gnICE

The gnICE is a fast and low cost USB JTAG In-Circuit-Emulator for Blackfin processors.
It's designed to provide reliable JTAG debugging and CFI NOR Flash programming via USB.
The gnICE JTAG Adapter Cable is an Open Source Hardware,
based on the FT2232 chip from Future Technology Devices International Ltd.

See also: gnICE+

Schematics can be found here:

Schematics

See the buy_stuff page for purchasing information.

This emulator does NOT work with VisualDSP++

Features:

- USB 2.0 Full Speed
- JTAG clock (TCK) 6 MHz

- Link Status LED

Supported OSes:

- Linux (any distribution)
- Mac OS X
- Windows 2000 and newer

Software

- The Blackfin toolchain fully supports this ICE

Supported Blackfin derivatives:

- All listed at features

Example gnICE gdb debugging

It's likely that your Blackfin toolchain already supports the gnICE USB Hardware. Connect the gnICE to your Hardware platform and start bfin-gdbproxy.

For more information follow the links here:

- debuggers
- gdbproxy

```
$ <strong>bfin-gdbproxy -q bfin</strong>
Initializing on FTDI device 0456:F000
IR length: 5
Chain length: 1
Device Id: 00110010011110100101000011001011 (0x00000000327A50CB)
  Manufacturer: Analog Devices
  Part(0):         BF533
  Stepping:     3
  Filename:
/opt/uClinux/bfin-uclinux/bin/../share/urjtag/analog/bf533/bf533
warning:   bfin: no board selected, BF533 is detected
notice:    bfin-gdbproxy: waiting on TCP port 2000
```

Now start your favorite debugger and connect to the target using a TCP connection to localhost port 2000.

Example gnICE flash programming

Also take a look here: flash-programmer Faster programming including support for SPI flashes.

For further information on the jtag tools (bfin-jtag) visit the page here:

UrJTAG Universal JTAG library, server and tools

Start bfin-jtag

```
$ <strong>bfin-jtag -q</strong>
jtag>
```

Initialize the gnICE cable driver

```
jtag> <strong>cable gnICE</strong>
Connected to libftdi driver.
```

Alternatively use: **cable gnICE ftdi-mpsse 0456:F000**

See also: Using multiple gnICE USB cables simultaneously

Scan JTAG Interface

```
jtag> <strong>detect</strong>
IR length: 5
Chain length: 1
Device Id: 00110010011110100101000011001011 (0x00000000327A50CB)
  Manufacturer: Analog Devices
  Part(0):         BF533
  Stepping:     3
  Filename:
/opt/uClinux/bfin-uclinux/bin/../share/urjtag/analog/bf533/bf533
```

Start bus driver according to your hardware

```
jtag> <strong>initbus bf533_stamp</strong>
```

Detect flash at 0x20000000 (Bank0)

```
jtag> <strong>detectflash 0x20000000</strong>
Query identification string:
        Primary Algorithm Command Set and Control Interface ID Code: 0x0002
(AMD/Fujitsu Standard Command Set)
        Alternate Algorithm Command Set and Control Interface ID Code: 0x0000
(null)
Query system interface information:
        Vcc Logic Supply Minimum Write/Erase or Write voltage: 2700 mV
        Vcc Logic Supply Maximum Write/Erase or Write voltage: 3600 mV
        Vpp [Programming] Supply Minimum Write/Erase voltage: 11500 mV
        Vpp [Programming] Supply Maximum Write/Erase voltage: 12500 mV
        Typical timeout per single byte/word program: 16 us
        Typical timeout for maximum-size multi-byte program: 0 us
        Typical timeout per individual block erase: 1024 ms
        Typical timeout for full chip erase: 0 ms
```

```
        Maximum timeout for byte/word program: 512 us
        Maximum timeout for multi-byte program: 0 us
        Maximum timeout per individual block erase: 16384 ms
        Maximum timeout for chip erase: 0 ms
Device geometry definition:
        Device Size: 4194304 B (4096 KiB, 4 MiB)
        Flash Device Interface Code description: 0x0002 (x8/x16)
        Maximum number of bytes in multi-byte program: 1
        Number of Erase Block Regions within device: 4
        Erase Block Region Information:
                Region 0:
                        Erase Block Size: 16384 B (16 KiB)
                        Number of Erase Blocks: 1
                Region 1:
                        Erase Block Size: 8192 B (8 KiB)
                        Number of Erase Blocks: 2
                Region 2:
                        Erase Block Size: 32768 B (32 KiB)
                        Number of Erase Blocks: 1
                Region 3:
                        Erase Block Size: 65536 B (64 KiB)
                        Number of Erase Blocks: 63
Primary Vendor-Specific Extended Query:
        Major version number: 1
        Minor version number: 0
        Address Sensitive Unlock: Required
        Erase Suspend: Read/write
        Sector Protect: 1 sectors per group
        Sector Temporary Unprotect: Not supported
        Sector Protect/Unprotect Scheme: 29BDS640 mode (Software Command
Locking)
        Simultaneous Operation: Not supported
        Burst Mode Type: Supported
        Page Mode Type: Not supported
```

