*2);                          // скорость фронта вывода = медленный
}

void PortC_Pin2_init(void){
  MDR_RST_CLK->PER_CLOCK |= (1UL << 23);                    // разрешение тактирования порта C
  MDR_PORTC->OE =      (0 << 2);                            // направление передачи данных = Вход
  MDR_PORTC->ANALOG = (1 << 2);                             // режим работы контроллера = Цифровой
  MDR_PORTC->FUNC =    (0 << 2*2);                          // режим работы вывода порта = Порт
  MDR_PORTC->PULL =    (0 << 2);                            // запрещение подтяжки к GND
  MDR_PORTC->PULL =    (1 << 2) << 16;                      // разрешение подтяжки к VCC
  MDR_PORTC->PD =      (1 << 2) << 16;                      // режим работы входа = триггер Шмитта вкл.
  MDR_PORTC->PWR =     (1 << 2*2);                          // скорость фронта вывода = медленный
  MDR_PORTC->GFEN =    (1 << 2);                            // входной фильтр включен
}

int main (void)
{
 CPU_init();
 PortB_Pin0_init();
 PortC_Pin2_init();
while(1)                                                    // Основной  цикл  работы  программы
  {
   if (MDR_PORTC->RXTX & SW1)                               // Если  бит  установлен (кнопка  отпущена)
    {
      MDR_PORTB->RXTX &= ~ VD5;                             // Установили  порт PB0 в 0 (выкл. светодиод)
    }
else
    {
        MDR_PORTB->RXTX |= VD5;                             // Установили  порт PB0 в 1 (вкл. светодиод)
    }
  }
}
//================================================================================
```

Далее компилируем, загружаем, получившийся .hex-файл в МК, посредством программы 1986UARTWSD и радуемся тому, что теперь светодиод отображает состояние кнопки, выполняя программу на максимальной частоте.

Вопросы для дополнительных примеров/разделов.

Дальше напишу несколько вопросов, на которые хотелось бы не просто получить ответ из нескольких строчек, а полноценную, развёрнутую инструкцию из принципа не «а как это вообще делать», а «как оптимальнее сделать».

1. Функции. Общий подход к созданию функций, а также разъяснение того, в каких случаях это оправдано, а в каких-то может быть даже вредно. Примеры, простые, средние.
2. Ассемблерные вставки. Когда возникает необходимость применения? Примеры, простые, средние.
3. Работа с Миландровской библиотекой. В каких случаях оптимальнее работать без неё/с ней? Пример использования, простой и средний.
4. Работа по прерываниям. Установка приоритетов в NVIC, простейшие примеры, где идёт работа по прерываниям с разным приоритетам.
5. Работа с графическим индикатором MT-12864A. Как лучше, с использованием системной шины или отдельно от неё. Самый-самый простейший пример работы с индикатором.
6. Общие подходы к распределению задач по времени, приоритету прерываний. Примеры простого уровня сложности.
7. USB. Разбор примера преобразователя USB/RS-232 или какой-нибудь самый простейший пример разобрать в подробностях. Желательно и Device и Host.
8. Работа с несколькими АЦП одновременно, синхронная обработка. Можно общими словами, но с рисунками в виде блок-схем или просто с простыми примерами.