Плагины Eclipse для ускорения разработки программ цифровых систем управления

А.В. Ескин, В.А. Жмудь, В.Г. Трубин
ФГБОУ ВПО НГТУ, Новосибирск, Россия

Аннотация: **Рассматриваются вопросы встраивания в интегрированную среду разработки (IDE) Eclipse механизма аппаратной отладки программ и упрощённой настройки периферии микроконтроллеров STM32.**[3]

***Ключевые слова*: Eclipse, Plugin, GDB отладка, SWD, JTAG, свободное ПО, ST-LINK, microXplorer, STM32VLDISCOVERY.**

ВВЕДЕНИЕ

Любая уважающая себя интегрированная среда разработки должна состоять из как минимум четырех компонентов: текстового редактора (с подсветкой синтаксиса), встроенной программы компилятора (с удобными настройками параметров), средствами сборки исполняемого файла и хорошего отладчика (с возможностью пошаговой отладки и установки точек останова). На современном уровне развития, интегрированные среды разработки перестали ограничиваться данным перечнем и стали развиваться дальше, по пути повышения удобства работы и ускорения процесса создания новых программ.

Постоянное улучшение функциональности сред, в классическом варианте, приводит к появлению на рынке их новых версий, которые, зачастую, отличаясь по внешнему виду (расположению кнопок, элементов меню и прочие) требуют времени на привыкание, так и дополнительных денежных затрат, если среда является коммерческим продуктом.

Этих недостатков лишена интегрированная среда разработки Eclipse [1], которая позволяет модульно наращивать свою функциональность за счет добавления новых частей (*Plugin*), и к тому же свободно распространяется и поддерживается по лицензии *Eclipse Public License* сообществом *Eclipse Foundation*.

Продолжая затронутую в [2] тему создания интегрированной среды разработки программ для микроконтроллеров фирмы STMicroelectronics, в данной статье будет описано добавление механизма отладки (без чего не обходится любая среда) и дополнительно коснемся вопроса упрощения и ускорения настройки периферии STM32 микроконтроллеров.

[3] Работа выполнена по заданию Министерства образования и науки РФ, проект №7.599.2011, Темплан, НИР № 01201255056.

На этих двух примерах будет продемонстрирована заложенная в *Eclipse* возможность расширения своей функциональности путем добавления новых *Plugin*, кстати сказать, из которых состоит сама среда.

ОТЛАДКА ПРОГРАММ ПРИ ПОМОЩИ GNU DEBUGGER

GNU Debugger (GDB) - отладчик пришедший к нам из мира *UNIX* подобных систем, но при этом позволяющий легко переносить его на другие платформы, в частности ОС *Windows*. Данный продукт может вести отладку на многих языках программирования: *C, C++, Free Pascal, FreeBASIC, Ada и Фортран и др.* [3].

Помимо возможности отлаживать программы для ОС (ориентация на процессор операционной системы) *GNU Debugger* позволяет работать и с внешними процессорами, например микроконтроллерами, при помощи возможности удаленной отладки программ.

Под удаленной отладкой подразумевается механизм, когда среда разработки условно располагается на одном компьютере, а целевая платформа (программа нуждающаяся в отладке) располагается на другом. Обмен данными между двумя этими устройствами происходит по специальному протоколу через *TCP/IP* канал.

В случае с микроконтроллерами фирмы *STMicroelectronics*, отладка ведется именно по такому пути, но усложняется тем, что доступ к модулю отладки целевой архитектуры (в данном случае *ARM Cortex M3*) организуется через встроенный порт *SWJ-DP (Serial Wire JTAG Debug Port)*.

Данный порт работает в двух режимах: *JTAG (Joint Test Action Group)* и *SWD* (*Serial wire debug* - двух проводной последовательный канал связи). В принципе можно использовать оба этих интерфейса, но на плате *STM32VLDISCOVERY* уже имеется интерфейс отладки *ST-Link*, который ведет обмен отладочной информацией по *SWD*. Связь управляющего компьютера с *ST-Link* физически организуется через *USB* интерфейс. *ST-Link*, по сути является преобразователем физических интерфейсов *USB* в *SWD*.

Чтобы согласовать на программном уровне интерфейсы *ST-Link* с отладчиком *GDB* используется дополнительная программная прослойка в виде так называемого *GDB-Server*. Она представляет собой урезанный *GNU Debugger*, который умеет исполнять те команды

GNU Debugger, которые поддерживает *ST-Link*. С *GNU Debugger* (далее в тексте будем назвать его *GDB*-клиент), как уже было сказано обмен информацией происходит по каналу *TCP/IP*. Приведем на *рис*. 1, для ясности, получившуюся программно-аппаратную конструкцию системы отладки.

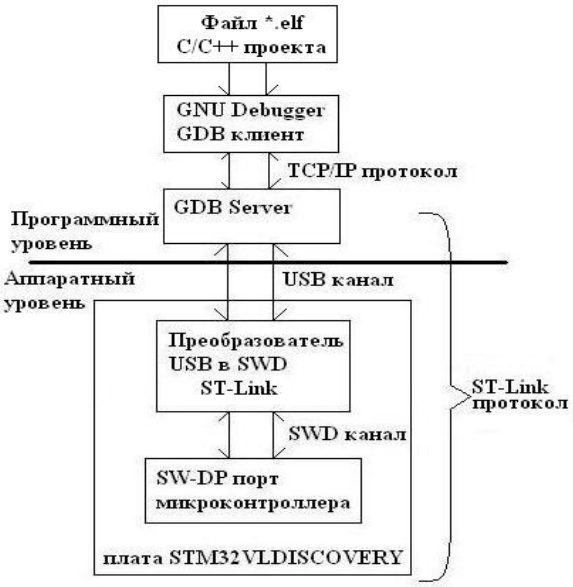


Рис. 2 - Схема программно-аппаратной конструкции системы отладки

Стоит отметить, что разнесение программ *GDB*-клиента и *GDB*-сервера на разные персональные компьютеры не обязательно, можно эти две программы разместить на одном компьютере. Именно эта возможность будет использована здесь.

Программа *GDB*-клиента обычно поставляется вместе с набором компиляции *GCC* (так как является, на ряду с компилятором *GCC*, частью большого проекта *GNU* [4]). В предыдущей статье [2] был использован пакет компиляции *Sourcery CodeBench Lite Edition*. Если, при установке этого пакета в папку по умолчанию, пройти по пути «*C:\Program Files\CodeSourcery\ Sourcery_CodeBench_Lite_for _ARM_EABI\bin*», то можно обнаружить файл «*arm-none-eabi-gdb.exe*», который как раз является программой *GDB*-клиентом (*GNU Debugger*).

