**USB-UART-FT232 (в режиме BitBang) в качестве JTAG программатора/отладчика**

1. Обновляем список и устанавливаем необходимые пакеты.

# apt update && apt-get install -y \

build-essential \

libusb-1.0-0-dev \

jimsh \

tclsh \

libusb-dev \

pkg-config \

libtool \

autoconf \

pip \

git

1. Получаем исходники OpenOCD и переходим в пиректорию с ними.

# git clone <https://github.com/ntfreak/openocd.git> && cd openocd

1. Читаем README “less README” и создаем сценарий настройки.

# ./bootstrap

Пример выхлопа:

root@openocd:~/openocd# ./bootstrap

+ aclocal --warnings=all

+ libtoolize --automake --copy

+ autoconf --warnings=all

+ autoheader --warnings=all

+ automake --warnings=all --gnu --add-missing --copy

configure.ac:22: installing 'build-aux/compile'

configure.ac:34: installing 'build-aux/config.guess'

configure.ac:34: installing 'build-aux/config.sub'

configure.ac:17: installing 'build-aux/install-sh'

configure.ac:17: installing 'build-aux/missing'

Makefile.am: installing './INSTALL'

Makefile.am: installing 'build-aux/depcomp'

Makefile.am:24: installing 'build-aux/mdate-sh'

Makefile.am:24: installing 'build-aux/texinfo.tex'

Setting up submodules

Подмодуль «jimtcl» (https://github.com/msteveb/jimtcl.git) зарегистрирован по пути «jimtcl»

Подмодуль «src/jtag/drivers/libjaylink» (https://repo.or.cz/libjaylink.git) зарегистрирован по пути «src/jtag/drivers/libjaylink»

Подмодуль «tools/git2cl» (https://repo.or.cz/git2cl.git) зарегистрирован по пути «tools/git2cl»

Клонирование в «/root/openocd/jimtcl»…

Клонирование в «/root/openocd/src/jtag/drivers/libjaylink»…

Клонирование в «/root/openocd/tools/git2cl»…

Подмодуль по пути «jimtcl»: забрано состояние «2d66360c61d2a89d4008e8bad12ae3aa5f0331e2»

Подмодуль по пути «src/jtag/drivers/libjaylink»: забрано состояние «9aa7a5957c07bb6e862fc1a6d3153d109c7407e4»

Подмодуль по пути «tools/git2cl»: забрано состояние «8373c9f74993e218a08819cbcdbab3f3564bbeba»

Generating build system...

libtoolize: putting auxiliary files in AC\_CONFIG\_AUX\_DIR, 'build-aux'.

libtoolize: copying file 'build-aux/config.guess'

libtoolize: copying file 'build-aux/config.sub'

libtoolize: copying file 'build-aux/install-sh'

libtoolize: copying file 'build-aux/ltmain.sh'

libtoolize: putting macros in AC\_CONFIG\_MACRO\_DIRS, 'm4'.

libtoolize: copying file 'm4/libtool.m4'

libtoolize: copying file 'm4/ltoptions.m4'

libtoolize: copying file 'm4/ltsugar.m4'

libtoolize: copying file 'm4/ltversion.m4'

libtoolize: copying file 'm4/lt~obsolete.m4'

configure.ac:43: installing 'build-aux/ar-lib'

configure.ac:37: installing 'build-aux/compile'

configure.ac:30: installing 'build-aux/missing'

libjaylink/Makefile.am: installing 'build-aux/depcomp'

Bootstrap complete. Quick build instructions:

./configure ....

root@openocd:~/openocd#

Скрипт должен отработать без ошибок. Если что то не хватает – доустанавливаем и запускаем bootstrap снова.

