1) Добавьте в ваше рабочее пространство новый проект для написания программы на языке ассемблер. Для этого на шаге выбора инструментов в окне «Toolchain» необходимо выбрать «*ST Assembler Linker*»

Шаблон проекта состоит из файлов: «*main.asm*», «*mapping.inc*» и «*mapping.asm*». Файл «*mapping.inc*» содержит константы деления ОЗУ на сегменты. Здесь первые две страницы ОЗУ выделены в сегменты *RAM*0 (256 байт) и *RAM*1 (4864 байта), остальные 1024 байта отданы под сегмент стека. В файле «*mapping.asm*» определяются сегменты: *ram*0, *ram*1, *stack*, *eeprom*, *rom*, *vectit*.

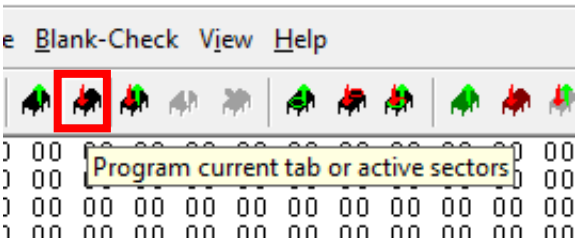
Ассемблерный файл в *STVD* начинается строкой '*stm*8/' и заканчивается строкой *'end'*. В файле «*main.asm*», в сегменте ‘*vectit’* содержится таблица прерываний. Первым в таблице расположен вектор прерывания *Reset*. Каждый вектор прерывания занимает в памяти микроконтроллера 4 байта. Первый байт должен содержать код операции *INT* (0x82) следующие 3 Байта содержат адрес ячейки памяти, в которой будет расположена функция обработчика прерывания. Директива *dc.l* позволяет записать в память 32 разрядные данные. Запись $82000000+*main* означает, ч в самом начале сегмента памяти *‘vectit'* в первую ячейку запишется 0x82, а в следующие три байта запишется адрес метки main. Так как метка *main.l* расположена в самом начале сегмента *‘rom’* то он принимает значение первой ячейки этого пространства памяти и равна 0x008080, что соответствует началу *Flash program memory*.

Все остальные вектора прерывания, по умолчанию, содержат адрес ячейки, на которую указывает метка *NonHandledInterrupt*. Соответственно у всех остальных прерываний на данный момент одна функция обработчик прерывания. Там содержится только одна инструкция – инструкция возврата из прерывания *iret*.

В начале области памяти, на которую указывает метка *main*, осуществляется загрузка в регистр указателя стека объявленного значения *stack_end*, равного 0x17*FF*, очищаются сектора памяти *RAM0*, *RAM1* и стек.

Далее расположен бесконечный цикл, выполненный в виде метки *infinite_loop.l*, указывающей на ячейку памяти программ, в которой расположена инструкция *jra infinite_loop* (безусловный относительный переход на метку).

2) Перед загрузкой программы откройте *STVisual Programmer*, в окне Hardware необходимо выбрать *ST_LINK*, в окне *Programming mode* выберете SWIM. В окне *Device* нужно выбрать микроконтроллер *STM8S207RB*. Далее откроется окно с пустым бланком памяти микроконтроллера. Используя кнопку загрузки текущего шаблона загрузить его в память микроконтроллера. Это нужно для очистки памяти программ вашего микроконтроллера, чтобы при дальнейшей работе не видеть ничего лишнего. После успешной загрузки *STVisual Programmer* нужно закрыть и вернуться в *STVisual Developer*



Архитектура микроконтроллеров STM8.

В основе микроконтроллеров семейства STM8 (рис 2) лежит CISC-ядро собственной разработки STMicroelectronics, сделанное на основе более ранних 8-разрядных контроллеров ST7. Ядро спроектировано по гарвардской архитектуре, подразумевающей разделение данных и команд. ЦП имеет полную 8-битную архитектуру с 16-битными операциями над индексными регистрами (для вычисления адреса). Шесть внутренних регистров позволяют эффективно манипулировать 8-битными данными. Центральный процессор способен выполнять 80 основных инструкций. Он имеет 20 режимов адресации и может адресовать 6 внутренних регистров и 16 Мбайт памяти/периферийных регистров.