Flash u-boot at address 0x20000000 in ASYNC Bank0

```
jtag> <strong>flashmem 0x20000000 u-boot.bin</strong>
Chip: AMD Flash
        Manufacturer: ST/Samsung
        Chip: M29W320DB
        Protected: 00ff
program:
flash_unlock_block 0x20000000 IGNORE

block 0 unlocked
flash_erase_block 0x20000000
.............................................................flash_eras
e_block 0x20000000 DONE
erasing block 0: 0
flash_unlock_block 0x20004000 IGNORE
```

```
block 1 unlocked
flash_erase_block 0x20004000
.....................................................................flash_eras
e_block 0x20004000 DONE
erasing block 1: 0
flash_unlock_block 0x20006000 IGNORE

block 2 unlocked
flash_erase_block 0x20006000
.....................................................................flash_erase
_block 0x20006000 DONE
erasing block 2: 0
flash_unlock_block 0x20008000 IGNORE

block 3 unlocked
flash_erase_block 0x20008000
.....................................................................flash_erase
_block 0x20008000 DONE
erasing block 3: 0
flash_unlock_block 0x20010000 IGNORE

block 4 unlocked
flash_erase_block 0x20010000
.....................................................................flash_erase
_block 0x20010000 DONE
erasing block 4: 0
flash_unlock_block 0x20020000 IGNORE

block 5 unlocked
flash_erase_block 0x20020000
.....................................................................:flash_erase
_block 0x20020000 DONE
erasing block 5: 0
addr: 0x20023212
verify:
addr: 0x20023212
Done.
jtag>
```

List of supported Bus Drivers

Even when your target board / processor module is not explicitly listed it may be the case that you can use one of the existing bus drivers with your platform.

Example: You can use bf561_ezkit bus driver together with the CM-BF561 Core Module from Bluetechnix, as well as the bf548_ezkit bus driver together with the CM-BF548 Core module. It's also likely that you can use one of the generic bf53x bus driver with a variety of BF53x platforms or the BF52x with various BF52x platforms.

To this day - All bus driver support address ranges mapped into Asynchronous Memory Banks 0-3 (/AMS0..3) typically 4MB.

Except those listed below, supporting only /AMS0:

