

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА
ШЕВЧЕНКА

Фізичний факультет
Кафедра ядерної фізики

USB зв'язок на платі YJ-DZ

Виконав:

Студент 2 курсу магістратури групи ФВЕ

Пядик Юрій Сергійович

Викладач:

Єрмоленко Руслан Вікторович

Київ, 2015

Зміст

Вступ.....	3
Налаштування робочого середовища.....	3
Створення прошивки для USB мікросхеми.....	3
Процедура прошивки USB мікросхеми.....	8
Прошивка FPGA мікросхеми.....	10
Розробка програми користувача.....	11

Вступ

Плата YJ-DZ цікава тим, що на ній по мимо потужного FPGA чіпа Cyclone IV (Altera) розведено також USB мікросхему від Cypress. Даний документ описує процедуру налаштування робочого середовища для розробки та програмування прошивок як для FPGA так і для USB мікросхеми. Процес демонструється на прикладі BulkLoop, який перенаправляє назад потік даних з комп'ютера, попередньо їх інвертуючи.

Налаштування робочого середовища

Перш за все необхідно встановити середовище Quartus для розробки та програмування прошивки для FPGA мікросхеми:

<https://www.altera.com/downloads/download-center.html>

Для програмування FPGA через JTAG інтерфейс використовується програматор Bluster для якого потрібно буде попередньо встановити драйвер. Слава Богу він встановлюється автоматично після першого під'єднання до комп'ютера. Проте може знадобитись доступ до мережі Інтернет.

Демонстраційний проект Quartus для прошивки FPGA можна знайти в папці `firmware_fpga/quartus_build`.

Для розробки прошивки до USB мікросхеми від Cypress необхідне спеціальне середовище розробки (на базі Eclipse), що входить до складу EZ-USB FX3 SDK:

<http://www.cypress.com/documentation/software-and-drivers/ez-usb-fx3-software-development-kit>

Туди ж входить спеціальна програма для запису прошивки в мікросхему – Control Center. Також в SDK входять відповідний драйвер (його можна знайти і у папці `drivers`) щоб USB мікросхема розпізнавалася комп'ютером та C++ бібліотека з документацією для розробки програми користувача (також у `software/3rdparty`).

Проте для компіляції прошивки ще бракує відповідного компілятора – SDCC:

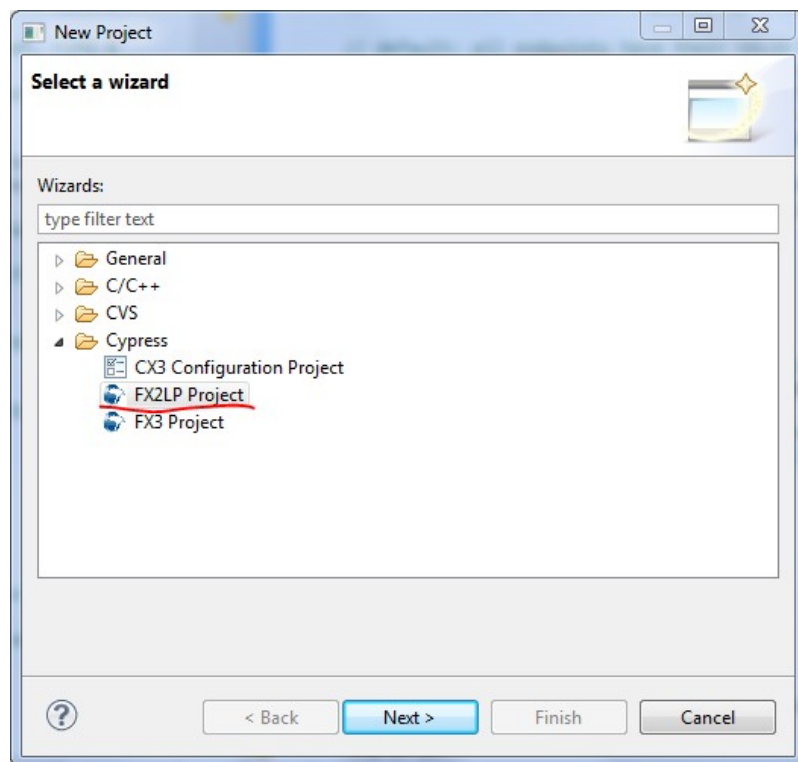
<http://sdcc.sourceforge.net/index.php#Download>

Для програмування USB мікросхеми на постійній основі прошивку необхідно заливати не безпосередньо в мікросхему, а в EEPROM пам'ять, що розпаяна поряд з нею та під'єднується по I2C інтерфейсу. Таким чином мікросхема автоматично прошиватиметься з EEPROM пам'яті кожен раз при ввімкненні живлення. Щоб сконвертувати прошивку у відповідний формат (hex у iic) для запису в EEPROM необхідна утиліта `hex2bix` з пакету CY3684 EZ-USB FX2LP DVK (також у `software/3rdparty/bin`):

