УДК 004.4′24
DOI: 10.22213/2410-9304-2021-4-98-110

Виртуальная лаборатория для изучения архитектуры и программирования микроконтроллеров STM32

К. С. Кузнецов, ИжГТУ имени М. Т. Калашникова, Ижевск, Россия
В. Г. Тарасов, кандидат технических наук, профессор, ИжГТУ имени М. Т. Калашникова, Ижевск, Россия

В настоящее время проектирование платформ на микроконтроллерах является очень перспективным направлением из-за их удешевления и увеличения производительности микросхем. Микроконтроллеры устанавливают в промышленное оборудование, в смартфоны и аудиоплееры, в видеотехнику и многое другое.

В статье приводятся теоретические основы для работы с микроконтроллерами серии STM32, их назначение, системная архитектура, программная модель, а также особенности ядра ARM Cortex M3. Данное 32-битное ядро имеет много преимуществ, но основное из них – универсальность. Описывается структура программно-технического комплекса для изучения и программирования микроконтроллеров, размещенных на удаленном сервере, с применением режима пошаговой отладки. Рассмотрены особенности написания программ для микроконтроллеров серии STM32.

Виртуальная лаборатория является платформой для наглядного изучения микроконтроллеров и языка программирования Ассемблер. Система позволяет программировать, прошивать и отлаживать микроконтроллеры без их приобретения. Пользователь может тестировать свой код на реальном устройстве и в реальном времени просматривать области памяти, регистры и другую отладочную информацию, а также с помощью видеотрансляции наблюдать за всеми изменениями в периферийных устройствах. Веб-интерфейс GDB GUI позволяет не устанавливать дополнительные приложения, а отлаживать программу прямо в браузере. К системе могут быть подключены всевозможные датчики и индикаторы, с которыми пользователь может работать. Приведены примеры программ, написанных на языке программирования Ассемблер, для демонстрации работы виртуальной лаборатории. Предложенную платформу можно интегрировать в обучающий курс по изучению языка программирования Ассемблер.

Ключевые слова: микроконтроллер, архитектура, программирование, ассемблер, виртуальная лаборатория, отладка, удаленное управление.

Введение

Микроконтроллер STM32 – семейство 32-битных микроконтроллеров производства STMicroelectronics. Использования всего потенциала микроконтроллера можно достичь только с помощью низкоуровневого языка программирования Ассемблер.

Ассемблер – это низкоуровневый язык программирования: код гораздо теснее связан с принципами работы компьютера, нежели с принципами мышления человека.

Ассемблерный код напрямую связан с машинным кодом процессора: каждой команде соответствует определенная последовательность байт и ничего лишнего не добавляется, в отличие от высокоуровневых языков программирования. Например, JavaScript и Python используют универсальный набор команд, который транслируется в громоздкий машинный код. Из-за необходимости дополнительных проверок в нем появляются дополнительные команды. При написании кода на Ассемблере программист заранее продумывает все ограничения и не пишет ничего лишнего. Поэтому код на Ассемблере выполняется быстрее, чем на других языках программирования.

Кроме того, машинный код, созданный Ассемблером, будет занимать мало места в памяти. Это может быть критично на устройствах, память которых сильно ограничена. Удаленное программирование и отладка микроконтроллеров позволяет более наглядно изучать язык программирования Ассемблер, а также является хорошим инструментом для взаимодействия с кодом.

Микроконтроллер принимает и передает информацию внутрисхемному отладчику через доступные интерфейсы связи. Внутрисхемный отладчик через интерфейс связи с компьютером направляет информацию в программное обеспечение, где поступившие сведения обрабатываются и отображаются в удобном для пользователя виде.

Современная линейка ARM-процессоров под названием Cortex делится на три группы:

- Cortex-A – это полноценные процессоры общего назначения для самых различных задач, создания приложений, требующих высокой производительности; чаще всего на них запускается linux, android и им подобные ОС;
- Cortex-R – предназначены для систем реального времени, где существует необходимость в быстрой и точной реакции на внешние

события с гарантированным временем отклика – для применений в промышленности, медицине, автомобилестроении и пр.;

• Cortex-M – для встраиваемых систем. Как обычно, это не очень быстрый процессор, но со встроенной памятью для программ, оперативной памятью и различной периферией – такой, как General-purpose input/output (GPIO), Universal Asynchronous Receiver-Transmitter (UART), Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I2C) и т. д.

Cortex-M обладает хорошей производительностью, низким энергопотреблением и относительно низкой ценой, именно поэтому в статье будет рассмотрена работа с этим процессором.

Существует два варианта запуска тестов для встраиваемых систем.

1. Запуск на симуляторе.

Преимущества: нет обязательных требований наличия самого устройства, возможность симуляции всего списка необходимой периферии.

Недостатки: ограниченная точность симуляции работы микроконтроллера и окружающей среды, трудность создания и конфигурации такого симулятора.

2. Запуск на самом микроконтроллере.

Преимущества: можно работать с периферией микроконтроллера.

Недостатки: обязательный доступ к самому устройству.

Для симуляции работы микроконтроллеров серии AVR существуют несколько программных средств: Atmel Studio, Simulavr и simavr [1]. Для симуляции 32-битных STM32 можно использовать программу Proteus. Моделирование аналого-цифровых преобразований – яркий пример ограничений симуляции, так как вычисления происходят со значительной погрешностью, и симуляция не может учитывать внешние факторы.

В статье рассматривается удаленный вариант запуска тестов на реальном микроконтроллере. В таком случае пользователь получает доступ к реальному устройству, расположенному на удаленном сервере. Этот подход обеспечивает максимальную точность и близость к реальным условиям без наличия самого устройства.

Существуют также отечественные микроконтроллеры на ядре Cortex M3 – 1986ВЕ9х [2] от компании АО «ПКК «Миландр». Между STM32 и российским аналогом имеется несколько несущественных отличий. В микроконтроллерах от STMicroelectronics для освобождения процессорного времени используют линии коммуникации между периферийными блоками, а в отечественном процессоре периферийные устройства связаны только с ядром. Российский микроконтроллер можно применять с некоторыми доработками там же, где успешно работал STM32. Широкому применению отечественных аналогов препятствует ряд ограничений: их малое количество на рынке и высокая стоимость.

Предложенная платформа интегрируется в учебные курсы Института «Информатика и вычислительная техника» по изучению языка программирования Ассемблер, проектированию компонентов системного программного обеспечения (ассемблеров, компоновщиков, загрузчиков) и дополнительно используется для макетной проработки в научно-исследовательских проектах.