На данный момент существует два способа получения программы *GDB*-сервера: из дистрибутива бесплатной облегченной версии среды ***Atollic TrueSTUDIO Lite*** компании *Atollic* [5] или портированным на *Windows Linux* приложением *stlink* от автора T*exane* [6], которое свободно распространяется в исходных кодах. Остановимся подробнее на способе получения программы GDB-сервера в обоих случаях.

Если загрузить себе на компьютер дистрибутив бесплатной версии *Atollic TrueSTUDIO for ARM Lite*, пройдя по ссылке [7], то для того, чтобы получить программу *GDB*-сервер, необходимо открыть этот само-распаковывающийся архив любой программой архиватором. Здесь очень важно именно открыть архив, как папку, а не распаковать его или запустить на выполнение. В открывшейся структуре папок нас будет интересовать директория *Servers*, как приведено на *рис*. 2.

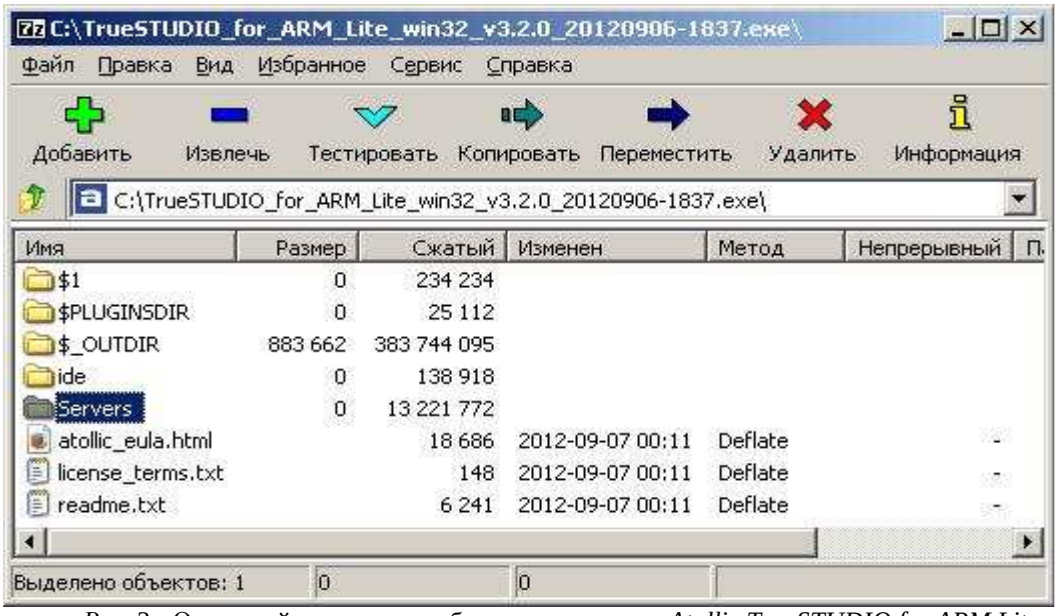


Рис. 3 - Открытый архив дистрибутива программы *Atollic TrueSTUDIO for ARM Lite*

Внутри этой папки следует найти папку *ST-LINK_gdbserver* и извлечь её из архива в удобное Вам место. Программой *GDB*-сервером является файл *ST-LINK_gdbserver.exe* внутри этой папки.

Теперь проект *stlink* от автора *Texane*. Так как он распространяется в исходных кодах, то можно

было бы загрузить эти файлы и откомпилировать их под *Windows* (создать порт), но для этого, дополнительно, необходимо устанавливать компилятор *GCC* для *Windows*, что сложно. К счастью, работа по портированию уже выполнена другими людьми (например [8]) и нам остается только воспользоваться плодами их усилий. Для этого следует пройти по ссылке [9], загрузить и далее распаковать архив в удобное Вам место. Программой *GDB*-сервером здесь является *st-util.exe* в папке *bin* распакованного архива.

Последней оставшейся частью является установка в системе *USB* драйвера для преобразователя *ST-Link*, дистрибутив которого можно скачать с сайта фирмы *STMicroelectronics* пройдя по ссылке [10] для *Windows XP*, *Vist* или *7*, и [11] для *Windows 8*.

НАСТРОЙКА GDB ОТЛАДКИ В ECLIPSE

В стандартную поставку среды *Eclipse* для *C/C++* разработчиков не входит механизм поддержки удаленной аппаратной отладки приложений при помощи *GDB*, но он входит как плагин *C/C++ GDB Hardware Debugging* в проект *CDT* (*C/C++ Development Tooling*).

Чтобы установить плагин *C/C++ GDB Hardware Debugging* потребуется открыть Eclipse (об установке самой среды, разворачивании рабочего пространства — *Workspace* и настройке было рассказано в предыдущей статье [2]), выбрать необходимое местоположение *Workspace* и вызвать пункт меню *Help → Install New Software*. Дальше возможно два варианта: установка из репозитария по интернет адресу или установка из архива расположенного на жестком диске компьютера. В первом случае необходимо ввести в поле *«Work With»* строку «http://download.eclipse.org/tools/cdt/releases/helios», как на *рис*. 3.

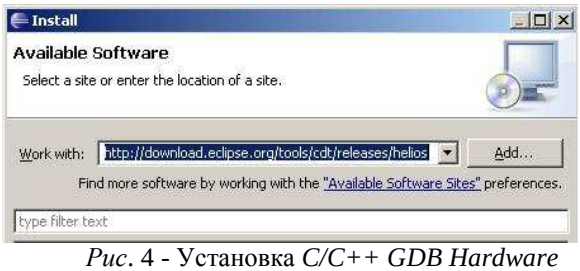


Рис. 4 - Установка *C/C++ GDB Hardware Debugging* из репозитария

Во втором случае нажать кнопку *Add...* (справа от поля ввода интернет строки) и в появившемся окне *Add repository* нажать кнопку *Archive...* Затем следует указать архив проекта *Eclipse CDT*, который можно загрузить пройдя по ссылке [12]. Дальше необходимо выбрать, как указано на *рис*. 4, пункт C/C++ GDB Hardware Debugging.

