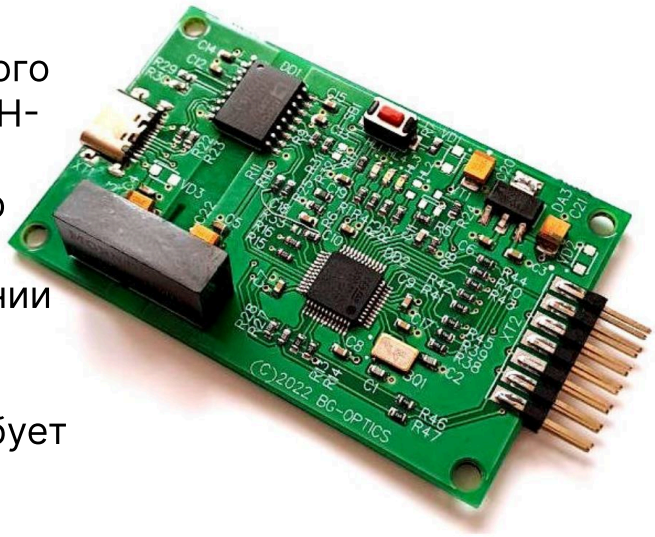
КС ИНФРАСТРУКТУРА
+7 (495) 009-10-35
ksi@csc.ru infra.csc.ru

1Б2541.003: 1.5КВ ИЗОЛИРОВАННЫЙ JTAG/SWD/UART/GPIO ОТЛАДЧИК-ЭМУЛЯТОР

Описание изделия:

Изделие 1Б2541.003 представляет собой высоковольтный изолированный полнофункциональный отладчик-эмулятор. Оно предназначено для отладки исполняемого кода и программирования внутренней FLASH-памяти микроконтроллеров и систем на кристалле (СнК) на базе процессорных ядер ARM Cortex.



Работа устройства основана на использовании отладчика GNU GDB с применением MI-протокола (Machine Interface) для обмена отладочной информацией. Отладчик не требует установки дополнительного внешнего программного обеспечения (демонов, драйверов и т.д.) и совместим с операционными системами GNU/Linux, Windows и macOS.

По своей функциональности изделие является полным аналогом проекта Black Magic Probe (https://github.com/blacksphere/blackmagic/wiki).

Изделие специально разработано для отладки и программирования высоковольтных и силовых целевых систем (target-систем). К таким системам относятся контроллеры мощных электродвигателей, высоковольтные индукторы, трансформаторы, линии передачи энергии и другие аналогичные устройства. Конструкция обеспечивает гальваническую развязку с напряжением пробоя изоляционного барьера до 1,5 кВ.

141190, Московская Область, г.о. Фрязино, г. Фрязино, проезд Окружной, дом 13Б

+7 (495) 009-10-35
ksi@csc.ru infra.csc.ru

Приложения:

– отладка, программирование в FLASH разрабатываемого кода устройств с процессорными ядрами ARM Cortex-A, ARM Cortex-M высоковольтных и силовых target-систем;
– использование как мост host USB→UART device;
– чтение/установка логических сигналов на device-разъёме адаптера (GPIO).

Особенности:

- Малые масса и габариты.
- Гальваническая развязка между host- и target-доменами.
- Полная функциональная совместимость с Black Magic Debug Probe.
- Простота эксплуатации.
- Не требует установки драйверов в ОС GNU/Linux.
- Не требует запуска отдельного GDB-сервера или иного промежуточного ПО.
- Высокая скорость операций отладки и программирования.
- Поддержка отладочных интерфейсов: JTAG, SWD, SWDIO.
- Связь с хост-компьютером по интерфейсу USB.
- Светодиодная индикация состояния.
- Возможность обновления встроенного программного обеспечения.
- Активная разработка встроенного ПО, направленная на:
  – расширение списка поддерживаемых target-систем (семейств микроконтроллеров и SoC);
  – добавление новых функциональных возможностей.

Поддерживаемые семейства микроконтроллеров с ядром ARM Cortex-M:

Производитель	Семейства
STMicroelectornics	STM32 L/F/G/H
Atmel	SAM D09/D20/D21/E70/S7x//3N/3X/3N/3S/3U/4L/4S/5X

+7 (495) 009-10-35
ksi@csc.ru infra.csc.ru

Производитель	Семейства
Nordic	NRF 51/52
NXP	LPC 8xx/11xx/15xx/43xx/546xx
Texas Instruments	LM3S TM4C MSP432
Freescale	KL25/27/02 KE04
Silicon Labs	EFM32 EZR32
Rasbery	RP2040
Renesas	RA2/4/6

Поддерживаемые семейства SOC с ядром ARM Cortex-A:

Производитель	Семейства
AMD/Xilinx	Zynq-7000 (dual-core Cortex-A9)
Broadcom	BCM2836 (quad-core Cortex-A7)

Электрические параметры:

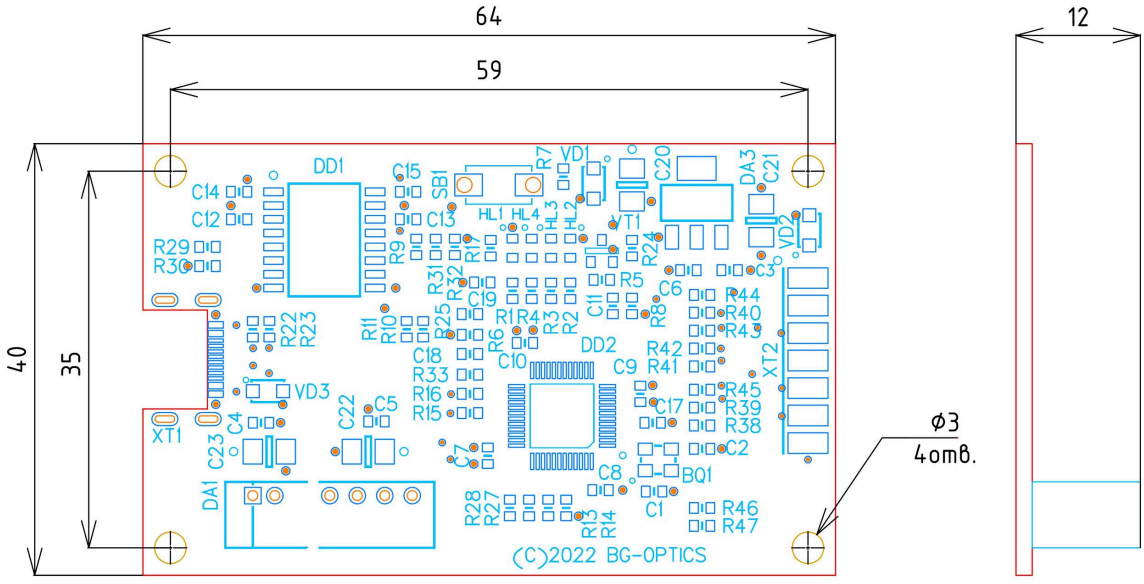
Параметр	Обозначение	Значение	Ед. изм.
Напряжение пробоя барьера изоляции	Uiso	1.5 (5.7*)	кВ
Напряжение питания (шина USB)	Uпит	5	В

+7 (495) 009-10-35
ksi@csc.ru
infra.csc.ru

Параметр	Обозначение	Значение	Ед. изм.
Рабочая температура	Top	-25...+60	°C
Максимальный ток питания внешней нагрузки 3.3 В	Itarget_max	150**	мА
Максимальный потребляемый ток по шине USB	Iop_usb	200***	мА
Масса	m	0.01	кг
Проходная емкость USB развязки, сигнальные линии	Ccross_usb	2.2	пФ
Проходная емкость преобразователя питания	Ccross_dc/dc	60	пФ

* Исполнение с преобразователем питания на дискретных элементах с напряжением пробоя изолирующего барьера не менее 5.7 кВ.
** Максимальный ток питания целевого модуля зависит от исполнения преобразователя напряжения.
*** Максимальный ток потребления по шине USB зависит от исполнения преобразователя напряжения.

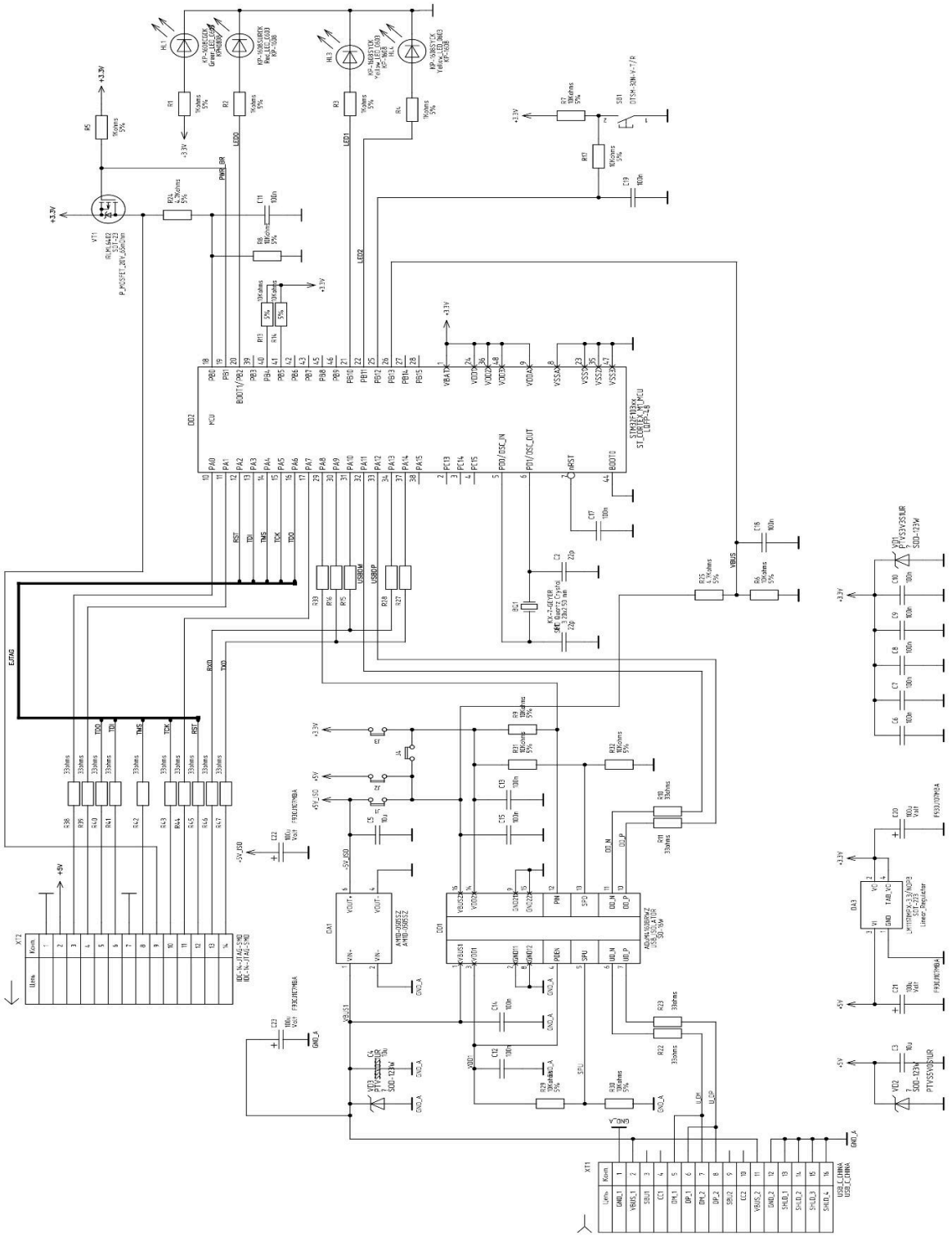
Эскиз изделия:



+7 (495) 009-10-35
ksi@csc.ru
infra.csc.ru

Модуль изготавливается в виде печатной платы с монтажными отверстиями. Исполнение без корпуса позволяет монтаж изделия в плату/шасси отлаживаемой системы

Электрическая схема изделия:



+7 (495) 009-10-35
ksi@csc.ru infra.csc.ru

Электрический разъём:

Назначение выводов в режиме JTAG:

2	4	6	8	10	12	14
UART_RX self swd_io	AUX_PIN_04	+3.3V_OUT/ VSENSE	GND	TDO	AUX_PIN_12	GND
UART_TX self swd_clk	nRST	TCK	TMS	TDI	AUX_PIN_11	+5V OUT
1	3	5	7	9	11	13

Назначение выводов в режиме SWD/SWDIO:

2	4	6	8	10	12	14
UART_RX self swd_io	AUX_PIN_04	+3.3V_OUT/ VSENSE	GND	SWO	AUX_PIN_12	GND
UART_TX self swd_clk	nRST	SWCLK	SWDIO	AUX_PIN_ 09	AUX_PIN_11	+5V OUT
1	3	5	7	9	11	13

Схема включения: TODO

+7 (495) 009-10-35
ksi@csc.ru infra.csc.ru

Руководство по использованию:

Термины и определения

Изделие является комбинацией оборудования и программного обеспечения к нему. В связи с тем, что отладчик является результатом доработки ПО Black Magic Debug(**BMD**) и переделки Black Magic Probe(**BMP**), приведём немного истории по этому проекту. Black Magic Debug (**BMD**) изначально был разработан как просто код прошивки для специально созданного оборудования под названием Black Magic Probe (**BMP**), которое может вести последовательный обмен с GNU Debugger (**GDB**). Исходное оборудование **BMP** также называют «родным», поскольку со временем были добавлены дополнительные хост-платформы, на которые можно было загрузить прошивку (таким хост-оборудованием может быть плата disco-stm32f4 или оборудование ST-Link и многое другое). Кроме того, **BMD** также может быть скомпилирован как отдельное приложение для ПК (Black Magic App или **BMA** и официально известно как **pc-hosted**), которое может действовать как промежуточное программное обеспечение, аналогично тому, как работает **OpenOCD**. В этом случае **BMP** действует только как транслятор **USB** в **JTAG/SWD**, а Black Magic App (**BMA**) выполняет всю тяжелую работу. Кроме того, это также позволяет использовать **BMA** с другим аппаратным обеспечением отладчиков, несовместимых с BMD, например таких как: JLink, ST-Link, sw-link, FTDI и т. д.

- **Black Magic Debug** (**BMD**) — это название всего проекта, но также выступающий в качестве термина, описывающего программное обеспечение (код прошивки) для аппаратного обеспечения отладчика.
- **Black Magic Firmware** (**BMF**) — специфичная сборка кода **BMD** под определенную плату изделия.
- Код прошивки Black Magic (**BMF**). Специфически собранный(компиляция и линковка с указанием аппаратной реализации) код **BMD**, загружаемый на оборудование **Black Magic Probe (BMP)**, либо на любое другое поддерживаемое хост-оборудование как в данном случае — аппаратное обеспечение отладчика, изделие **1Б2541.003**.
- **Black Magic Application**(**BMA**), так же называемое как **pc-hosted,** система отладки **Black Magic**, скомпилированная как приложение для ПК, а не как прошивка для аппаратной части отладчика. Данный вариант отладки, позволяет выполнять код **BMD** на ПК, вместо того, чтобы выполнять его на аппаратном устройстве (**hosting** в контексте проекта).
- **Black Magic Probe** (**BMP**) — родное аппаратное обеспечение, которое было специально разработано и предназначено для проекта **BMD**. Это аппаратное обеспечение предлагается на таких интернет площадках, как 1BitSquared, Adafruit.
- **Black Magic Core** (**BMC**) — основная часть кода проекта **BMD**, который реализует протокол JTAG/SWD, функции загрузчика (старт, обновление прошивки), обнаружение **целевых устройств** и все функции, специфичные для них. Он не включает специфичный для хост-платформы код,

например, код, который переключает GPIO и реализует код расширений.

- **Хост-платформа** — определяет тип оборудования, на котором работает программное обеспечение **BMD** (прошивка или приложение). Это может быть ПК, STM32, lm4f или какое-то более специфическое оборудование, такое как **BMP**, ST-Link, sw-link и тд.
- **Зонд, отладочный кабель** — относится к оборудованию, которое подключено между ПК, на котором работает **GDB** (и/или **BMA**), и **целевым устройством**. В простейшем случае это преобразователь USB в JTAG/SWD, такой как микросхема FTDI, но в большинстве случаев это специальное оборудование, работающее либо с **BMF**, либо с какой-либо другой прошивкой. В Нашем случае это аппаратное обеспечение отладчика, изделие **1Б2541.003**.
- **Целевое устройство (Target)** - аппаратное обеспечение, которое разработчик хочет отлаживать с помощью **BMD**.

Далее и везде для упрощения под отладчиком будем понимать **GNU GDB**, под аппаратным обеспечением отладчика **BMP** — в нашем случае изделие **1Б2541.003.** Под програмным обеспечением **BMD**, будем понимать специально разработанную версию прошивки для **1Б2541.003.**

Команды GDB

На этой странице представлен краткий обзор наиболее часто используемых команд **GDB** с **BMP**. Он предоставляется только в качестве введения, для получения более полной информации следует обращаться к руководству по **GDB**. Онлайн-справочная система **GDB** является хорошим источником информации.

Используйте команду help <command> для получения дополнительной информации о любой команде GDB.

Подключение GDB к BMP и целевому устройству отладки.