Микроконтроллеры STM32

В данном разделе речь идет о структуре и особенностях микроконтроллеров STM32, что необходимо для дальнейшей работы с микроконтроллерами.

Назначение

На сегодняшний день большую известность среди разработчиков электронной аппаратуры разного назначения получили микроконтроллеры компании STMicroelectronics. Это связано с тем, что данные микроконтроллеры имеют ряд преимуществ перед существующими аналогами [3].

Семейство микроконтроллеров STM32, основанное на ядре ARM Cortex M3 [4], обеспечивает основу для создания широкого диапазона встраиваемых систем: от простых ключей с батарейным питанием до сложных систем реального времени, таких как автопилоты вертолетов. Это семейство компонентов включает в себя десятки различных конфигураций, обеспечивающих широкий выбор объемов памяти, доступных периферийных устройств, производительности и мощности.

Микроконтроллеры предназначены для сбора информации с датчиков и управления различными электронными устройствами: индикаторы, экраны, двигатели. Микроконтроллеры, как правило, не работают в одиночку, а собираются в схему, содержащую и другие периферийные устройства.

ARM Cortex

Семейство ARM Cortex – это новое поколение процессоров со стандартизованной архитектурой для решения широкого круга технологических задач. В отличие от других ядер ARM, семейство Cortex представляет собой законченное процессорное ядро со стандартным центральным процессорным устройством (ЦПУ) и системной архитектурой.

В STM32 применяется профиль Cortex-M3, разработанный специально для систем, сочетающих в себе высокую производительность

и низкое энергопотребление [5]. Практически все микроконтроллеры с ядром ARM Cortex-M3 могут функционировать на частотах более 50 МГц. STM32 имеет тактовую частоту 72 МГц, от 256 до 512 кбайт встроенной памяти (Flash), до 64 кбайт оперативной памяти (SRAM), а также некоторые модели обладают поддержкой параллельного интерфейса для подключения внешнего экрана. Микроконтроллеры имеют до 13 коммуникационных интерфейсов, в том числе Universal Serial Bus (USB) и Controller Area Network (CAN).

Cortex-M3 – это стандартизованное микроконтроллерное ядро, выходящее за пределы ЦПУ, представляющее собой законченное «сердце» микроконтроллера (включает систему прерываний, системный таймер, систему отладки и карту памяти). 4 Гбайта адресного пространства Cortex M3 разбито на конкретные области для кода приложения, SRAM, периферийных и системных устройств.

Программная модель

ARM Cortex является RISC (компьютер с набором коротких команд) процессором с выделенным доступом к памяти [6]. За счет сокращения набора команд производительность значительно увеличивается. Также она называется Load/Store архитектурой. Изменять значения в основной памяти, в отличие от архитектуры регистр/память, здесь нельзя. Для обработки данных в памяти необходимо загрузить (load) их в регистр процессора, провести с ними необходимые операции и поместить (store) их обратно в память.

Процессор содержит 13 32-битных регистром общего назначения R0-R12. Они являются абсолютно равноправными. Следующие три регистра имеют особое назначение. Регистр R13 является указателем стека SP. В Cortex-M есть два стека MSP (указатель основного стека) и PSP (указатель стека процессора). Активный стек всегда один R13, но он может быть как MSP, так и PSP. Это позволяет реализовать многозадачную операционную систему. Например, RTOS (операционная система реального времени) может выполнять системный код отдельно от основного и в защищенном режиме. Регистр связи LR (R14) хранит адрес для возврата из подпрограммы. При выполнении команды BL (переход на указанный адрес) в регистр LR записывается адрес этой команды. Если уровень вложенных подпрограмм большой, то адреса возвратов автоматически сохраняются в стек. Регистр R15 – счетчик команд PC. Он не может быть использован как регистр общего назначения. При считывании этого регистра его значение будет на 8 больше адреса текущей команды процессора из-за особенностей работы конвейера. Также существуют и регистры состояния, которые содержат информацию о состоянии процессора и его флагах. Полный список регистров представлен в таблице.

Регистры микроконтроллера
Microcontroller registers

Регистр	Название	Назначение
Регистры общего назначения	R0, R1, R2...R12	Как заранее созданные переменные для ассемблера
Указатель стека - Stack Pointer	R13 (SP)	Хранит адрес вершины стека
Регистр связи – Link Register	R14 (LR)	Хранит адрес возврата при вызове функции
Счетчик команд – Program Counter	R15 (PC)	Хранит адрес текущей команды
Регистры состояния	PSR, ASPR, IPSR, EPSR, PRIMASK, FAULTMASK, BASEPRI, CONTROL	Каждый бит в этих регистрах указывает на какое-нибудь событие (произошло или нет)

Отладочная система CoreSight

Все ЦПУ ARM имеют свою внутрисхемную отладочную систему. При помощи этих отладочных средств можно подключаться к ЦПУ и загружать программный код во встроенное оперативное запоминающее устройство (ОЗУ) или Flash-память, использовать функции отладки (пошаговое исполнение программы, постановка точек останова и т. д.), а также просматривать содержимое областей памяти. Но в ядре Cortex представлена новая отладочная система, называемая CoreSight.

CoreSight выполняет функции трассировки и управления выполнением программы. Преимущество этой системы в том, что она может оставаться работающей во время пребывания STM32 в спящем режиме. Система сделала большой шаг в развитии отладочных систем.

Трассировщик просмотра данных позволяет видеть содержимое областей памяти, не изменяя режима работы ЦПУ. Отладочная система CoreSight может оставаться активной во время пребывания ядра Cortex в режиме низкого энергопотребления или спящем режиме. Это имеет огромное значение при отладке низко потребляющего приложения. Кроме того, система может останавливать таймеры STM32 в моменты, когда ЦПУ находится в состоянии ожидания. Это позволяет осуществлять пошаговое исполнение программы и оставлять таймеры синхронными с выполнением команд. Отладочная система значительно улучшает реально-временные характеристики отладки STM32 по сравнению с предыдущими версиями, не увеличивая его стоимость.