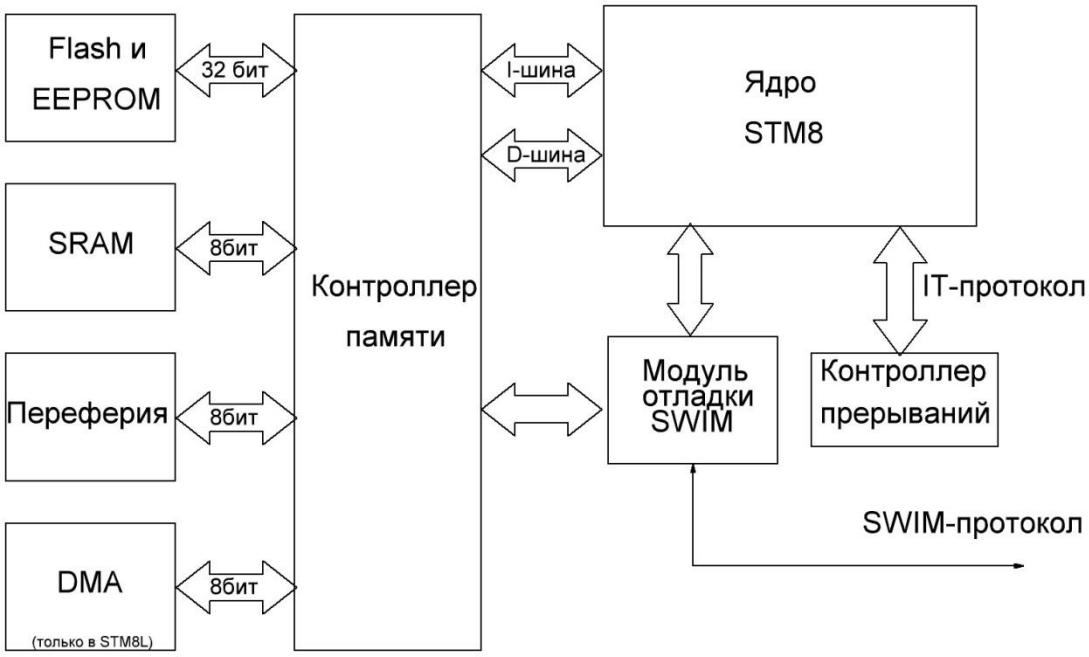


Рис.2.

Регистры ЦП.

Аккумулятор (A) - это 8-разрядный регистр общего назначения, используемый для хранения операндов и результатов арифметических и логических вычислений, а также операций с данными.

Индексные регистры (X и Y) – эти 16-битные регистры используются для вычисления адреса или в качестве области временного хранения для манипуляций с данными.

Программный счетчик (PC) – это 24-битный регистр (PCE+PCH+PCL), используемый для хранения адреса следующей инструкции, которую должен выполнить ЦП. Он автоматически обновляется после каждой обработанной инструкции. В результате ядро STM8 может получить доступ к 16-мегабайтной памяти.

Указатель стека (SP) – представляет собой 16-битный регистр. Он содержит адрес следующей свободной ячейки в стеке. В зависимости от модели микроконтроллера старшие значащие биты могут быть аппаратно предустановлены в то или иное значение.

Стек используется для сохранения контекста ЦП при вызовах подпрограмм или возникновении прерываний. Пользователь также может напрямую использовать его с помощью инструкций POP и PUSH.

После сброса микроконтроллера указатель стека устанавливается в свое максимально допустимое верхнее значение. Затем он уменьшается после помещения данных в стек и увеличивается после извлечения данных из стека. Когда указатель достигнет нижней допустимой границы, его значение снова будет установлено в максимально допустимое значение. Тогда ранее сохраненные данные будет перезаписываться и, следовательно, они будут потеряны.

Вызов подпрограммы занимает две или три ячейки в стеке. Когда происходит прерывание, регистры ЦП (CC, X, Y, A, PC) сохраняются в стек. Операция вызова прерывания занимает 9 циклов ЦП и использует 9 байт в ОЗУ.

Глобальный конфигурационный регистр (CFG_GCR) – привязан к адресу в оперативной памяти. Он управляет конфигурацией процессора и включает в себя AL-бит:

AL: уровень активации

Если бит AL равен 0 (основной), IRET вызовет извлечение контекста из стека, и основная программа продолжит работу после инструкции WFI.

Если бит AL равен 1 (активно только прерывание), IRET (инструкция возврата из прерывания) заставит ЦП вернуться в режим WFI/HALT (коротко пояснить) без извлечения содержимого регистров из стека.

Этот бит используется для управления режимами пониженного энергопотребления микроконтроллера. В приложении с очень низким энергопотреблением микроконтроллер проводит большую часть времени в режиме WFI/HALT и пробуждается (через прерывания) в определенные моменты для выполнения своих задач. Эти задачи могут быть настолько короткими, что могут выполняться непосредственно в обработчике прерывания, без возвращения в основную программу. В таком случае флаг AL устанавливается в 1 переводом микроконтроллера в режим пониженного энергосбережения (с помощью инструкций WFI/HALT). В результате время выполнения обработчика прерывания уменьшается за счет того, что содержимое регистров ЦПУ не сохраняется и не восстанавливается каждый раз при возникновении прерывания.

