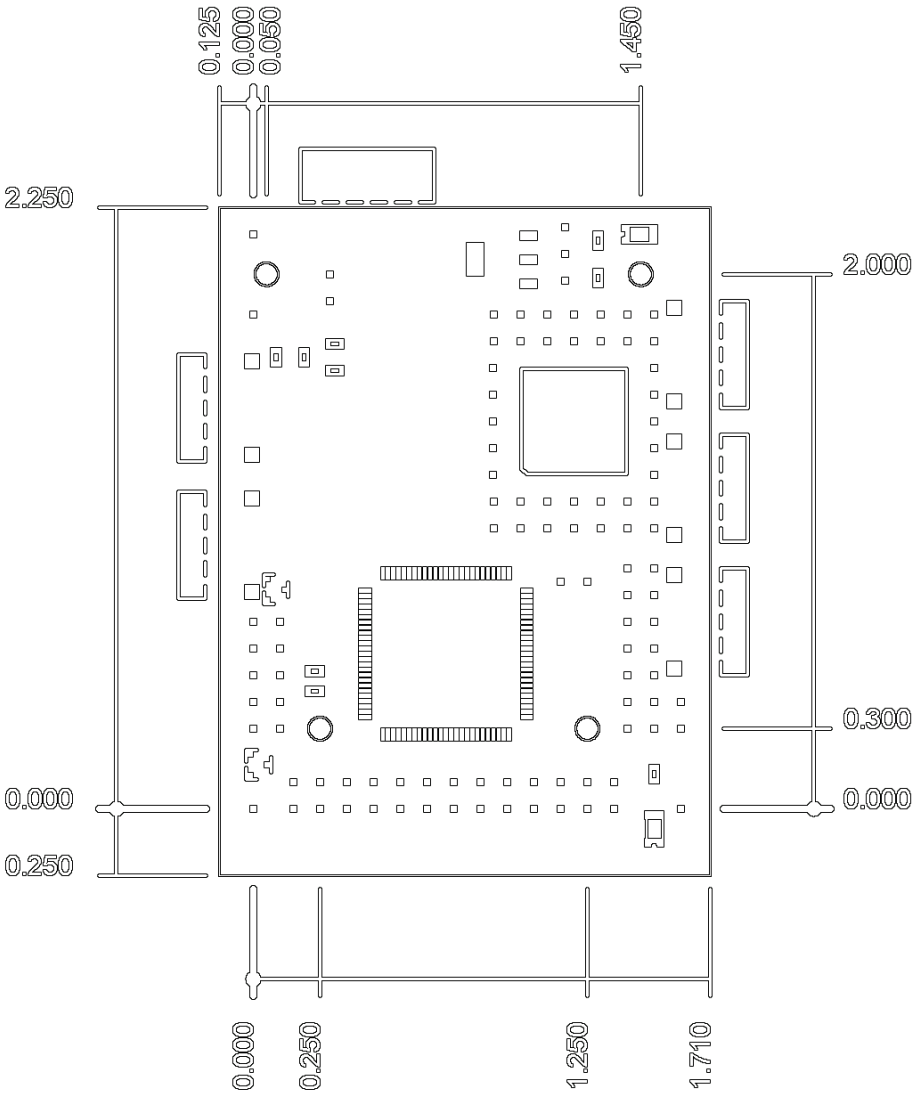
KNJN FX2 FPGA development boards

http://www.knjn.com/

This document applies to the following boards.

- Saxo (revision G)
- Saxo-L (revisions B & above)
- Saxo-Q
- Xylo (revision E)
- Xylo-E (revisions B & above)
- Xylo-EM
- Xylo-L
- Xylo-LM
- Dragon-E (revision B & above)



Last revision **April 13, 2017**

Table of Contents

1 Welcome

1.1 This guide

Welcome to the KNJN FX2 FPGA development board guide. It is partitioned in short and easy to read chapters, and explains how to work with your new FPGA board.

1.2 FPGAs the easy way

Although FPGA boards can be intimidating, KNJN FPGA boards are easy to use. KNJN FX2 boards work right out of the box with a simple USB connection, so that you can get up to speed quickly and concentrate on your task.

For example, one of the first tasks this document walks you through is the FPGA configuration. Basically you plug your board into one of your Windows PC's USB ports, go through the Windows wizard to install a provided USB driver, open the FPGA configuration software, select one of the provided bitfiles and click on the “Configure!” button.

Voila! You've configured the FPGA within just a few minutes.

That's only the beginning. Through this document, you will see how easy it is to program the FPGA boot-PROM, speak to I2C peripherals, transform your board into a digital oscilloscope, transfer data from the FPGA to the PC, drive an LCD...

Welcome to the FPGA world.

1.3 ARM processor

If your board has an ARM processor, you also need the “KNJN FX2 ARM boards” guide. The KNJN guides are available from http://www.knjn.com/docs/

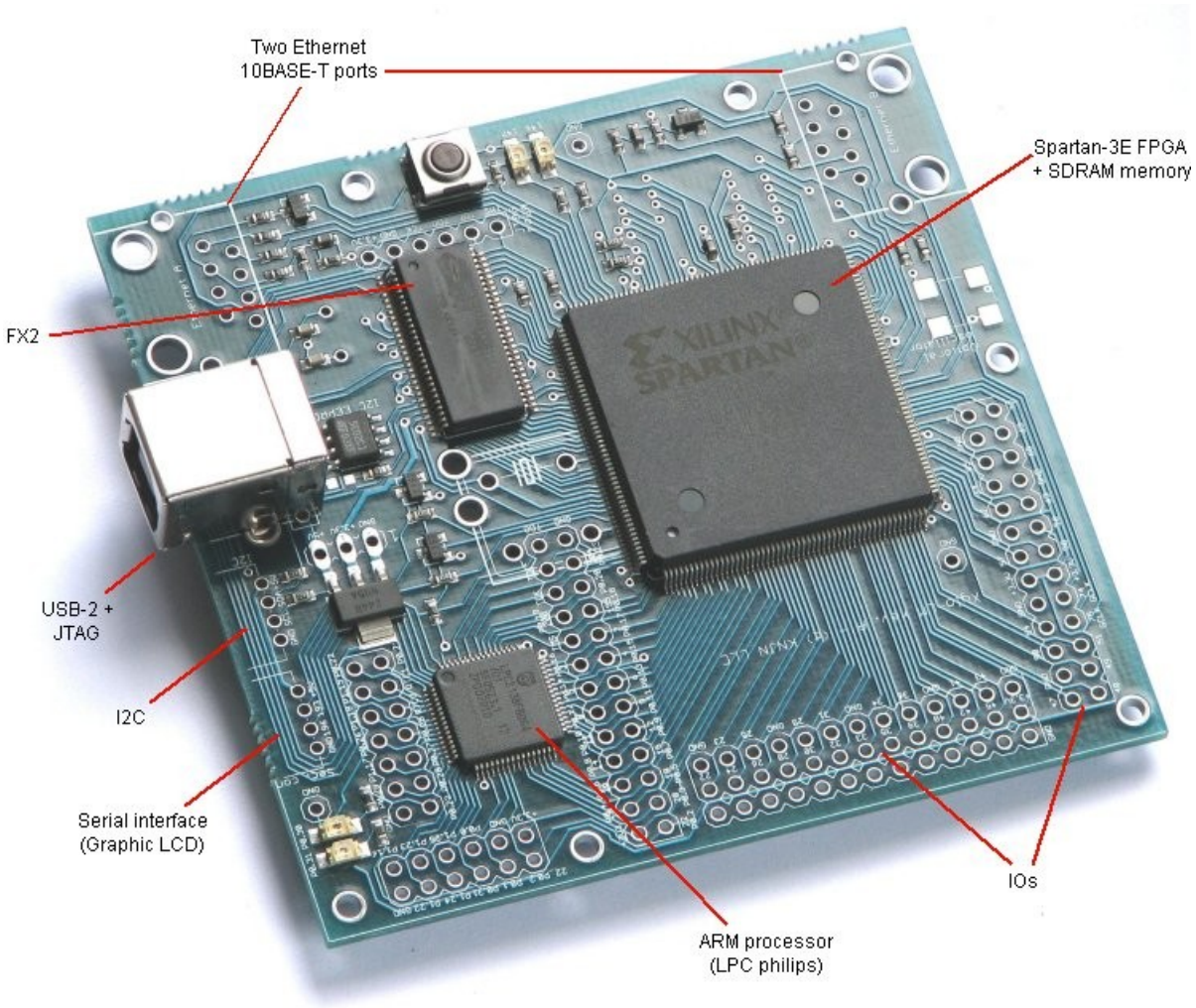
1.4 Windows and other OSes

The KNJN boards are fully supported on the latest versions of Windows (XP / Vista / 7 / 8 / 8.1 / 10). They can also be used in JTAG mode with other OSes like Linux and Mac OS (chapter 35).