Системная архитектура STM32

Ядро Cortex соединяется с Flash-памятью через шину команд (I-bus). Шина данных (D-bus) и системная шина (System) соединяются с матрицей высокоскоростных шин ARM Advanced High Speed Buses (AHB). Внутреннее статическое ОЗУ подключается к матрице шин AHB, как и модуль прямого доступа к памяти (ПДП). Периферийные устройства располагаются на двух шинах ARM Advanced Peripheral Buses (APB). Каждая из APB-шин замыкается на матрицу шин AHB. Матрица шин тактируется с той же частотой, что и ядро. Однако шины AHB имеют отдельные делители частоты и могут тактироваться со сниженной частотой для снижения энергопотребления. Полная схема STM32 изображена на рис. 1.

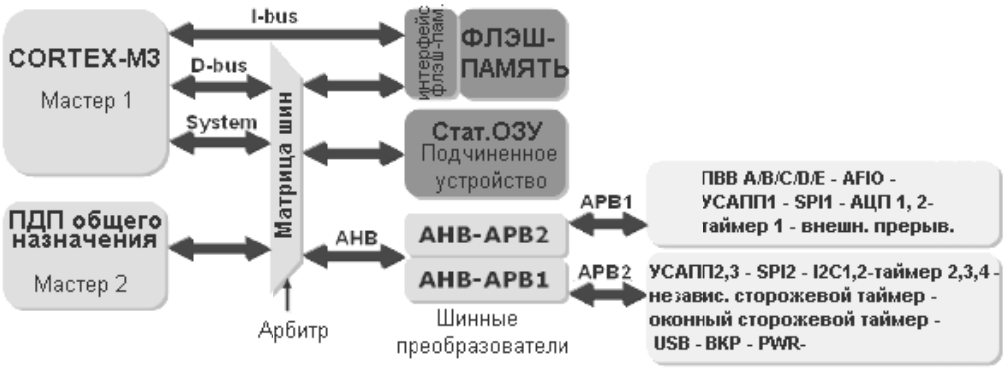

Рис. 1. Системная архитектура
Fig. 1. System architecture

Периферийные устройства

STM32 оснащен портами ввода-вывода общего назначения GPIO и может иметь до 80 двунаправленных выводов [8, 9]. Они сгруппированы на пять портов по 16 линий ввода-вывода на каждый.

Порты обозначаются символами латинского алфавита от A до E. Многие из внешних выводов микроконтроллера вместо выполнения функций ввода-вывода могут использоваться для обслуживания пользовательских периферийных устройств. Кроме этого, модуль внешних прерываний позволяет распределить 16 линий между портами ввода-вывода.

Каждый порт GPIO содержит два 32-битных конфигурационных регистра. Два регистра объединяются для получения 64-битного регистра. Среди этих 64 бит для каждого порта ввода-вывода выделяется четыре бита, задающие его характеристики. Четырехбитовое поле состоит из двухбитового поля выбора режима и двухбитового конфигурационного поля. Поле выбора режима позволяет пользователю определять вывод как вход или выход, в то время как конфигурационное поле задает управляющую характеристику вывода.

Карта памяти микроконтроллера

Поскольку микроконтроллеры STM32 созданы на основе ARM Cortex, то у них используется стандартное распределение памяти. Flash-память начинается с адреса 0x08000000. В ней записывается сама программа. Встроенное статическое ОЗУ начинается с адреса 0x20000000. Все ячейки статического ОЗУ расположены в области хранения бит. Регистры периферийных устройств (УВВ на рис. 2) представлены в карте памяти, начиная с адреса 0x40000000. Наконец, регистры Cortex находятся в их стандартном месте, начиная с адреса 0xE0000000.

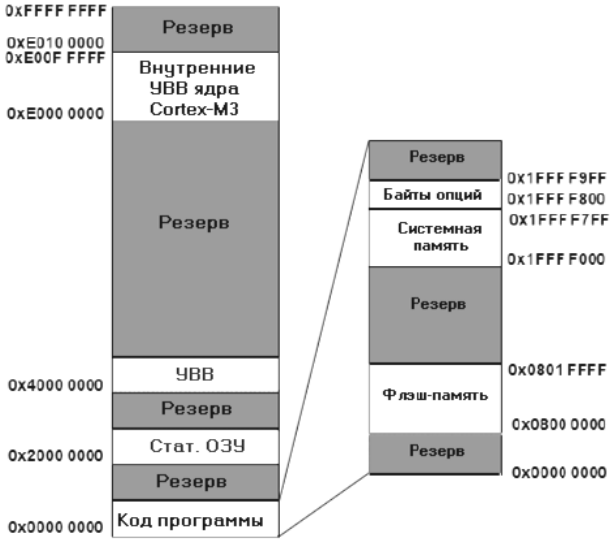

Рис. 2. Распределение памяти
Fig. 2. Memory allocation

Программное обеспечение микроконтроллера STM32

Отладочные средства

Становление ARM как популярных ядер стандартных микроконтроллеров послужило причиной массового производства отладочных средств для них. Все ведущие производители компиляторов, такие как GCC, Keil, IAR и Tasking, предлагают также компиляторы и для ARM-ядер. С появлением процессоров Cortex все эти средства отладки получили расширение для поддержки системы команд Thumb-2. Набор инструкций Thumb-2 является смесью 16- и 32-битных инструкций. Инструкции Thumb-2 дают улучшение плотности кода на 26 % по сравнению с 32-битными инструкциями ARM и производительности на 25 % по сравнению с 16-битными инструкциями Thumb.

Существует два компилятора для систем Cortex. Первый – GCC-компилятор [10, 11] – открытое средство разработки, свободно доступное для скачивания и использования. Этот компилятор интегрирован во многие коммерческие среды разработки и отладчики в составе недорогих отладочных и оценочных наборов.

Второй – коммерческий компилятор ARM Real View [14] – оригинальный и проработанный Си-компилятор, разработанный компанией ARM для своих ядер. Компилятор RealView поставляется в отладочном наборе ARM RealView.

Языки программирования

Ассемблер является языком самого низкого уровня. При этом он наиболее полно раскрывает все возможности микроконтроллеров и позволяет получать максимальное быстродействие и компактный код.

Программа на языке С и С++ лучше понятна человеку. Достоинством этих языков является огромное число программных средств и библиотек, позволяющих просто создавать необходимый код [12, 13].

Процесс подготовки программы и ее загрузка в микроконтроллер

В работе используется микроконтроллер STM32F103C8T6, язык программирования Ассемблер, а также GNU-компилятор.

При включении микроконтроллер:

1. Значение из памяти по адресу 0x08000000 загружает в регистр указателя стека SP. Обычно стек начинается с конца доступной ОЗУ и при заполнении движется от старших адресов к младшим.