Регистр статуса (CC) – это 8-битный регистр, в который заносятся флаги результата только что выполненной инструкции ЦПУ. Эти флаги могут быть проанализированы независимо друг от друга специальными инструкциями, в результате чего могут быть приняты те или иные действия.

Данный регистр содержит следующие флаги:

- **V: флаг переполнения (overflow)** – если флаг V установлен, то это показывает, что при выполнении последней знаковой арифметической операции старший бит - MSB - был установлен в единицу. Для подробностей смотрите описание инструкций: **INC, INCW, DEC, DECW, NEG, NEGW, ADD, ADC, SUB, SUBW, SBC, CP, CPW**.
- **I1** и **I0: маска прерывания первого и нулевого уровня** – Этот флаг работает совместно с флагом **I0** и определяет уровень обрабатываемого прерывания согласно таблице, приведенной ниже. Эти флаги могут быть установлены программно следующими инструкциями: **RIM, SIM, HALT, WFI, IRET, TRAP и POP**. Или они могут быть установлены автоматически аппаратным способом при входе в обработчик прерывания.

Инструкции микроконтроллеров семейства STM8S.

Команды загрузки и передачи	LD, LDF, LDW, CLR, MOV, EXG, LDW, CLRW, EXGW,
Инструкции операций со стеком	PUSH, POP, PUSHW, POPW
Инструкции инкремента/декремента	INC, DEC, INCW, DECW
Инструкции сравнения	CP, TNZ, BCP, CPW, TNZW
Инструкции логических операций	AND, OR, XOR, CPL, CPLW
Инструкции битовых операций	BSET, BRES, BCPL, BCCM
Инструкции условного битового перехода	BTJT, BTJF
Инструкции арифметических операций	NEG, ADC, ADD, SUB, SBC, MUL, DIV, DIVW, NEGW, ADDW, SUBW
Инструкции сдвига (и поворота)	SLL, SRL, SRA, RLC, RRC, SWAP, SLLW, SRLW, SRAW, RLCW, RRCW, SWAP, RLWA, RRWA
Инструкции безусловного перехода или вызова	JRA, JRT, JRF, JP, JPF, CALL, CALLR, CALLF, RET, RETF, NOP
Инструкции условного ветвления/выполнения	JRxx, WFE
Инструкции управления прерываниями	TRAP, WFI, HALT, IRET
Инструкции модификации флагов состояния	SIM, RIM, SCF, RCF, CCF, RVF
Инструкция точка останова	BREAK

Описать инструкции, используемые в сгенерированном коде.

Инструкция чтения из памяти *LDW* выполняет загрузку данных размером 16 бит из src (*source*) в dst (*destination*). В качестве dst и src могут выступать 16-разрядные регистры (*X*,*Y* или *SP*) или любое 16-разрядное число или адрес ячейки памяти.

Инструкция отчистки *CLR* выполняет загрузку числа 0 в адрес ячейки памяти, соответствующий операнду инструкции. Адрес ячейки памяти может быть указан, как значение, содержащееся в аккумуляторе, регистрах *X*,*Y* или *SP*, или в виде адреса.

Инструкция инкремента слова *INCW* выполняет увеличение содержимого регистра индекса (X или Y) на 1.

Инструкция сравнения слова *CPW* используется для сравнения содержимого регистров индекса (*X* или *Y*) с 16-разрядным числом или

содержимым памяти. В случае, если разность равна нулю, то устанавливается флаг состояния *C*. Если же значение содержимого в регистре индекса меньше, чем число, с которым оно сравнивается, то устанавливаются флаги состояния *N* и *Z*. Если же разность содержимого регистра индекса и сравниваемого числа вызывает переполнение (результат вычитания не может быть записан в виде 16-разрядного числа), то устанавливается флаг состояния *V*.

Инструкция *JRA* это безусловный относительный переход. Значение программного счетчика обновляется знаковой суммой содержимого программного счетчика и адресом перехода, чтобы указать, где необходимо продолжить выполнение программы.

Инструкция *JRULE* это условный относительный переход, который соответствует условию <= (меньше либо равно). Значение программного счетчика обновляется знаковой суммой содержимого программного счетчика и адресом перехода, если выполняется условие сравнения, указанное выше. Если условие не выполняется, то данная инструкция не выполняется и программа продолжает выполняться последовательно.

3) После блока очистки стека написать программу, которая заполняет N ячеек памяти числами от a до b.

4) Выполнить компиляцию программы, сборку и ее загрузку в микроконтроллер.

5) Если нет вспомогательных окон просмотра содержимого регистров ядра микроконтроллера (*Core Registers*) и памяти (*Memory*), добавить их через меню *View*.

6) В окне просмотра содержимого памяти можно увидеть, что в адресах с 0x6000 по 0x67*FF* содержится 2 кБайта *boot ROM*.