1. Указываем нужные нам опции сборки (поддержка чипа r232r, поддержка режима MPSSE и префикс директории, куды мы будем устанавливать собранный openocd)

# ./configure --enable-ft232r --enable-ftdi --prefix=$HOME/.opt

Все должно пройти без ошибок. Результат работы конфигуратора (список поддерживаемых устройств):

OpenOCD configuration summary

--------------------------------------------------

MPSSE mode of FTDI based devices yes

ST-Link Programmer yes (auto)

TI ICDI JTAG Programmer yes (auto)

Keil ULINK JTAG Programmer yes (auto)

Altera USB-Blaster II Compatible yes (auto)

Bitbang mode of FT232R based devices yes

Versaloon-Link JTAG Programmer yes (auto)

TI XDS110 Debug Probe yes (auto)

CMSIS-DAP v2 Compliant Debugger yes (auto)

OSBDM (JTAG only) Programmer yes (auto)

eStick/opendous JTAG Programmer yes (auto)

Olimex ARM-JTAG-EW Programmer yes (auto)

Raisonance RLink JTAG Programmer yes (auto)

USBProg JTAG Programmer yes (auto)

Andes JTAG Programmer yes (auto)

CMSIS-DAP Compliant Debugger no

Nu-Link Programmer no

Cypress KitProg Programmer no

Altera USB-Blaster Compatible no

ASIX Presto Adapter no

OpenJTAG Adapter no

Linux GPIO bitbang through libgpiod no

SEGGER J-Link Programmer yes (auto)

Bus Pirate yes (auto)

Use Capstone disassembly framework no

1. Собираем OpenOCD

# make

1. Что бы не засорять систему, мы создадим директорию в хомяке, куда установим OpenOCD.

Так же добавим в файл “.bashrc” путь до директории в переменную $PATH, куда будет установлен бинарник openocd. Это необходимо для того, что бы ваш интерпретатор знал, где искать исполняемый файл, установленный в ручную мимо менеджера пакетов.

# mkdir $HOME/.opt

# nano $HOME/.bashrc

Добавьте в конец файла:

if [ -d "$HOME/.opt/bin" ] ; then

PATH="$HOME/.opt/bin:$PATH"

fi

И перечитаем переменные среда командой

# source $HOME/.bashrc

1. Наконец устанавливаем собранный OpenOCD

# make install

1. Копируем свежеустановленный файл конфигурации интерфейса и берем его за основу для настройки нашего отладчика на базе микросхемы ft232rl

cp -a $HOME/.opt/share/openocd/scripts/interface/ft232r/radiona\_ulx3s.cfg $HOME/ft232rl\_jtag.conf

Открываем в нашем любимом, текстовом редакторе и допиливаем необходимый минимум.

Для этого необходимо прописать в него VID&PID нашего адаптера. VID и PID, можно посмотреть в выводе команды lsusb при подключенном (usb-uart ft232) адаптере.

root@openocd:~# lsusb

Bus 001 Device 002: ID 0403:6001 Future Technology Devices International, Ltd FT232 Serial (UART) IC

Bus 001 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub

Здесь мы видим, что в первой строке, устройство 002 на шине 001, с идентификатором производителя и устройства: 0403:6001и есть наш USB-UART Future Technology Devices International, Ltd FT232 Serial (UART) IC . Эти два шестнадцатеричных числа, нужно указать аргументами в параметре “ft232r\_vid\_pid”, не забыв указать перед ними “0x”.

Так же нужно подключить конфигурационный файл для нашего таргета (cc2538), дописав в конце

“source [ find target/cc2538.cfg ]” (сам файл лежит здесь $HOME/.opt/share/openocd/scripts/target/cc2538.cfg). Еще нужно немного подправить наш файл, что бы OpenOCD не ругался на deprecated формат конфигурации (убрать андерлайны). Настройка маппинга ножек, описывается так “ft232r tdo\_num CTS” – здесь вывод микросхемы ft232 “CTS”, используется как вывод “DO” JTAG отладчика.