Processor	bfin-jtag Accessible Address Space
BF533_stamp	1 MByte
bf548_ezkit	64 MByte
bf561_ezkit	64 MByte

```
jtag> <strong>help initbus</strong>
Usage: initbus BUSNAME
Initialize new bus driver for active part.

BUSNAME       Name of the bus

List of available buses:
au1500     AU1500 BUS Driver via BSR
avr32      Atmel AVR32 multi-mode bus driver, requires <mode> parameter
           valid <mode> parameters:
               x8:   8 bit bus for the uncached HSB area, via OCD registers
               x16:  16 bit bus for the uncached HSB area, via OCD registers
               x32:  32 bit bus for the uncached HSB area, via OCD registers
               OCD : 32 bit bus for the OCD registers
               HSBC: 32 bit bus for the cached HSB area, via SAB
               HSBU: 32 bit bus for the uncached HSB area, via SAB
bcm1250    Broadcom BCM1250 compatible bus driver via BSR
<em>bf526_ezkit Blackfin BF526 EZ-KIT board bus driver via BSR
bf527_ezkit Blackfin BF527 EZ-KIT board bus driver via BSR
bf52x      Generic Blackfin BF52x bus driver via BSR
bf533_stamp Blackfin BF533 Stamp board bus driver
bf533_ezkit Blackfin BF533 EZ-KIT board bus driver via BSR
bf537_stamp Blackfin BF537 Stamp board bus driver via BSR
bf537_ezkit Blackfin BF537 EZ-KIT board bus driver via BSR
bf538f_ezkit Blackfin BF538F EZ-KIT board bus driver via BSR
bf53x      Generic Blackfin BF53x bus driver via BSR
bf548_ezkit Blackfin BF548 EZ-KIT board bus driver
bf561_ezkit Blackfin BF561 EZ-KIT board bus driver</em>
ejtag      EJTAG compatible bus driver via PrAcc
fjmem      FPGA JTAG memory bus driver via USER register, requires parameters:
           opcode=<USERx OPCODE> [len=<FJMEM REG LEN>]
ixp425     Intel IXP425 compatible bus driver via BSR
jopcyc     JOP.design Cyclone Board compatible bus driver via BSR
h7202      H7202 compatible bus driver via BSR
lh7a400    Sharp LH7A400 compatible bus driver via BSR (flash access only!)
mpc5200    Freescale MPC5200 compatible bus driver via BSR
mpc824x    Motorola MPC824x compatible bus driver via BSR
ppc405ep   IBM PowerPC 405EP compatible bus driver via BSR
ppc440gx_ebc8 IBM PowerPC 440GX 8-bit EBC compatible bus driver via BSR
prototype  Configurable prototype bus driver via BSR, requires parameters:
```

```
          amsb=<addr MSB> alsb=<addr LSB> dmsb=<data MSB> dlsb=<data LSB>
          ncs=<CS#>|cs=<CS> noe=<OE#>|oe=<OE> nwe=<WE#>|we=<WE>
[amode=auto|x8|x16|x32]
pxa2x0     Intel PXA2x0 compatible bus driver via BSR
pxa27x     Intel PXA27x compatible bus driver via BSR
s3c4510x   Samsung S3C4510B compatible bus driver via BSR
sa1110     Intel SA-1110 compatible bus driver via BSR
sh7727     Hitachi SH7727 compatible bus driver via BSR
sh7750r    Hitachi SH7750R compatible bus driver via BSR
sh7751r    Hitachi SH7751R compatible bus driver via BSR
SHARC_21065L SHARC_21065L compatible bus driver via BSR
slsup3     SLS UP3 compatible bus driver via BSR
tx4925     Toshiba TX4925 compatible bus driver via BSR
zefant-xs3 Simple Solutions Zefant-XS3 Board compatible bus driver via BSR

jtag>
```

JTAG Communications

Using multiple gnICE USB cables simultaneously

Every gnICE cable features a unique Serial number. To determine the Serial number programmed into your device use the Linux lsusb command.

```
$ <strong>lsusb -v</strong>
...
Bus 002 Device 004: ID 0456:f000 Analog Devices, Inc.
...
  idVendor           0x0456 Analog Devices, Inc.
  idProduct          0xf000
  bcdDevice            5.00
  iManufacturer           1 Analog Devices Inc.
  iProduct                2 Blackfin gnICE
  iSerial                 3 <strong>ADREYL49</strong>
...
```

Then plug that serial number into the cable command.

```
jtag> <strong>cable gnICE help</strong>
Usage: cable gnICE [vid=VID] [pid=PID] [desc=DESC] [driver=DRIVER]

VID        vendor ID (hex), defaults to 0456
PID        product ID (hex), defaults to F000
```

```
DESC       Some string to match in description or serial no.
DRIVER     usbconn driver, either ftdi-mpsse or ftd2xx-mpsse
           defaults to ftdi-mpsse if not specified

jtag> cable gnICE <strong>desc=ADREYL49</strong>
Connected to libftdi driver.
jtag>
```

Troubleshooting Hints and Tricks

- Use USB JTAG without root privilege
- Using bfin-jtag scripts and default init rc files