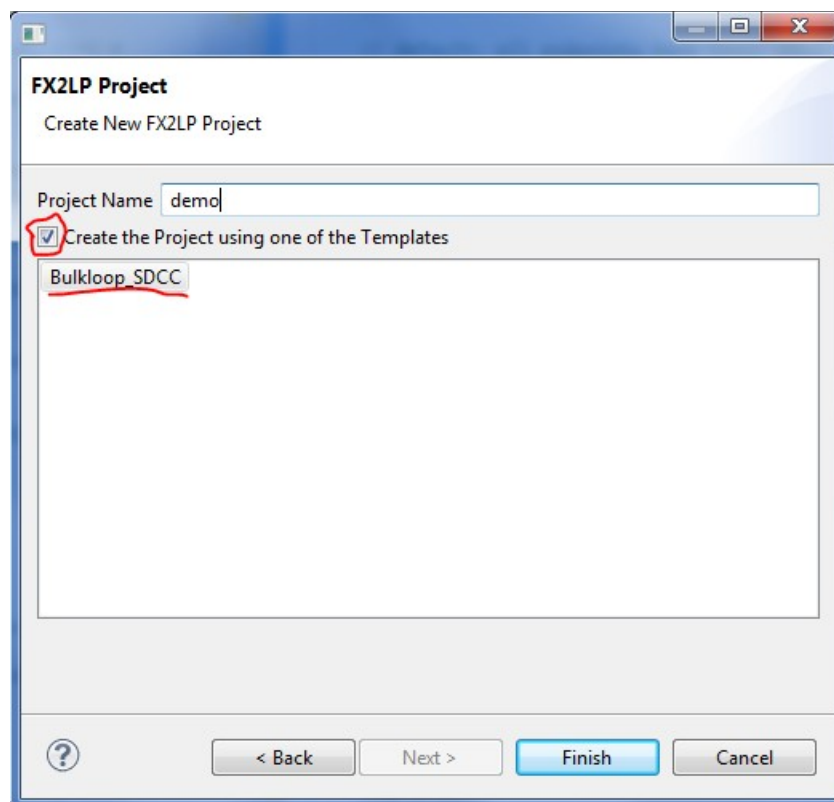
<http://www.cypress.com/documentation/development-kitsboards/cy3684-ez-usb-fx2lp-development-kit>

Створення прошивки для USB мікросхеми

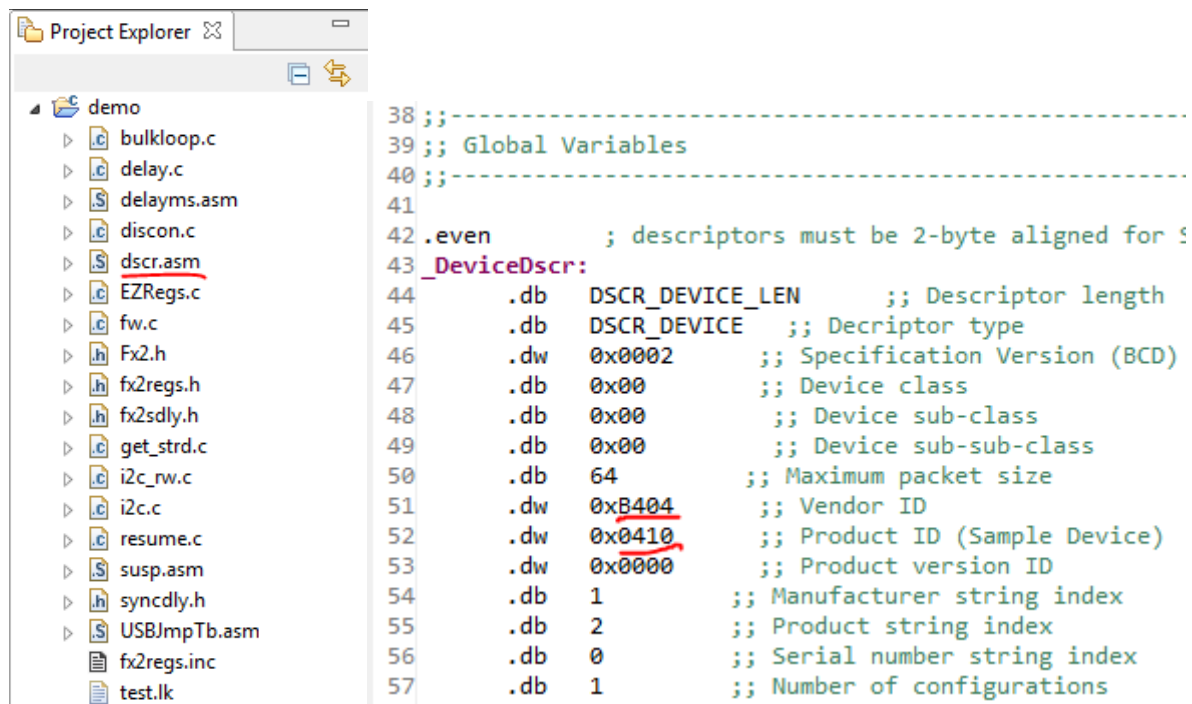
Запустимо EZ USB Suite (спеціалізоване середовище Eclipse) та створимо новий FX2LP проект:



На базі прикладу BulkLoop, що компілюється за допомогою компілятора SDCC:



Задамо у файлі dscr.asm бажані VID та PID:



```
38 ;;-----
39 ;; Global Variables
40 ;;-----
41
42 .even          ; descriptors must be 2-byte aligned for :
43 _DeviceDscr:
44     .db  DSCR_DEVICE_LEN      ;; Descriptor length
45     .db  DSCR_DEVICE          ;; Descriptor type
46     .dw  0x0002               ;; Specification Version (BCD)
47     .db  0x00                 ;; Device class
48     .db  0x00                 ;; Device sub-class
49     .db  0x00                 ;; Device sub-sub-class
50     .db  64                   ;; Maximum packet size
51     .dw  0xB404               ;; Vendor ID
52     .dw  0x0410               ;; Product ID (Sample Device)
53     .dw  0x0000               ;; Product version ID
54     .db  1                    ;; Manufacturer string index
55     .db  2                    ;; Product string index
56     .db  0                    ;; Serial number string index
57     .db  1                    ;; Number of configurations
```

Зауважте, що це двобайтові величини і записувати їх треба байтами навиворіт. В тому ж файлі можна задати назву вашого USB пристрою тощо. Але рекомендується залишити все як є щоб не довелося підлаштовувати та перевстановлювати USB драйвер для вашого пристрою (для BulkLoop він вже настроєний).

Тепер залишається замінити функцію TD_Init() в файлі bulkloop.c на наступну:

```
void TD_Init(void)           // Called once at startup
{
    // Registers which require a synchronization delay, see section 15.14

    // FIFORESET      FIFOPINPOLAR
    // INPKTEND       OUTPKTEND
    // EPxBCH:L       REVCTL
    // GPIFTCB3       GPIFTCB2
    // GPIFTCB1       GPIFTCB0
    // EPxFIFOPFH:L   EPxAUTOINLENH:L
    // EPxFIFOCFG     EPxGPIFFLGSEL
    // PINFLAGSxx     EPxFIFOIRQ
    // EPxFIFOIE      GPIFIRQ
    // GPIFIE         GPIFADRH:L
    // UDMACRCH:L     EPxGPIFTRIG
    // GPIFTRIG

    // Note: The pre-REVE EPxGPIFTCH/L register are affected, as well...
    // ...these have been replaced by GPIFTC[B3:B0] registers

    // default: all endpoints have their VALID bit set
    // default: TYPE1 = 1 and TYPE0 = 0 --> BULK
    // default: EP2 and EP4 DIR bits are 0 (OUT direction)
    // default: EP6 and EP8 DIR bits are 1 (IN direction)
    // default: EP2, EP4, EP6, and EP8 are double buffered

    // set the CPU clock to 48MHz (internally multiplies xtal 24 MHz ref clk by 2)
    CPUCS = ((CPUCS & ~bmCLKSPD) | bmCLKSPD1) ;
```

```

// will hurt
// REVCTL = bmNOAUTOARM | bmSKIPCOMMIT;
// SYNCDELAY;

// setup the slave FIFO interface
// internal clock | 30 MHz | output to IFCLK pin | output inverted | Slave FIFO (ext master)
mode IFCONFIG = bmIFCLKSRC/* | bm3048MHZ*/ | bmIFCLKOE | bmIFCLKPOL | bmIFCFGMASK;

// FLAGA=PF, FLAGB=FF, FLAGC=EF, FLAGD=EP2PF
PINFLAGSAB = 0x8E; // FLAGA - EP6FF; FLAGB - EP2EF
SYNCDELAY;
PINFLAGSCD = 0x0A; // FLAGC - EP6EF
SYNCDELAY;
PORTACFG |= 0x80; // programmable FLAGD
SYNCDELAY;

EP1OUTCFG = 0xA0;
EP1INCFG = 0xA0;
SYNCDELAY;

// VALID DIR TYPE1 TYPE0 | SIZE 0 BUF1 BUF0
EP2CFG = 0xA2; // out 512 bytes, 4x, bulk
SYNCDELAY;

EP4CFG = 0xA0; // out 512 bytes, 2x, bulk
SYNCDELAY;

EP6CFG = 0xE2; // in 512 bytes, 4x, bulk
SYNCDELAY;

SYNCDELAY;
FIFORESET = 0x80; // activate NAK-ALL to avoid race conditions
SYNCDELAY; // see TRM section 15.14
FIFORESET = 0x02; // reset, FIFO 2
SYNCDELAY; //
FIFORESET = 0x04; // reset, FIFO 4
SYNCDELAY; //
FIFORESET = 0x06; // reset, FIFO 6
SYNCDELAY; //
FIFORESET = 0x08; // reset, FIFO 8
SYNCDELAY; //
FIFORESET = 0x00; // deactivate NAK-ALL
SYNCDELAY;

// handle the case where we were already in AUTO mode...
// ...for example: back to back firmware downloads...
SYNCDELAY; //
EP2FIFOCFG = 0x00; // AUTOOUT=0, WORDWIDE=1

// core needs to see AUTOOUT=0 to AUTOOUT=1 switch to arm endp's

SYNCDELAY; //
EP2FIFOCFG = 0x11; // AUTOOUT=1, WORDWIDE=1

SYNCDELAY; //
EP6FIFOCFG = 0x09; // AUTOIN=1, ZEROLENIN=0, WORDWIDE=1

EP8CFG = 0xE0; // in 512 bytes, 2x, bulk
SYNCDELAY;

EP4BCL = 0x80; // arm EP4OUT by writing byte count w/skip.

