

Общее описание архитектуры ARM и 32-разрядных микроконтроллеров STM

Процессоры **ARM** являются ключевым компонентом для большого количества успешных 32-битных встраиваемых систем. Процессоры **ARM** широко используются в мобильных телефонах, планшетах и других портативных устройствах. **ARM** основаны на **RISC-архитектуре**, что позволяет уменьшить потребление энергии процессором и, таким образом, делает их идеальным выбором для встраиваемых систем.

Хотя **ARM** основаны на **RISC-архитектуре**, они не полностью повторяют принципы построения таких систем. Для того, чтобы сделать **ARM** более приспособленными к использованию во встраиваемых системах, пришлось пойти на следующие отклонения от принципов **RISC**:

1. Переменное количество циклов выполнения для простых инструкций. Простые инструкции **ARM** могут потребовать на выполнение более одного цикла. Например, выполнение инструкций **Load** и **Save** зависит от количества регистров, которые им переданы.
2. Возможность соединять команды сдвига и вращения с командами обработки информации.
3. Условное выполнение – инструкция выполняется только в том случае, если выполняется конкретное условие. Это увеличивает производительность и позволяет избавиться от операторов ветвления.
4. Улучшенные инструкции – процессоры **ARM** поддерживают улучшенные **DSP-инструкции** для операций с цифровыми сигналами.

Программист может рассматривать ядро **ARM** как набор функциональных блоков – **ALU**, **MMU** и др., – соединенных шиной данных. Данные поступают в процессор через шину данных. Декодер инструкций обрабатывает инструкции перед их выполнением. **ARM** могут работать только с данными, которые записаны в регистрах, поэтому перед выполнением инструкций в регистры записываются данные для их выполнения. **ALU** считывает данные из регистров, выполняет необходимые операции и записывает результат обратно в регистр, откуда его можно записать во внешнюю память.

Процессоры **ARM** содержат до **18 регистров**: 16 регистров данных и 2 регистра процессов. Все регистры содержат **32 бита** и именуются от **R0** до **R15**. Регистры **R13**, **R14**, **R15** используются для выполнения определенных специфических задач:

- **R13** используется в качестве указателя стека;
- **R14** используется как связывающий регистр;
- **R15** играет роль счетчика.

В зависимости от контекста эти регистры могут использоваться как регистры общего назначения. Также имеется два **программных регистра**, которые называются **CPSR** (Current Program Status Register) и **SPSR** (Saved Program Status Register), которые используются для сохранения состояния процессора и программы.

Одними из последних процессоров для встраиваемых систем, являются процессоры, основанные на архитектуре **ARM Cortex-M4**. Эти процессоры предназначены для использования в цифровой обработке сигналов (Digital Signal Processing, DSP). В общем виде микроконтроллеры, основанные на базе **ARM Cortex-M4** имеют следующие внутренние модули (**рисунок 1**): Микроконтроллер, установленный на рассматриваемой плате, **STM32F407VG**, в качестве основы использует именно решение **ARM Cortex-M4**.



Рисунок 1 - Встроенные модули ARM Cortex-M4

Семейство микроконтроллеров STM32

Семейство микроконтроллеров **STM32** построено с использованием **32-разрядного ядра Cortex** различных версий (в микроконтроллере, установленном на плате используется ядро **Cortex-M4**). Некоторые основные характеристики ядра микроконтроллеров STM32 представлены в таблице 1.

Таблица 1- Основные характеристики ядра микроконтроллеров STM32

Характеристика	Значение
Ширина слов для данных, разряд	
Архитектура	Гарвард
Конвейер	3-ступенчатый
Набор инструкций	RISC
Организация памяти программ	32
Буфер предвыборки, разряд	2x64
Средний размер инструкции, байт	2
Тип прерываний	Векторизированные
Задержка реагирования на прерывания	12 циклов
Режимы управления энергопотреблением	Сон, сон по выходу, глубокий сон
Отладочный интерфейс	ST-LINK, JTAG

Микроконтроллеры данного типа построены на гарвардской архитектуре и имеют **3-ступенчатый конвейер**, который минимизирует время выполнения команд. Они разработаны для построения систем с максимальной энергоэффективностью и имеют несколько режимов управления энергопотреблением. В них используются внутренние интерфейсы памяти шириной больше, чем средняя длина инструкции. Это минимизирует число доступов к шине памяти, а, следовательно, и потребление электроэнергии, связанное с операциями по шине и чтением энергонезависимой памяти. Технология непрерывной обработки прерываний с исключением внутренних операций над стеком (tail chaining) сокращает время реакции на прерывания и исключает лишние операции со стеком.

На **рисунке 2** представлено упрощенное представление цифрового периферийного устройства. Периферийный узел может быть разделен на два главных блока. Первый блок – это ядро, которое содержит конечные автоматы, счетчики и любой вид комбинаторной или последовательной логики. Оно предназначено для выполнения задач, не требующих участия процессора, таких как простые задачи передачи данных, управления аналоговыми входами или выполнения функций, привязанных к синхросигналам. Ядро периферийного узла связывается с внешним миром через порты ввода/вывода МК. Внешние соединения могут состоять из нескольких сигналов или сложных шин. Второй блок – настройка и управление периферией, которые осуществляются приложением через регистры, соединенные с внутренней шиной, разделяемой с другими ресурсами МК.