В итоге, наш готовый конфиг выглядит так:

adapter driver ft232r

adapter speed 1000

ft232r vid\_pid 0x0403 0x6001

ft232r tck\_num DSR

ft232r tms\_num DCD

ft232r tdi\_num RI

ft232r tdo\_num CTS

ft232r trst\_num RTS

ft232r srst\_num DTR

source [ find target/cc2538.cfg ]

1. Подключение.

Из наших настроек, видно, что ножки ft232, линий данных последовательного интерфейса: DSR, DCD, RI, CTS, RTS, DTR, определены как линии данных JTAG: TCK, TMS, TDI, TDO, TRST, SRST соответственно.

DSR > TCK

DCD > TMS

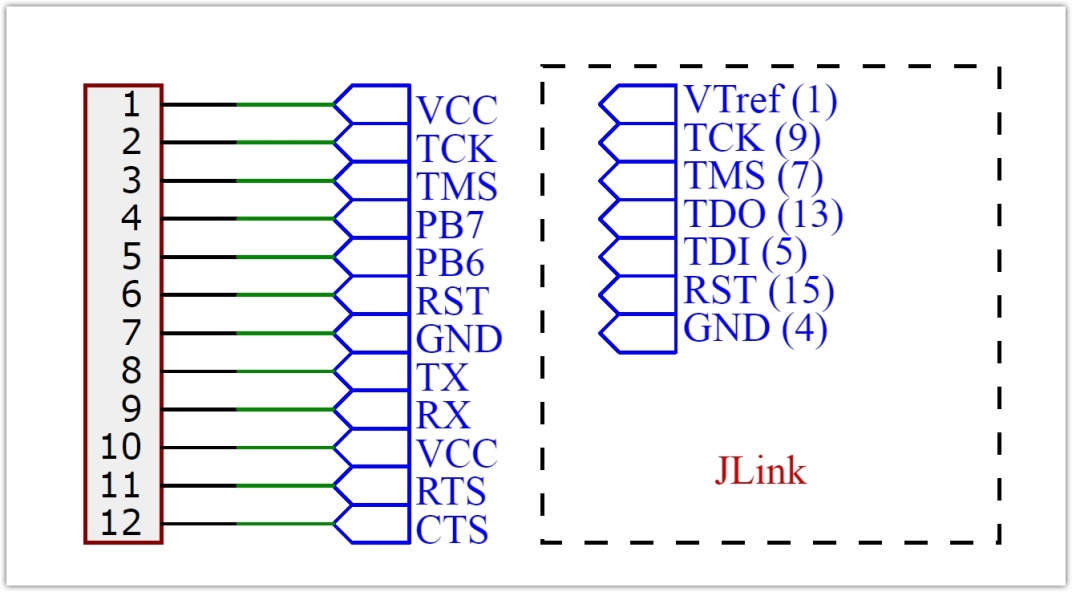
RI > TDI

CTS > TDO

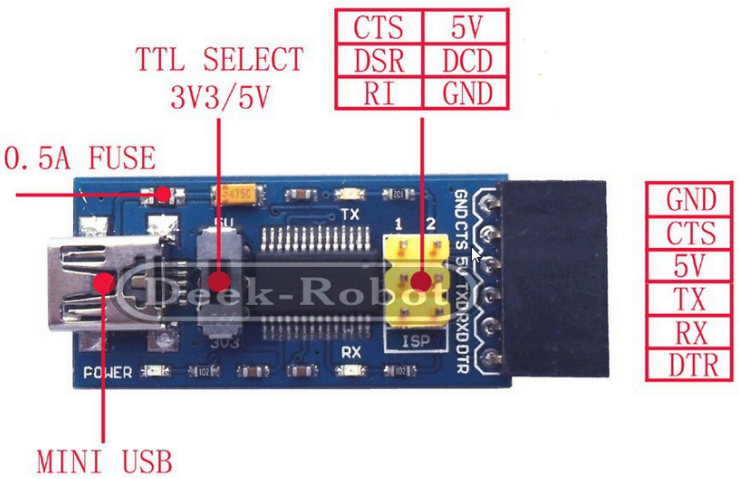
RTS > TRST

DTR > SRST

Я использую только 4 линии JTAG (TCK, TMS,), без RST – этого вполне хватит.