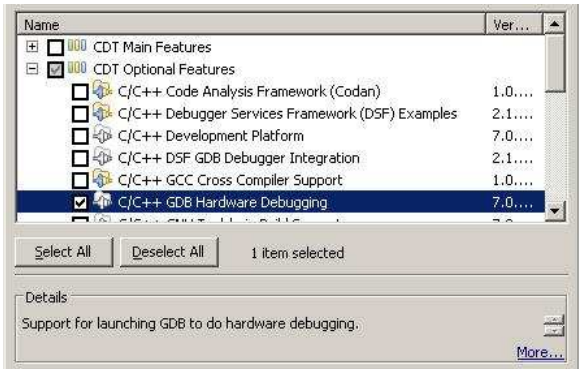


Рис. 5 - Выбор плагина C/C++ GDB *Hardware Debugging*

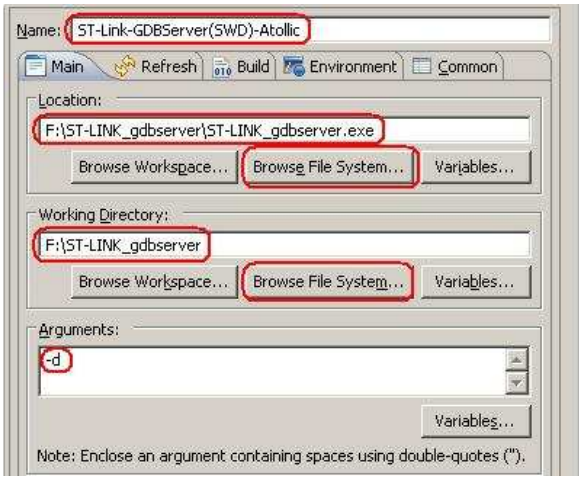
При открытии последующих двух окон Нажать на кнопку *Next...*, а на третьем указать пункт *«I accept the terms of the license agreement»* и нажать кнопку *«Finish»*. После установки плагина среда попросит Вас перезапустить *Eclipse*, для выполнения этого нажмите кнопку *«Restart Now»*.

Теперь создадим в *Eclipse* настройки конфигурации запуска *GDB*-сервера для чего вызовем пункт меню *Run → External Tools → External Tools Configurations...* В появившемся диалоге *External Tools Configurations* выбираем в списке слева пункт *«Program»* и затем в левом верхнем углу найдем кнопку, показанную на *рис*. 5, для создания новой конфигурации и нажмём её.

Рис. 6 - Создание новой конфигурации для *GDB-Server*

В созданной пустой конфигурации заполняем первую вкладку *Main* данными, которые определяются выбранным типом *GDB*-сервера.

Для программы от Atollic следует заполнить поля ввода как на рисунке 6.



Puc. 7 - Настройка конфигурации *GDB*-сервера, вкладка *Main* для *GDB*-сервера от *Atollic*.

Заполнение поля *Working Directory:* не обязательно. В поле *Name:* может быть введена любая строка. В поле *Location:* полный путь до файла *GDB*-сервера.

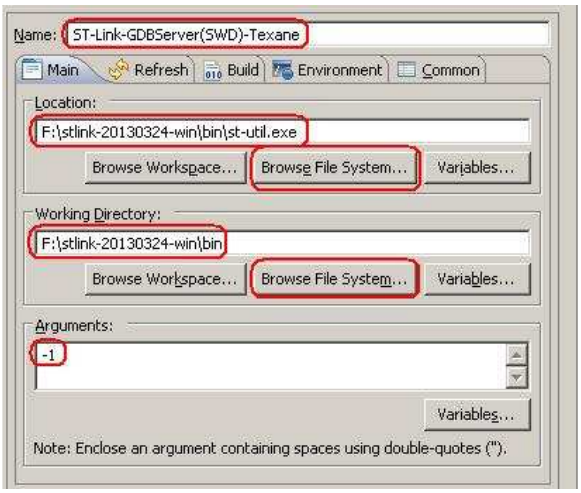
Важным полем здесь является *Arguments:* где приводятся ключи вызова *GDB*-сервера. Сделаем некоторое пояснение: задаваемые в этом окне параметры позволяют каждый раз запускать, при активации данной конфигурации, *GDB*-сервер из командной строки, а в поле *Arguments:* задаются ключи для его вызова.

GDB-сервер от Atollic поддерживает всего 7 ключей командной строки:

1. **-e** - активирует режим в котором программа сервер не заканчивает свою работу при закрытии клиентской программы;
2. **-f <Log-File>** - определяет имя лог-файла (*Log-File*) для записи в него процесса обмена отладочными сообщениями. Пример использования: «*-f c:\tmp\debug.log»;*
3. **-l <Log-Level>** - определяет степень подробности вывода информации в лог-фай: *Log-Level = 0* — отключение записи в файл, *Log-Level >=1* - разрешение записи сообщений об ошибках, *Log-Level >=2* - добавляется вывод предупреждений, *Log-Level >=4* - добавляется вывод специфических команд отладочной информации. *Log-Level >=8* - добавляется вывод всех информационных сообщений, *Log-Level >=16* — добавляется вывод специфических *HW* сообщений;
4. **-p <Port-Number>** - определяет номер порта *TCP*, с которого должны поступать отладочные сообщения от клиента *GDB*, по умолчанию этот *Port-Number* равен *61234;*
5. **-v** - вводит такой режим работы командной строки, что все сообщения которые поступают от клиентской части выводятся в консоль;
6. **-r <delay-sec>** - определяет максимальную задержку, в секундах, при обновления текущего состояния сервера;
7. **-d** - переводит (программным образом) микроконтроллер в режим обмена данными по каналу *SWD*, по умолчанию активен режим *JTAG*.

Таким образом, установка ключа -d запускает работу преобразователя *ST-Link* в режиме обмена данными по последовательному двух проводному интерфейсу.

Теперь приведем, на *рис*. 7 параметры конфигурации для программы сервера st-utils от Texane.



Puc. 8 - Настройка конфигурации *GDB*-сервера, вкладка *Main* для *GDB*-сервера от *Texane*

1. Обратим наше внимание на возможные значения поля *Arguments:*. Здесь допустимы следующие ключи:
2. **-h** или **--help** - выводит в командную строку помощь по основным ключам данной программы;
3. **-vXX**, или **--verbose=XX** - определяет уровень подробности вывода информации в консоль, как и *GDB*-сервер *Atollic;*
4. **-v** или **--verbose** - разрешает подробный вывод всех сообщений в командную строку;
5. **-s X** или, **--stlink_version=X** - устанавливает какая версия протокола *ST-Link* используется (на плате *STM32VLDISCOVERY* развернута версия *ST-Link 1*), по умолчанию задана *2* версия;
6. **-1** или **--stlinkv1** - принудительно устанавливает версию 1 *ST-Link;*
7. **-p 4242** или **--listen_port=1234** - определяет номер порта *TCP*, с которого должны поступать отладочные сообщения от клиента *GDB*, по умолчанию номер порта равен *4242*.