2. Значение из памяти по адресу 0x0800 0004 загружает в регистр PC.

3. Осуществляется исполнение программы по адресу, загруженному в PC.

Рассмотрим процесс подготовки и запуска программы в микроконтроллере на примере.

Файл `main.s`

```
.syntax unified                    @ директива выбора варианта синтаксиса ас-
семблера
.thumb                             @ тип используемых инструкций Thumb
.cpu cortex-m3                     @ семейство микроконтроллера

.equ StackPointer, 0x20008000

.section .text                     @ секция кода

.word StackPointer                 @ указатель на вершину стека
.word Reset + 1                    @ указатель на программу

Reset:
      add r0, 1                    @ добавить 1 к регистру R0
B Reset                        @ бесконечный цикл
```

Данный код на ассемблере представлен в качестве примера. Первые 3 строчки программы являются директивами ассемблера и необходимы для дальнейшей компиляции этого файла. С помощью директив `.word` в память записываются указатель на стек и адрес начала программы. Далее в бесконечном цикле к регистру R0 добавляется единица. С помощью этой простой программы можно продемонстрировать пошаговую отладку микроконтроллера.

После создания ассемблерного файла нужно его скомпилировать. GCC-компилятор ассемблера имеет название `arm-none-eabi-as.exe`. Команда для компиляции файла `main.s` и получения объектного файла

```
arm-none-eabi-as.exe -g -o main.o main.s
```

Аргументы:

-g – таблица символов для дебага;

-o – объектный файл результат, `main.s` – файл-исходник

Также можно создать листинг из полученного объектного файла. Это можно сделать с помощью команды.

```
arm-none-eabi-objdump -d main.o >main.o.lst
```

Файл `main.o.lst`

```
00000000 <Reset-0x8>:
   0: 20008000    .word 0x20008000
   4: 00000009    .word 0x00000009

00000008 <Reset>:
   8: f100 0001   add.w r0, r0, #1
   c: e7fc        b.n   8 <Reset>
```

Первый столбик листинга показывает адрес команды, начиная с 0. Адреса сдвигаются исходя из того, сколько каждая команда занимает места в памяти. Второй столбик показывает машинный код. Третий столбик показывает исходную ассемблерную команду с аргументами. Например, последняя строчка листинг-файла говорит о том, что команда B Reset будет записана со смещением 0xC и состоит из 2 байт 0xE7 и 0xFC. Иногда трудно найти информацию о машинных кодах для определенной команды. Этот файл удобно использовать при создании собственного транслятора, с помощью этого файла можно смотреть, какой машинный код генерируется из соответствующей ассемблерной команды. В дальнейшем можно сравнить код, загруженный в микроконтроллер, с машинным кодом из листинга.

Далее из файла `main.o` при помощи программы-линковщика нужно сделать файл прошивки. Для программы-линковщика необходим файл настройки, в котором и будет описано, какую часть нашей программы и куда нужно поместить.

Линковщик GCC – это утилита `arm-none-eabi-ld.exe`. Используем ключ `-T` для указания файла карты памяти, и ключ `-o` для задания имени файла-прошивки, это будет файл с расширением `.elf`.

```
arm-none-eabi-ld.exe -o main.elf -T stm32f103.ld main.o
```

Файл `main.elf` можно загружать в микроконтроллер. Файл содержит сам код, а также отладочную информацию. ELF – это сокращение от Executable and Linkable Format (формат исполняемых и связываемых файлов) и определяет структуру бинарных файлов, библиотек и файлов ядра.

Для прошивки микроконтроллера STM32 использована утилита OpenOCD [15] – это популярная система отладки для микроконтроллеров. Для загрузки кода в память микроконтроллера применяется команда

```
openocd -f interface/stlink-v2.cfg -f target/stm32f1x.cfg -c "program main.elf verify reset exit"
```

Аргументы:

-f interface – файл конфигурации адаптера;

-f target – файл конфигурации микроконтроллера;

-c “program main.elf verify reset exit” – загрузка main.elf в память микроконтроллера, верификация данных, перезагрузка и выход из программы.

Запуск программы и удаленная отладка микроконтроллера

Для прошивки и отладки микроконтроллера необходим также программатор. ST-LINK/V2 – внутрисхемный программатор/отладчик для микроконтроллеров серии STM8 и STM32 производства фирмы STMicroelectronics. Отладчик подключается к отладочным платам посредством стандартного JTAG/SWD-интерфейса (микроконтроллеры на базе ядра STM32).

STMicroelectronics предоставляет несколько утилит для визуальной отладки микроконтроллеров. Одна из них утилита ST-LINK (рис. 3) – программа для прошивки, просмотра памяти, регистров STM32.