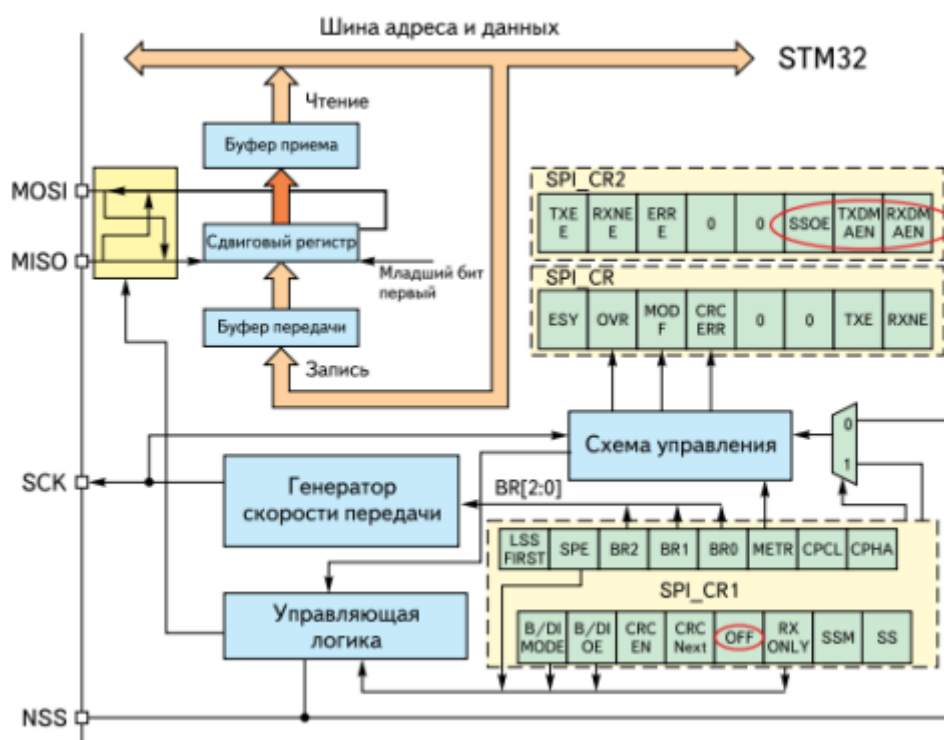


Рисунок 2 - Представление цифрового периферийного устройства

Краткое описание платы STM32F4 Discovery

Плата **STM32F4 Discovery** (рисунок 3) предназначена для ознакомления с возможностями **32-битного МК** на основе **ARM-архитектуры**, а также для реализации собственных устройств и приложений с использованием аппаратного обеспечения платы.

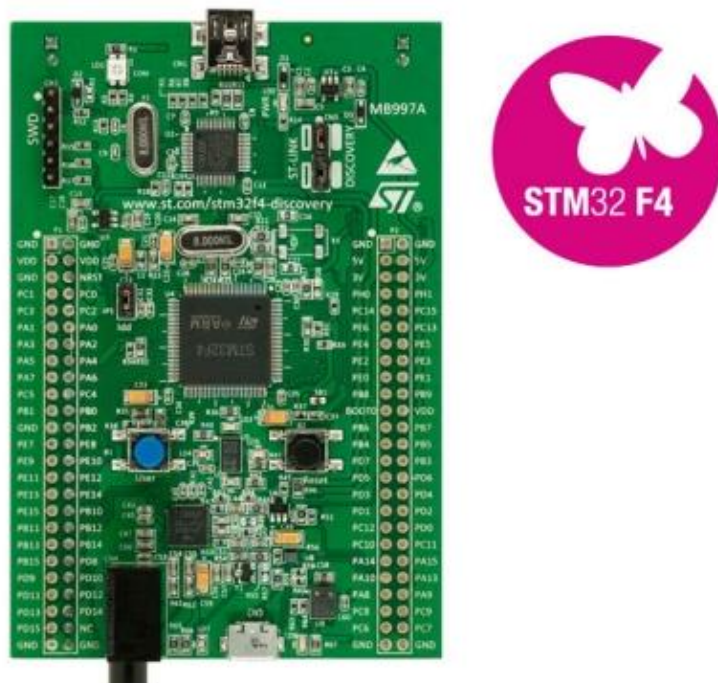


Рисунок 3 - Внешний вид платы STM32F4 Discovery

Плата **STM32F4 Discovery** оснащена:

- микроконтроллером **STM32F407VGT6** с ядром **Cortex-M4F** тактовой частотой **168 МГц**, **1 Мб** Flash-памяти, **192 кб** RAM в корпусе **LQFP100**;
- отладчиком **ST-Link/V2** для отладки и программирования МК;
- питанием платы через **USB** или от внешнего источника питания **5 В**;
- датчиком движения **LIS302DL** и выходами цифрового акселерометра по трем осям;
- датчиком звука **MP45DT02**;
- звуковым ЦАП **CS43L22**;
- восемью светодиодами: **LD1** (красный/зеленый) для USB-подключения, **LD2** (красный) для питания **3.3 В**, четыре пользовательские светодиода: **LD3** (оранжевый), **LD4** (зеленый), **LD5** (красный), **LD6** (синий) и два светодиода для USB On-The-Go – **LD7** (зеленый) и **LD8** (красный);
- двумя кнопками (для программирования пользователем и для перезапуска).

Таким образом, отладочная плата оснащена большим количеством периферии, что позволяет сразу же реализовывать на ней примеры различной сложности.

Начало работы с отладочной платой STM32F4 Discovery

Для начала работы следует установить среду разработки. Далее рассмотрен процесс установки и начала работы с STM32 - **STM32CubeIDE**. Рассмотрим детали установки и использования среды разработки **STM32CubeIDE**.

Скачать программу можно с сайта производителя – <https://www.st.com/en/development-tools/stm32cubeide.html>. Нажав на кнопку **Get Software** (Рисунок 4)