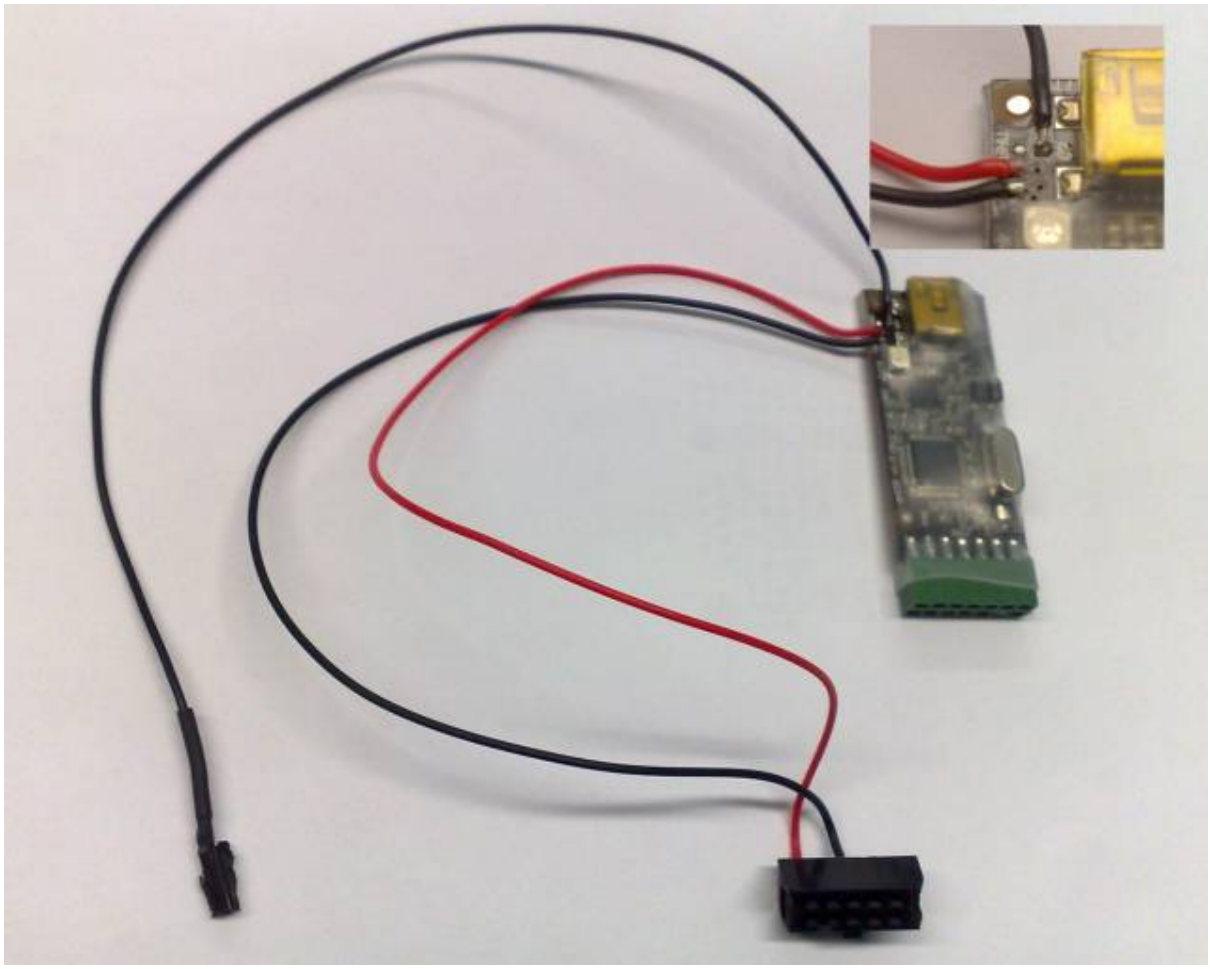
The rc Script file contains startup instructions for an application program, usually a text file containing commands of the sort that might have been invoked manually once the system was running but are to be executed automatically each time the system starts up.

```
$ <strong>ls -l ~/.jtag/</strong>
total 20
-rw-r--r-- 1 michael users   55 2008-12-08 12:51 flash
-rw------- 1 michael users 2282 2009-02-17 14:17 history
-rw-r--r-- 1 michael users   19 2009-02-10 09:42 rc

$ <strong>cat ~/.jtag/rc</strong>
cable gnICE
detect
```

- Other useful Blackfin specific bfin-jtag commands bfin_command

gnICE hacking – enabling auxiliary functions



The gnICE features four auxiliary signals routed to solder pads, located besides the Mini-USB Jack.