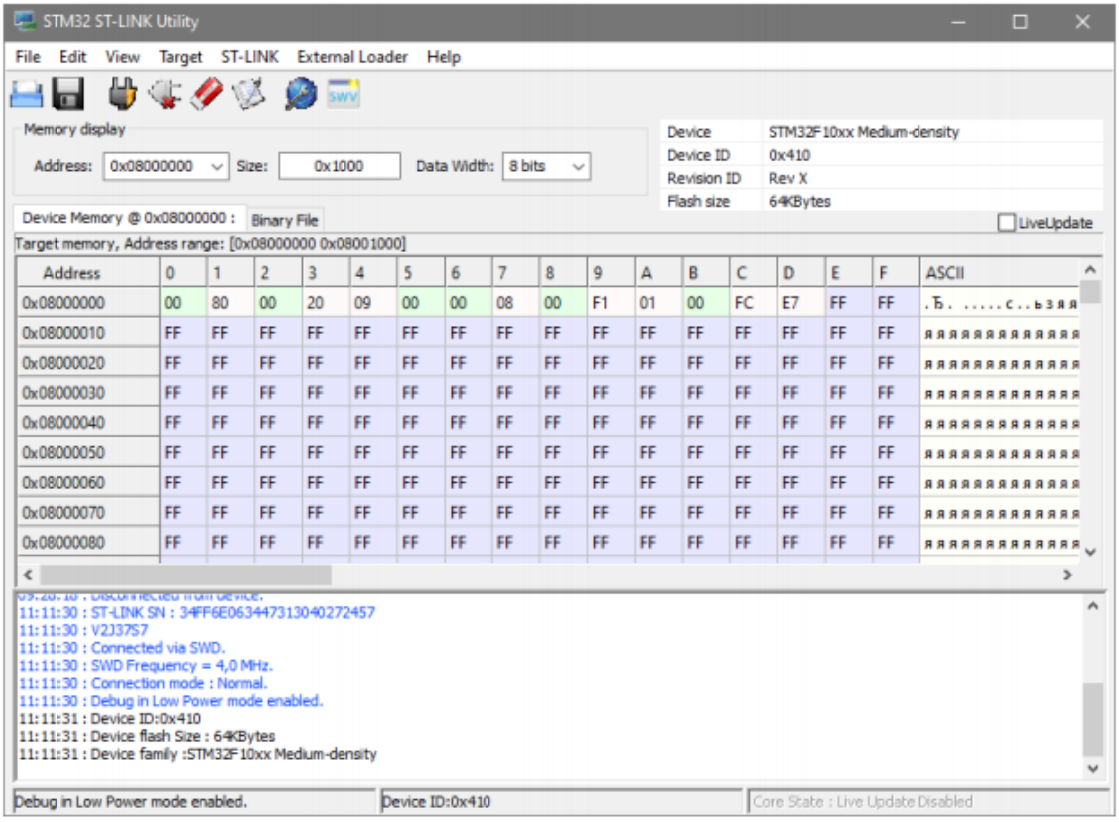

Рис. 3. Содержание ячеек памяти, отображаемое утилитой ST-LINK
Fig. 3. Contents of memory cells displayed by the ST-LINK utility

В главном окне программы мы можем выбрать, с какого адреса и какой размер памяти нужно считать из микроконтроллера. После загрузки программы в память микроконтроллера можно заметить, что по адресу 0x08000000 идет конец стека, затем указатель начала программы и сами инструкции.

Со скриншота видно, что команды записаны в обратном порядке, от младших байтов к старшим. Этот порядок записи байтов называется little-endian, или интеловский порядок. При работе с многобайтовыми числами наибольшую ценность представляют младшие байты, например, в операциях вычитания и сложения вычисление начинается с самого младшего байта, так как может происходить перенос. ARM Cortex M3 может обращаться к байтам в памяти в формате с прямым порядком или обратным. Процессор содержит вывод конфигурации BIGEND, который позволяет выбрать, какой порядок байт использовать. Но обычно производители микроконтроллеров выставляют little-endian-формат и не дают пользователям менять его в дальнейшем.

Первые 4 байта полностью совпадают с листингом – это указатель на вершину стека. Следующие 4 байта указывают на адрес самой программы в памяти. В листинге адрес записывался как 0x00000009, но уже в памяти самого микроконтроллера он записывается как 0x08000009. Это связано с работой компоновщика. Он преобразует смещение в реальный адрес. Дальнейшие команды совпадают с листингом, только записаны в обратном порядке.

Также с помощью этой программы можно провести отладку, контролируя регистры R0 и PC (рис. 4).

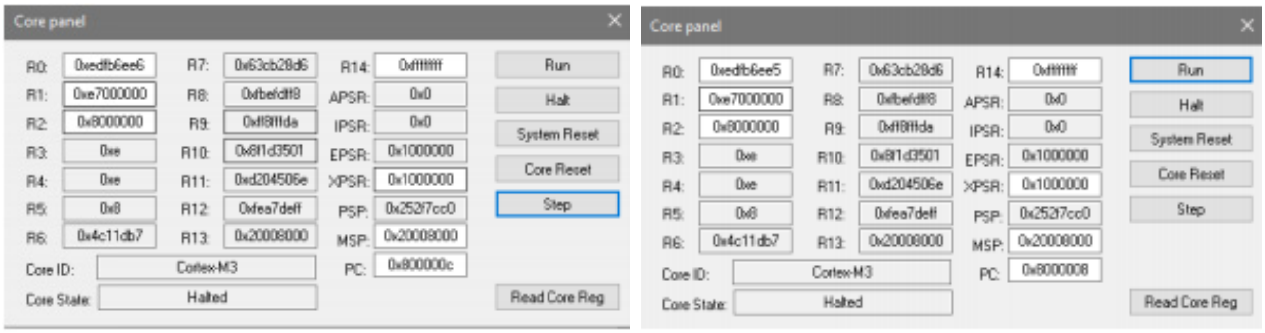

Рис. 4. Содержание регистров R0– R15, PC и управление режимами отладкой
Fig. 4. Contents of registers R0– R15, PC and control of debugging modes

Как видно из значений регистров, с каждый шагом регистр PC идет по бесконечному циклу и в каждой итерации этого цикла выполняется команда add r0, 1.

Структура виртуальной лаборатории

Главная цель этой работы – это создание программно-технического комплекса для изучения и работы микроконтроллера без его приобретения, а с размещением его на удаленном сервере. При отладке микроконтроллера разработчик может видеть отладочную информацию, которая работает на реальном физическом устройстве. Он может видеть изменения в регистрах, памяти, а вследствие этого и периферийных устройств микроконтроллера.

Изучение ассемблера станет гораздо нагляднее, если после выполнения одной ассемблерной команды на устройстве загорится светодиод, демонстрирующий, что бит в памяти микроконтроллера, отвечающий за состояние GPIO порта, изменился. Мигание светодиодом – это всего лишь пример, также возможно управление электромотором, выводом сообщения на LCD-экран и многое другое.

В этой части рассмотрим размещение микроконтроллера на удаленном сервере и создание веб-интерфейса, через который будут возможны компиляция, прошивка и отладка микроконтроллера STM32.

В роли удаленного сервера может выступать виртуальная машина (ВМ на рис. 5), размещенная на сервере кафедры «Программное обеспечение» ИжГТУ имени М. Т. Калашникова.