2 Boards features

2.1 The KNJN FX2 FPGA family of boards

The KNJN FX2 FPGA boards are based on Xilinx and Altera FPGAs, plus LPC213x ARM processors.



"Xylo-LM", member of the KNJN FX2 family of boards.

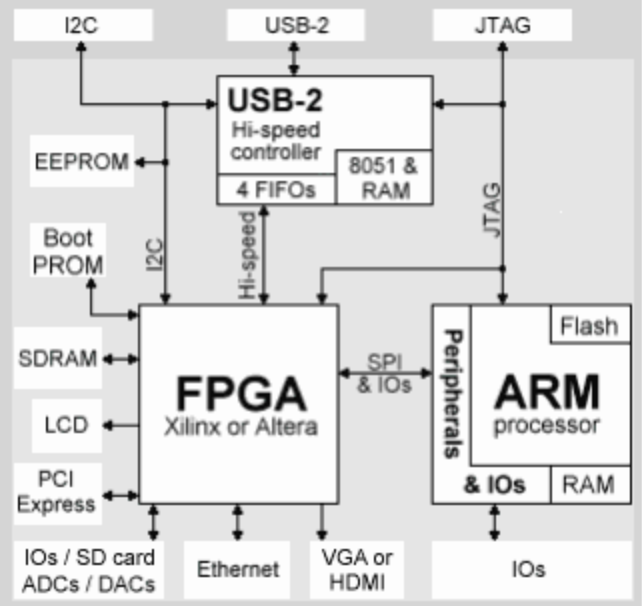
Like all KNJN FPGA boards, the FX2 boards features an easy to use link to a PC. Here, that's a high-speed USB-2 port, which allows typical PC ↔ FPGA communication speeds of 30MB/s to 40MB/s (240Mbit/s to 320Mbit/s).

The KNJN FX2 FPGA family of boards currently has eight members. Each member has unique features, the latest ones are:

- Xylo-E with HDMI, SDRAM plus optional SD card and Ethernet.
- Saxo-Q with several high-speed analog channels (ADCs and DACs).
- Dragon-E with a PCI Express interface.

2.2 Block diagram and features checklist

Here's the KNJN FX2 FPGA family of boards block diagram, check the table below for the particular features of each board.



Board name	Saxo	Saxo-L	Xylo	Xylo-E	Xylo-EM	Xylo-L	Xylo-LM	Saxo-Q	Dragon-E
Main PC interface(s)	USB	USB	USB	USB	USB	USB	USB	USB	PCI Express & USB
FPGA	Altera EP1C3	Altera EP1C3	Altera EP1C3	Xilinx XC6SLX9	Altera EP2C5	Xilinx XC3S500E	Xilinx XC3S500E	Altera EP2C5	Xilinx XC5VLX20T
Datasheet	Cyclone	Cyclone	Cyclone	Spartan-6	Cyclone II	Spartan-3E	Spartan-3E	Cyclone II	Virtex-5
Logic cells	2910	2910	2910	9152	4608	10476	10476	4608	12480
FPGA boot-PROM	1Mbit	1Mbit	1Mbit	4Mbit	4Mbit	4Mbit	4Mbit	4Mbit	8Mbit
IOs / clocks	36 / 4	40 / 4	36 / 4	32 / 5	17 / 3	104 / 7	58 / 5	0 / 3	130 / 13
PLLs / DCMs	1 / 0	1 / 0	1 / 0	2 / 4	2 / 0	4 / 0	4 / 0	2 / 0	1 / 2
ARM processor	-	LPC2132/ LPC2138	-	-	-	LPC2132/ LPC2138	LPC2132/ LPC2138	-	-
SDRAM	-	-	-	16Mbit	16Mbit	-	256Mbit	-	-
SD card	-	-	-	optional	-	-	-	-	-
Ethernet	-	-	10BASE-T	optional	10BASE-T	10BASE-T	10BASE-T x 2	-	-
I2C master/bus	yes	yes	yes	yes	yes	yes	yes	yes	yes
High-speed USB-2	yes	yes	yes	yes	yes	yes	yes	yes	yes
Text LCD connector	yes	yes	yes	yes	-	-	-	-	yes
Graphic LCD ready	yes	yes	yes	yes	yes	yes	yes	yes	yes (LVDS)
Video connector	-	-	VGA	HDMI	VGA (note 2)	-	-	-	-
Analog outputs (DAC)	-	-	-	-	-	-	-	2 x 10-Bit 165MSPS	-
Analog inputs (ADC)	- (note 1)	- (note 1)	- (note 1)	- (note 1)	- (note 1)	- (note 1)	- (note 1)	4 x 8-Bit 200MSPS	- (note 1)
Dimensions	41x60mm	44x60mm	57x60mm	42x56mm	57x60mm	81x82mm	81x77mm	84x77.5mm	122x89mm (3)

Note 1: FlashyD ready (two analog inputs). FlashyD is sold separately.
Note 2: Five VGA IOs are shared with the IOs of the main board header.
Note 3: Dimensions don't include the PCI Express connector.

3 Software tools

3.1 Important downloads

Each KNJN FPGA board is provided with a “startup kit” that includes documentation and other files (mainly example source codes). The startup kit doesn't include some important software tools that are required to work with the FPGA board:

- An “FPGA software”
- A C/C++ compiler

3.2 FPGA software

An FPGA software will allow you to generate FPGA bitfiles.

Each FPGA vendor provides a software with a complete tool chain (synthesis and P&R), nicely integrated together. The software is available in a free version (which may be all you'll ever need) and a non-free version (with typically more features and support for bigger devices).

Get the software that matches your board.

- For Xylo-E/-L/LM & Dragon-E, download ISE WebPACK 14.7
- For Saxo, Saxo-L, Saxo-Q, Xylo and Xylo-EM, get Quartus II Web Edition 11.0 SP1

3.3 C/C++ compiler

A C/C++ compiler is optional but you'll need one for many projects.

Here are different C compilers that can be used:

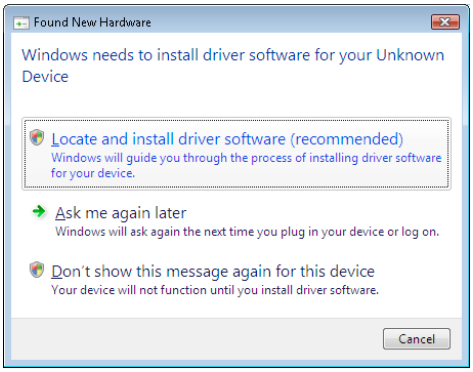
- Microsoft Visual Studio 2010 and 2015, and older 6.0 version
- Digital Mars (free download)
- Jacob Navia's lcc-win32 (free download)

4 Board installation

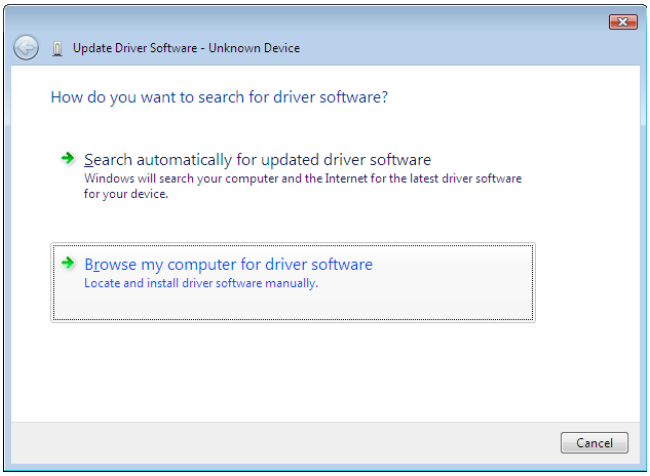
4.1 USB driver installation

Let's plug your board in!