Предполагается,что **BMP** подключен кабелем USB к ПК, а целевое устройство по кабелю JTAG/SWD к **BMP** и запитано.

Подключение GDB к BMP

```
target extended-remote <port>
```

Эта команда подключает **GDB** к **BMP** для удаленной отладки целевого устройства. Он не выполняет никаких действий JTAG или SWD и не подключает GDB к процессору целевого устройства. Параметр порта должен быть именем последовательного интерфейса сервера **GDB**, в роли которого выступает **BMP**.

В Linux:

В современных системах Linux имеется так называемый функционал udev, который выполняет работу по идентификации, старту и останову динамически подключаемого устройства. При наличии файла udev-правил, системе можно сообщить некоторую информацию о подключаемом устройстве и упростить определение имени файла последовательного устройства. В комплекте с изделием предоставляется соответствующий файл 99-kgp-tools.rules, который необходимо скопировать в /etc/udev/rules.d/99-kgp-tools.rules:

```
#           1Б2541.003 (BMP)
# устройство реализует два интерфейса: к GDB и к UART
# скопируйте этот файл udev-правил в /etc/udev/rules.d/99-kgp-tools.rules
# выполните обновление правил sudo udevadm control -R
ACTION!="add|change", GOTO="bmp_rules_end"
SUBSYSTEM=="tty", ACTION=="add", ATTRS{interface}=="Black Magic GDB Server",
SYMLINK+="kgp/bmp"
SUBSYSTEM=="tty", ACTION=="add", ATTRS{interface}=="Black Magic UART Port",
SYMLINK+="kgp/uart"
SUBSYSTEM=="usb", ATTR{idVendor}=="1d50", ATTR{idProduct}=="6017", MODE="0666",
GROUP="plugdev", TAG+="uaccess"
SUBSYSTEM=="usb", ATTR{idVendor}=="1d50", ATTR{idProduct}=="6018", MODE="0666",
GROUP="plugdev", TAG+="uaccess"
LABEL="bmp_rules_end"
```

после обновления udev-правил и переподключения кабеля USB должны появится файлы двух устройств /dev/kgp/bmp и /dev/kgp/uart

```
(gdb) target extended-remote /dev/kgp/bmp
```

в OS X:

```
(gdb) target extended-remote /dev/cu.usbmodemE2C0C4C6
```

в Windows:

```
(gdb) target extended-remote \\.\COM10
```

в последних двух случаях необходимо будет уточнить имена последовательных устройств.

Список основных команд BMP

```
(gdb) monitor help
General commands:
        version -- Display firmware version info
        help -- Display help for monitor commands
        jtag_scan -- Scan JTAG chain for devices
        swdp_scan -- Scan SW-DP for devices
        targets -- Display list of available targets
        morse -- Display morse error message
        connect_srst -- Configure connect under SRST: (enable|disable)
        hard_srst -- Force a pulse on the hard SRST line - disconnects target
        debug_bmp -- Output BMP "debug" strings to the second vcom: (enable|disable)
```

Список доступных команд зависит от типа подключенного целевого устройства. После подключения к определенным целевым устройствам список доступных команд дополняется специфическими командами. Например, STM32 предоставляют `(GDB) monitor erase_mass`, который недоступен, пока вы не

просканируете доступные целевые устройства и не подсоединитесь к STM32 командой `(GDB) attach <n>`

Сканирование и подсоединение к целевому устройству

для физического подключения по интерфейсу JTAG

```
(GDB) monitor jtag_scan
```

для физического подключения по интерфейсу SWD

```
(GDB) monitor swdp_scan
```

Эти команды **BMP** выполняют сканирование отладочных цепочек подключенных целевых устройств с использованием интерфейсов JTAG или SWD и вывод списка доступных для подсоединения. Любое подсоединённое целевое устройство будет отключено этой командой.

Изделие 1Б2541.003 содержит изолирующий преобразователь напряжения, обеспечивающий развязку по питанию от шины USB 5В. Выход преобразователя 5В выведен на контакт отладочного разъема и позволяет запитывать 5В маломощные устройства, такие как отладочные платы. Также имеется возможность по команде подключать на соответствующий контакт отладочного разъема выход стабилизатора напряжения 3.3В

```
(GDB) monitor tpwr enable
```

по умолчанию выход стабилизатора напряжения 3.3В отключен. Данная команда позволяет подключать и отключать выход стабилизатора.

Подсоединенение к целевому устройству

```
(GDB) attach <n>
```

Команда логического подсоединения **GDB** к целевому устройству, подключенному физически по интерфейсу JTAG/SWD к **BMP**. Аргумент должен быть идентификатором цели, отображаемым в списке, выведенном командой сканирования. Подсоединение не приведет к сбросу целевого устройства, для этого используйте команды `(GDB) run` или `(GDB) start`

Разрешение доступа к адресам, отображаемым в область памяти ввода-вывода из GDB

```
(GDB) set mem inaccessible-by-default off
```

Эта команда разрешает **GDB** доступ к памяти за пределами известной карты памяти устройств, чтобы обеспечить доступ в область памяти ввода-вывода.

Поскольку вышеперечисленные команды являются общими, которые вы, вероятно, будете запускать каждый раз при запуске **GDB**, вы можете включить их в свой файл .gdbinit. Если текстовый файл с именем .gdbinit скопировать в домашний каталог или каталог, из которого вы запускаете **GDB**, он

автоматически его считает и выполнит эти команды при запуске. Так же практически все интегрированные среды разработки (**IDE**) позволяют настроить скрипт запуска **GDB** для каждого проекта в отдельности.

Загрузка программы в память целевого устройства

```
(GDB)load [filename]
```

Эта команда запишет двоичный файл программы в память целевого устройства. Если адреса программы размещены в флэш-памяти, флэш-память будет стерта и запрограммирована.

```
(GDB) compare-sections
```

Эта команда сравнит контрольные суммы CRC32 целевой памяти и секций двоичного файла, сообщит, если будет обнаружено несоответствие. В связи с особенностями стирания и ограниченного ресурса полезно проверять результат программирования флэш-памяти.