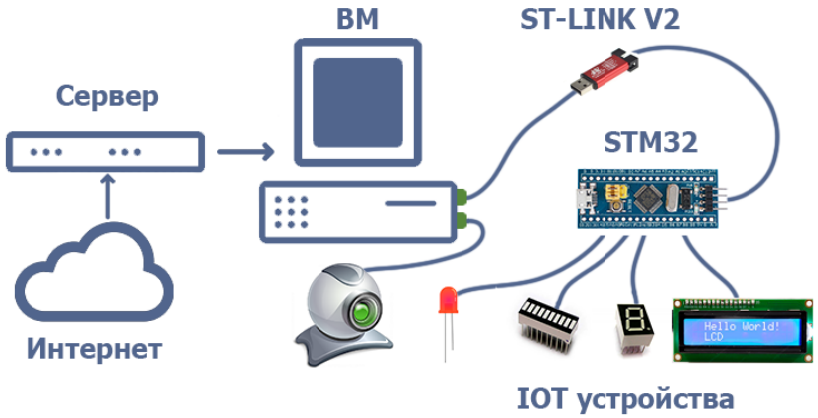

Рис. 5. Структурная схема микропроцессорной системы
Fig. 5. Block diagram of the microprocessor system

К нему подключен микроконтроллер STM32 через отладчик ST-LINK/V2. К микроконтроллеру может быть подключена дополнительная периферия, например, светодиоды, индикаторы, двигатели, датчики. Для наглядной отладки также необходима веб-камера, с помощью нее можно отслеживать изменения в периферийных устройствах. Необходимо установить компиля-

тор GCC-ARM, а также утилиту OpenOCD. Команды компиляции и прошивки остаются прежними. Структура изображена на рис. 5. В качестве IOT-устройств могут быть любые датчики, светодиоды, индикаторы, экраны, электрические двигатели. Все эти устройства необходимы для визуальной отладки.

Пользователь имеет доступ к виртуальной машине; с помощью терминала он может подготовить программу, загрузить ее в память микроконтроллера и начать процесс отладки. Также сервер в режиме реального времени транслирует видео с веб-камеры, направленной на периферию микроконтроллера.

Для начала отладки нужно запустить утилиту OpenOCD. Она запустит GDB-сервер.

```
openocd -f interface/stlink-v2.cfg -f target/stm32f1x.cfg -c "init; reset halt;"
```

Gdbgui [16] – браузерная оболочка для GDB-отладчика, позволяющая отлаживать программу в браузере.

Преимущества gdbgui состоят в следующем.

- Активно разрабатывается и совместим с последней версией GDB (7.12).
- Предназначен лишь для отладки программ – ничего лишнего.
- Дизайн разработан под влиянием отладчика Chrome.
- Написан на широко используемых языках (Python и JavaScript).
- Инструмент бесплатен, исходный код находится в свободном доступе.

Команда для запуска gdbgui-сервера:

```
gdbgui.exe main.elf --gdb-args='--eval-command="target remote localhost:3333"'
```

После этой команды можно открыть браузер на хосте. По адресу виртуальной машины на порте 5000 будет работать gdbgui (рис. 6).

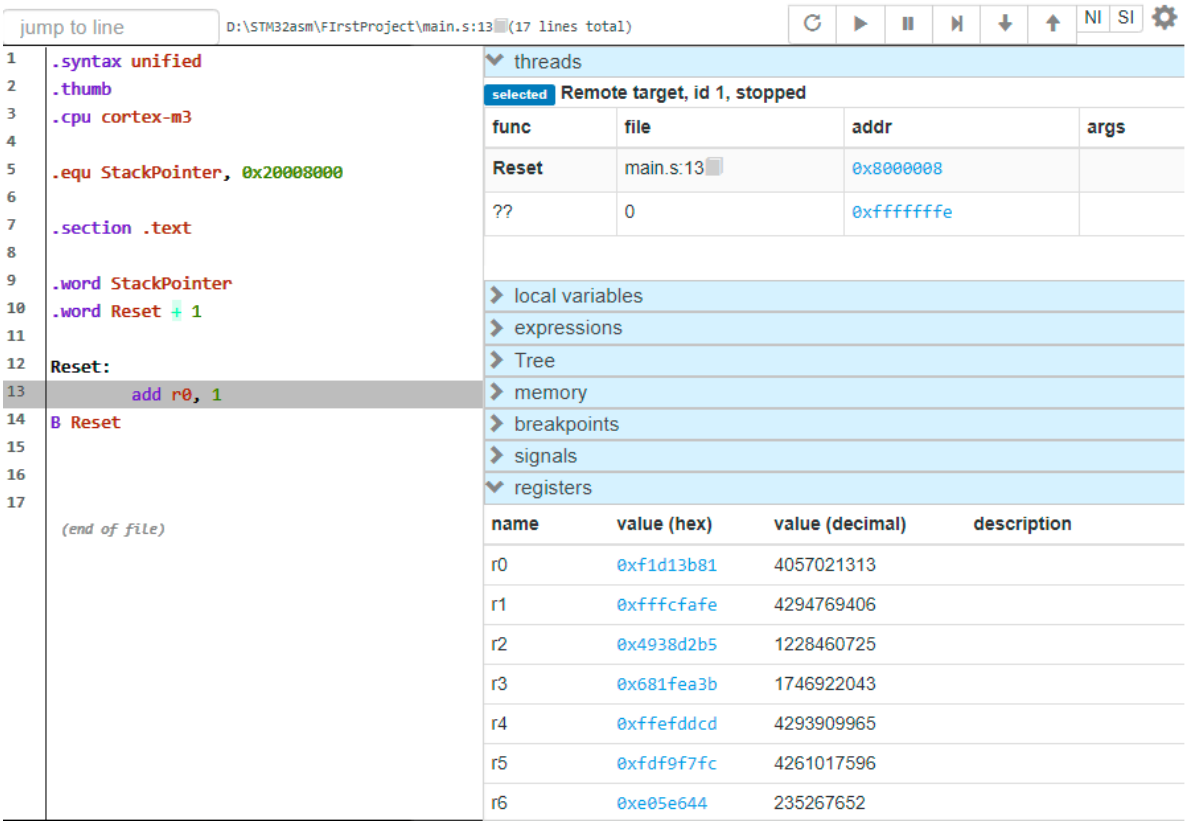

Рис. 6. Отладка микроконтроллера в браузере
Fig. 6. Debugging the microcontroller in the browser

Данная оболочка позволяет просматривать исходный код, регистры, участки памяти, устанавливать точки остановки и многое другое. Удобный интерфейс дает наглядное представление о работе программы.

Удаленная отладка. Управление светодиодом

Для того чтобы продемонстрировать визуальную отладку, необходимо задействовать периферию микроконтроллера. Например, можно использовать светодиод, расположенный на самой плате STM32, который находится на порту C и имеет порядковый номер 13. Для того чтобы включить тактирование портов ввода-вывода, нужно поставить соответствующие значения в RCC (reset and clock control) регистрах.