1. Connect your board to one of your PC's USB ports. Windows detects the board and starts its “Hardware Update Wizard”.

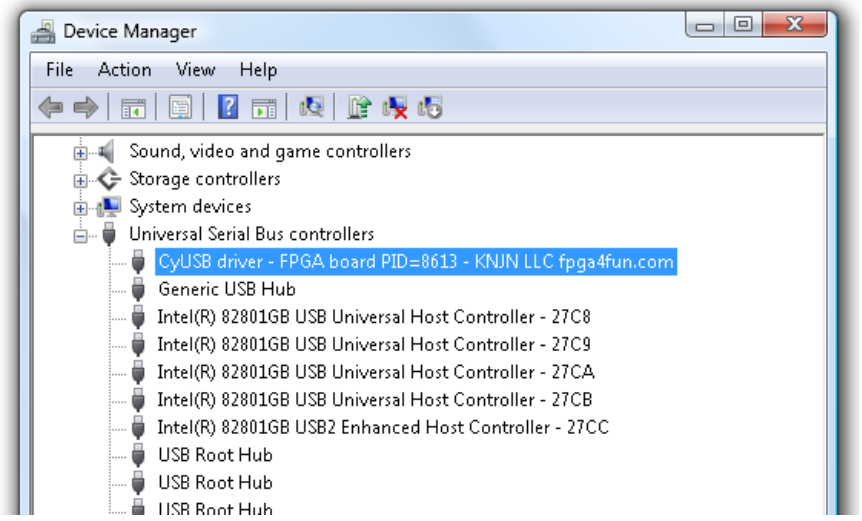


2. Instruct the wizard to use CyUSB (from the startup kit “USB drivers” directory). If Windows asks about the driver not being Windows-certified, select “continue anyway”.
   For Windows 7 and above, the wizard may not offer the ability to select the driver directory and then fail... The workaround is simple. Complete the wizard and with the board still connected, open Windows Device Manager, find the board in the USB devices list, and update the driver there.



4.2 Windows device manager

Once the driver is installed, Windows device manager shows the board entry in the USB controllers section.



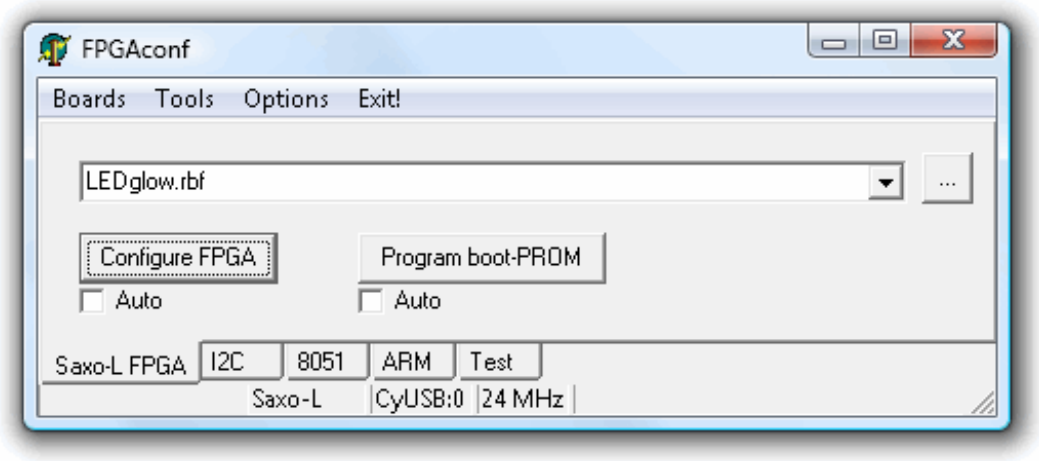
5 FPGA configuration

Now that the board is connected and the USB driver installed, let's configure the FPGA.

1. Run FPGAconf
2. Go to menu → Boards and select your FPGA board.
3. Go to menu → Options → USB driver and select the USB driver you chose earlier (CyUSB usually). If you are unsure which driver is used, open Windows device manager (see 4.2).

We are ready to configure the FPGA:

4. Click on the browse button to select an FPGA bitfile (the browse button is shown as "..."). For your convenience, sample "ledblink" and "ledglow" bitfiles are provided in the startup kit ("FPGA Project - LED").
5. Click on "Configure FPGA".
   After a few seconds, the FPGA is configured.



Now **right-click** on the "Configure FPGA" button and select "Un-configure FPGA". You'll notice that when the FPGA is powered but not configured, it is idle and the FPGA LEDs glow slightly. With practice, you'll be able to recognize immediately if the FPGA is configured or not.

6 FPGA boot-PROM

Your board has an FPGA boot-PROM. Let's learn what it is and how to use it.

6.1 FPGA at power up

An FPGA starts un-configured (idle) and every time it's unpowered, it loses its configuration.

If you need to use your FPGA without a PC attached (in standalone mode), that's a problem. The workaround is to configure the FPGA automatically at power-up. The easiest way to do that is with an FPGA boot-PROM – and that's what KNJN boards do.

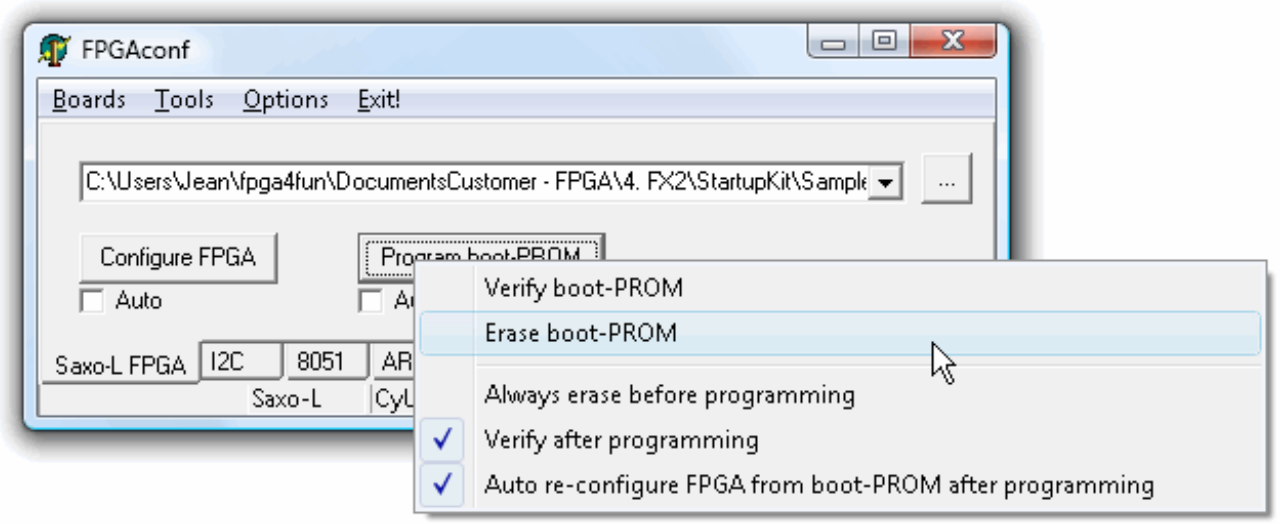
Here's how it works. The boot-PROM is a small memory that is attached to the FPGA. You program it with an FPGA bitfile. At power-up, the FPGA tries to read the boot-PROM (that's automatic and takes a split-second). If the boot-PROM contains a valid bitfile, the FPGA gets configured. If the boot-PROM is empty or its content is invalid, the FPGA is not configured and stays idle.