Мой адаптер выглядит так (только с красной маской)



Переводим переключатель уровней в положение 3V3!

Питание таргета от пина 5V – не допустимо! На нем всегда 5V, вне зависимости от положения переключателя 3V3/5V!

При питании таргета от внешнего источника тока 3.3V, не забывайте, что между отладчиком и таргетом, должна быть общая земля!

Подключаем линии данных JTAG, между нашим отладчиком (USB-UART-FT232R) и таргетом (CC2538) в соответствии с распиновкой вашего адаптера и файла настройки. Проверяем все цепи!

Подключаем отладчик в USB порт компьютера и подаем питание на таргет.

1. Запускаем OpenOCD указав ему нашу конфигурацию.

# openocd -f $HOME/ft232rl\_jtag.conf

Если видим в что то типа:

Error: JTAG scan chain interrogation failed: all ones

Error: Check JTAG interface, timings, target power, etc.

Error: Trying to use configured scan chain anyway...

Error: cc2538.jrc: IR capture error; saw 0x00 not 0x01

то еще раз проверяем соединения и все напряжения питания!

Если наш отладчик удачно определил таргет, то вывод будет примерно такой:

Info : only one transport option; autoselect 'jtag'

Info : Listening on port 6666 for tcl connections

Info : Listening on port 4444 for telnet connections

Info : clock speed 100 kHz

Info : JTAG tap: cc2538.jrc tap/device found: 0x8b96402f (mfg: 0x017 (Texas Instruments), part: 0xb964, ver: 0x8)

Info : JTAG tap: cc2538.cpu enabled

Info : cc2538.cpu: hardware has 6 breakpoints, 4 watchpoints

Info : starting gdb server for cc2538.cpu on 3333

Info : Listening on port 3333 for gdb connections

Отладчик нашел таргет и готов с ним работать. Нам нужно telnet подключение, доступное на пору 4444. Tcl доступен на порту 6666, GDB сервер на порту 3333. Открываем свободный терминал и c помощью telnet клиента, подключаемся на локальный порт 6666 (так же можно указать в конфигурации параметр “bindto 0.0.0.0” для удаленного подключения к хосту с другого компьютера):

openocd@openocd:~$ telnet localhost 4444

Trying ::1...

Trying 127.0.0.1...

Connected to localhost.

Escape character is '^]'.

Open On-Chip Debugger

>

Получаем приглашение “>” командной строки. Справка по командам доступна по “help”.

Просмотр регистров командой reg:

> reg

===== arm v7m registers

(0) r0 (/32)

(1) r1 (/32)

(2) r2 (/32)

(3) r3 (/32)

(4) r4 (/32)

(5) r5 (/32)

(6) r6 (/32)

(7) r7 (/32)

(8) r8 (/32)

(9) r9 (/32)

(10) r10 (/32)

(11) r11 (/32)

(12) r12 (/32)

(13) sp (/32)

(14) lr (/32)

(15) pc (/32)

(16) xPSR (/32)

(17) msp (/32)

(18) psp (/32)

(20) primask (/1)

(21) basepri (/8)

(22) faultmask (/1)

(23) control (/3)

===== Cortex-M DWT registers

Делаем доступным BSL загрузчик командами :

> mww 0x400D300C 0x7F800

> mww 0x400D3008 0x0205

В дальнейшем можно пользоваться [cc2538-bsl](https://github.com/JelmerT/cc2538-bsl) для загрузки прошивки в модуль через UART:

# ./cc2538-bsl.py -p /dev/ttyUSB0 -b 115200 -w ./cc2538-prog/MODKAMRU\_V3\_UART-no-flow-control.hex

Drag (ɔ)