Все адреса регистров, а также все их возможные значения записываются в файл определения для каждого микроконтроллера STM32. В файле stm32f10x.inc находится адрес для регистра RCC – RCC_APB2ENR, а также значение, позволяющее включить тактирование порта C – RCC_APB2ENR_IOPCEN. Все, что остается сделать, это поместить значение RCC_APB2ENR_IOPCEN по адресу RCC_APB2ENR. Это можно сделать следующими ассемблерными командами:

```
mov r1, RCC_APB2ENR_IOPCEN
ldr r2, =RCC_APB2ENR
str r1, [r2]
```

Далее нужно настроить вывод OC13 на выход. Это делается через регистр GPIOC_CRH (control register high). За каждый пин отвечают 4 бита в этом регистре. В файле определения можно воспользоваться значением GPIO_CRH_MODE13, которое выставит режим работы ножки на выход. Для того чтобы выбрать нужный режим работы ножки, можно воспользоваться кодом выше, только поменять адрес и значение:

```
mov r1, GPIO_CRH_MODE13
ldr r2, =GPIOC_CRH
str r1, [r2]
```

Еще один регистр, который нам понадобится, – это регистр выходных данных GPIOC_ODR (output data register). При записи в регистр какого-либо значения, это значение устанавливается на выходных линиях соответствующего порта. Для того чтобы включить светодиод, мы должны установить соответствующий бит. Это можно сделать, использовав значение GPIO_ODR_ODR13. А для того чтобы выключить светодиод, можно просто сбросить данный бит, т. е. записать значение 0.

Включение и выключение светодиода должны сменять друг друга и выполняться в бесконечном цикле, но если это сделать без какой-либо задержки, то визуально будет нереально определить – мигает светодиод иди просто горит. Для решения этой проблемы необходимо добавить задержки между переключениями. Это можно сделать с помощью цикла на 1 миллион итераций, на который процессор потратит несколько миллисекунд.

```
Delay:
            ldr   r2, =0x00100000
Delay_loop:
            subs r2, r2, 1
            BNE   Delay_loop
```

Полный код программы представлен далее:

```
.syntax unified                     @ директива выбора варианта синтаксиса ас-
семблера
.thumb                              @ тип используемых инструкций Thumb
.cpu cortex-m3                      @ семейство микроконтроллера

.include "stm32f10x.inc"            @ файл определений для stm32f10x

.equ StackPointer, 0x20008000

.section .text

.word StackPointer                  @ указатель вершину стека
.word Reset + 1                     @ указатель на программу

Reset:
      @ Включаем тактирование для RCC
      mov r1, RCC_APB2ENR_IOPCEN
      ldr r2, =RCC_APB2ENR
      str r1, [r2]
```

```
      @ Настройка выводы PC13 на выход
      mov r1, GPIO_CRH_MODE13
      ldr r2, =GPIOC_CRH
      str r1, [r2]

Loop:
      @ GPIOC13 HIGH
      mov r1, GPIO_ODR_ODR13
      ldr r2, =GPIOC_ODR
      str r1, [r2]

      @ Задержка
      BL    Delay

      @ GPIOC13 LOW
      mov r1, 0
      ldr r2, =GPIOC_ODR
      str r1, [r2]

      @ Задержка
      BL    Delay

B Loop

Delay:
            ldr   r2, =0x00100000     @ повтор цикла задержки 0x0010 0000 раз.
Delay_loop:
            subs r2, r2, 1
            BNE   Delay_loop
            BX    LR
```

После компиляции этого кода и загрузки его в микроконтроллер можно сразу обнаружить, что светодиод на плате микроконтроллера начал мигать (рис. 7). Это говорит об успешной работе.

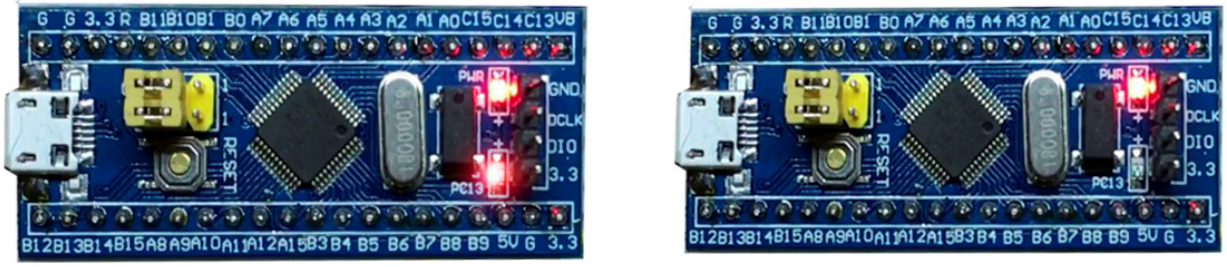

Рис. 7. Мигание светодиода
Fig. 7. Blinking LED

Также можно открыть gdbgui и с помощью режима пошаговой отладки отследить, на каком именно моменте светодиод загорается и гаснет.

Заключение

Описанный выше способ программирования и тестирования микроконтроллеров сочетает в себе плюсы и метода запуска программы на симуляторе, и метода запуска на самом микроконтроллере. Пользователю не нужно покупать микроконтроллер и всю периферию. Тесты, запущенные на удаленном сервере, выполняются в реальных условиях, и программист будет получать максимально правдоподобную информацию. Также виртуальная лаборатория является кросс-платформенной системой, так как вся отладка производится в браузере. Поэтому ее можно использовать на любом устройстве, на котором установлен браузер.

Представлены все необходимые инструменты компиляции, загрузки и отладки программы. Теоретическая база, описанная в статье, позволяет свободно использовать микроконтроллеры для решения всевозможных задач. Например, микроконтроллеры STM32 используются для

управления 3-осевыми станками с числовым программным управлением. 32-битный процессор позволяет быстро вычислить требуемое количество импульсов для шаговых двигателей, чтобы перемещать рабочий инструмент в нужную точку с минимальной погрешностью. Удаленное изучение и работа с микроконтроллерами без их приобретения позволяет заинтересовать больше людей для работы с ними. Не все могут позволить купить микроконтроллер и работать с ним, но с помощью предлагаемой виртуальной лаборатории порог вхождения существенно понизится.