After power-up, the boot-PROM gets “out of the way” so that you can always configure the FPGA.

6.2 Boot-PROM and USB

With FPGAconf, the boot-PROM can easily be programmed, verified and erased through USB.

- Programmed: click on the “Program boot-PROM” button.
- Verified or erased: right-click on the button and use the drop-down menu.



If you have troubles with the boot-PROM, check 8.3.

6.3 Boot-PROM and JTAG

The boot-PROM can also be programmed through the FPGA JTAG port.

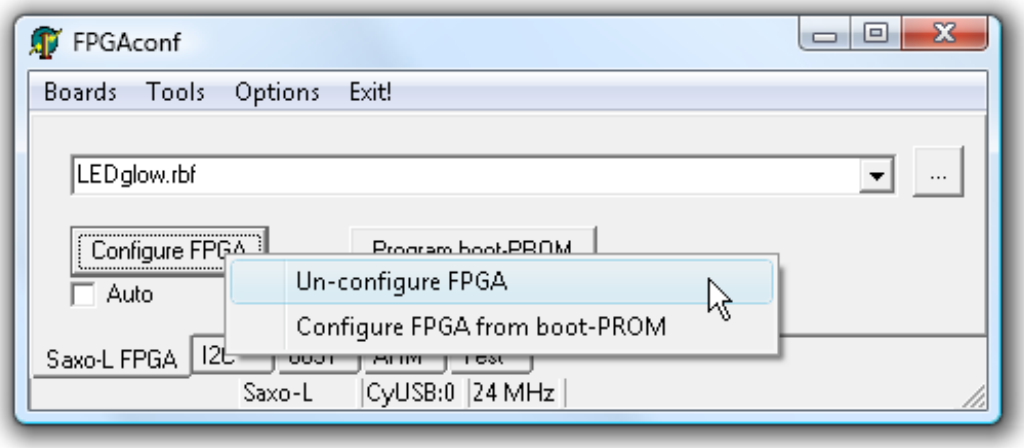
- For Xilinx FPPAs, check http://www.xilinx.com/bvdocs/appnotes/xapp951.pdf
- For Altera FPGAs, check http://www.altera.com/literature/an/an370.pdf and also chapter 24 of this document.

7 Advanced FPGA control

7.1 More FPGA actions

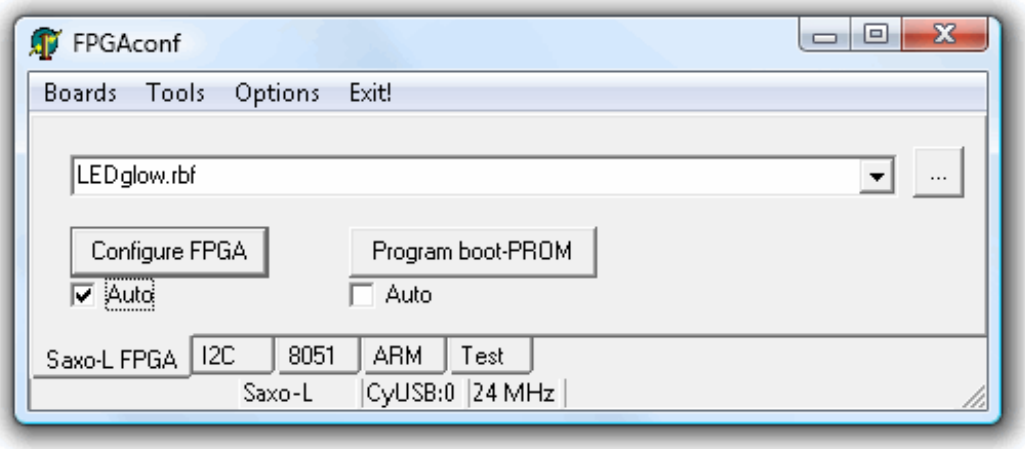
Right-click on the “Configure FPGA” button to see more available FPGA actions:

- Un-configure FPGA.
- Configure FPGA from boot-PROM.



7.2 Auto-mode

Two “Auto” check-boxes (below the “Configure FPGA” and “Program boot-PROM” buttons) allow configuring the FPGA or programming the boot-PROM every time the FPGA bitfile is updated on the hard-drive (like when Quartus-II or ISE finishes a compilation).



7.3 Options

Go to menu → Options to see the list of options available. Important ones are:

- USB driver. Makes sure this matches the driver installed (see 4.1 and 4.2).
- FX2 clock speed (12, 24 or 48 MHz). This takes effect when FPGAconf configures the FPGA.

FPGAconf saves all settings upon closing.

7.4 Log

If you increase the size of the FPGAconf window, you get access to the log panel which remembers and time-stamps each action performed.

8 Your own FPGA project

8.1 LEDblink

Here's a simple Verilog source file that makes an LED blink on the board. It is simple and uses only 2 pins so is a good candidate for a first FPGA project.

```
module LEDblink(input clk, output LED);

// first create a 32 bits counter
reg [31:0] cnt;
always @(posedge clk) cnt <= cnt + 1;

// and take one of the counter bit to drive the LED
assign LED = cnt[23];  // here 24th bit... change that to make the LED blink faster or slower
endmodule
```

To create the FPGA project, follow the instructions from chapter 9 and 10, and make the correct pin assignments using information from chapter 11.

8.2 The FPGA doesn't configure?

If FPGAconf fails to configure the FPGA using one of your own FPGA bitfile, check the following:

- Try first with a pre-compiled FPGA bitfile from the startup kit (like LEDblink or LEDglow) since they are known to be good (this makes sure the board works).
- Now with your own FPGA bitfile:
  - Make sure all the pin assignments are correct.
  - Make sure you followed all the recommendations listed in the chapter 9 or 10 (in particular the one regarding unused pins/IOBs).

8.3 The boot-PROM fails to configure the FPGA?

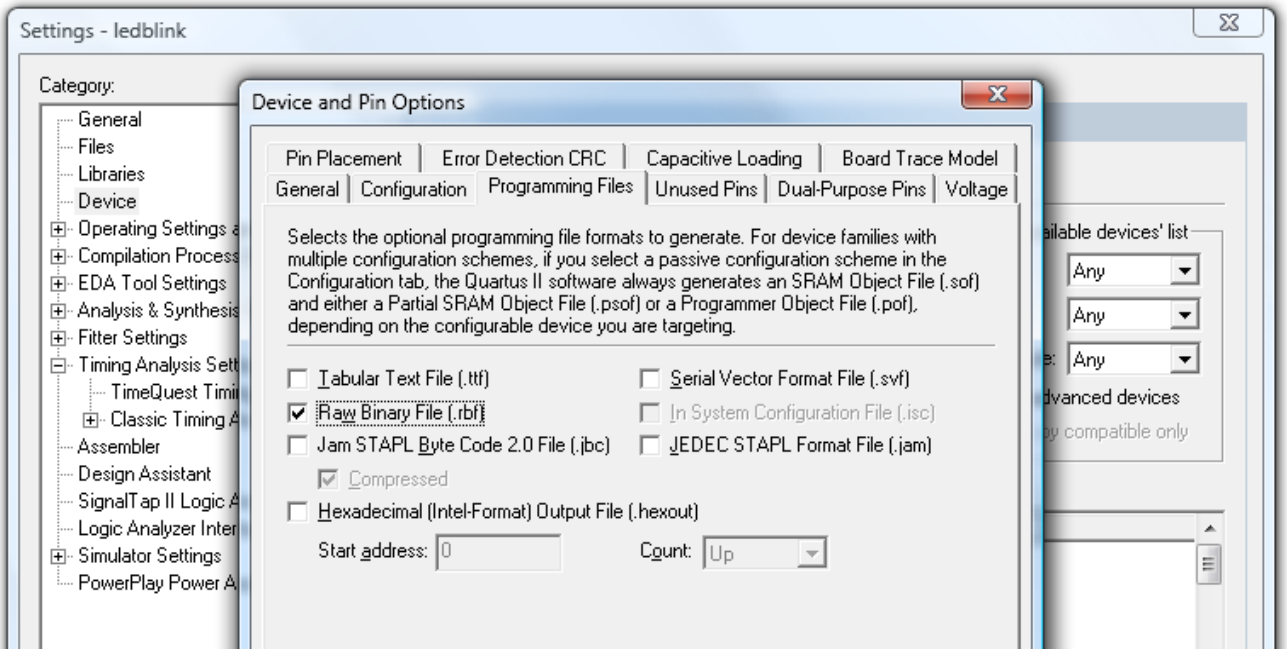
Sometimes you have an FPGA bitfile that works in the FPGA, but when programmed in the boot-PROM, it fails to configure the FPGA.