Основной задачей загрузчика является загрузка программы во внутреннюю память через встроенные периферийные устройства (*UART, SPI* или *CAN*) без использования протокола *SWIM* и специального оборудования. Данные предоставляются любым устройством (хостом), способным отправлять информацию через один из вышеупомянутых последовательных

интерфейсов. Загрузчик позволяет загружать программное обеспечение в память устройства, включая ОЗУ, память программ и данных, используя стандартные последовательные интерфейсы. Это дополнительное решение для программирования через интерфейс отладки *SWIM*. Код загрузчика хранится во внутренней памяти загрузочного ПЗУ. После сброса код загрузчика проверяет, является ли программная память пустой или установлен ли определенный байт опции, позволяющий модифицировать код. Если эти условия не выполняются, загрузчик возобновляет работу и запускается пользовательское приложение. В случае успешной проверки выполняется загрузчик. При запуске процедуры загрузчика основными задачами являются:

• Опрос всех поддерживаемых последовательных интерфейсов для проверки используемого периферийного устройства.

• Загрузка программного кода, данных, байтов опций и/или векторных таблиц по адресу(ам), полученному от хоста.

Каждое устройство *STM8* включает в себя определенный код загрузчика, общий для всей группы устройств *STM8*.

7) Найти ячейку памяти с адресом 0*x*7*F*00, согласно документации, в этой ячейке хранится содержимое аккумулятора, после сброса должно быть равно 0. Ячейки с адресами 0*x*7*F*01 - 0*x*7*F*03 хранят значения программного счетчика, должно быть равно 0x006000. Следовательно, адрес ячейки в которой лежит следующая команда равен 0*x00*6000, а это первая ячейка в области *boot ROM*.

8) Убедиться, что значение указателя стека равно 0*x*17*FF*, оно должно храниться в ячейках памяти с адресами 0*x*7*F*08 и 0*x*7*F*09.

9) В ячейках памяти с адресами, лежащими в диапазоне от 0*x*8000 до 0*x*807*F*, находятся 32 вектора прерываний. Каждый из них занимает 4 Байта. Первый байт должен содержать код операции *INT* (0x82) следующие 3 Байта содержат адрес ячейки памяти, в которой будет начинаться обработчик прерывания. По умолчанию в проекте вектора описаны следующим образом:

segment 'vectit'

```
        dc.l {$82000000+main}                   ; reset
        dc.l {$82000000+NonHandledInterrupt}    ; trap
        dc.l {$82000000+NonHandledInterrupt}    ; irq0
….
        dc.l {$82000000+NonHandledInterrupt}    ; irq29
```

Директива *dc.l* позволяет записать в память константу размером 32 бита. Таким образом в сегменте памяти «*vectit*» записаны 128 Байт в формате {0*x*82000000 + адрес метки}. По умолчанию в проекте у вектора прерывания «*reset*» стоит метка main, она при компиляции примет значение 0*x*8080, так как это адрес начала сегмента «*rom*». Соответственно в векторе прерывания «*reset*» (в ячейках с адреса 0x8000 по 0x8003) будет следующее содержимое: 0*x*82 0*x*00 0*x*80 0*x*80 и при возникновении прерывания «*reset*» в программный счетчик запишется 0*x*008080.

Рассмотрим содержание ячеек памяти начиная с адреса 0*x*008080.

AE 17 FF 94 AE 00 00 7F 5C A3 00 FF 23 F9 AE 01 00 7F 5C A3 13 FF 23 F9 AE 14 00 7F 5C A3 17 FF 23 F9 20 FE 80

ldw X,#stack_end код операции ldw X, #word = 0xAE, значение stack_end = 0x17FF

ldw SP,X код операции = 0x94

ldw X,#ram0_start код операции ldw X, #word = 0xAE, значение ram0_start = 0x0000

clr (X) код операции = 0x7F

incw X код операции = 0x5C

cpw X,#ram0_end код операции cpw X,#word = 0xA3, значение ram0_end = 0x00FF

jrule clear_ram0 код операции jrule MEM= 0x23, в следующей ячейке памяти лежит сдвиг (на сколько нужно изменить программный счетчик). В момент выполнения текущей команды в программном счетчике содержится адрес следующей ячейки со следующей операцией (AE). Для того, чтобы попасть на метку «clear_ram0» (ячейка 7F) необходимо уменьшить значение программного счетчика на 7, в дополнительном коде -7=0xF9 (0xF9 +7 =0), следовательно, сдвиг, лежащий в следующей ячейке равен 0xF9.

ldw X,#ram1_start код операции ldw X, #word = 0xAE, значение ram1_start = 0x0100

clr (X) код операции = 0x7F

incw X код операции = 0x5C

cpw X,#ram1_end код операции cpw X,#word = 0xA3, значение ram1_end = 0x13FF

jrule clear_ram1 код операции jrule MEM= 0x23, в следующей ячейке памяти лежит сдвиг (на сколько нужно изменить программный счетчик). В момент выполнения текущей команды в программном счетчике содержится адрес следующей ячейки со следующей операцией (AE). Для того, чтобы попасть на метку «clear_ram1» (ячейка 7F) необходимо уменьшить значение