Итак, *GDB*-сервер запускается с единственным ключом *-1*, который принудительно устанавливает версию протокола *ST-Link*. Здесь необходимо заметить, что сервер от *Texane* не будет работать со стандартным драйвером от *STMicroelectronics* по причине несовместимости используемой при написании этой программы специфической библиотеки *libusb* с этим драйвером. Требуется заменить стандартный драйвер аналогом под *Windows* для совместимости. Для этого есть достаточно простая бесплатная программа *Zadig*, которую можно загрузить пройдя по ссылке [13]. Дальнейшую работу следует проводить после подсоединения отладочной платы *STM32VLDISCOVERY* к *USB* порту компьютера. После запуска программы следует установить галочку в пункте меню *Options → List All Devices*. В результате чего окно программы должно измениться, как показано на *рис*. 8.

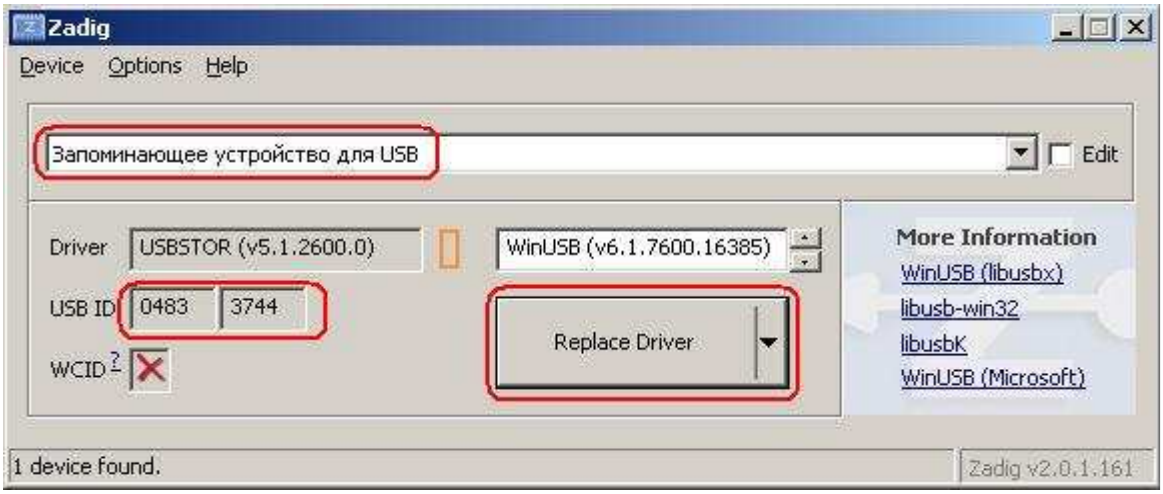


Рис. 9 - Указание в программе *Zadig* заменяемого драйвера *ST-Link*

Выберите из выпадающего списка устройство *USB Mass Storage Device* (Запоминающее устройство для *USB*) с *USB ID*, равным 0483 3744 (*ST-Link v1*) и нажмите кнопку *Replace Driver:* Программа предупредит, что изменяется системный драйвер, показав окно *Warning-System Driver*, где следует нажать кнопку *Yes*. После окончания замены драйвера программа выдаст соответствующее сообщение, что драйвер установлен успешно. На этом работа программы *Zadig* завершена. Следует здесь отметить, что этими действиями мы фактически удалили стандартный драйвер *ST-Link*, тем самым нарушив нормальную работу сервера от Atollic. Поэтому обе программы — серверы (от *Atollic* и *Texane*) не могут работать одновременно — нужно выбрать один из этих вариантов.

Следующая вкладка, которая нас будет интересовать при настройке конфигурации *GDB*-сервера это *Build*, на которой всего лишь следует снять галочку *Build before launch*. На этом настройка конфигурации *GDB*-сервера завершена. В подтверждение этого нужно последовательно нажать кнопки *Apply* и *Close* внизу окна настройки конфигурации (*рис*. 9).

Рис. 10 - Расположение кнопок *Apply* и

Теперь, при наличие конфигурации для серверной части механизма отладки, обратим наше внимание на клиентскую часть. Настройку начнем с вызова пункта меню *Run → Debug Configurations...*, после чего увидим окно *Debug Configurations*, где создадим новую конфигурацию подобно тому как это было сделано с серверной частью. Выделим пункт *GDB Hardware Debugging* и нажмем кнопку *New launch configuration*, что изменит внешний вид окна. Начнем последовательно заполнять вкладки данными начиная с *Main*. Итак, заполняем эту вкладку данными как на *рис*. 10.

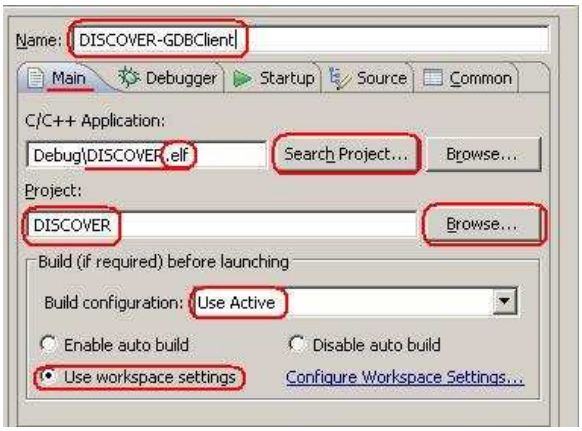


Рис. 11 - Настройка конфигурации *GDB*-клиента, вкладка *Main*

Сначала дадим имя данной конфигурации

(поле *Name:*), например *DISCOVER-GDBClient*. Дальше выберем проект, который будет подвергаться отладки. В предыдущей статье [2] был собран тестовый проект с сайта фирмы *STMicroelectronics*, для него и создадим конфигурацию клиентской части *GDB* отладки. Воспользовавшись кнопкой *Browse...* найдем этот проект в текущем *Workspace*. Может произойти так, что поле *C/C++ Application:* заполнится автоматически, но если этого не произошло, то найдем файл *DISCOVER.elf* в выбранном проекте, воспользовавшись кнопкой *Search Project...*. Остальные параметры, желательно установить так как показано на *рис*. 10 — это делает настройку универсальной.

Переходим к следующей вкладке — *Debugger*. Вписываем содержимое полей как на *рис*. 11.