- First, use one of the FPGA bitfiles from the startup kit (like LEDblink or LEDglow) to program the boot-PROM and make sure it configures the FPGA at power-up.
- Then go back to your own bitfile, and compare its programming properties (Device and Pins options in Quartus, or the Programming File properties in ISE) with a known good project. In particular:
  - If you are using Quartus-II, make sure you didn't change the project's programming option (it should be "Active serial").
  - If you are using Dragon-E, reduce the Configuration Rate (in ISE "Programming File" properties → "Configuration Options" tab).

9 FPGA projects with Altera's Quartus (Saxo/-L/-Q and Xylo/-EM)

To start an FPGA project with Quartus-II, proceed as follows:

1. Create a Quartus-II project.
2. Go to “Assignments/Device”
   2.1. Choose the Cyclone family and the “EP1C3T100C8” device (for Saxo/-L and Xylo) or the Cyclone-II family and the “EP2C5T144C8” device (for Saxo-Q and Xylo-EM)
   2.2. Click on “Device & Pin Options…”
      a) Go to the “Programming Files” tab, select “Raw Binary File (.rbf)”. Otherwise, only SOF files are created (SOF files are used in JTAG mode, while RBF files are used in USB mode).



   b) Go to Unused Pins”, select “As inputs, tri-stated”
   c) Click “OK”
   2.3. Click “OK”
3. Use the menu → Assignments/Pins and assign the correct pins.

The option 2.2.b is there to prevent the FPGA from driving the pins that are not used in your project (by default, Quartus-II grounds all the unused pins, which often ends-up creating IO contentions). You may change this option back once you know that all the pins of your project are correctly assigned.

For a graphical walk-through, check the Altera Quartus II quick-start guide

10 FPGA projects with Xilinx's ISE (Xylo-L/-LM and Dragon-E)

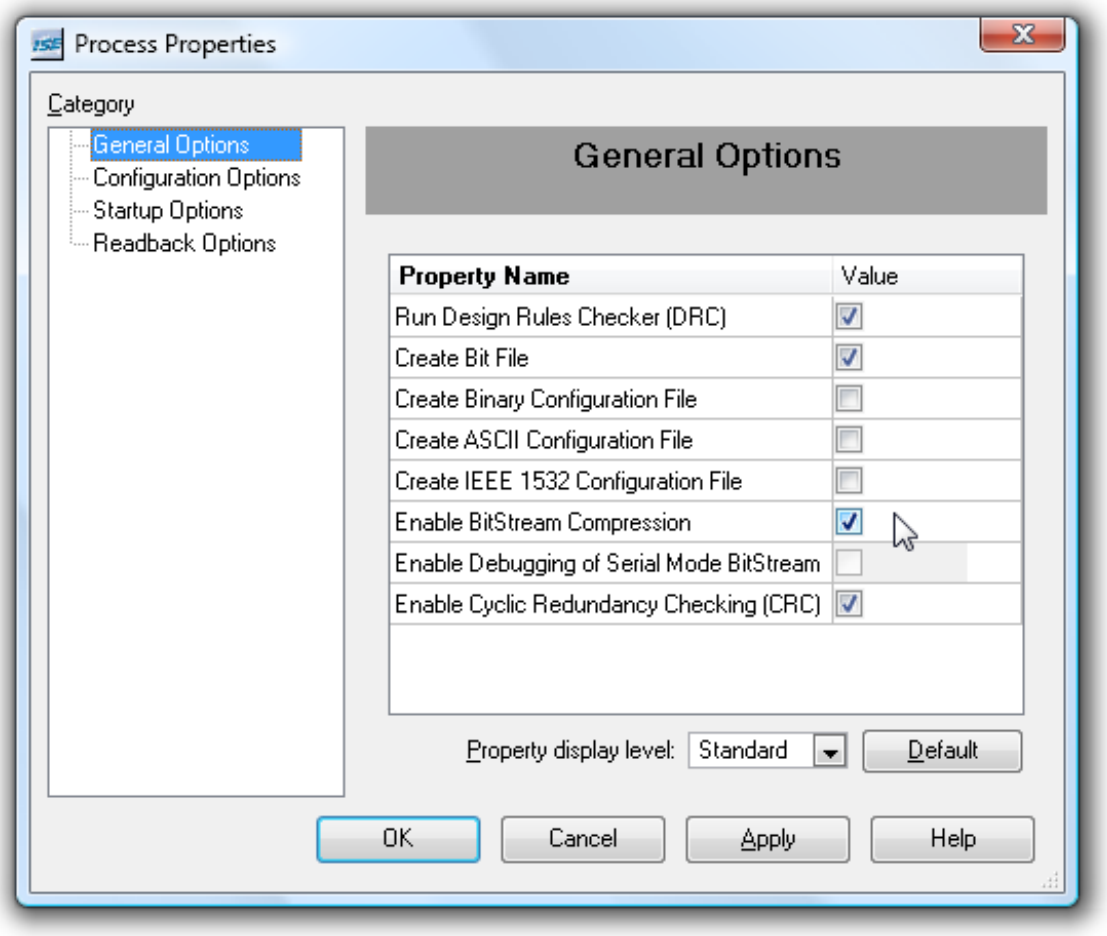
To start an FPGA project with ISE, proceed as follows:

1. Create a new project.
   In the New Project Wizard, select the right FPGA for your board:
   - Xylo-E: choose “Spartan-6” family and the “XC6SLX9” device in a “TQG144” package.
   - Xylo-L/-LM: choose “Spartan3E” family and the “XC3S500E” device in a “PQ208” package.
   - Dragon-E: choose “Virtex-5” family with the “XC5VLX20T-1” in “FF323” package.

   Optionally, select some source files before finishing the wizard.

2. Select the top-level design in your project, right-click on “Generate Programming File” and choose “Properties”. Select BitStream Compression.

   In “Configuration Options”, go to “Unused IOB Pins” and choose “Pull Up”.



3. If you are using Dragon-E, reduce the Configuration Rate (in ISE “Programming File” properties → ”Configuration Options” tab) as the maximum speed is not reliable.
4. Create a UCF file with the pin assignments.

For a graphical walk-through, check the Xilinx ISE quick-start guide

11 FPGA pins

11.1 Common pin assignments

This table shows which pins are used for clocks, LEDs and push-buttons.

Pin name	FPGA	Saxo	Saxo -L	Saxo -Q	Xylo	Xylo -E	Xylo -EM	Xylo -L	Xylo -LM	Dragon -E	Comments
CLK	Input	10	66	17	10	14	17	181	181	R9	Main clock. Software selectable 12, 24 or 48 MHz.
CLK_ADC	Input			88							75.7575MHz clock
CLK_DIL8	Input		10	89 (1)	34		91	183	183		Optional DIL-8 clock oscillator
CLK_SMD	Input			89 (1)				184	184	V6	Optional SMD clock oscillator
CLK_SMD2	Input								80		Second optional SMD clock oscillator
LED1	Output	91	35	92 (2)	91	27	72	146	146	V2	Active high
LED2	Output				97	30	71	147	147	V3	Active high
LED3	Output									D8	Active high
Push-button	Input				92		63	148	148		See 11.4
vga_h_sync					99		133				VGA H-sync, active low
vga_v_sync					100		132				VGA V-sync, active low
vga_red					2		112				VGA red
vga_green					1		113				VGA green
vga_blue					98		114				VGA blue

Note 1: Do not mount both oscillators, as they share the same board trace and FPGA pin.
Note 2: Shared with DAC_CTRL1 (see 28.2).