Библиографические ссылки

1. *Голубцов М. С.* Микроконтроллеры AVR: от простого к сложному. М. : СОЛОН-Пресс, 2003. 286 с.
2. *Благодаров А. В.* Программирование микроконтроллеров семейства 1986ВЕ9х компании Миландр. М. : Горячая линия-Телеком, 2017. 232 с.
3. STM32: эпоха 32-битных микроконтроллеров наступила: сайт. URL: https://www.compel.ru (дата обращения: 09.06.2021). Текст: электронный.
4. Joseph Y. The Definitive Guide to the ARM® Cortex-M3 – Elsevier Inc, 2010. 531 p.
5. RM0008. Reference manual. STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and. STM32F107xx advanced ARM®-based 32-bit MCUs. STMicroelectronics 2015. 1137 p. Режим доступа: www.st.com/resource/en/reference_manual/ cd00171190.pdf (дата обращения: 09.06.2021).
6. Joseph Y. Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language (2nd Edition) – E-Man Press LLC, 2016. 545 p.
7. Trevor M. The Insider's Guide to the STM32 ARM Based Microcontroller. Hitex Ltd., 2009. 106 p.
8. Geoffrey B. Discovering the STM32 Microcontroller. Indiana University, 2019. 244 p.
9. Carmine N. Mastering the STM32 Microcontroller. Leanpub, 2016. 819 p.
10. Warren G. Beginning STM32: Developing with FreeRTOS, libopencm3 and GCC. Apress, 2018. 430 p.
11. *Артур Г.* GCC. Настольная книга пользователей, программистов и системных администраторов. СПб. : ТИД "ДС", 2004. 624 с.
12. Vincent M. Assembly Language Programming – ISTE Ltd and John Wiley & Sons, Inc, 2012. 258 p.
13. Majid P. Advanced Programming with STM32 Microcontrollers. Elektor, 2020. 216 p.
14. Richard M. Using the GNU Compiler Collection. GNU Press, 2003. 458 p.
15. RealView® Development Suite. Getting Started Guide. URL: https://developer.arm.com/ documentation/dui0255/b (дата обращения: 09.06.2021).
16. OpenOCD - Open On-Chip Debugger. Reference Manual. URL: http://openocd.org/doc/pdf/ openocd.pdf (дата обращения: 09.06.2021).
17. A browser-based frontend to gdb (gnu debugger). Documentation. URL: https://www.gdbgui.com (дата обращения: 09.06.2021).

References

1. Golubtsov M.S. *Mikrokontrollery AVR: ot prostogo k slozhnomu* [AVR microcontrollers: from simple to complex]. Moscow, Solon-press, 2003, 286 p. (in Russ.).
2. Blagodarov A.V. *Programmirovanie mikrokontrollerov semeistva 1986VE9kh kompanii Milandr* [Programming of microcontrollersof the 1986BE9x family of the Milandr company]. Moscow, Goryachaya liniya-Telekom, 2017, 232 p. (in Russ.).
3. RM0008. Reference manual. STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and. STM32F107xx advanced ARM® -based 32-bit MCUs. STMicroelectronics 2015. Available at: http://www.intuit.ru/studies/courses/4115/1230/info (accessed 28.06.2021).
4. STM32: epokha 32-bitnykh mikrokontrollerov nastupila [STM32: the era of 32-bit microcontrollers has arrived]. Available at: https://www.compel.ru (accessed 28.06.2021).
5. Trevor M. The Insider's Guide To The STM32 ARM Based Microcontroller. Shirley, Hitex, 2009. 106 p.
6. Geoffrey B. Discovering the STM32 Microcontroller. Bloomington, Indiana University, 2019. 244 p.
7. Vincent M. Assembly Language Programming. New Jersey, ISTE Ltd and John Wiley & Sons, Inc., 2012. 258 p.
8. Joseph Y. The Definitive Guide to the ARM® Cortex-M3. Amsterdam, Elsevier Inc., 2010. 531 p.
9. Joseph Y. Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language (2nd Edition). New York, E-Man Press LLC, 2016. 545 p.
10. Carmine N. Mastering the STM32 Microcontroller. Leanpub, 2016. 819 p.
11. Majid P. Advanced Programming with STM32 Microcontroller. Kansas City, Elektor, 2020. 216 p.
12. Warren G. Beginning STM32: Developing with FreeRTOS, libopencm3 and GCC. New York, Apress, 2018. 430 p.
13. Artur G. *GCC. Nastol'naya kniga pol'zovatelei, programmistov i sistemnykh administratorov* [GCC. Reference book of users, programmers and system administrators.]. Saint-Petersburg, TID "DS", 2004, 624 p. (in Russ.).
14. Richard M. Using the GNU Compiler Collection. Boston, GNU Press, 2003. 458 p.
15. RealView® Development Suite. Getting Started Guide. Available at: https://developer.arm.com/documentation/dui0255/b (accessed 28.06.2021).
16. OpenOCD - Open On-Chip Debugger. Reference Manual. Available at: http://openocd.org/doc/pdf/ openocd.pdf (accessed 28.06.2021).
17. A browser-based frontend to gdb (gnu debugger). Documentation. Available at: https://www.gdbgui.com (accessed 28.06.2021).

* * *

Virtual Laboratory for Studying and Programming Stm32 Microcontrollers

K. S. Kuznetsov, Student, Kalashnikov ISTU, Izhevsk, Russia
V. G. Tarasov, PhD in Engineering, Professor, Kalashnikov ISTU, Izhevsk, Russia

Currently, the design of platforms based on microcontrollers is a very promising direction due to their reduction in cost and increase in the performance of circuits. Microcontrollers are installed in industrial equipment, smartphones and audio players, video equipment, and much more.

The paper provides the theoretical foundations for working with microcontrollers of the STM32 series, their purpose, system architecture, software model, as well as the features of the ARM Cortex M3 core. 32-bit core has many advantages, but the main one is its versatility. The structure of a software and hardware complex for studying and programming microcontrollers located on a remote server using the step-by-step debugging mode is described. The paper also discusses the features of writing programs for microcontrollers of the STM32 series.

The virtual laboratory is a platform for the visual study of microcontrollers and the Assembler programming language. The system allows you to program, flash and debug microcontrollers without purchasing them. The user can test a code on a real device and view memory areas, registers and other debugging information in real time, as well as watch all changes in peripheral devices using video broadcast. The GDB GUI web interface allows you not to install additional applications, but to debug the program right in the browser. All kinds of sensors and indicators can be connected to the system, with which the user can work. Examples of programs written in the Assembler programming language are given to demonstrate the work of a virtual laboratory. The proposed platform can be integrated into a training course for learning the Assembler programming language.

Keywords: Microcontroller, architecture, programming, assembler, virtual laboratory, debugging, remote control.

Получено: 22.06.2021