программного счетчика на 7, в дополнительном коде -7=0xF9 (0xF9 +7 =0), следовательно, сдвиг, лежащий в следующей ячейке равен 0xF9.

ldw X,#stack_start код операции ldw X, #word = 0xAE, значение stack_start = 0x1400

clr (X) код операции = 0x7F

incw X код операции = 0x5C

cpw X,#stack_end код операции cpw X,#word = 0xA3, значение stack_end = 0x17FF

jrule clear_stack код операции jrule MEM= 0x23, в следующей ячейке памяти лежит сдвиг (на сколько нужно изменить программный счетчик). В момент выполнения текущей команды в программном счетчике содержится адрес следующей ячейки со следующей операцией (20). Для того, чтобы попасть на метку «clear_stak» (ячейка 7F) необходимо уменьшить значение программного счетчика на 7, в дополнительном коде -7=0xF9 (0xF9 +7 =0), следовательно, сдвиг, лежащий в следующей ячейке равен 0xF9.

jra infinite_loop код операции jra MEM= 0x20, в следующей ячейке памяти лежит сдвиг (на сколько нужно изменить программный счетчик). В момент выполнения текущей команды в программном счетчике содержится адрес следующей ячейки со следующей операцией (80). Для того, чтобы попасть на метку «infinite_loop» (ячейка 20) необходимо уменьшить значение программного счетчика на 2, в дополнительном коде -2=0xFE (0xFE +2 =0), следовательно, сдвиг, лежащий в следующей ячейке равен 0xFE.

На этом заканчивается основная программа main. В следующую ячейку, с номером 0*x*80*A*4 компилятор разместит код операции IRET=0*x*80, следовательно, метка «NonHandledInterrupt» будет указывать на ячейку с номером 0*x*80*A*4 и по умолчанию, все вектора прерываний, кроме «reset», будут записаны следующим образом: 0*x*82 0*x*00 0*x*80 0*x*A4

10) Добавить в автоматически сгенерированный код, перед меткой infinite_loop.l код настройки первого вывода порта A в режим выхода. Для этого, в документации, необходимо найти адрес регистра PA_DDR и используя команду mov longmem, #byte. Для записи в регистр PA_DDR, адрес которого равен 0x5002, команда будет выглядеть следующим образом:

mov $5002,#$02, где mov операция пересылки, знак $ означает шестнадцатеричную систему счисления, а знак # означает пересылку байта данных.

11) Выполнить компиляцию программы, сборку и ее загрузку в микроконтроллер.

12) Убедиться, что после загрузки, код операции mov $5002,#$02, равный 0x35 0x02 0x50 0x02, размещается в памяти программ начиная с ячейки памяти с адресом 0x80A2. Код операции mov longmem, #byte разен 0x35, следующая ячейка содержит байт данных, которые необходимо загрузить в регистр PA_DDR (0x02) и два байта, содержащие адрес этого регистра (0x50 0x02).

13) Поставить точку останова на команде записи в регистр PA_DDR числа 0x02, запустить приложение. После остановки, в программном счетчике должно лежать число 0x0080A2. Открыть в окне просмотра регистров периферии регистры порта A. В регистре PA_DDR должен быть записан 0x00. Выполнить один шаг в программе (Step Into (F11)). После этого в регистр PA_DDR должно положиться число 0x02. Определить значение программного счетчика, найти ячейки памяти, хранящие это значение.

14) Найти в документации на микроконтроллер (STM8S207 datasheet стр. 35) адреса регистров PA_CR1, PA_CR2, PA_ODR, настроить первый вывод порта A в режиме двухтактного выхода с частотой тактирования 10 МГц и вывести уровень логической «1».

15) Соединить вывод PA1 со светодиодом. Выполнить компиляцию программы, сборку и ее загрузку в микроконтроллер. Проверить правильность работы программы.

16) Поместить в программу, между командой jrule clear_stak и меткой infinite_loop.l следующий код:

```
        mov $5002,#$02   ;PA_DDR=0x02
        mov $5003,#$02   ;PA_CR1=0x02
        mov $5004,#$02   ;PA_CR2=0x02
    blink.l
        mov $5000,#$02   ;PA_ODR=0x02
        mov $00,#$FF
        mov $01,0
        mov $02,#$5
    delayon.l
        dec 0
        jrne delayon
        dec 1
        jrne delayon
        dec 2
        jrne delayon

        mov $5000,#$00   ;PA_ODR=0x00

        mov $00,#$FF
        mov $01,0
        mov $02,#$5
    delayoff.l
        dec 0
        jrne delayoff
        dec 1
        jrne delayoff
        dec 2
        jrne delayoff
```

```
mov $5000,#$02   ;PA_ODR=0x02
jra blink
```

17) Выполнить компиляцию программы, сборку и ее загрузку в микроконтроллер. В режиме отладки изучить логику работы программного кода. Дать краткие комментарии по работе программы.