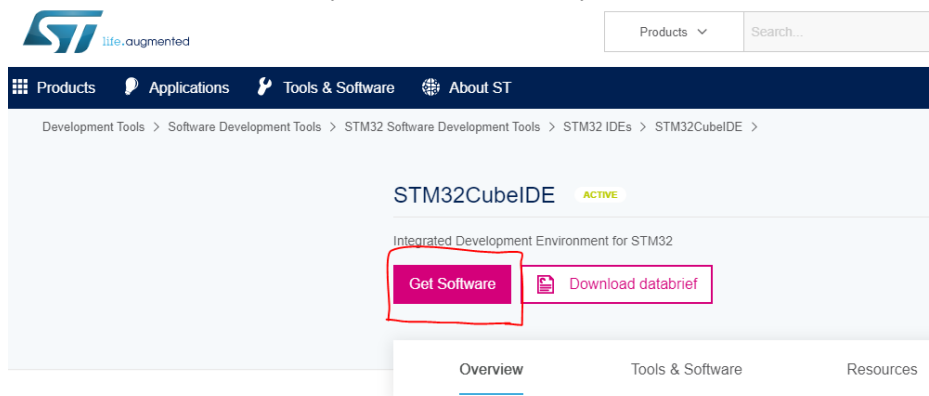


Рисунок 4 – Веб-интерфейс страницы скачивания STM32CubeIDE

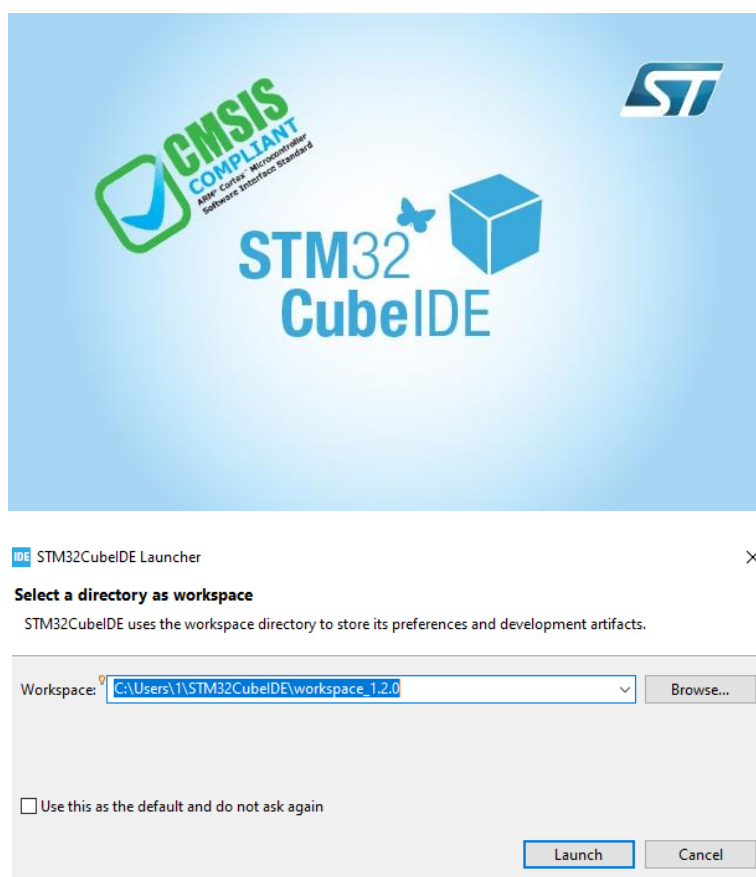


Рисунок 5 – Выбор рабочего пространства IDE

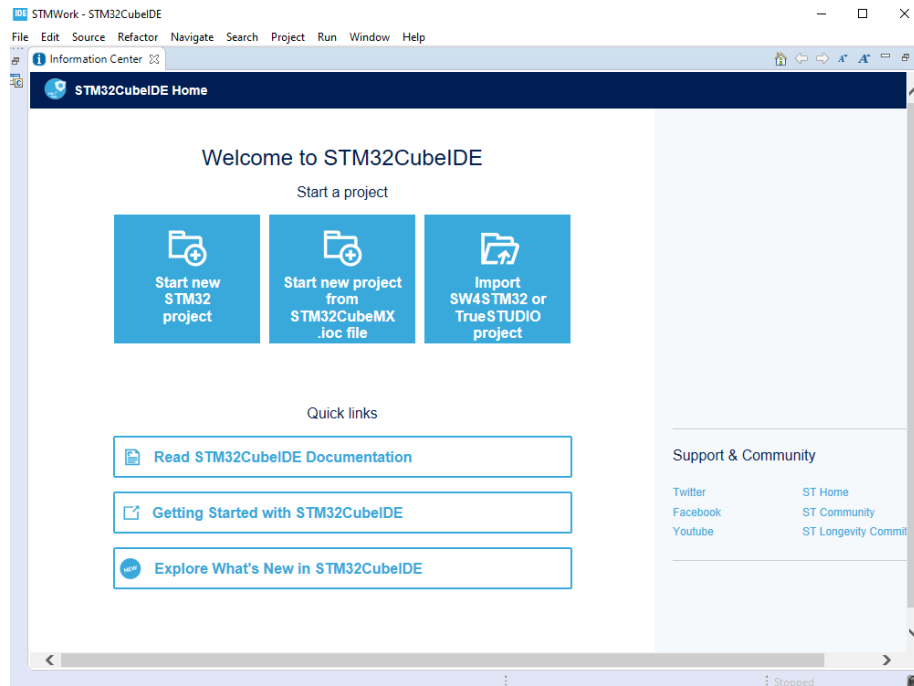


Рисунок 6– Стартовый экран STM32CubeIDE

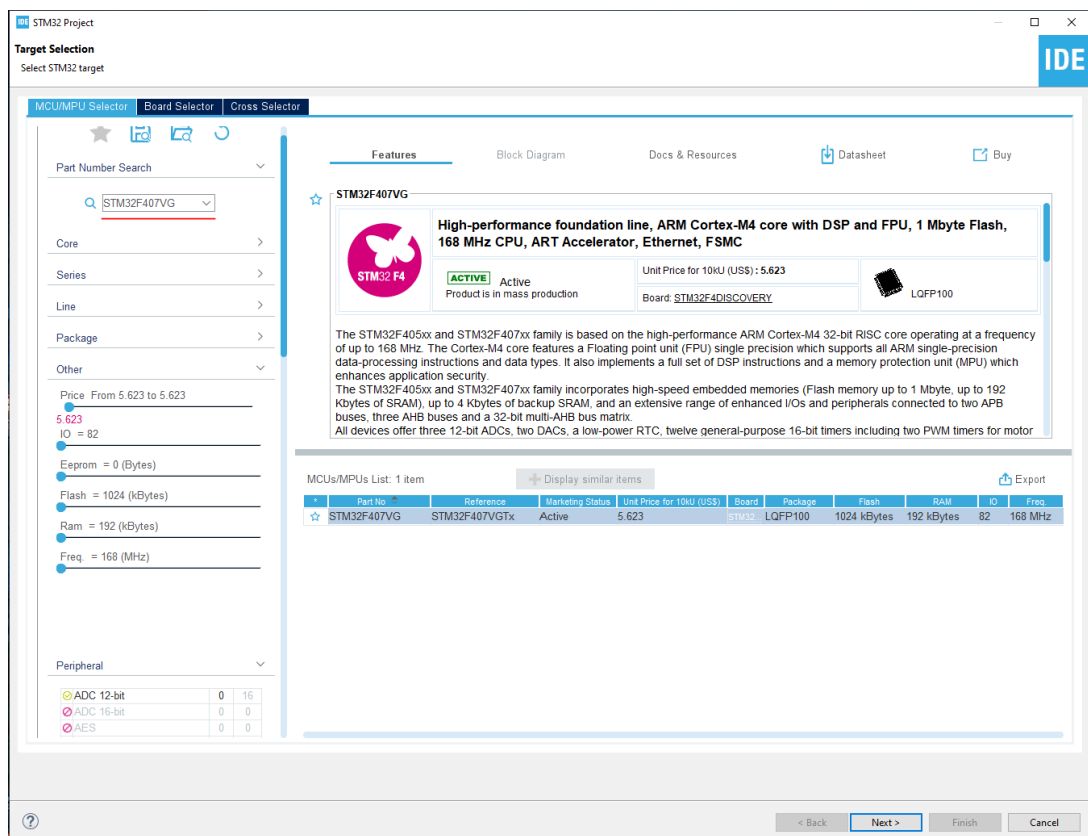


Рисунок 7 –Встроенный в IDE STM32CubeMx

Создание нового проекта — *File/New/STM32Project*. Затем появляется окно выбора микроконтроллера. Далее предлагается