Отладка с помощью GDB

Запуск программы

```
(GDB) start
```

```
(GDB) run
```

Эти команды сбрасывают и начинают выполнение программы на целевом устройстве. `(GDB) run` позволяет целевому объекту продолжить выполнение команд программы до тех пор, пока не произойдет прерывание или другое событие, в то время как `(GDB) start` выполнит сброс устройства, исполнение кода инициализации и обеспечит останов при входе в функцию **main**.

В ходе исполнения программы, ее можно прервать, набрав **Ctrl-C** в консоли **GDB** или отправив **GDB** сигнал **SIGINT**.

Установка условий остановки

```
(GDB) break <function>
(GDB) break <file>:<line>
(GDB) watch <var>
```

Эти команды устанавливают точки останова и точки наблюдения. Точки останова прерывают выполнение программы при достижении определенного места в коде, а точки наблюдения прерывают программу при изменении определенной переменной.

Вывод состояния программы и целевого устройста

```
(GDB) print <expr>
(GDB) x <address>
```

+7 (495) 009-10-35
ksi@csc.ru infra.csc.ru

```
(GDB) info registers
```

Эти команды могут использоваться для проверки переменных, памяти или основных регистров процессора.

```
(GDB) dump ihex memory <file> <start> <stop>
```

Эту команду можно использовать для создания дампа содержимого памяти целевого устройства в шестнадцатеричный файл формата Intel hex. Существуют и другие варианты для альтернативных форматов, используйте команду `(GDB) help dump` для получения дополнительной информации.

Отсоединение от целевого устройства

```
(GDB) detach
(GDB) kill
```

Эти команды можно использовать для отключения **BMP** от целевого устройства. `(GDB) kill` также сбросит цель при отсоединении.

Автоматизация програмирования FLASH Automation

Во многих случаях полезно автоматически прошивать и тестировать загруженную прошивку. GDB имеет встроенный язык сценариев, а также привязки к Python.

Вот несколько способов автоматизировать процесс программирования FLASH.

Командная строка

Одной командой:

```
arm-none-eabi-gdb -nx --batch \
  -ex 'target extended-remote /dev/ttyACM0' \
  -ex 'monitor swdp_scan' \
  -ex 'attach 1' \
  -ex 'load' \
  -ex 'compare-sections' \
  -ex 'kill' \
  firmware.elf
```

Замечание: имя файла отладчика-эмулятора отличается для различных операционных систем, так например `/dev/ttyACMx` для Linux, в Windosw имя файла имеет вид `COMx`, и `/dev/cu.usbmodemX` для Mac OS.

Замечание: в случае использования интерфейса JTAG необходимо использовать команду `monitor jtag_scan`, в случае интерфейса SWD команду `monitor swdp_scan`.

Замечание: изделие позволяет запитывать маломощную нагрузку со стороны высоковольтного сегмента напряжением 3.3 вольта. Это удобно в случае отладки малопотребляющих целевых модулей и отладочных плат, не прибегая к автономному питанию в высоковольтном сегменте. Так же, данная возможность полезна

в случае не-3.3В логики на целевом устройстве, для запитки интерфейсных трансляторов уровней. Для подачи питания необходимо выполнить команду `monitor tpwr enable`. Следует учитывать, что максимальный ток потребления запитываемого устройства не должен превышать указаного в характеристиках значения.

Приведенный выше вызов командной строки можно упростить, если поместить большинство команд -ex в файл сценария GDB. Вы можете создать файл с именем, например, bmp_probe_flash.scr со следующим содержимым:

```
monitor swdp_scan
attach 1
load
compare-sections
kill
```

Затем приведенный выше вызов сводится к следующему:

```
arm-none-eabi-gdb -nx --batch \
  -ex 'target extended-remote /dev/ttyACM0' \
  -x bmp_probe_flash.scr \
  firmware.elf
```

Makefile

Если вы используете GNU Makefiles для своего проекта, вы можете добавить `make flash` цель. Повторное использование сценария из раздела командной строки над целью может выглядеть примерно так:

```
firmware: firmware.flash

BMP_PORT ?= /dev/ttyACM0

%.flash: %.elf
        @printf "  BMP $(BMP_PORT) $(*).elf (flash)\n"
        $(Q)$(GDB) -nx --batch \
                   -ex 'target extended-remote $(BMP_PORT)' \
                   -x bmp_probe_flash.scr \
                   $(*).elf
```

в данном случае предполагается, что имя двоичного файла заканчивается суффиксом .elf. Вы можете вызвать цель, запустив make flash, и она создаст firmware.elf, прошьет его и проверит, что flash запрограммирован.

Также можно явно указать файл последовательного устройства:

```
make flash BMP_PORT=/dev/ttyACM2
```

Пакетный файл команд .bat

Примерно также сделать подобные действия можно в среде ОС Windows с помощью пакетного `.bat` файла:

```
@echo off
rem: Note %~dp0 get path of this batch file
rem: Need to change drive if My Documents is on a drive other than C:
set driverLetter=%~dp0
set driverLetter=%driverLetter:~0,2%
%driverLetter%
cd %~dp0
rem: get all parameters that we will be using and make sure the slashes are all correct
set working_directory=%~p0
set toolchain_path=%~1
set toolchain_path=%toolchain_path:/=\%
set bmp_gdb_port=%2
set bmp_gdb_port=%bmp_gdb_port:/=\%
set elf_file=%3
set elf_file=%elf_file:/=\%
%toolchain_path%\arm-kgp-eabi-gdb.exe --batch -nx ^
        -ex "target extended-remote %bmp_gdb_port%" ^
        -x %working_directory%..\shared\bmp_gdb_upload_swd.scr ^
        %elf_file%
```

Приведенный выше скрипт принимает на вход три параметра: `toolchain_bin_dir`, `debug probe GDB port`, `binary elf file`. Пример его запуска в консоли:

```
bmp_upload.bat C:\\gcc-arm-kgp-eabi\bin COM1 firmware.elf
```

Программирование при серийном производстве

Вы можете использовать изделие для массового программирования большого количества устройств в производстве. Самый простой способ добиться этого — использовать один из описанных выше методов и запустить их в цикле. Также полезно иметь звуковой сигнал, когда процесс флэш-памяти прошел успешно. Вы можете добавить воспроизведение аудиофайла в конце скрипта GDB, и оно будет воспроизводиться только после успешного цикла флэш-памяти.