18) Между командой jrule clear_stak и меткой infinite_loop.l настроить первый вывод порта A в режим двухтактного выхода с любой частотой тактирования и первый вывод порта B в режим входа с подтяжкой к питанию и с запрещенным внешним прерыванием. Внутри бесконечного цикла содержимое регистра PB_IDR помещать в регистр PA_ODR.

19) На плате верхний вывод разъема P25(слева от матричной клавиатуры) соединить с выводом GND, а крайний левый вывод разъема P28 (снизу матричной клавиатуры) соединить первым выводом порта B. Таким образом, первый вывод порта B оказывается соединен с крайней верней левой кнопкой матричной клавиатуры. Когда кнопка не нажата, благодаря подтяжки к питанию, на первом выводе порта B сигнал логической «1», при нажатии кнопки, данный вывод соединяется с «землей» и сигнал сменяется на логический «0».

20) Выполнить компиляцию программы, сборку и ее загрузку в микроконтроллер. После запуска программы проверить реакцию схемы на нажатие и отпускание кнопки. Объяснить работы схемы.

21) Убрать из бесконечно цикла помещение содержимого регистра PB_IDR в регистр PA_ODR. Выполнить компиляцию программы, сборку и ее загрузку в микроконтроллер. После запуска программы проверить реакцию схемы на нажатие и отпускание кнопки. Объяснить работы схемы.

22) Изменить настройку первого вывода порта B таким образом, чтобы внешние прерывания стали разрешены. Добавить пустую операцию nop между меткой NonHandledInterrupt.l и операцией iret. Выполнить компиляцию программы, сборку и ее загрузку в микроконтроллер. Поставить точку

останова на операции nop, запустить выполнение программы и проверить, перейдет ли контроллер в обработчик прерываний после нажатия кнопки.

23) Несмотря на то, что прерывания на первом выводе порта B разрешены, микроконтроллер не будет переходить в обработчик прерываний в связи с тем, что по умолчанию все прерывания маскированы. Для сброса маскирования всех прерываний необходимо после строки настройки разрешения внешнего прерывания первого вывода порта B добавить операцию rim (Reset Interrupt Mask). Выполнить компиляцию программы, сборку и ее загрузку в микроконтроллер. Поставить точку останова на операции nop, запустить выполнение программы и проверить, перейдет ли контроллер в обработчик прерываний после нажатия кнопки. Обратите внимание на содержимое указателя стека, объясните полученное значение. Напишите адреса ячеек стека, куда в результате ухода в обработчик прерывания было записано содержимое регистров, укажите содержание этих ячеек и уточните содержимое какого регистра было записано в каждую конкретную ячейку.

На данный момент все прерывания микроконтроллера, за исключением «reset» имею один обработчик прерывания, так как в таблице векторов прерывания указана одна и та же метка.

24) Найти в документации на микроконтроллер (STM8S207 datasheet стр. 46) номер внешнего прерывания порта B и соответствующий вектор внести изменение названия метки, например, вместо $0x82000000+NonHandledInterrupt написать $0x82000000+PB_Ext_Interrupt. Таким образом в векторе прерывания будет указан адрес метки PB_Ext_Interrupt, которую необходимо создать в коде программы, добавив после операции iret стандартного вектора следующие строки:

```
interrupt PB_Ext_Interrupt
PB_Ext_Interrupt.l
    nop
    iret
```

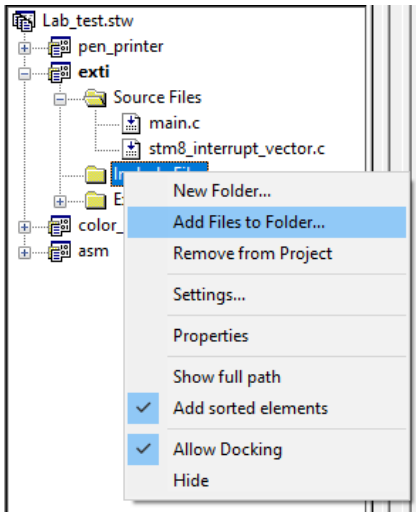
25) Инструкция nop не обязательна, но нужна нам для установки на нее точки останова. Выполнить компиляцию программы, сборку и ее загрузку в микроконтроллер. Поставить точку останова на обе операции nop (в обоих обработчиках прерывания), запустить выполнение программы и проверить, в какой из обработчиков прерываний перейдет микроконтроллер после нажатия кнопки.