выбрать имя проекта, расположение, язык программирования C/C++, исполняемый файл/статическая библиотека и будет ли проект сгенерирован с помощью CubeMX. Выберем тип проекта *Empty* — финиш.

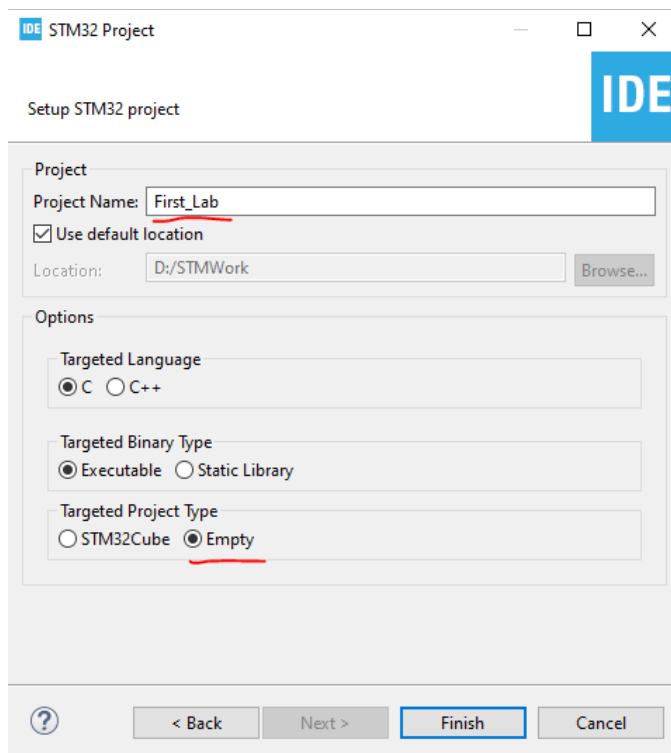


Рисунок 8 – Создание проекта

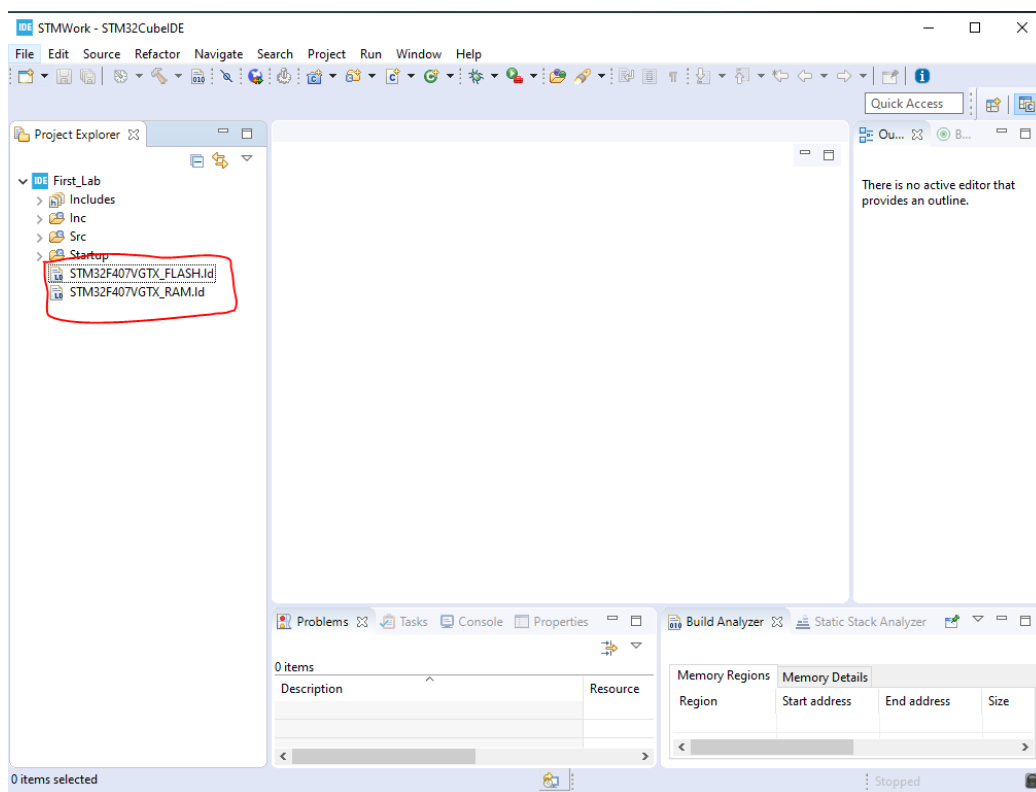


Рисунок 9 – Основной интерфейс STM32CubeIDE

Слева, в окне **Project Explorer**, появилось дерево проекта. Удаляем всё, кроме скрипта линкера т.е. файла с расширением **.ld**.