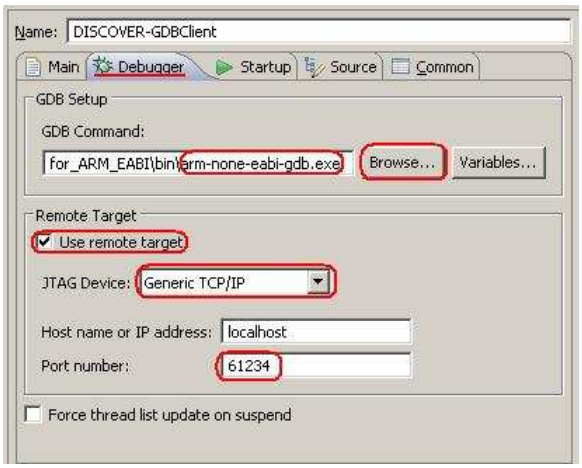


Рис. 12 - Настройка конфигурации GDB-клиента, вкладка *Debugger*

Здесь важно в поле *GDB Command* вписать полный путь до *GDB* клиента. Если пакет *Sourcery CodeBench Lite* был установлен в папку, предлагаемую по умолчанию, то нужно записать: «*C:\ProgramFiles\CodeSourcery\Sourcery_CodeBench_Lite_for_ARM_EABI\bin\arm-none-eabi-gdb.exe*». В поле *Port number* следует записать тот порт, по которому клиентская программа будет отправлять запросы серверной. По умолчанию, для сервера от *Atollic*, этот порт равен *61234*, а для сервера от *Texane – 4242*.

В следующей вкладке *Startup*, прокрутив содержимое окна вниз следует поставить всего две галочки: *Set breakpoin at:* и *Resume* как показано на *рис*. 12 и в поле напротив *Set breakpoin at:* ввести *main*.

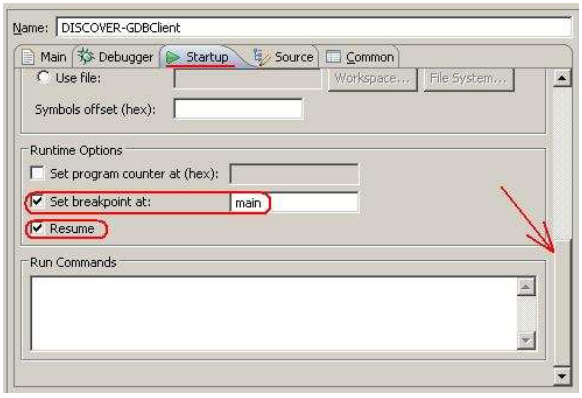


Рис. 13 - Настройка конфигурации *GDB*-клиента, вкладка *Startup*

Это позволит остановить дальнейшее выполнение программы, после её загрузки в память микроконтроллера, для выполнения пошаговой отладки.

На этом конфигурация клиентской части закончена. Применяем введенные настройки и закрываем окно, нажав последовательно на клавиши *Apply* и *Close*.

В основном настройка работы механизма *GDB* отладки завершена, но использование его будет не очень удобным. Так, например, для того, чтобы отладить какой-либо проект Вам сначала потребуется откомпилировать проект, затем запустить конфигурацию сервера и, наконец, конфигурацию клиента. К счастью, разработчики *Eclipse* предусмотрели возможность упрощения этого механизма до нажатия одной кнопки. Он называется группой запуска (*Launch Group*).

Доступ к этой возможности организуется через пункт меню *Run → Run Configurations...*. В результате открывается окно настройки конфигураций, схожее с тем, что было раньше, на предыдущих шагах. Выбираем *Launch Group* и создаем новую конфигурацию запуска. В результате этого должно появится пустое окно, которое следует заполнить при помощи кнопки *Add...* как на *рис*. 13.

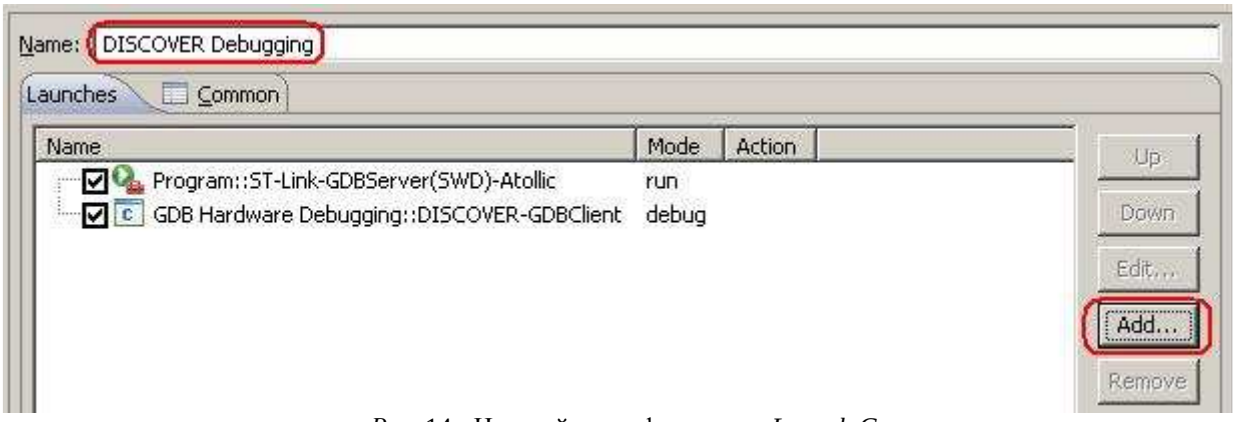


Рис. 14 - Настройка конфигурации *Launch Group*

Обязательно необходимо выполнить правильную последовательность добавления конфигураций: сначала конфигурацию сервера, а затем конфигурацию клиента. Рассмотрим добавление только серверной части, клиентская часть выполняется схожим образом за исключением некоторых моментов, о которых будет сказано отдельно. Итак, после нажатия кнопки *Add...* появляется окно *Add Launch Configurations*, где необходимо из выпадающего списка *Launch mode* выбрать *run* и затем, раскрыв пункт *Program* в окне ниже, выбрать пункт соответствующий имени созданной ранее конфигурации сервера. Должно получиться как на *рис*. 14.