В среде ОС семейсва UNIX, часть скрипта, выполняющего подобные действия будет выглядеть так:

```
while true; do sleep 1; arm-none-eabi-gdb --batch -nx \
  -ex "target extended-remote /dev/ttyACM0" \
  -x gdb_upload.scr \
  firmware.elf; done
```

Замечание: полезно иметь короткую задержку между вызовами gdb, иначе трудно прервать цикл комбинацией клавиш Ctrl-C. Большинство современных систем UNIX также поддерживают более короткие периоды, чем 1 сек, в качестве параметра команды sleep. Вы можете использовать `do sleep 0,5;` для полусекундного периода задержки.

Строки скрипта звуковой сигнализации для Linux будут выглядеть следующим образом:

```
paplay /usr/share/sounds/sound-icons/start
```

```
sleep 3
```

В среде Mac OS:

```
afplay /System/Library/Sounds/Ping.aiff
sleep 3
```

Замечание: дополнительные 3 секунды сна после успешной загрузки прошивки дают оператору время, чтобы отключить изделие от программируемого устройства без случайного многократного перепрограммирования.

Замечание: Если вы отсоедините кабель в середине процесса прошивки, вы можете получить наполовину запрограммированную или повторно стертую FLASH программируемого устройства.

Полный скрипт, который должен работать на компьютере с Ubuntu, с подробными комментариями:

```
# Print BMPM version
monitor version
# To make sure the target is not in a "strange" mode we tell BMPM to reset the
# target using the reset pin before connecting to it.
monitor connect_srst enable
# Enable target power (aka. provide power to the target side of the level shifters)
monitor tpwr enable
# Scan for devices using SWD interface
monitor swdp_scan
# Alternatively scan for devices using JTAG. (comment out the above line...)
# monitor jtag_scan
# Attach to the newly found target if available. (if it fails the script exits)
attach 1
# Success! Lets make some sound!
shell paplay /usr/share/sounds/ubuntu/stereo/message.ogg
# Load aka. flash the binary
load
# Check if the flash matches the binary
compare-sections
# Reset the target and disconnect
kill
# Write to the terminal that we succeeded
echo "Upload success!!!"
# Finished flashing success! Lets make some more sound!
shell paplay /usr/share/sounds/ubuntu/stereo/system-ready.ogg
```

Автоопределение имени COM порта интерфейса GDB в Windows PowerShell

Вручную определять имя последовательного порта и каждый раз вводить его в вызов сценария не всегда удобно. Следующая команда в Windows PowerShell обнаружит порт GDB и установит переменную среды, которую затем можно будет использовать для вызова.

```
Get-CimInstance -ClassName Win32_SerialPort -Filter "PNPDeviceID like 'USB\\
VID_1D50&PID_6018&MI_00%'" | `
  Select -ExpandProperty DeviceID | Set-Variable -Name BMP_GDB_PORT
```

```
echo $BMP_GDB_PORT
```

Semihosting

Semihosting позволяет целевой системе выполнять ввод с клавиатуры, вывод на экран и файловый ввод-вывод на хосте. Для реализации данной возможности необходимо в целевом коде вызывать стандартный набор функций **open(), close(), read(), write(), lseek(), rename(), unlink(), stat(), isatty(), system()**, которые будут выполнены на хосте отладки, изделие и GDB обеспечать прозрачный транспортныей интерфейс между вызовом функции в коде на целевой машине и ее исполнением на хосте. Это очень похоже на удаленный вызов процедур. Код реализации этих функций и сопутствующего функционала обычно является частью реализации **libc**, для линковки этого кода в целевой необходимо линкеру передать флаги

```
LD_FLAGS += --specs=rdimon.specs  -lrdimon
```

Если вы собираетесь использовать **stdin**, **stdout** или **stderr** (например, через **printf()/scanf()**) и не используете среду выполнения **newlib** (путем указания **-nostartfiles**), вам нужно добавить это в свою инициализацию:

```
void initialise_monitor_handles(void);
initialise_monitor_handles();
```

Пользователи Arm Arduino могут использовать полухостинговую библиотеку из диспетчера библиотек.

Детальное описание данного механизма и его реализация дается в документации ARM, описывающей низкоуровневый интерфейс, который должен реализовать целевой объект:

- *Semihosting for ARM* PDF HTML

SWD / TRACESWO

Изделие в режиме Serial Wire Debug (не JTAG) может получать диагностический поток TRACESWO.

Чипы ARM Cortex включают в себя комплексную структуру трассировки, вкратце, есть встроенная макроячейка трассировки (ETM), которая может генерировать выходные данные трассировки для различных видов событий, таких как точки наблюдения, и есть макроячейка инструментальной трассировки (ITM), которая аналогична, но ориентирована к сообщениям журнала, сгенерированным программным обеспечением.

Инструментарий можно оставить включенным в поставленном продукте. Потоки ETM и ITM объединяются в TPIU, который генерирует серию кадров в указанном формате, этот поток может быть отправлен по 1-проводной, 2-проводной или 4-проводной синхронной последовательной шине, если это разрешено в вашем проекте, или его можно отправить на вывод TRACESWO, который представляет собой асинхронный последовательный поток 1-Wire. Выход TRACESWO может быть в формате UART (старт, 8 битов данных, стоп) или в формате манчестерской кодировки.