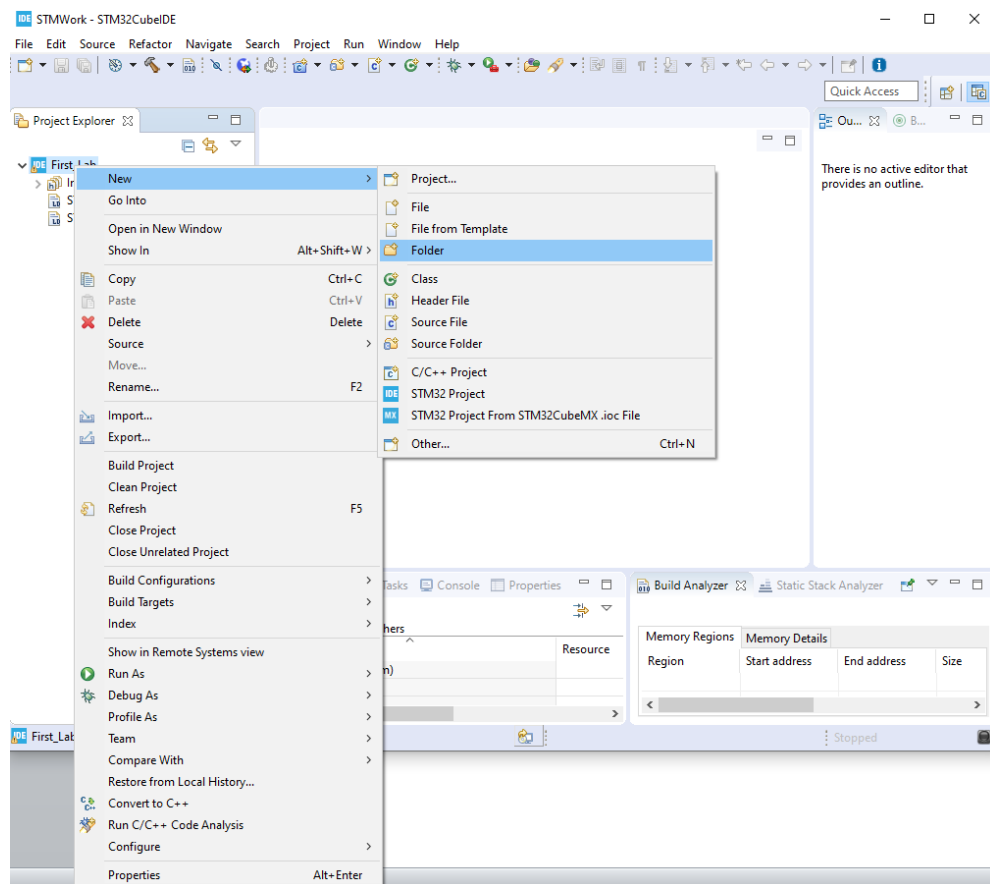


Рисунок 10 – Создание папки в проекте

Все манипуляции с папками и файлами можно проводить как в проводнике, так и внутри **IDE**, нажав правой кнопкой на название проекта, к примеру: **правая кнопка** → **new** → **Folder**. Если структура проекта изменялась вне **IDE**, то нужно просто обновить проект: **правая кнопка** → **Refresh**.

Структура проекта выглядит так (Рисунок 11):

- **Startup** – здесь будет храниться скрипт линкера, тот самый, оставшийся от сгенерированного проекта, а также startup файл взятый из CMSIS
- **CMSIS\src** и **CMSIS\inc** – здесь будут лежать исходники, файлы с расширением **.c** в папке **src** и заголовочные файлы с расширением **.h** в папке **inc** соответственно, относящиеся к библиотеке **CMSIS**
- **Core\src** и **Core\inc** – здесь будет расположен собственно сам проект **main.c** и **main.h**

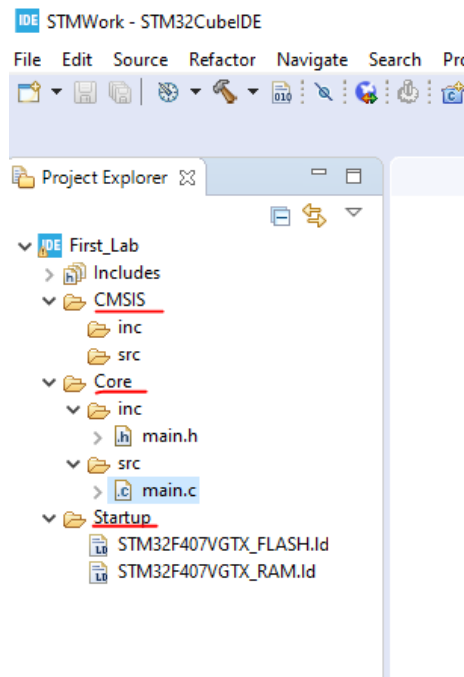


Рисунок 11 – Структура проекта

Теперь нужно перенести файлы библиотеки **CMSIS** в проект. Библиотека состоит из файлов ядра и файлов периферии. Файлы ядра начинаются с **core_** или **cmsis_** они общие для всех микроконтроллеров, использующих данное ядро. Файлы периферии содержат в названии наименование микроконтроллера **stm32** и специфичны для конкретного производителя, в данном случае, компании **STM**.

Нужно скопировать (Рисунок 12):

В CMSIS\inc:

- Drivers\CMSIS\Include\cmsis_compiler.h
- Drivers\CMSIS\Include\cmsis_gcc.h
- Drivers\CMSIS\Include\cmsis_version.h
- Drivers\CMSIS\Include\core_cm0.h
- Drivers\CMSIS\Device\ST\STM32F0xx\Include\stmf0xx.h
- Drivers\CMSIS\Device\ST\STM32F0xx\Include\stm32f072xb.h
- Drivers\CMSIS\Device\ST\STM32F0xx\Include\system_stm32f0xx.h

В CMSIS\src:

- Drivers\CMSIS\Device\ST\STM32F0xx\Source\Templates\system_stm32f0xx.c

В Startup:

- Drivers\CMSIS\Device\ST\STM32F0xx\Source\Templates\gcc\startup_stm32f072xb.s

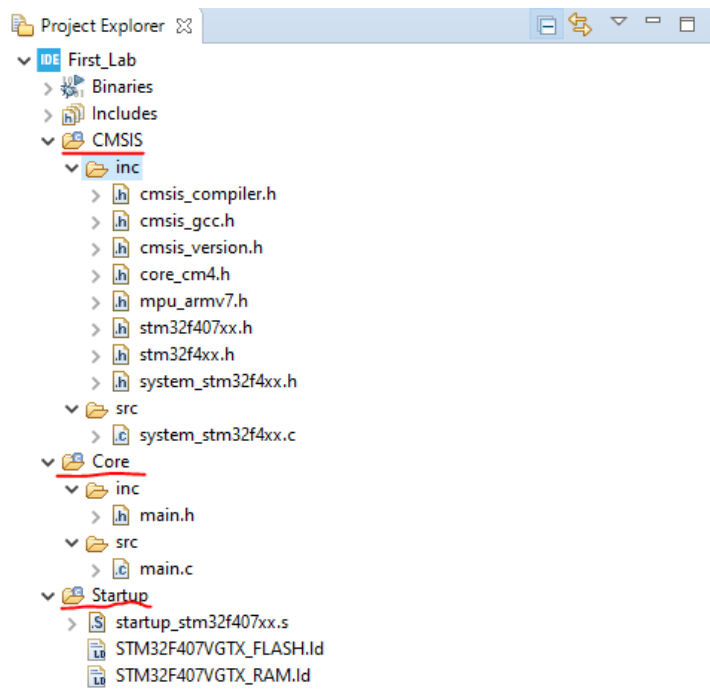


Рисунок 12 – Проект в заполненном виде

Так как были проведены некоторые манипуляции с папками проекта, нужно отобразить это в настройках. **Правая кнопка по названию проекта -> Properties -> C/C++ Build -> Settings -> Tool Settings -> MCU GCC Linker -> General** – здесь нужно указать новое расположение скрипта линкера с помощью кнопки **Browse...**

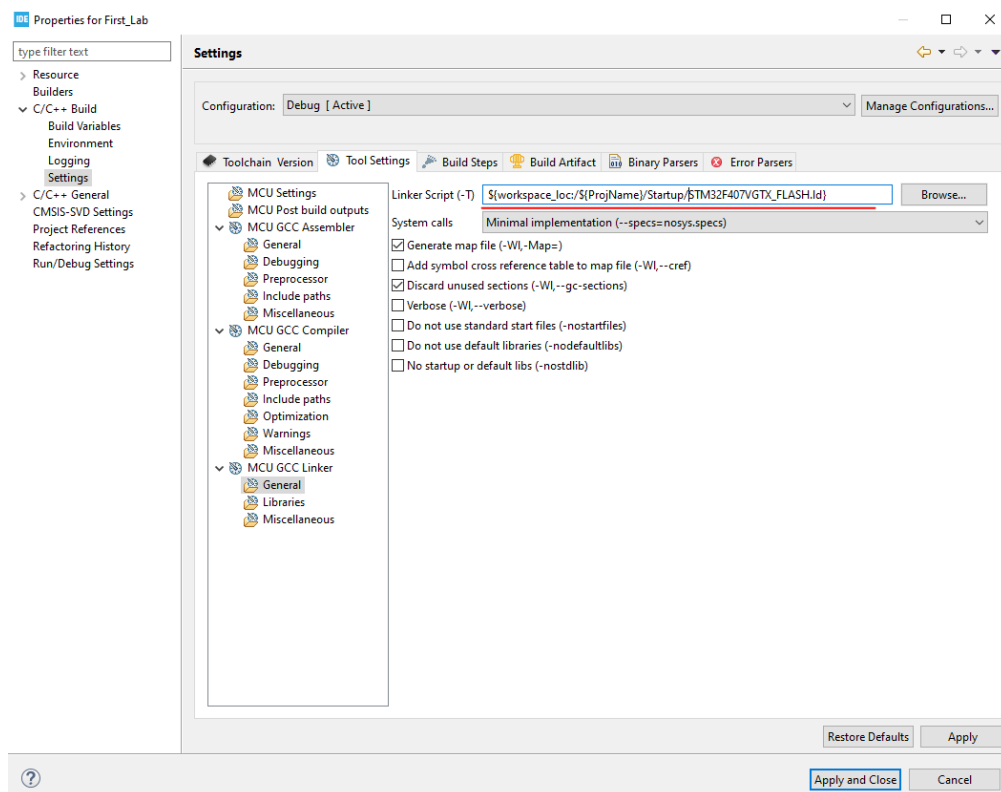


Рисунок 13 – Окно настроек проекта

Также нужно указать пути к файлам проекта **Properties -> C/C++ General -> Includes**
Properties -> C/C++ General -> Source Location

В **Includes** пути к папкам **inc**, а в **Source Location** логично было-бы к папкам **src**, но если так сделать, то в дереве проекта будут отдельно добавлены эти папки.

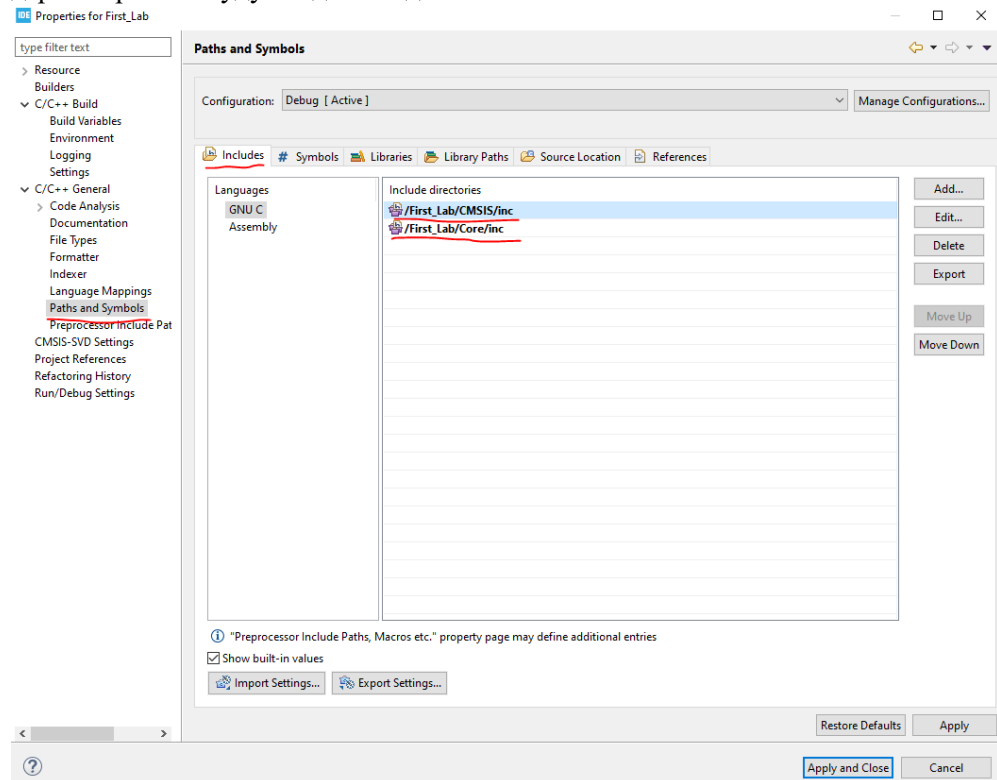


Рисунок 14 – Вкладка Includes с прописанными путями к файлам проекта

Чтобы не загромождать визуально дерево, в **Source Location** можно указать корневые папки **Core**, **CMSIS** и **Startup**.