```

```

    SYNCDELAY;
    EP4BCL = 0x80;
    SYNCDELAY;

    // enable dual autopointer feature
    AUTOPTRSETUP |= 0x01;
}

```

Вона налаштовує кінцеві точки EP2OUT та EP6IN на автоматичний вивід/ввід через інтерфейс Slave FIFO.

Треба відповідно видалити код, що відповідав за зациклення цих двох кінцевих точок в прикладі, з функції TD_Poll() зразу після TD_Init() в тому ж файлі:

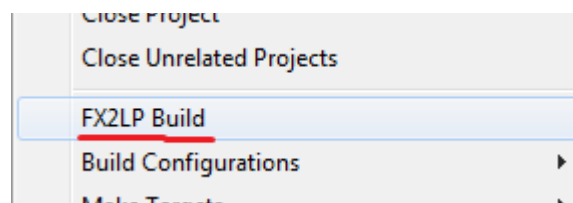
```

// if(!(EP2468STAT & bmEP2EMPTY))
// { // check EP2 EMPTY(busy) bit in EP2468STAT (SFR), core set's this bit when FIFO is empty
//   if(!(EP2468STAT & bmEP6FULL))
//   { // check EP6 FULL(busy) bit in EP2468STAT (SFR), core set's this bit when FIFO is full
//     APTR1H = MSB( &EP2FIFOBUF );
//     APTR1L = LSB( &EP2FIFOBUF );
//
//     AUTOPTRH2 = MSB( &EP6FIFOBUF );
//     AUTOPTL2 = LSB( &EP6FIFOBUF );
//
//     count = (EP2BCH << 8) + EP2BCL;
//
//     // loop EP2OUT buffer data to EP6IN
//     for( i = 0x0000; i < count; i++ )
//     {
//       // setup to transfer EP2OUT buffer to EP6IN buffer using AUTOPOINTER(s)
//       EXTAUTODAT2 = EXTAUTODAT1;
//     }
//     EP6BCH = EP2BCH;
//     SYNCDELAY;
//     EP6BCL = EP2BCL;      // arm EP6IN
//     SYNCDELAY;
//     EP2BCL = 0x80;      // re(arm) EP2OUT
//   }
// }

```

Тепер ми готові компілювати проект:

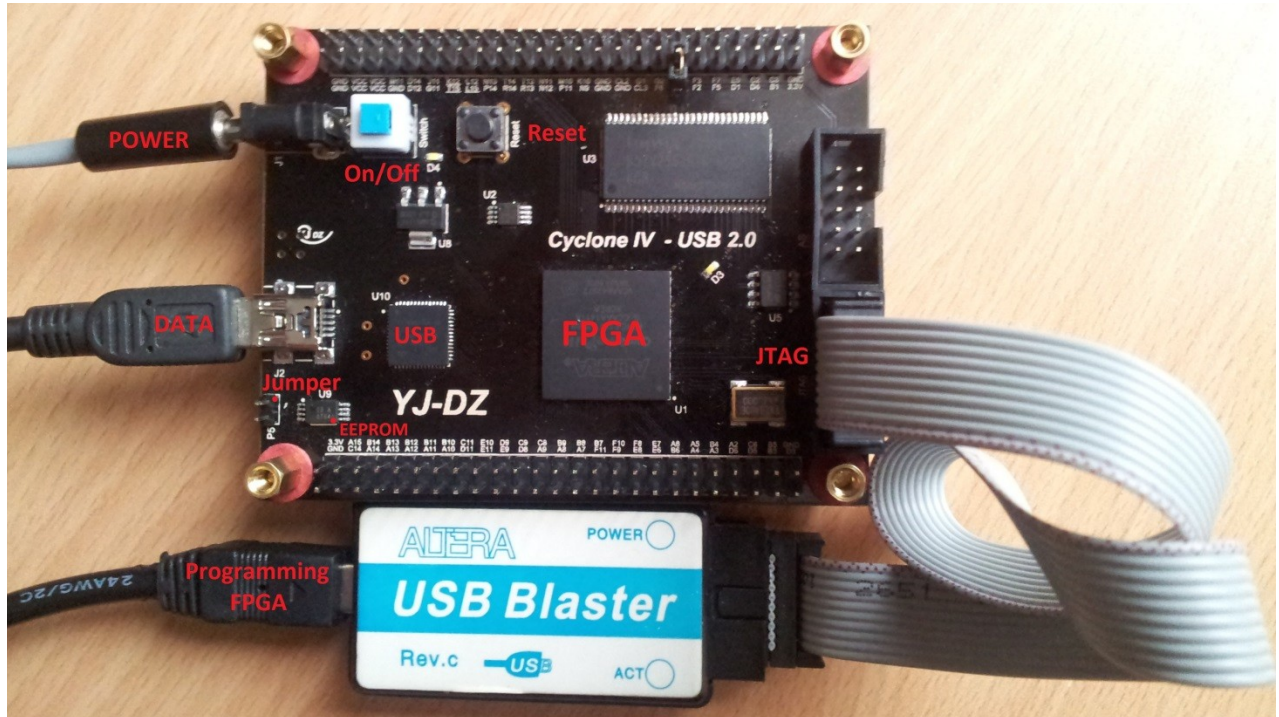
ПКМ на проекті – FX2LP Build



Це займе декілька секунд і вам згенерує в папці Release файл прошивки <projectName>.hex.

Увага: в жодному разі не перейменовуйте bulkloop.c як вам заманеться чи на назву проекту.

Процедура прошивки USB мікросхеми



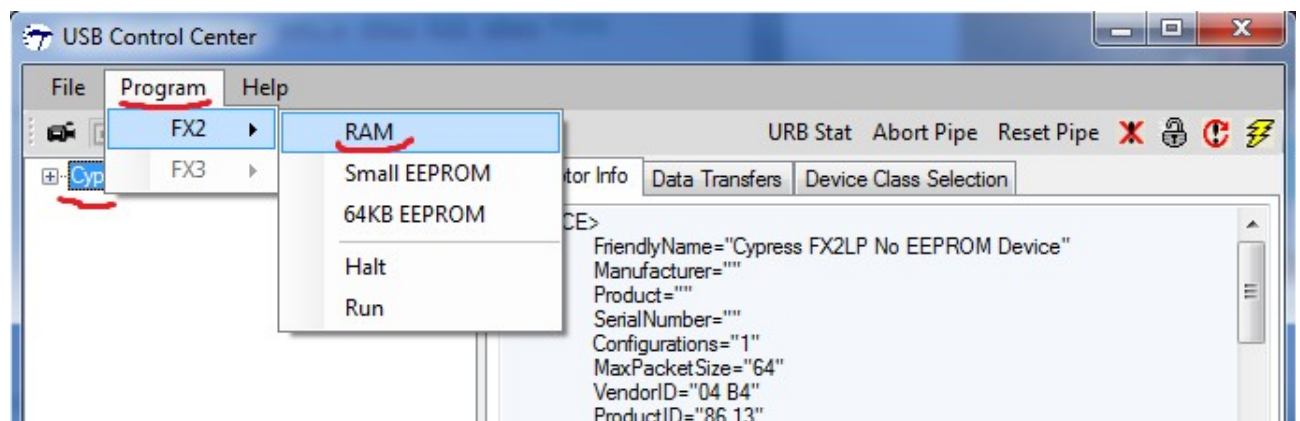
Існує два способи прошити USB мікросхему від Cypress:

1. Миттєвий

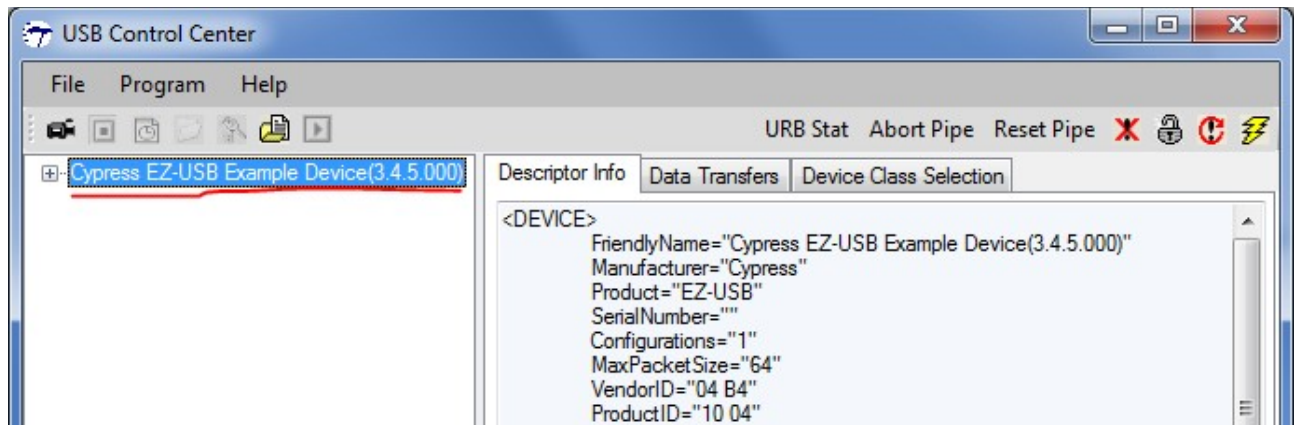
В даному випадку виконання прошивки почнеться зразу після успішного завершення програмування. Мікросхема автоматично «перепідключиться» до комп'ютера вже з новою прошивкою.