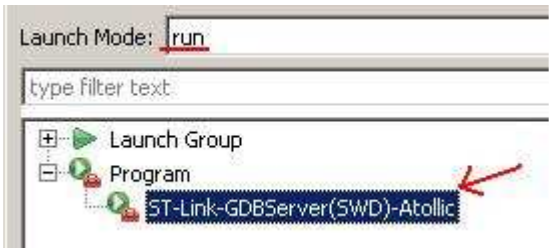


Рис. 15 - Добавление конфигурации сервера в группу запуска на примере программы от *Atollic*

Добавление клиентской части аналогично, только следует выбрать из списка *Launch mode* режим *debug* и далее, в списке ниже, раскрыв группу *GDB Hardware Debugging* выбрать имя конфигурации сервера После добавления обоих конфигураций (серверной и клиентской) нужно ещё определить точку вызова данного функционала из меню программы: Для этого следует перейти на вкладку *Common* и в оке *Display in favorites menu* поставить галочку напротив *Ran*, как на *рис*. 15.

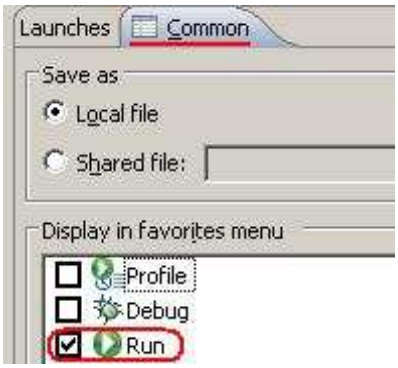


Рис. 16 - Конфигурация параметров отображения группы запуска в элементах меню

Подтверждаем введенные параметры нажатием кнопки *Apply* и закрываем окно кнопкой *Close*.

Попробуем то, что получилось. Сначала необходимо подключить плату *STM32VLDISCOVERY* к *USB* порту компьютера и проверить замкнуты ли перемычками соседние контакты разъема *CN3* на плате Discovery, как было упомянуто в предыдущей статье [2]. Наконец, закрыв окно обнаруженного диска (для установленного стандартного USB драйвера), выполнить пункт меню *Run → Run History → DISCOVER Debugging* (или название группы конфигурации, которое Вы ей дали). Или ещё другой способ: нажать кнопку меню как показано на *рис*.16.

Рис. 17 - Место запуска отладки *GDB*

Для режима отладки существует специальная перспектива (видовой экран) которая называется *Debug*. Переключиться в нее возможно нажав одноименную клавишу в верхнем правом углу окна среды как на *рис*. 17.

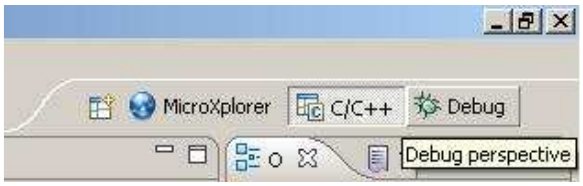


Рис. 18 - Место переключение перспектив

Если к этому моменту Вы ещё этого не сделали, то система спросит Вас о переключении в данную перспективу появлением окна *Confirm Perspective Switch*. При ответе положительно (нажатие кнопки *Yes*) происходит автоматическое переключение. При этом счетчик программ будет установлен на первую команду из функции main, как было задано в настройках. Обратим наше внимание на пожалуй самое главное окно в этой перспективе — *Debug*. Внешний вид этого окна показан на *рис*. 18.

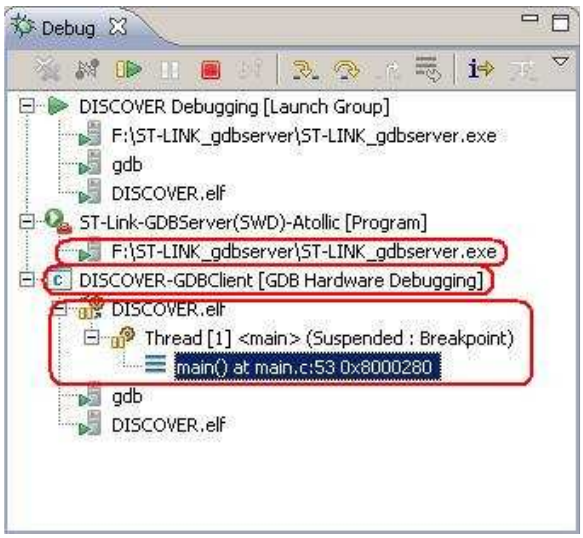


Рис. 19 - Окно дерева отладки

Здесь выделены (снизу вверх) запущенные составные части механизма отладки: сервер, клиент и файл откомпилированной программы в *elf* формате. Все управление процессом отладки (пошаговое выполнение запуск и остановка процесса отладки сосредоточено в верхней части этого окна). Для человека имеющего хотя бы малейший опыт работы с отладчиками разобраться в остальном не составит особого труда.

ПЕРСПЕКТИВА MICROXPORER

Всегда, при выборе того или иного микроконтроллера, встает вопрос: подходит ли микроконтроллер в выбранном корпусе для решения поставленной задачи? Зачастую, богатый набор периферии устройства еще не гарантирует полную реализацию всех возможностей. Это может быть вызвано тем, что при малом количестве выводов, каждому из них может быть назначено сразу несколько функций разных периферийных модулей. Разрешить все возможные конфликты между этими модулями задача не такая уж простая, особенно при большом количестве выводов и периферийных модулей заложенных в микроконтроллер.

Помимо этого, разработчик программы сталкивается со значительными затратами времени на задание настроек регистров той или иной периферии. Задача трудна из-за своего большого объема рутиной работы и требует известной степени внимательности, так как ошибка в одном бите может привести к отличной от требуемой настройки периферии. Компания *STMicroelectronics* попыталась здесь помочь разработчику, упростив эти процессы и сделав их более наглядными. Речь идет о бесплатном программном продукте *microXplorer* [14].

Существует два варианта этого продукта: автономное *Java* приложение и плагин для *Eclipse*. Оба этих продукта по своему внешнему виду и по возможностям идентичны друг другу, поэтому, применительно к нашей ситуации, рассмотрим только плагин.

Начнем с описания процесса установки плагина в среду. Компания *STMicroelectronics* не предоставляет возможность загрузки через интернет адрес, поэтому остается только один вариант - загрузить архив с плагином, пройдя по ссылке [15]. Процесс установки аналогичен плагину *C/C++ GDB Hardware Debugging*, за исключением того, что во время установки будет выдано предупреждения об отсутствии подписи у данного продукта, на что следует ответить нажатием кнопки «*OK*».

На момент написания статьи последней версией *microXplorer* считается версия 3.2. Чтобы открыть окно плагина в среде *Eclipse* нужно вызвать окно *Open Perspective*, выполнив пункт меню *Window → Open Perspective → Other...* и выбрать в нём *microXplorer*. После этих манипуляций вид активного окна среды должен измениться, так как открылась новая перспектива. Вернутся обратно, в другую перспективу, можно выбрав соответствующую вкладку, приведенную на рисунке 17.