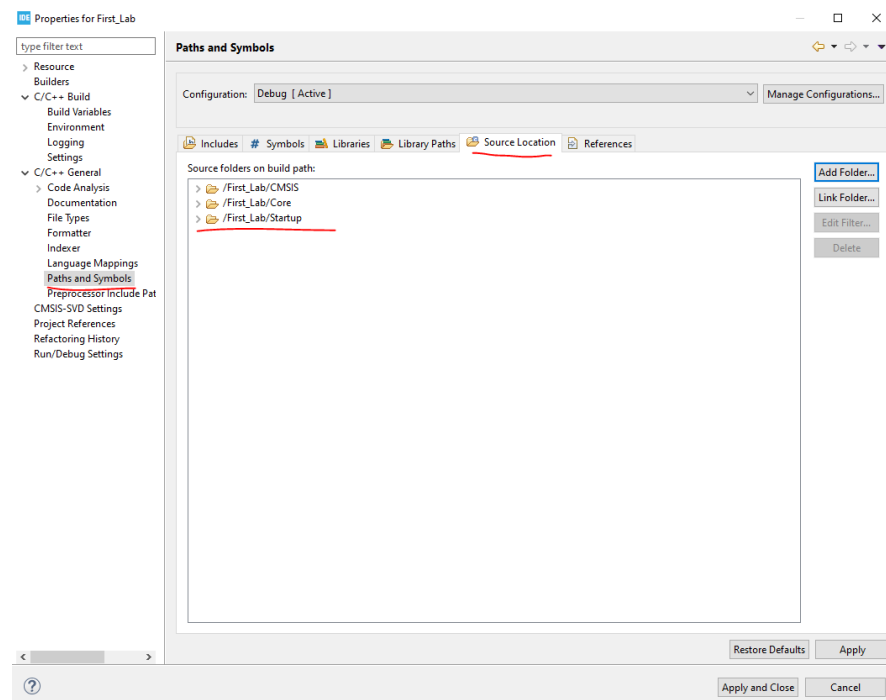


Рисунок 15 – Вкладка Source Location с прописанными корневыми папками проекта

Для того чтобы проект скомпилировался нужно раскомментировать в файле **stm32f4xx.h** строку с названием микроконтроллера и в **main.c** добавить функцию **main**.