Для цього замкніть джампер на платі перед тим як включати живлення. Це відключить EEPROM пам'ять від USB мікросхеми (в силу особливості протоколу I2C) та заставить її завантажити внутрішню стандартну заводську прошивку. Опісля під'єднайте плату по USB до комп'ютера. Відкрийте Control Center із EZ-USB FX3 SDK і плата повинна бути ідентифікована як Cypress FX2LP No EEPROM Device. Виділіть її та залийте прошивку:

Меню Program – FX2 – RAM – виберіть hex-файл і дочекайтесь успішного завершення



Тепер плату буде ідентифіковано як Cypress EZ-USB Example Device(3.4.5.000):



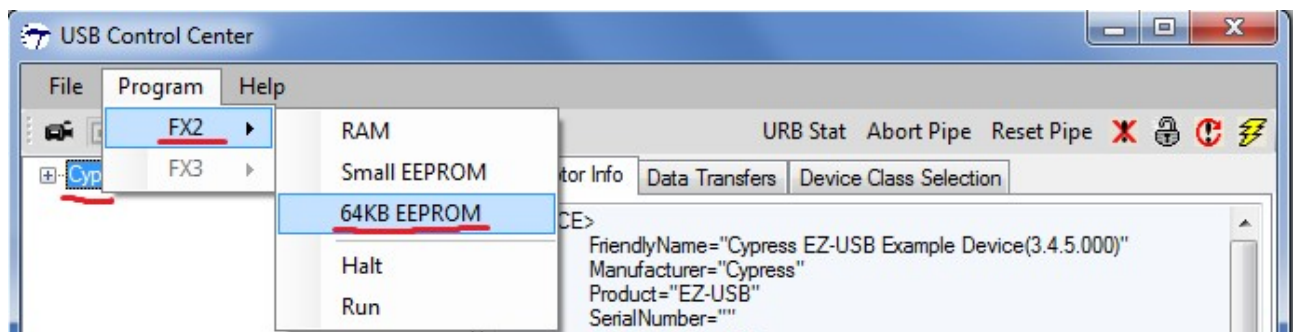
Якщо пропаде живлення, то прошивку буде втрачено і після відновлення живлення знову завантажиться заводська (бо джампер замкнутий).

2. Постійний

Даним методом програмується фінальна версія прошивки USB мікросхеми коли перший метод зручний на етапі її розробки.

Повторіть все як у першому методі лише безпосередньо перед прошивкою (плата вже ідентифікована із заводською прошивкою) розімкніть джампер. Також потрібно замість hex-файлу прошивати ііс-файл через пункт меню:

Меню Program – FX2 – 64KB EEPROM – виберіть ііс-файл і дочекайтесь успішного завершення



Щоб застосувати нову прошивку потрібно вимкнути/ввімкнути живлення. Це заставить USB мікросхему після відновлення живлення автоматично прошитися з EEPROM пам'яті (куди ми щойно залили нову прошивку).

Конвертація hex-файлу прошивки у ііс-файл

Знайдіть у CY3684 EZ-USB FX2LP DVK утиліту Hex2bix.exe та скористайтесь командним рядком:

```
Hex2bix.exe -i -m 0xffff -f 0xC2 -V 04b4 -P 1004 -o <fileName>.iic <fileName>.hex
```

Або скористайтесь скриптом convert.bat із папки software/3rdparty/bin щоб не вводити команду кожен раз з нуля:

```
C:\Windows\system32\cmd.exe

D:\Projects\YJ-DZ\software\3rdparty\bin>Hex2bix.exe -i -m 0xffff -f 0xC2 -U 04b4
-P 1004 -o ../../../../firmware_usb/bulkloop/Release/bulkloop.iic ../../../../firmwar
e_usb/bulkloop/Release/bulkloop.hex
Intel Hex file to EZ-USB Binary file conversion utility
Copyright (c) 2012-2013, Cypress Semiconductor Inc.
3464 Bytes written.
Total Code Bytes = 3395
Conversion completed successfully.

D:\Projects\YJ-DZ\software\3rdparty\bin>pause
Press any key to continue . . .
```