Процесс работы с этой программой начинается с создания новой конфигурации, для чего нужно: либо выполнить пункт главного меню *File → new Config...*, либо нажать комбинацию клавиш *Shift + N*, либо нажать на кнопку *Add New Configuration* под главным меню (первая слева). Затем в появившемся окне *New Configuration Settings* выбрать интересующий контроллер. Давайте, для примера, выберем *STM32F100RB* находящийся на плате *STM32VLDISCOVERY*, как изображено на рисунке

19.

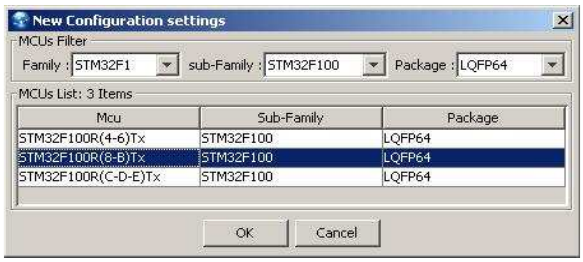


Рис. 20 - Выбор типа микроконтроллера

После создание конфигурации вид перспективы снова изменится. В этом окне справа будет приведён перечень доступной для данного микроконтроллера периферии, а слева внешний вид выбранного корпуса. При выборе периферии окно станет выглядеть как на *рис*. 20.

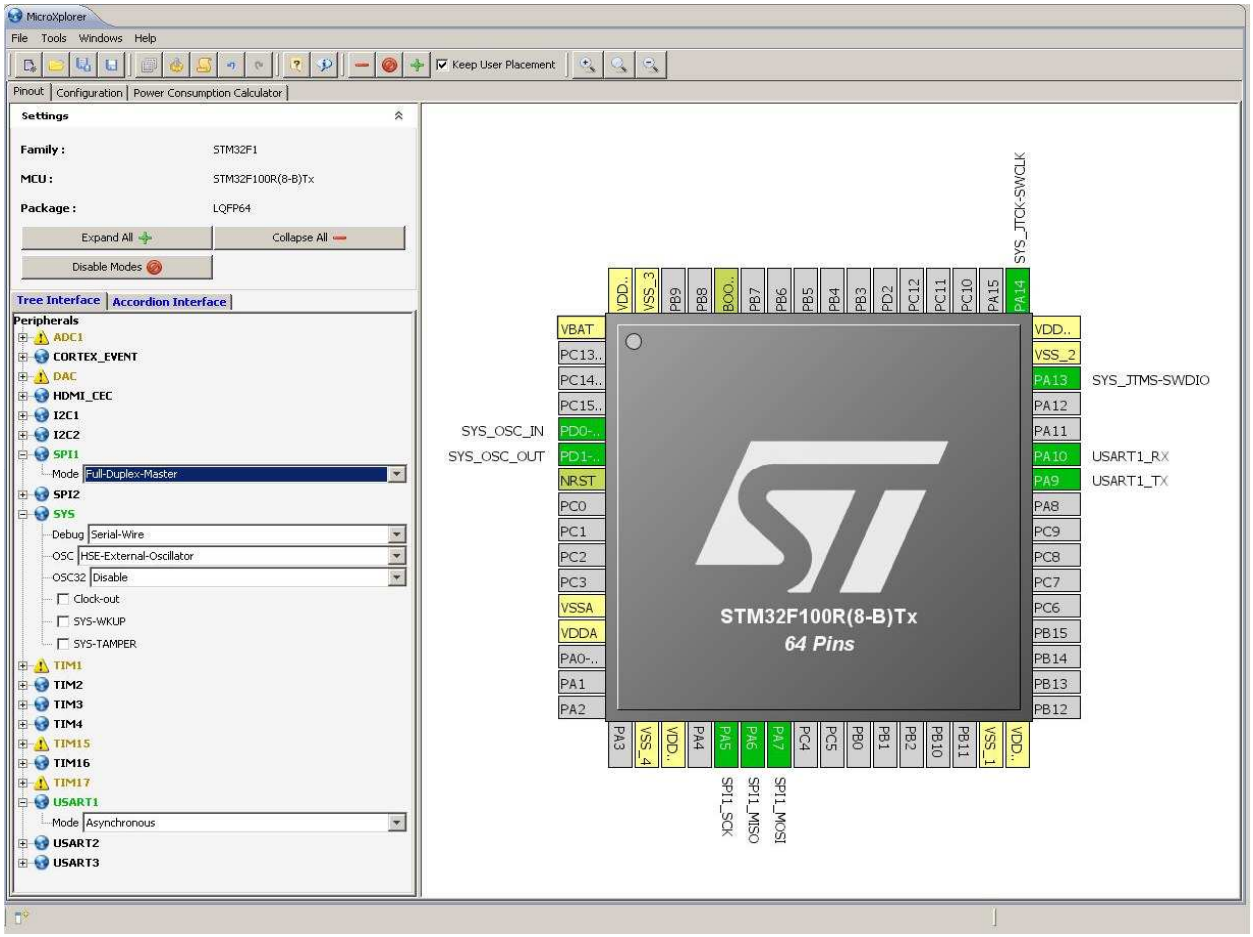


Рис. 21 - Внешний вид перспективы microXplorer

На *рис*. 20, в частности, приведен результат задания следующих функций: работа *SPI1* модуля в режиме полнодуплексного обмена данными с функцией ведущего (*Master*), выбрана отладка и программирование ядра через канал *SWD*, заняты ножки под внешний кварцевый резонатор и наконец модуль *USART1* переведен в режим асинхронной передачи данных. В правом окне зеленным выделены занятые этими функциями выводы. В левом окне восклицательным знаком выделены те модули функциональность которых пострадала в результате этих действий.

Правое окно служит не только для отображения данных. Если щелкнуть на интересующем выводе правой кнопкой мыши, то можно увидеть перечень допустимых функций для данного вывода, а если при щелчке левой кнопкой ещё зажать кнопку *Ctrl*, то можно увидеть, выделенные синим, возможные варианты перенесения данной функции на другие выводы при помощи механизма переназначения выводов *Remap*. Таким образом, можно изменить вручную конфигурацию любого вывода.

Полное руководство по данному плагину можно получить обратившись к [16]. Отметим только наиболее интересные особенности.

Для получения результатов проведенных действий можно распечатать в формате PDF отчет выполнив команду главного меню *Tools* → *Generate Report*.