For more pin assignments, check your board layout (chapter 38).

11.2 Clocks

CLK is the main board clock. It is generated by the USB-2 controller (named “FX2” in this document) and is also the clock used by the internal FX2 bus (see paragraph 12.2). CLK defaults at 12MHz at power-up but can also run at 24 or 48MHz (see paragraph 7.3).

You can add more clocks:

- One or several PLL or DLL in the FPGA are available to generate custom frequencies (see your FPGA documentation for details).
- On some boards, a 3.3V oscillator can be added with a specific clock frequency. The oscillator is in DIL-8 (socket-able) or SMD5x7 (to solder) form.
- Many other pins can also be used as clock sources. They are shown as CLK, DPCLK, GCLK, GC or CC pins on the chapter 38 drawings.

11.3 LEDs

The LEDs are active high, so to light up an LED, output a “1” (from the right FPGA pin – see table above).

Note that when the FPGA is not configured, the LEDs glow slightly (because of weak-pullups in the FPGA IOs that are then activated).

11.4 Push-button

The push-button is an input to the FPGA.

- It is usually active low (reads as “1” when the push-button is not pressed, and “0” when it is).
- On newer board Xylo revision H and Xylo-L/-LM revision D and above, it is active high (reads as “0” when the push-button is not pressed, and “1” when it is).

11.5 VGA (Xylo/-EM only)

The VGA interface uses five FPGA pins and is able to generate eight colors on a VGA monitor. The pong game provides an example design that makes use of the VGA. Note that the RGB impedance matching resistors are already present on the board, so to connect a VGA monitor, simply solder a DB-15HD connector to your board.

11.6 Secondary connector

The secondary connector is a small connector that has only four pins. It provides power plus two IOs.
On some boards, the two IOs have serial terminations, plus weak over-voltage protections:

- Serial terminations: useful for high-speed serial output signals (like a graphic LCD, chapter 25).
- Weak over-voltage protection: useful for a low speed I2C bus.

Note that Dragon-E has two secondary connectors (one of them powered with +5V and the other by +3.3V).

Connector pin	Saxo / Xylo	Saxo-L	Saxo-Q	Xylo-E	Xylo-EM	Xylo-L / Xylo-LM	Dragon-E bottom	Dragon-E top	Comment
1. VCC	+5V	+5V	+5V	+3.3V	+5V	+5V	+5V	+3.3V	Power pin
2. IO1	65	91	143	83	64	193	T12	T16	RxD or serclk or other
3. IO2	68	97	144	84	57	196	U10	V15	TxD or serdata or other
4. GND	0V	0V	0V	0V	0V	0V	0V	0V	Power pin
Termination/ WOVP	Yes	Yes	No	No	Yes	Yes	No	No	Serial termination and weak over-voltage protection

11.7 Text LCD (Saxo/-L & Xylo/-E only)

See chapter 30 for more info.

11.8 I2C (all boards)

The I2C bus is connected to the FX2 hard-macro I2C controller. See chapter 16 for details.

On Saxo and Xylo, the I2C bus is internally connected to the FPGA pins 51 (SDA) and 52 (SCL).

On Saxo-L/-Q and Xylo-E/-EM/-L/-LM, the I2C bus is not internally connected to the FPGA. You can connect the FPGA to the I2C bus externally by using wires or an I2C switch that links the secondary connector to the I2C connector. Then configure the FPGA with a soft I2C controller (master or slave).

Check KNJN's I2C accessories page for I2C switches and other peripherals.

11.9 Ethernet signals (Xylo/-E/-EM/-L/-LM only)

For Xylo-E, an optional Ethernet adapter is available. The other boards have one or two dedicated Ethernet headers. Each header uses three FPGA pins named RD, TDp and TDn.

Pin names	Xylo	Xylo-L	Xylo-LM	Xylo-EM
RD1, TDp1, TDn1	21, 3, 4	150, 151, 152	150, 151, 152	143, 139, 142
RD2, TDp2, TDn2			77, 78, 82	

11.10 HDMI (Xylo-E only)

Check the “StartupKit\FPGA project – HDMI” folder.

11.11 SD card (Xylo-E only)

The SD card is an Xylo-E option that mounts on the bottom of the board. Check the “StartupKit\FPGA project – SD” folder for more details.

12 USB/FX2 interface

The USB interface makes using your FPGA board easy as it allows to:

1. Configure the FPGA
2. Communicate with the FPGA (once it is configured).

This chapter describes the communication with the FPGA. Chapter 13 describes the PC function calls and chapter 14 gives complete examples of communication.

Your board uses a Cypress CY7C68013 high-speed USB-2 chip (nicknamed “FX2”) that is the interface between the PC's USB-2 link and the FPGA. It achieves typical application level speeds of 30MB/s to 40MB/s (240Mbps to 320Mbps).

12.1 Bulk vs. Isochronous

The USB protocol defines two types of data packets:

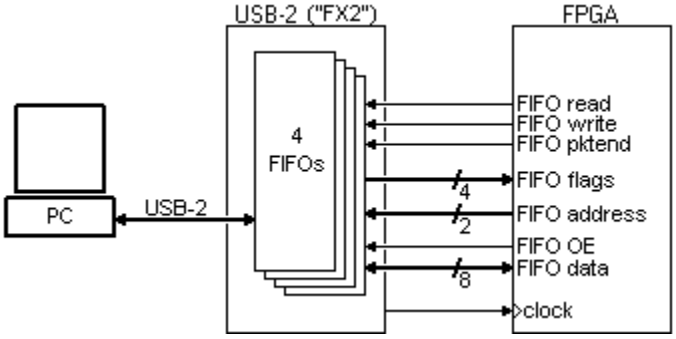
- Bulk packets (guaranteed delivery)
- Isochronous packets (guaranteed bandwidth)

Bulk packets guarantee delivery, which means that if there is an error on the line, the data is retransmitted automatically until it is received correctly. This is done in the background at the USB hardware level, so that the FPGA ends up only receiving guaranteed good data from the PC, and the PC ends up receiving guaranteed good data from the FPGA. Note that bulk packets can only be up to 512 bytes long (for the USB 2.0 high-speed mode used by the FX2), but multiple of them are automatically sent back-to-back for longer transmissions.

Isochronous packets don't guarantee delivery, so KNJN FX2 boards use bulk packets exclusively.

12.2 FX2 interface and FIFOs

The FX2 is connected to the FPGA through an 8bit bidirectional data bus, a clock, and a few extra signals.



The FX2 implements four hardware FIFOs, named FIFO2, FIFO3, FIFO4 and FIFO5.

- FIFO2 and FIFO3 are used to send (i.e. PC → FPGA). The PC (through the USB link) writes to these FIFOs, and the FPGA reads from them.
- FIFO4 and FIFO5 are used to receive (i.e. PC ← FPGA). The FPGA writes to these FIFOs, and the PC (through the USB link) reads from them.
- Each FIFO can hold up to 1024 bytes of data (or more precisely two times 512 bytes, more details later).

12.3 PC to FPGA

When the PC software sends data to FIFO2 or FIFO3, it specifies how many bytes to send. Anything bigger than 512 bytes is automatically sliced on the USB line, so the FPGA doesn't need to be aware of the USB packets boundaries.

12.4 FPGA to PC

When the FPGA wants to send data to the PC, it writes data to FIFO4 or FIFO5. What happens if the FPGA wants to send less than 512 bytes?

- The FPGA can keep writing (dummy data) until it reaches 512 bytes (padding method).
- The FPGA can assert a special line named “FIFO pktend” that signals to the FX2 that a packet is complete.

Note that:

- Data is committed to the FIFO (and available for the PC to read) only when one of the two previous conditions is met.
- Each FIFO can hold 2 packets. So each FIFO can hold up to 1024 bytes, but may be full by having less than that if the packets are completed using “FIFO pktend”.