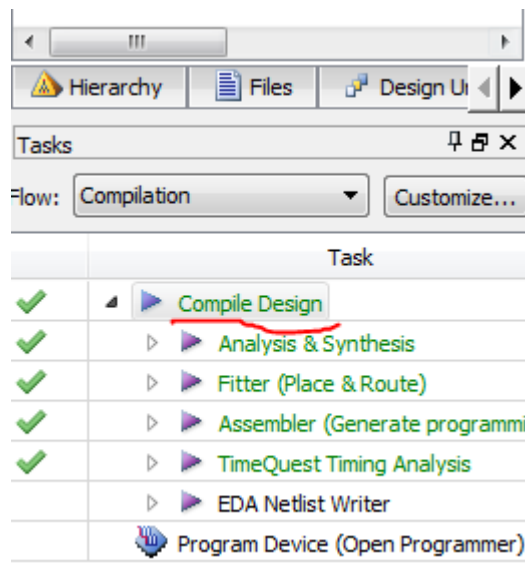
Прошивка FPGA мікросхеми

Яким би способом не була прошита USB мікросхема, кожен раз після ввімкнення живлення потрібно буде з нуля прошивати FPGA мікросхему через JTAG інтерфейс за допомогою програматора Blaster.

Драйвер програматора Blaster повинен автоматично встановитися після його першого підключення, але для цього може знадобитися доступ до мережі Інтернет.

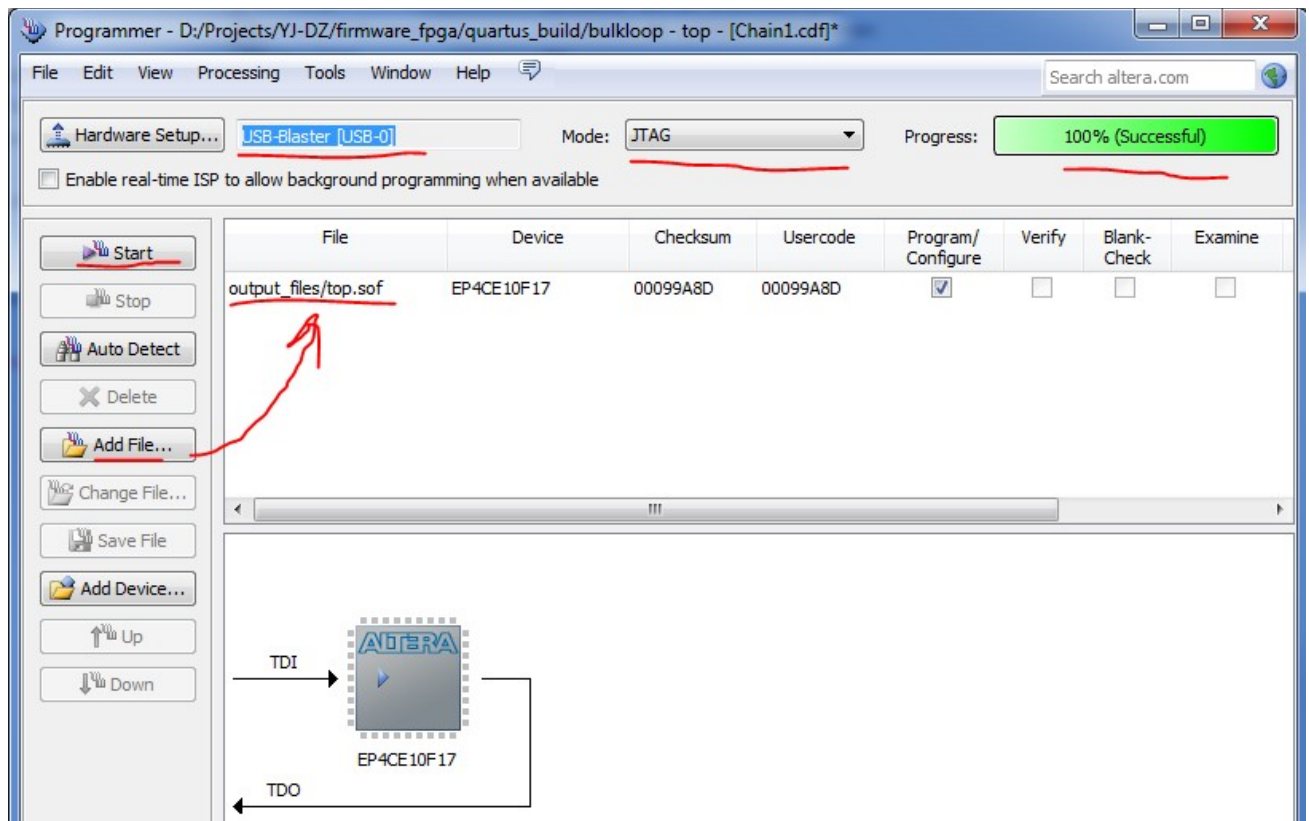
Відкрийте за допомогою Quartus демонстраційний проект для FPGA мікросхеми. Він відсилає назад всі дані що приходять по USB попередньо їх інвертуючи (побітово).