Label	Description	Note
RX	Channel B (BDBUS1) RXD	UART RXD (3.3V IO) signal should connect to Blackfin TXD
TX	Channel B (BDBUS0) TXD	UART TXD (3.3V IO) signal should connect to Blackfin RXD
GP	Channel A (ACBUS2) GPIOH2	This is a General Purpose IO Signal, which can be used to RESET the board
GND	Signal Ground	In case the gnICE is connect to the board via the JATG header this can be left unconnected

RX and TX are 3.3Volt TTL Signals they may not be connected to RS232 typical +/- 10V UART signals

Utilizing the Serial Interface

The gnICE auxiliary Serial Interface can be used with the standard Linux ftdi_sio driver. In case your kernel version already includes following patch, ignore this section. Otherwise it's recommended that your patch your HOST PC kernel with this patch.

gnICE gnICE+

It is also possible to utilize the Serial interface without this patch applied.

- Make sure the driver isn't already loaded

```
modprobe -r ftdi_sio
```

- Then load the driver module with given gnICE VID/PID arguments

```
modprobe ftdi_sio vendor=0x0456 product=0xF000
```

The Linux kernel of your host system will create a virtual serial device called /dev/ttyUSBx where 'x' is a sequentially assigned number. If you don't have any other USB serial converters attached to your machine, the device name will be /dev/ttyUSB1. You can use your favorite terminal emulator (minicom, cu, zc, ...) just like for any other/real serial port.

JTAG and Serial can be used together.

```
usbcore: registered new interface driver usbserial
usbserial: USB Serial support registered for generic
usbcore: registered new interface driver usbserial_generic
usbserial: USB Serial Driver core
usbserial: USB Serial support registered for FTDI USB Serial Device
ftdi_sio 5-1.3:1.0: FTDI USB Serial Device converter detected
ftdi_sio: Detected FT2232C
usb 5-1.3: <strong>FTDI USB Serial Device converter now attached to
ttyUSB0</strong>
ftdi_sio 5-1.3:1.1: FTDI USB Serial Device converter detected
ftdi_sio: Detected FT2232C
usb 5-1.3: <strong>FTDI USB Serial Device converter now attached to
ttyUSB1</strong>
usbcore: registered new interface driver ftdi_sio
ftdi_sio: v1.4.3:USB FTDI Serial Converters Driver
```

Without the patch the driver will detect and configure two USB Serial TTYs. **You should only use the second one**. Since the first one uses the same signals on the FT2232 Channel A shared with the JTAG.

You may damage your hardware!

Controlling the auxiliary GPIO

A simple control application can be found here:

file: trunk/debug-helpers/gnice-reset/gnice-reset.c

```
/* Blackfin gnICE JTAG GPIO reset utility
 * Michael Hennerich Copyright 2009 Analog Devices Inc.
 *
 * Licensed under the GPL-2 or later
 *
 * For more information see:
 * - gnICE schematics
 * - libftdi
 * - Future Technology Devices International Limited (FTDI)
 *   Application Note AN_108 Command Processor for MPSSE and
 *   MCU Host Bus Emulation Modes
 */

#include <getopt.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <ftdi.h>

#define ARRAY_SIZE(a) (sizeof(a) / sizeof(a[0]))

#define GNICE_VID 0x0456
const int pids&#91;&#93; = &#123;
        0xF000,         /* gnICE */
        0xF001, /* gnICE+ */
&#125;;

#ifdef WIN32
# define usleep(x) Sleep((x) / 1000000 ? : 1)
#endif

const char *argv0;

#define FLAGS "g:hl:s:"
#define a_argument required_argument
static struct option const long_opts&#91;&#93; = &#123;
        &#123;"gpio",     a_argument, NULL, 'g'&#125;,
        &#123;"help",    no_argument, NULL, 'h'&#125;,
        &#123;"led",      a_argument, NULL, 'l'&#125;,
        &#123;"serial",   a_argument, NULL, 's'&#125;,
&#125;;

void usage&#40;int exit_status&#41;
&#123;
        int i;
        FILE *fp = exit_status ? stderr : stdout;
        fprintf&#40;fp, "Usage: %s [options]\n\nOptions: -[%s]\n", argv0,
FLAGS&#41;;
        for &#40;i = 0; i < ARRAY_SIZE&#40;long_opts&#41;; ++i&#41;
                fprintf&#40;fp, "  -%c, --%s\n", long_opts&#91;i&#93;.val,
long_opts&#91;i&#93;.name&#41;;
```

```
        exit&#40;exit_status&#41;;
&#125;

int main&#40;int argc, char *argv&#91;&#93;&#41;
&#123;
        struct ftdi_context con;
        unsigned char buf&#91;3&#93;;
        const char *desc;
        int i, f, gpio, led, bitmask;

        argv0 = argv&#91;0&#93;;

        gpio = 2;        /* ACBUS2 */
        led = 3;        /* ACBUS3 */
        desc = NULL;
        while &#40;&#40;i = getopt_long&#40;argc, argv, FLAGS, long_opts,
NULL&#41;&#41; != -1&#41; &#123;
                switch &#40;i&#41; &#123;
                        case 'g': gpio = atoi&#40;optarg&#41;; break;
                        case 'h': usage&#40;0&#41;;
                        case 'l': led = atoi&#40;optarg&#41;; break;
                        case 's': desc = optarg; break;
                        default:  usage&#40;1&#41;;
                &#125;
        &#125;
        bitmask = &#40;1 << gpio&#41; | &#40;1 << led&#41;;

        ftdi_init&#40;&con&#41;;
        ftdi_set_interface&#40;&con, INTERFACE_A&#41;;

        if &#40;desc&#41; &#123;
                for &#40;i = 0; i < ARRAY_SIZE&#40;pids&#41;; ++i&#41; &#123;
                        f = ftdi_usb_open_desc&#40;&con, GNICE_VID,
pids&#91;i&#93;,
                                                NULL, argv&#91;2&#93;&#41;;
                        if &#40;f >= 0&#41;
                                break;
                &#125;
        &#125; else &#123;
                for &#40;i = 0; i < ARRAY_SIZE&#40;pids&#41;; ++i&#41; &#123;
                        f = ftdi_usb_open&#40;&con, GNICE_VID,
pids&#91;i&#93;&#41;;
                        if &#40;f >= 0&#41;
                                break;
                &#125;
        &#125;

        if &#40;f < 0 && f != -5&#41; &#123;
                fprintf&#40;stderr, "unable to open ftdi device: %d (%s)\n",
f,
                        ftdi_get_error_string&#40;&con&#41;&#41;;
```

```
                exit&#40;-1&#41;;
        &#125;

        ftdi_disable_bitbang&#40;&con&#41;;
        /* in order to use ACBUS we need to enable MPSSE mode */
        ftdi_set_bitmode&#40;&con, 0x0, BITMODE_MPSSE&#41;;

        /* drive LED and RESET GPIO HIGH */
        buf&#91;0&#93; = SET_BITS_HIGH;
        buf&#91;1&#93; = bitmask;        /* VAL */
        buf&#91;2&#93; = bitmask;        /* DIR */

        f = ftdi_write_data&#40;&con, buf, 3&#41;;
        if &#40;f < 0&#41;
                fprintf&#40;stderr,
                        "write failed error %d (%s)\n",
                        f, ftdi_get_error_string&#40;&con&#41;&#41;;
        usleep&#40;50000&#41;; /* 50ms */

        /* drive LED and RESET GPIO LOW */
        buf&#91;0&#93; = SET_BITS_HIGH;
        buf&#91;1&#93; = 0;
        buf&#91;2&#93; = bitmask;

        f = ftdi_write_data&#40;&con, buf, 3&#41;;
        if &#40;f < 0&#41;
                fprintf&#40;stderr,
                        "write failed error %d (%s)\n",
                        f, ftdi_get_error_string&#40;&con&#41;&#41;;
        usleep&#40;50000&#41;; /* 50ms */

        /* ACBUS HIGH Input */
        buf&#91;0&#93; = SET_BITS_HIGH;
        buf&#91;1&#93; = 0;
        buf&#91;2&#93; = 0;

        f = ftdi_write_data&#40;&con, buf, 3&#41;;
        if &#40;f < 0&#41;
                fprintf&#40;stderr,
                        "write failed error %d (%s)\n",
                        f, ftdi_get_error_string&#40;&con&#41;&#41;;

        /* cleanup and exit */
        ftdi_disable_bitbang&#40;&con&#41;;
        ftdi_usb_close&#40;&con&#41;;
        ftdi_deinit&#40;&con&#41;;

        return 0;
&#125;
```