12.5 FIFO flags

The FX2 provides one flag signal per FIFO. The flags are named FLAG2, FLAG3, FLAG4 and FLAG5.

- FLAG2 and FLAG3 are “empty” flags. They indicate that FIFO2 and FIFO3 are empty
- FLAG4 and FLAG5 are “full” flags. They indicate that FIFO4 and FIFO5 are full.

The flag signals are read by the FPGA. The FPGA controls when it reads and writes from the FIFO, but it has to follow these rules:

- The FPGA can only read from FIFO2 or FIFO3 if it is not empty.
- The FPGA can only write to FIFO4 or FIFO5 if it is not full.

12.6 FIFO data, address and control lines

The FPGA controls which FIFO he accesses using a 2-bits wide address bus named “FIFO address”.

FIFO address	FIFO accessed
00	FIFO2
01	FIFO3
10	FIFO4
11	FIFO5

Note that:

- The FPGA reads from FIFO2/FIFO3 by addressing one of them and asserting the “FIFO read” signal.
- The FPGA writes to FIFO4/FIFO5 by addressing one of them and asserting the “FIFO write” signal. While writing, the FPGA can signal the end of a packet using the “pktend” signal.
- The FPGA reads and writes to the FIFOs using the bidirectional 8bit wide data bus “FIFO data”. The FPGA also controls “FIFO OE” (which tells the FX2 to drive the data bus or not).

12.7 FIFO signal names

Here's the complete list of signals used internally by the FPGA to access the FX2 FIFOs.

- All are active high.
- For examples of use, check chapter 14.

Signal name	Width	Direction	Purpose
FIFO_CLK	1 bit	FX2 → FPGA	Clock (all the other signals are synchronous to this clock)
FIFO_RD	1 bit	FPGA → FX2	FPGA reads from FX2
FIFO_WR	1 bit	FPGA → FX2	FPGA writes to FX2
FIFO_PKTEND	1 bit	FPGA → FX2	FPGA indicates “End of packet” (i.e. FPGA is done writing)
FIFO_DATAIN_OE	1 bit	FPGA → FX2	FPGA wants the FX2 to drive the DATA bus
FIFO_DATAOUT_OE	1 bit	internal	FPGA drives the DATA bus
FIFO_DATAIN	8 bits	FX2 → FPGA	Data from FX2
FIFO_DATAOUT	8 bits	FPGA → FX2	Data to FX2
FIFO_FIFOADR	2 bits	FPGA → FX2	FPGA selects one FX2 FIFO (out of the four available)
FIFO2_empty / FIFO2_data_available	1 bit	FX2 → FPGA	FIFO2 is empty, or not
FIFO3_empty / FIFO3_data_available	1 bit	FX2 → FPGA	FIFO3 is empty, or not
FIFO4_full / FIFO4_ready_to_accept_data	1 bit	FX2 → FPGA	FIFO4 is full, or not
FIFO5_full / FIFO5_ready_to_accept_data	1 bit	FX2 → FPGA	FIFO5 is full, or not

13 PC access to the FX2 FIFOs

The PC (through the USB link) can write to FIFO2/3 and read from FIFO4/5. Depending of the USB driver installed (chapter 4), you must use a different set of software functions.

13.1 FIFO access with the CyUSB driver

The PC reads and writes to the FIFOs using the `XferData` C++ method:

```
bool XferData(PUCHAR buf, LONG &len)
```

When writing, the len parameter specifies how many bytes we are sending to the FPGA. For example, let's send one byte to FIFO2:

```
unsigned char c = 41;
LONG len = sizeof(c);
BulkOutPipe2->XferData(&c, len);  // send one byte (41) to FIFO2
```

The XferData method is also used for reading from FIFO4/5. In that case, the len parameter value must be the size of the data packet returned by the FX2, or a bigger value. In doubt, always use a multiple of 512 to avoid underflow problems.

For example:

```
unsigned char buf[512];
LONG len = sizeof(buf);
BulkInPipe4->XferData(buf, len);  // read up to 512 bytes
// now len contains the number of bytes actually read
```

13.2 FIFO access with the EzUSB driver

The PC writes to the FX2 FIFO2/3 using a C function like:

```
void USB_BulkWrite(ULONG pipe, void* buffer, WORD buffersize)   // pipe must be 2 or 3
```

The PC reads from the FX2 FIFO4/5 using a C function like:

```
WORD USB_BulkRead(ULONG pipe, void* buffer, WORD buffersize)   // pipe must be 4 or 5
```

With USB_BulkRead, the buffersize parameter value must be the size of the data packet from the FX2, or a bigger value. In doubt, use a multiple of 512 to avoid underflow problems.

The “pipe” number is the FIFO number. So to write to FIFO3, we could use

```
char* str = 'hello';
USB_BulkWrite(3, str, strlen(str));  // write the 5 bytes string “hello” to FIFO3
```

14 FX2 examples

14.1 Example 1: LED control

This is the simplest of our examples. We control the board's LEDs through USB-2.

Let’s use FIFO2. Every time a byte is sent from the PC to FIFO2, the FPGA reads it and updates the LEDs.

The C code looks like this:

```
int main()
{
    int i;
    USB_Open();

    for(i=0; i<100; i++)      // blink the LEDs for a few seconds
    {
        USB_BulkWrite(2, &i, 1);      // send one single byte (= the value of i) to FIFO2
        Sleep(50);                    // and wait 50ms
    }

    USB_Close();
}
```

The Verilog HDL looks like this:

```
assign FIFO_FIFOADR = 2'b00;         // select FIFO2
assign FIFO_RD = 1'b1;               // always read from FX2
assign FIFO_WR = 1'b0;               // never write to FX2
assign FIFO_DATAOUT = 8'h00;         // never write to FX2, so this value is not used
assign FIFO_DATAIN_OE = 1'b1;        // always read data from FX2
assign FIFO_DATAOUT_OE = 1'b0;       // never output data to FX2
assign FIFO_PKTEND = 1'b0;

reg [1:0] LED;
always @(posedge FIFO_CLK)
    if(FIFO2_data_available) LED <= FIFO_DATAIN[1:0];  // when one byte is available in FIFO2, then
                                                       // get 2 bits out (to control the 2 LEDs)
```

The complete code is available in the startup kit (“Projects\USB-2 (FX2) 1 - Blink leds”).

14.2 Example 2: Text LCD

This second example controls a text LCD display connected to the board. See chapter 30 for details on how to make the connection.

In our implementation, any data sent by the PC to FIFO2 is read by the FPGA and sent to the LCD. The FPGA uses a small state machine (just 2 states) to control how to read from FIFO2.

```
reg state;
always @(posedge FIFO_CLK)
case(state)
    1'b0: if(FIFO2_data_available) state <= 1'b1;   // wait for data
    1'b1: state <= 1'b0;                            // read data, go back waiting
endcase
```