Перекомпілюємо проект (подвійний клік на Compile Design):



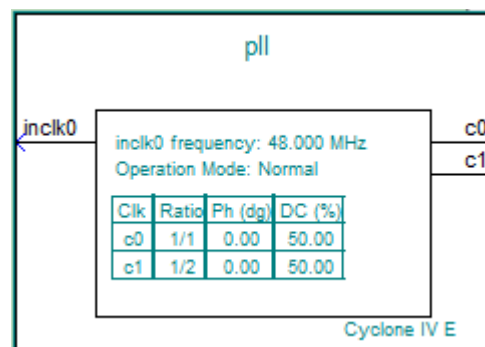
Та залийте:

Tools - Programmer



Увага: JTAG коннектор повинен бути підключений до ввімкнення живлення на платі, інакше можна пошкодити FPGA мікросхему!

Проект складається з одного VHDL файлу – `top.vhd`, а піни присвоювалися згідно схеми розведення компонентів на платі – `docs/Schematic.pdf`. Також проект повинен містити pll компонент з наступними параметрами:

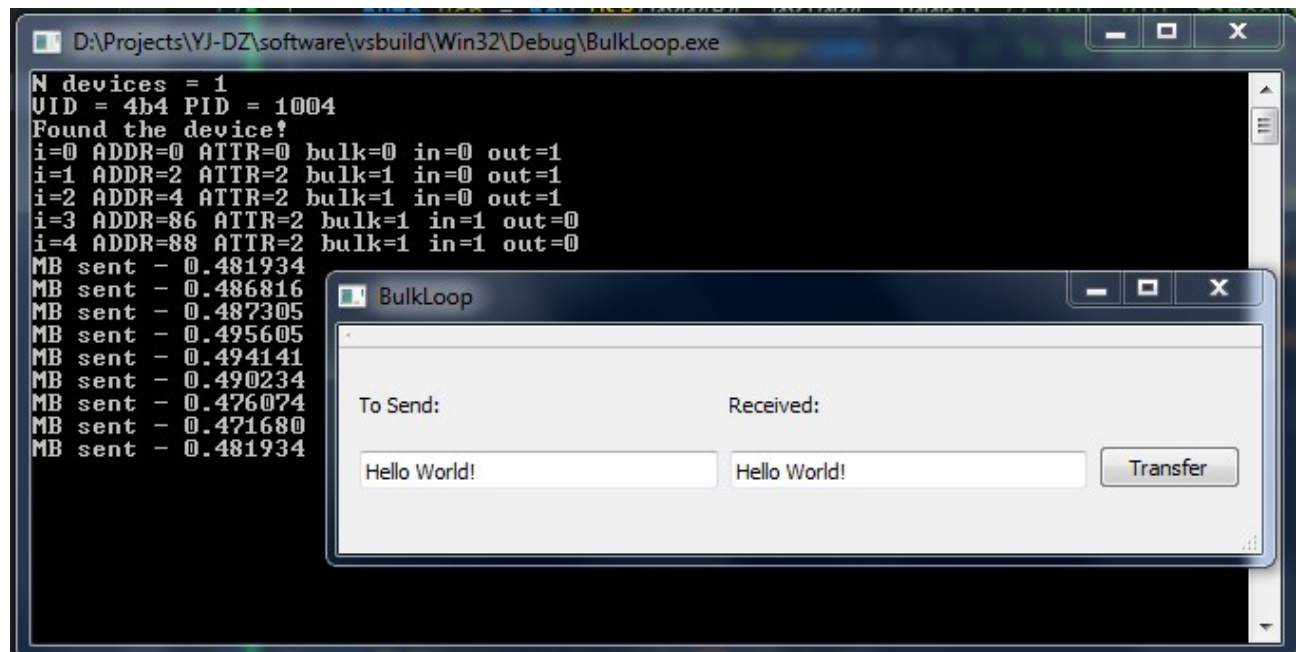


Розпіновку завантажується з фалу `firmware_fpga/src/top.qsf`.

Програма користувача

У EZ-USB FX3 SDK йде в комплекті C++ бібліотека CyAPI яка є програмним інтерфейсом до драйвера CyUSB. Демонстраційний проект використовує графічний інтерфейс Qt та статично лінкується з даною бібліотекою (вважається що читач знає як це робиться, єдиний нюанс, що потрібно ще додати до списку стандартну бібліотеку SetupAPI.lib).

При запуску програма намагається ідентифікувати USB пристрій та починає безперервно посилати спеціальний idle-код (0xFE), очікуючи отримати його інвертовану версію. Якщо отримується інше значення, то воно інтерпретується як корисні дані. Коли потрібно відправити корисні дані, то вони вставляються замість idle-кодів.



Роботу з USB заховано в спеціальному класі USB, який повинен виконуватись в паралельному потоці.

Для коректного обміну даними між різними потоками використовується стандартний для Qt механізм – сигнали і слоти. Існують три сигнали/слоти класу USB для яких є відповідні слоти/сигнали в класі головного вікна. Сигнал send імітується для вставки даних в чергу для відправки. Перед тим як імітувати цей сигнал потрібно емітувати waitFor через який передається очікуваний розмір відповіді. Слот received викликається автоматично, коли буде отримано вказану кількість даних.