Изделие поддерживает только манчестерский кодированный формат, что является разумным выбором, поскольку позволяет автоматически синхронизироваться со скоростью передачи данных цели (в пределах некоторого диапазона скоростей передачи данных). Обратите внимание, что изделие использует для этой задачи по существу UART только с битовым входом, реализованный в программном обеспечении с использованием вывода захвата входа таймера, и не поддерживает высокие скорости передачи данных. В основном контакт TRACESWO будет использоваться как удобный способ получения вывода в стиле printf() для хоста отладчика, не тратя впустую контакт ввода-вывода, назначив для этой функции выделенный UART. Схема ETM/ITM/TPIU перегружена для этой задачи, поэтому имеет смысл при этом отключить форматирование в TPIU. Это дает выходной поток, состоящий из 16-битных слов, каждое из которых содержит идентификатор потока и символ.

Простое приложение Linux для вывода вывода TRACESWO на стандартный вывод.

https://github.com/nickd4/bmp_traceswo

Он также содержит пример кода для вставки в приложение STM32, показывающий, как настроить и использовать ITM и TPIU для вывода в стиле **printf()**.

Обратите внимание, что случай STM32 немного необычен (содержит так называемый Pelican TPIU по адресу, не указанному ARM). Код в примере взят отсюда:

https://mcuoneclipse.com/2016/10/17/tutorial-using-single-wire-output-swo-with-arm...

Еще один пример, он также показывает, как включить вывод TRACESWO, отсутствующий в примере выше:

http://forum.segger.com/index.php?page=Thread&threadID=608

Основные IDE также могут интерпретировать пакеты TRACESWO, в частности форматирование, которое может быть более подходящим для расширенного использования, с отдельными окнами для разных потоков, поиском, фильтрацией и т. д.

Также следует изучить форк bmp_traceswo

https://github.com/tristanseifert/bmp_traceswo

Оригинальная версия брала по одному слову из потока TRACESWO, игнорируя старший байт и сбрасывая младший байт, в то время как эта модификация, похоже, принимает кадры. Если будет использоватся приведенный форк, необходимо взять скрипты правил udev **/etc/udev/rules.d/99-blackmagic.rules** с

https://github.com/nickd4/bmp_traceswo

Все данные передаются через один контакт JTAG TDO. С точки зрения разработчика встроенного ПО это выглядит так — в целевом коде выполняется настройка периферийного устройства, физически реализованного в отладочной части (TPIU) ядра процессора запись в его регистр данных потока байт. Настроенное изделие по линии SWO принимает поток данных, декодирует и заносит в буфер. В случае, если на хосте запущена программа чтения этого потока, она может открыть соответствующий

интерфейс (в терминах USB) и проводить чтение данных из этого буфера. В простейшем случае необходимо:

- соединить целевое устройство и изделие, включая линию SWO
- в целевом коде сконфигурировать отладочную периферию ядра для использования SWO
- сконфигурировать изделие для приема данных SWO
- запустить внешнее приложение, которое будет вычитывать с изделия SWO буфер и обрабатывать данные

PlatformIO

PlatformIO- экосистема с открытым исходным кодом для разработки IoT с кросс-платформенной системой сборки, менеджером библиотек и полной поддержкой Black Magic Probe и его клонов. Он работает на популярных ОС: macOS, Windows, Linux 32/64, Linux ARM (например, Raspberry Pi, BeagleBone, CubieBoard).

Поддерживаемые платформы: Atmel AVR, Atmel SAM, Espressif 32, Espressif 8266, Freescale Kinetis, Intel ARC32, Lattice iCE40, Maxim 32, Microchip PIC32, Nordic nRF51, Nordic nRF52, NXP LPC, Silicon Labs EFM32, ST STM32, Teensy, TI MSP430, TI Tiva, WIZNet W7500

SDK и библиотеки: Arduino, ARTIK SDK, CMSIS, Energia, ESP-IDF, libOpenCM3, mbed, Pumbaa, Simba, SPL, STM32Cube, WiringPi

PIO Unified Debugger

PIO Unified Debugger-решение для нескольких архитектур и платформ разработки, не требует конфигурации. Он поддерживает более 200 встроенных плат и самых популярных IDE, такие как:

- VSCode
- Atom
- Eclipse
- Sublime Text

Полный список совместимых отладочных плат: compatible boards for Black Magic Probe (“Debug” column).

Быстрый старт

1. Установка PlatformIO IDE
2. Создание нового проекта “PlatformIO Home > New Project”
3. Редактирование **Project Configuration File** platformio.ini,установка BMP средством по умолчанию для отладки, чтения и записи flash:

```
[env:someboard]
```

```
platform = …
framework = ...
board = …
debug_tool = blackmagic
upload_protocol = blackmagic
upload_port = !!!UPDATE_ME!!!
monitor_port = !!!UPDATE_ME!!!
```

4. Запуск отладочной сессии:
    - PlatformIO IDE for Atom: Menu > PlatformIO > Debug > Start a debug session
    - PlatformIO IDE for VSCode: Menu > Debug > Start Debugging

Отладка в командной строке CLI

Дополнительные сведения см. в документации по командам отладки платформы.

Расширенная конфигурация

Пожалуйста, посетите официальную документацию PlatformIO для расширенной конфигурации отладки.

STM32

Изделие поддерживает множество микросхем от многих производителей. Несмотря на попытки обеспечить унифицированный способ обработки целей, существуют некоторые особенности. На этой странице описаны некоторые из них.

Программирование памяти OTP STM32G0xx

Многие микросхемы STM32 имеют область флэш-памяти с возможностью однократного программирования. Эта область обычно используется для хранения данных только для чтения, например. безопасные ключи или любые очень стабильные энергонезависимые данные. Эта область не предназначена для хранения кода программ.

К семейству STM32 с поддержкой OTP относятся STM32G0xx .