26) Если все было сделано правильно, то микроконтроллер должен перейти в недавно созданный вами обработчик прерываний. **Результат показать преподавателю**

27) Добавьте в ваше рабочее пространство новый проект для написания программы на языке Си.

28) Через меню File -> New text file создайте пустой файл и сохраните его как main.h в папке с проектом.

29) В окне рабочего пространства в активном проекте на папке Include Files нажать правую кнопку мыши и выбрать добавление нового файла в папку (Add Files to Folder…). Выбрать ранее созданный файл main.h.



30) Открыть добавленный файл и добавить в него следующее содержимое: @far @interrupt void EXTI_PortC (void);

Таблица векторов прерываний находится в памяти программ (flash) в диапазоне адресов 0x008000 - 0x00807F. Каждый вектор занимает 4 байта. Всего векторов 32 и под них отведено 128 байт. Вектор прерывания представляет из себя блок из 4 байт, где первый байт - ассемблерная инструкция INT (машинный код 0x82), а остальные 3 байта - абсолютный 24-битный адрес функции обработчика прерываний. При вызове прерывания происходит переход по вектору прерывания (при этом программный счетчик PC не меняется), в стек загружаются регистры ядра (программный счетчик, регистры X и Y, аккумулятор, регистр состояния), затем в программный счетчик загружается 24-битный адрес из вектора прерывания. Выход из обработчика прерывания выполняется ассемблерной инструкцией IRET. Модификаторы типа функции @far @interrupt для void позволяют: @far:

получить именно 24-битное значение адреса функции в адресном пространстве программного кода. @interrupt: выполнять выход из функции инструкцией IRET.

31) Используя директиву #include в файлах main.c и stm8_interrupt_vector.c подключить содержимое заголовочного файла main.h. Также, в файле main.c, после функции main создать обработчик внешнего прерывания порта C.

```
@far @interrupt void EXTI_PortC (void)
{
}
```

32) В файле stm8_interrupt_vector.c в структуре векторов прерывания заменить в соответствующем прерыванию порта C векторе заменить имя обработчика прерывания с NonHandledInterrupt на EXTI_PortC.

33) Настроить первый вывод порта C в режим входа с подтяжкой к питанию и разрешить внешние прерывания.

34) Сбросить маскирование всех прерываний используя ассемблерную вставку: _asm("rim"); Данная команда снижает приоритет основной программы и дает возможность прервать ее выполнение при возникновении запроса на прерывание. Также в конце функции main необходимо добавить бесконечный цикл.

35) В обработчике внешнего прерывания порта C добавить операцию инкремента переменной, которую заранее глобально объявить, как long int.

36) Выполнить компиляцию программы, сборку и ее загрузку в микроконтроллер. Поставить точку останова на операции инкремента переменной. Первый вывод порта C соединить одной из кнопок, коммутирующей на «землю» (по аналогии с пунктом 18). Убедиться, что при нажатии на кнопку, с которой соединен первый вывод порта C, микроконтроллер заходит в обработчик прерывание и останавливается на точке останова.

37) Выполнить программный сброс выполнения программы, убрать точку останова и запустить приложение. Остановить выполнение программы (Stop Program (Shift+F5)). Открыть окно просмотра переменных, добавить ранее объявленную переменную, которая инкрементируется в обработчике внешнего прерывания порта C и посмотреть ее содержимое. Запустить программу и кратковременно нажать и отпустить кнопку, соединенную с первым выводом порта C.

38) Остановить выполнение программы (Stop Program (Shift+F5)), посмотреть на содержимое переменной. Исходя из логики работы программы, содержимое переменной должно было увеличиться на единицу, однако, это скорей всего не так и она увеличилась намного больше, чем ожидалось. Значит, за одно нажатие кнопки, микроконтроллер успел несколько раз обработать запрос на внешнее прерывание порта C.

Чтобы понять одну из причин, необходимо ознакомиться с регистром настройки внешнего прерывания EXTI_CR1. Данный регистр описан в документации на микроконтроллер (Reference manual) на стр. 70. В данном регистре осуществляется настройка чувствительности внешнего прерывания, а именно какое событие вызывает запрос на внешнее прерывание. Таких событий может быть 4 варианта, а именно: 1) спадающий фронт и низкий уровень сигнала; 2) только нарастающий фронт; 3) только спадающий фронт; 4) нарастающий и спадающий фронты сигнала. Согласно документации, после сброса микроконтроллера в данном регистре хранится значение «0», что настраивает чувствительность к внешним прерываниям на портах A, B, C и D по спадающему фронту и низкому уровню сигнала. Пока нажата кнопка, на первый вывод порта C подается сигнал логического нуля и это приводит к формированию запроса на внешнее прерывание.