Можно, помимо назначения выводов, создать программный код, написанный на языке *C* осуществляющий необходимую настройку периферии. В текущей версии пока возможна только настройка портов ввода/вывода. Доступ к этой возможности открывается через пункт меню *Tools → Generate Code*. Правда, предварительно необходимо перейти на вкладку Configuration, левого окна, и затем, щелкнув в правом окне на кнопке *GPIO* (*рис*. 21), задать необходимые функции для выбранных выводов.

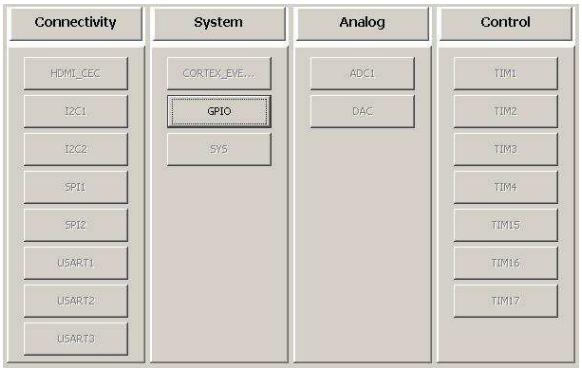


Рис. 22 - Место задание параметров настройки выводов периферии

Неактивная, пока, для выбранного нами микроконтроллера возможность по оценке энергопотребления в разных режимах работы доступна для семейства *STM32L1*. Обращение к ней организуется при переходе на вкладу *Power Consumption Calculator* в окне слева.

Таким образом, *microXplorer* хороший помощник на пути освоения новых микроконтроллеров и помогает ускорить работу разработчика за счёт сокращения количества рутинной работы.

ЗАКЛЮЧЕНИЕ

Приведенные в этой статье возможности лишь малая часть всех тех возможностей которые можно получить от Eclipse. В стандартной поставке среды, при загрузки с официального сайта, присутствуют иные плагины не менее полезные при профессиональной работе по программированию. И куда более обширная часть представлена в сети Интернет. Стоит отметить, что зачастую, у каждого плагина или набора плагинов всегда есть своя интернет страничка, на которой можно найти всю исчерпывающую информацию по нему. В этом свете, перед пользователем среды открываются широкие возможности по построению той среды, какой он хочет видеть её сам.

ВЫВОДЫ

- Различные способы и механизмы отладки заложенные в саму среду *Eclipse* могут быть расширены при помощи дополнительных плагинов.
- Есть возможность реализовать отладку микроконтроллеров фирмы *STMicroelectronics* не прибегая к платным программным продуктам причем не единственным способом.
- В среду *Eclipse* заложены возможности гибкой настройки частей механизма отладки.
- Плагин *microXplorer* помогает разработчику экономить время и избавляют его от рутинной работы.
- Заложенный в среду *Eclipse* мощный механизм расширения функциональности позволяет создать ту *IDE*, которую хочет видеть её пользователь.

ЛИТЕРАТУРА

[1] Eclipse (среда разработки). Материал из Википедии — свободной энциклопедии. URL: http://ru.wikipedia.org/wiki/Eclipse_(среда_разработки) (дата обращения 18.11.13).

[2] А.В. Ескин, В.А. Жмудь, В.Г. Трубин. STM32VLDISCOVERY – средство для быстрой разработки опытных образцов цифровых систем управления. Автоматика и программная инженерия. 2013. № 3 (5). С. 42–47.

[3] GDB: The GNU Project Debugger. URL: https://sourceware.org/gdb/ (дата обращения 18.11.13).

[4] GNU. Материал из Википедии — свободной энциклопедии URL: http://ru.wikipedia.org/wiki/GNU (дата обращения 18.11.13).

[5] Atollic TrueSTUDIO. URL: http://atollic.com/index.php/truestudio (дата обращения 18.11.13).

[6] Stm32 discovery line linux programmer. texane / **stlink. GitHub URL: https://github.com/texane/stlink (дата обращения 18.11.13).**

[7] Download Atollic TrueSTUDIO® for ARM® Lite. URL: http://atollic.com/index.php/download/truestudio-for-arm (дата обращения 18.11.13).

[8] STLINK GDB Server. Emb4fun. URL: http://www.emb4fun.de/archive/stlink/index.html (дата обращения 18.11.13).

[9] Download STLINK v0.5.3 for Windows, tested with Windows XP (942 KB) http://www.emb4fun.de/download/arm/stlink/stlink-20121227-win.zip (дата обращения 18.11.13).

[10] STSW-LINK003 ST-LINK/V2 USB driver for Windows 7, Vista and XP. STMicroelectronics. URL: http://www.st.com/web/en/catalog/tools/PF258167 (дата обращения 18.11.13).

[11] STSW-LINK006 ST-LINK/V2 USB driver for Windows 8. STMicroelectronics. URL: http://www.st.com/web/en/catalog/tools/PF259459. (дата обращения 18.11.13).

[12] Download CDT 7.0.2 for Eclipse Helios URL: http://www.eclipse.org/downloads/download.php?file=/tools/cdt/releases/helios/dist/cdt-master-7.0.2.zip (дата обращения 18.11.13).
[13] Zadig. USB driver installation made easy. URL: http://zadig.akeo.ie/ (дата обращения 18.11.13).
[14] MicroXplorer MCU graphical configuration tool. STMicroelectronics. URL: http://www.st.com/web/catalog/tools/FM147/CL1794/SC961/SS1533/PF251717?s_searchtype=partnumber# (дата обращения 18.11.13).
[15] STSW-STM32095. MicroXplorer Eclipse plugin, graphical tool to configure STM32 microcontrollers. URL: http://www.st.com/web/en/catalog/tools/PF257931# (дата обращения 18.11.13).
[16] UM1568. User manual. MicroXplorer 3.x graphical configuration tool URL: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/DM00063255.pdf (дата обращения 18.11.13).

Алексей Викторович Ескин - ведущий инженер ООО «КБ Автоматика»,
E-mail: kba-elma@bk.ru

Вадим Аркадьевич Жмудь – заведующий кафедрой Автоматики НГТУ, профессор, доктор технических наук, автор более 200 научных статей, включая 10 патентов и 6 учебных пособий. Область научных интересов и компетенций – теория автоматического управления, электроника, лазерные системы, оптимизация, измерительная техника.
E-mail: oao_nips@bk.ru

Виталий Геннадьевич Трубин - зав. лаб. кафедры Автоматики НГТУ, директор ООО «КБ Автоматика». Автор 18 научных статей. Область интересов – разработка специализированной электроники.
E-mail: trubin@ngs.ru