Не рекомендуется записывать область OTP из GDB как минимум по двум причинам:

- Из-за разного размера страницы/блока области OTP по сравнению с основной флэш-памятью программ можно заметить ошибочное поведение GDB в случае программирования файла в области с разнородными размерами блоков (в частности, некоторые разделы программного кода отображаются разделенными при отправке через отладчик). Программирование такого файла, вероятно, не удастся.
- OTP по определению программируется один раз, тогда как использование GDB многократно.

+7 (495) 009-10-35
ksi@csc.ru infra.csc.ru

По этим причинам предпочтительным способом программирования одноразовых паролей является **hosted BMA**.

Установка **Read Out Protection level 2** и **OTP** программирование в STM32G0xx являются необратимыми, для дополнительной защиты реализованы команды

```
$ blackmagic -M "irreversible enable|disable"
```

Примечание: эта команда монитора может измениться в любое время в будущем.

Затем необходимо явно указать начальный адрес записи OTP, чтобы предотвратить использование BMD по умолчанию в область 0x08000000 флэш-памяти(код программ). Для STM32G0xx адрес области OTP 0x1FFF73E0. Обратите внимание, что любые данные размером, кратным 8 байтам, могут быть записаны в 8-байтовый выровненный свободный адрес (еще не запрограммированный данными, отличными от 0xFF). Следовательно, можно запрограммировать часть всей области OTP, а остальную часть (или часть остального) запрограммировать позже:

```
$ blackmagic -M "irreversible enable" -a 0x1FFF7000 otp_1024bytes.bin
Irreversible operations: enabled
Flash Write succeeded for 1024 bytes,    x.yyy kiB/s
```

пример программирования 8-байтного блока по адрессу 0x1FFF73E0 :

```
$ blackmagic -M "irreversible enable" -a 0x1FFF73E0 otp_1024bytes.bin
Irreversible operations: enabled
Flash Write succeeded for 8 bytes,    1.143 kiB/s
```

Подготовка OTP бинарных данных

Пример определения данных OTP в исходном файле C:

```
/* OTP */
const uint32_t otp_tab[2] __attribute__ ((section (".otp"))) = {
        0xAABBAABB, 0xCCDDCCDD
};
```

Скрипт компоновщика можно заполнить разделом .otp следующим образом:

```
MEMORY
{
        /* rom and ram definitions here */
        otp (r) : ORIGIN = 0x1FFF7000, LENGTH = 1K
}
SECTIONS
{
        .otp : {
                . = ALIGN(8);
                KEEP(*(.otp))
                . = ALIGN(8);
        } >otp
}
```

+7 (495) 009-10-35
ksi@csc.ru infra.csc.ru

После создания файла в формате elf, выходной раздел .otp можно извлечь с помощью **objcopy** пакета **binutils** :

```
$ arm-none-eabi-objcopy -j ".otp" -O binary firmware.elf otp_data.bin
```

Справочные материалы:

Проекты связанные с BMP:

- atom-gdb-debugger - GDB integration for Github's Atom editor
- gcc-arm-embedded - GNU ARM Embedded Toolchain
- libopencm3 (on Github) - Open source Cortex-M microcontroller library (examples)
- gdb - TheGNUProjectDebugger
- dfu-util - Device Firmware Upgrade Utilities
- stm32flash - Flash STM32 over built-in serial bootloader

Среды разработки печатных плат:

- KiCad
- gEDA/PCB

ПО, связанное с отладкой:

- OpenOCD - Open On-Chip Debugger
- pyOCD - Open source python library for programming and debugging ARM Cortex-M microcontrollers using CMSIS-DAP
- probe-rs - embedded debugging toolkit written in Rust
- pystlink - Python tool for flashing and debugging STM32 devices using ST-LINK/V2
- texane/stlink - STM32 Discovery Line Linux Programmer

Описание протокола MCU Debugger Protocols:

- CMSIS-DAP

+7 (495) 009-10-35
ksi@csc.ru infra.csc.ru

Примечание.

По требованию заказчика возможно внесение конструктивных изменений в изделие (например: изменение габаритов, посадочных размеров, схемы питания, напряжений логических уровней и т. д.) и модификация его программного обеспечения для расширения функциональных и эксплуатационных возможностей.

Возможно исполнение изделия с повышенным напряжением пробоя изолирующего барьера до 5,7 кВ и повышенной мощностью выходного питания по цепям 5 и 3,3 вольта.

История версий документа:

Версия 0.1 | Дата: 10.03.2023
Черновой вариант, верстка.

Версия 0.2 | Дата: 20.03.2023
Добавлено описание цепочки отладки, история создания, термины и определения.

Версия 0.3 | Дата: 15.04.2023
Добавлено описание способов автоматизации программирования FLASH.

Версия 0.4 | Дата: 11.06.2023
Изменен эскиз изделия и электрическая схема на актуальную.

Версия 0.5 | Дата: 12.06.2023
Исправлена масса обнаруженные орф. ошибок (caxapa.ru: Ыыукпу, pavel2000).

Версия 0.6 | Дата: 13.06.2023
Ошибки по сноскам в таблице основных параметров (caxapa.ru: Dingo).

Версия 0.7 | Дата: 13.06.2023
Изменен эскиз изделия и электрическая схема на актуальную.

+7 (495) 009-10-35
ksi@csc.ru infra.csc.ru

TODO: doc
Описание механизма RTT;
интерфейс симулятора внешней среды.

TODO: software
Реализация интерфейса симулятора внешней среды.

TODO: hardware
- Нанести маску с обозначением распиновки разъема;
- добавить 2 канала DAC 12бит 0..3.3V;
- добавить 8 каналов ADC 12бит 0..3.3V;
- добавить двунаправленный выход питание 5V;
- добавить выход питание 3.3V;
- перепланировать разъем XX (вывод/вывод новых сигналов, удобное положение питания и земли);
- выполнить переход на stm32f411.