39) Настроить чувствительность внешнего прерывания на первом выводе порта C только к спадающему фронту сигнала. Выполнить компиляцию программы, сборку и ее загрузку в микроконтроллер. Проверить реакцию на нажатие кнопки аналогично пунктам 36 и 37. Случай, когда

переменная за одно нажатие кнопки увеличивается больше чем на один, можно объяснить «дребезгом контактов». Если на входе микроконтроллера после начала выполнения обработчика прерывания возникает сигнал вызывающий запрос на внешнее прерывание, то после выхода из обработчика внешнего прерывания микроконтроллер снова выполнит процедуру ухода в обработчик прерывания.

40) Настроить еще одно внешнее прерывание на шестом выводе порта D с чувствительностью к нарастающему фронту сигнала. Написать код программы обеспечивающий инверсию состояния двух светодиодов, соединенных с первым и вторым выводами порта A, по нажатию кнопок, соединенных с первым выводом порта C и шестым выводом порта D. Нажатие на одну кнопку должно изменять состояние одного светодиода, а нажатие на другую, изменять состояние второго светодиода. Выполнить компиляцию программы, сборку и ее загрузку в микроконтроллер. Проверить правильность работы программы. Зафиксировать в какой момент (нажатия или отпускания кнопки) изменяется состояние светодиода, ответ пояснить.

41) Изменить оба обработчика внешних прерываний таким образом, чтобы при нажатии кнопки соответствующий светодиод изменил своё состояние 10 раз. Задержка после изменения состояния должна быть от 500 мс – до 1с. Выполнить компиляцию программы, сборку и ее загрузку в микроконтроллер. Проверить корректность работы программы. В случае правильной работы системы, нажать на кнопку, соединенную с портом C и не дожидаясь окончания обработки данного прерывания нажать на вторую кнопку. Описать логику работы микроконтроллера. Изменить порядок вызова прерываний, сначала нажать на кнопку, соединенную с портом D, и не дожидаясь окончания обработки данного прерывания нажать на вторую кнопку. Описать логику работы микроконтроллера.

Программный приоритет прерываний можно настроить с помощью регистров ITC_SPR1 - ITC_SPR8 (Reference manual стр. 69). После сброса данные регистры содержат значение 0xFF. Данные регистры содержат

битовые поля VECTxSPR[1:0], где вместо x указывается номер вектора прерывания. Например битовое поле VECT0SPR[1:0] отвечает за настройку приоритета внешнего прерывания высшего уровня (TLI), VECT3SPR[1:0] отвечает за настройку приоритета внешнего прерывания порта A и т.д. Биты VECT0SPR[1:0] регистра ITC_SPR1 устанавливаются в 1 аппаратно, так как у прерывания TLI должен быть высший приоритет. Биты [7:4] регистра ITC_SPR8 зарезервированы и устанавливаются в 1 аппаратно. **Запись во все битовые поля значения 10_2 не допускается! В случае записи значения 10_2 содержимое битого поле не меняется и хранит прежнее значение!** Когда микроконтроллер прекращает выполнять основной код программы и уходит в обработчик прерывания, содержимое битового поля VECTxSPR[1:0] текущего прерывания загружается в регистр CCR в биты I1 и I0. Существует 4 уровня приоритета прерываний:

Бит I1 в регистре CCR	Бит I0 в регистре CCR	Уровень	
1	0	0	Самый низкий приоритет
0	1	1	↓
0	0	2	
1	1	3	Самый высокий приоритет

По умолчанию, основная программа имеет наивысший приоритет (11_2), в связи с этим прерывания не могут остановить выполнение основного кода программы. После выполнения операции RIM приоритет основной программы понижается до уровня 0 (10_2), что означает, что все прерывания с этого момента имеют более высокий приоритет.

42) Установить программный приоритет внешнего прерывания порта D равный второму уровню. Установить программный приоритет внешнего прерывания порта C равный первому уровню. Выполнить компиляцию

программы, сборку и ее загрузку в микроконтроллер. Запустить программу, нажать на кнопку, соединенную с портом C и не дожидаясь окончания обработки данного прерывания нажать на вторую кнопку. Описать логику работы микроконтроллера. Изменить порядок вызова прерываний, сначала нажать на кнопку, соединенную с портом D, и не дожидаясь окончания обработки данного прерывания нажать на вторую кнопку. Описать логику работы микроконтроллера.

43) Установить программный приоритет внешнего прерывания порта D равный второму уровню. Установить программный приоритет внешнего прерывания порта C равный третьему уровню. Выполнить компиляцию программы, сборку и ее загрузку в микроконтроллер. Запустить программу, нажать на кнопку, соединенную с портом C и не дожидаясь окончания обработки данного прерывания нажать на вторую кнопку. Описать логику работы микроконтроллера. Изменить порядок вызова прерываний, сначала нажать на кнопку, соединенную с портом D, и не дожидаясь окончания обработки данного прерывания нажать на вторую кнопку. Описать логику работы микроконтроллера.

44)