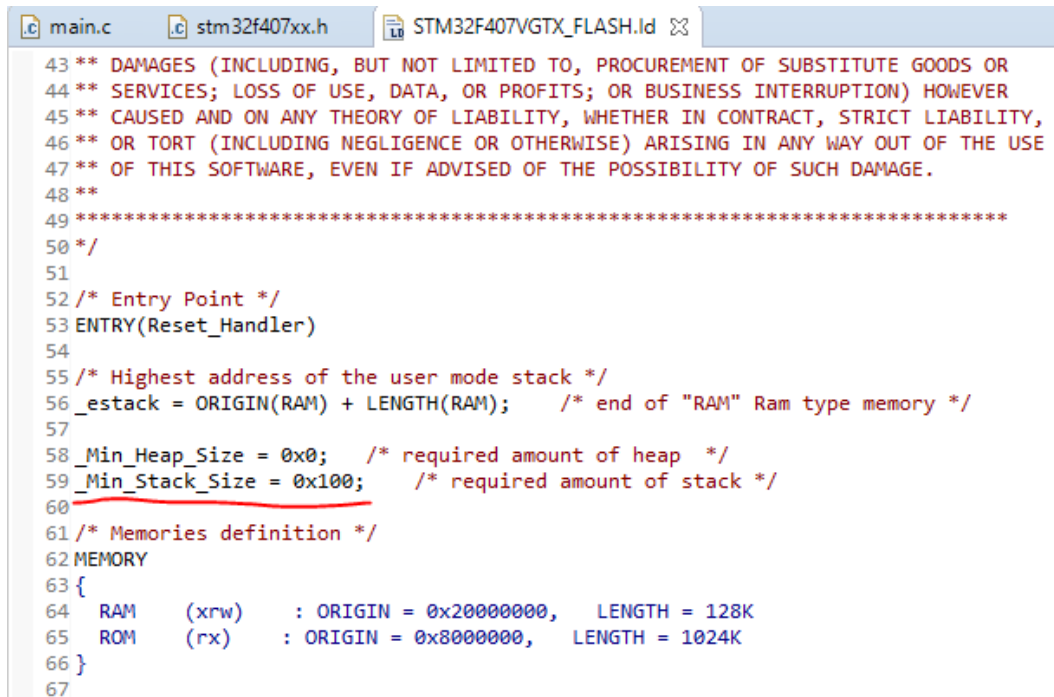
```
stm32f4xx.h
67 */
68 #if !defined (STM32F4)
69 #define STM32F4
70 #endif /* STM32F4 */
71
72 /* Uncomment the line below according to the target STM32 device used in your
73 application
74 */
75 #if defined (STM32F405xx) && defined (STM32F415xx) && defined (STM32F407xx) && defined (STM32F417xx) && \
76 !defined (STM32F427xx) && !defined (STM32F437xx) && !defined (STM32F429xx) && !defined (STM32F439xx) && \
77 !defined (STM32F401xC) && !defined (STM32F401xE) && !defined (STM32F410Tx) && !defined (STM32F410Cx) && \
78 !defined (STM32F410Rx) && !defined (STM32F411xE) && !defined (STM32F446xx) && !defined (STM32F469xx) && \
79 !defined (STM32F479xx) && !defined (STM32F412Cx) && !defined (STM32F412Rx) && !defined (STM32F412Vx) && \
80 !defined (STM32F412Zx) && !defined (STM32F413xx) && !defined (STM32F423xx)
81 #define STM32F405xx /*!< STM32F405RG, STM32F405VG and STM32F405ZG Devices */
82 #define STM32F415xx /*!< STM32F415RG, STM32F415VG and STM32F415ZG Devices */
83 #define STM32F407xx /*!< STM32F407VG, STM32F407VE, STM32F407ZG, STM32F407IE and STM32F407IE Devices */
84 #define STM32F417xx /*!< STM32F417VG, STM32F417VE, STM32F417ZG, STM32F417ZE, STM32F417IG and STM32F417IE Devices */
85 #define STM32F427xx /*!< STM32F427VG, STM32F427VI, STM32F427ZG, STM32F427ZI, STM32F427IG and STM32F427II Devices */
86 #define STM32F437xx /*!< STM32F437VG, STM32F437VI, STM32F437ZG, STM32F437ZI, STM32F437IG and STM32F437II Devices */
87 #define STM32F429xx /*!< STM32F429VG, STM32F429VI, STM32F429ZG, STM32F429ZI, STM32F429BG, STM32F429BG, STM32F429NG,
88 STM32F439NI, STM32F429IG and STM32F429II Devices */
89 #define STM32F439xx /*!< STM32F439VG, STM32F439VI, STM32F439ZG, STM32F439ZI, STM32F439BG, STM32F439BI, STM32F439NG,
90 STM32F439NI, STM32F439IG and STM32F439II Devices */
91 #define STM32F401xC /*!< STM32F401CB, STM32F401CC, STM32F401RB, STM32F401RC, STM32F401VB and STM32F401VC Devices */
92 #define STM32F401xE /*!< STM32F401CD, STM32F401RD, STM32F401VD, STM32F401CE, STM32F401RE and STM32F401VE Devices */
93 #define STM32F410Tx /*!< STM32F410TB and STM32F410TB Devices */
94 #define STM32F410Cx /*!< STM32F410CB and STM32F410CB Devices */
95 #define STM32F410Rx /*!< STM32F410RB and STM32F410RB Devices */
96 #define STM32F411xE /*!< STM32F411CC, STM32F411RC, STM32F411VC, STM32F411CE, STM32F411RE and STM32F411VE Devices */
97 #define STM32F446xx /*!< STM32F446NC, STM32F446NE, STM32F446RC, STM32F446RE, STM32F446VC, STM32F446VE, STM32F446ZC,
98 and STM32F446ZE Devices */
99 #define STM32F469xx /*!< STM32F469AI, STM32F469II, STM32F469BI, STM32F469BG, STM32F469IG, STM32F469BG,
100 STM32F469NG, STM32F469AE, STM32F469IE, STM32F469BE and STM32F469NE Devices */
101 #define STM32F479xx /*!< STM32F479AI, STM32F479II, STM32F479BI, STM32F479BG, STM32F479NI, STM32F479AG, STM32F479IG, STM32F479BG
102 and STM32F479NG Devices */
103 #define STM32F412Cx /*!< STM32F412CEU and STM32F412CGU Devices */
104 #define STM32F412Zx /*!< STM32F412ZET, STM32F412ZGT, STM32F412ZE and STM32F412ZG Devices */
105 #define STM32F412Vx /*!< STM32F412VET, STM32F412VGT, STM32F412VEH and STM32F412VGH Devices */
106 #define STM32F412Rx /*!< STM32F412RET, STM32F412RGT, STM32F412REY and STM32F412RGY Devices */
107 #define STM32F413xx /*!< STM32F413CH, STM32F413RH, STM32F413VH, STM32F413ZH, STM32F413CG, STM32F413MG,
108 STM32F413RG, STM32F413VG and STM32F413ZG Devices */
109 #define STM32F423xx /*!< STM32F423CH, STM32F423RH, STM32F423VH and STM32F423ZH Devices */
110 #endif
```

Рисунок 16 – Листинг файла stm32f4xx.h

```
main.c
1 #include "stm32f4xx.h"
2 #include "main.h"
3
4 volatile uint32_t delayTimerValue = 0;
5
6
7 // SysTick interrupt handle
8 void SysTick_Handler(void)
9 {
10     delayTimerValue--;
11 }
12
13
14 // set delay in milliseconds using sysTick timer
15 void delayMs(uint32_t delay)
16 {
17     delayTimerValue = delay;
18     while(delayTimerValue);
19 }
20
21
22
23
24 int main(void)
25 {
26     RCC -> AHB1ENR |= (1<<21);
27     GPIOD -> MODER |= 0x55550000;
28     while(1) {
29         GPIOD -> ODR |= 0x0000FF00;
30         delayMs(1000);
31
32         GPIOD -> ODR &= ~(0x0000FF00);
33         delayMs(1000);
34     }
35 }
36
```

Рисунок 17– Листинг файла main.c

Безошибочная компиляция и сразу же куда-то подевалось целых полтора килобайта памяти ОЗУ она же RAM. Для исправления ситуации необходимо исправить величину стека и кучи указанной в файле скрипта линкера, что с расширением **.ld**. Эти значения находятся в начале файла в виде меток **_Min_Heap_Size/_Min_Stack_Size** с указанием размера в шестнадцатеричном виде (Рисунок 18).



```
43 ** DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
44 ** SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
45 ** CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
46 ** OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
47 ** OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
48 **
49 ****
50 */
51
52 /* Entry Point */
53 ENTRY(Reset_Handler)
54
55 /* Highest address of the user mode stack */
56 _estack = ORIGIN(RAM) + LENGTH(RAM); /* end of "RAM" Ram type memory */
57
58 _Min_Heap_Size = 0x0; /* required amount of heap */
59 _Min_Stack_Size = 0x100; /* required amount of stack */
60 
61 /* Memories definition */
62 MEMORY
63 {
64   RAM    (xrw)  : ORIGIN = 0x20000000, LENGTH = 128K
65   ROM    (rx)   : ORIGIN = 0x80000000, LENGTH = 1024K
66 }
67
```

Рисунок 18– Листинг скрипта линкера