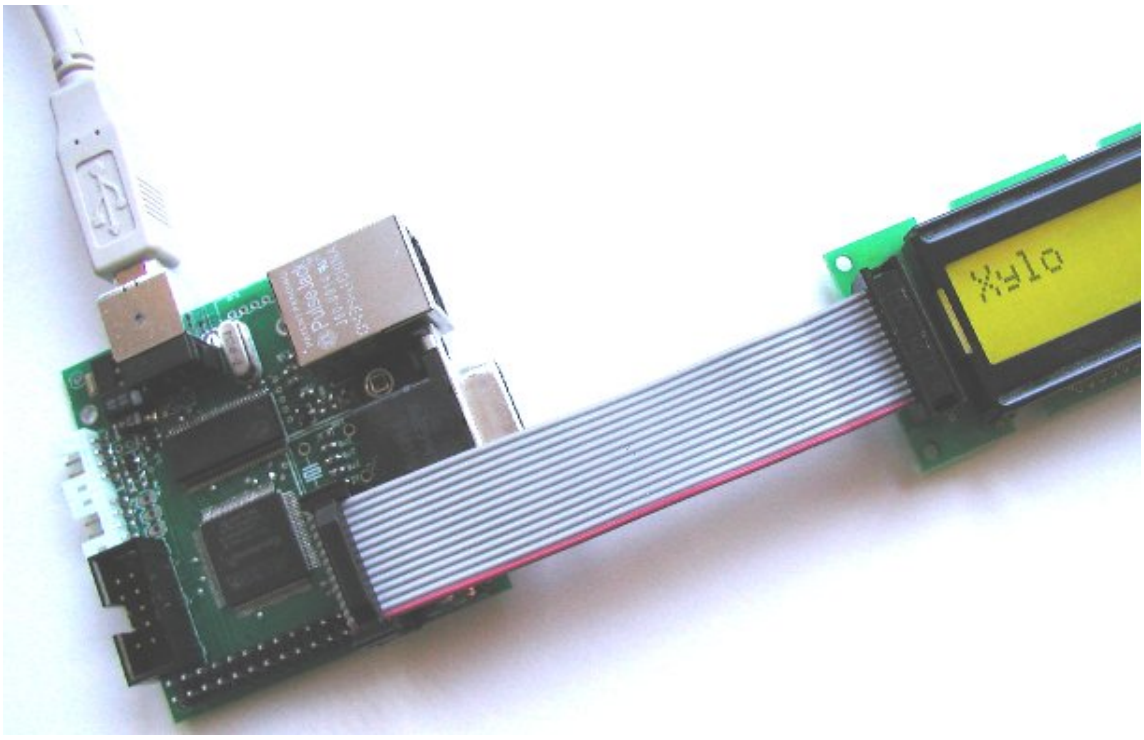
USB transmits data much faster than the LCD can accept it. This example assumes that the PC sends the bytes to the LCD one by one - not very efficient from the USB point of view, but this keeps this example simple.

For LCD command bytes, we send 0x00 followed by a command byte. It is ok to send the 2 bytes at once (in the same USB packet) because the first one (0x00) is not sent to the LCD module but is just there to indicate to the FPGA that the second byte is a command to the LCD.

The C code looks like this:

```
void main()
{
    USB_Open();

    // Commands - Initialize the LCD
    USB_WriteWord(0x3800);    // remember, the PC is little-endian, so that's 0x00 followed by 0x38 !!
    USB_WriteWord(0x0F00);
    USB_WriteWord(0x0100);
    Sleep(2);

    // Data
    USB_WriteChar('X');
    USB_WriteChar('y');
    USB_WriteChar('l');
    USB_WriteChar('o');
    USB_Close();
}
```

The complete code is available in the startup kit (“Projects\USB-2 (FX2) 2 - Text LCD display”).



14.3 Example 3: Bidirectional communication

Here the FPGA waits until some data is available in FIFO2. It counts the number of bytes available and sends the count back to FIFO4.

The HDL code looks like this:

```
reg [2:0] state;
always @(posedge FIFO_CLK)
case(state)
    3'b000: if( FIFO2_data_available) state <= 3'b001;  // wait for data packet in FIFO2
    3'b001: if(~FIFO2_data_available) state <= 3'b100;  // wait until end of data packet
    3'b100: state <= 3'b101;  // switch to FIFO4, turnaround cycle
    3'b101: state <= 3'b110;  // write data
    3'b110: state <= 3'b000;  // end packet, turnaround cycle
    default: state <= 3'b000;
endcase

assign FIFO_FIFOADR = {state[2], 1'b0};  // FIFO2 or FIFO4
assign FIFO_RD = (state==3'b001);

// count the number of bytes received
reg [7:0] cnt;
wire read_byte = (state==3'b001) & FIFO2_data_available;
always @(posedge FIFO_CLK) if(read_byte) cnt <= cnt+8'h1;

// now write the count back to FIFO4
assign FIFO_DATAOUT = cnt;
assign FIFO_WR = (state==3'b101);
assign FIFO_PKTEND = (state==3'b110);
assign FIFO_DATAIN_OE = ~state[2];
assign FIFO_DATAOUT_OE = (state==3'b101);
```

Care needs to be taken so that FIFO_DATA_IN and FIFO_DATA_OUT are never driven together (to avoid a data bus contention). Other timing issues may need to be reviewed. For example, it may be wise to have one idle clock cycle (when nobody's driving the bus) during a bus turnaround, especially before allowing the FPGA to drive the DATA bus when the FPGA just read from the FX2. Refer to the FX2 datasheet for timing information.

The complete code is available in the startup kit (“Projects\USB-2 (FX2) 3 - Bidirectional communication”).

14.4 Example 4: SDRAM (Xylo-EM/LM only)

The example shows one simple way to read and write to the SDRAM from the PC. The SDRAM is used in AUTO-PRECHARGE mode with burst length=1, CAS latency=2.

The source code is available in the startup kit (“FPGA Project - USB-2 (FX2) 4 – SDRAM”).

Note: this FX2 SDRAM design is not provided for Xylo-E. Instead a more complete SDRAM controller is provided (similar to the design published on this fpga4fun's page).

14.5 Example 5: DDS (Saxo-Q only)

This example shows how to create and control a DDS in the FPGA.

The source code is available in the startup kit (“FPGA Project - USB-2 (FX2) 5 – DDS”). See also fpga4fun's DDS project for more information.

15 Ethernet (Xylo-E/-EM/-L/-LM only)

15.1 Ethernet board setup

With Xylo-E, connect the Ethernet adapter to the secondary connector. On the other boards, solder the provided PulseJack RJ-45 connector if it is not already present.

Use a regular network cable from a network hub/switch to the RJ-45. If you want to connect your board directly to a PC, use a “crossover” network cable.

15.2 Ethernet HDL reference design

A reference design is provided allowing bi-directional 10BASE-T communication. The reference design is provided in source code form only.

The design provides an example of UDP/IP transmission and reception.

- Transmission: a packet is sent at regular intervals (about one every second).
- Reception: every time a UDP packet is received, the board checks the packet validity and updates its LEDs (the first 2 bits of the UDP payload are used to update the 2 LEDs). The packet payload is also stored and sent back in the transmission packets.

Important: The reference design requires a 24MHz clock. Make sure FPGAconf’s clock option is set correctly (see paragraph 7.3 - FX2 clock speed).

15.3 Troubleshooting – the PC has troubles receiving

- Make sure the hub/switch light blinks every second or so, to indicate that a packet is transmitted by the board
- Make sure you are not running a firewall on the PC

15.4 Troubleshooting – the PC has troubles sending

Make sure the PC has the correct ARP entries. You can use “ARP –a” to check the ARP entries. The physical address of the board should be listed. If it is not, you can add it manually using “ARP” or “netsh”. See below examples when the FPGA uses IP 192.168.0.44 and physical address 00-12-34-56-78-90

OS	Command
Windows XP	*ARP –s 192.168.0.44 00-12-34-56-78-90*
Windows Vista to Windows 8.1	netsh interface ipv4 add neighbors "Local Area Connection" 192.168.0.44 00-12-34-56-78-90
Windows 10	netsh interface ipv4 add neighbors Ethernet 192.168.0.44 00-12-34-56-78-90

- With Windows XP, an additional IP parameter can be used in the ARP -s command to specify the IP of the interface where your FPGA board is plugged. This may be required if your machine has multiple Ethernet ports that are enabled concurrently.
- With Windows Vista and above, make sure you open an administrator command-line prompt (otherwise the netsh command fails). An administrator command-line prompt can be easily opened by typing “cmd” into the start menu search box, and then pressing Shift+Ctrl+Enter.

15.5 UDP tester

Use FPGAconf Tools → UDP tester, or the UDP utility found at the bottom of this page, or the Ethernet UDP sample C code from chapter 37.

```
Waiting for packets (Press Ctrl-C to exit)...
Received 18 bytes from 192.168.0.44
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11
Received 18 bytes from 192.168.0.44
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11
Received 18 bytes from 192.168.0.44
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11
Received 18 bytes from 192.168.0.44
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11
```

16 I2C bus

16.1 I2C controller

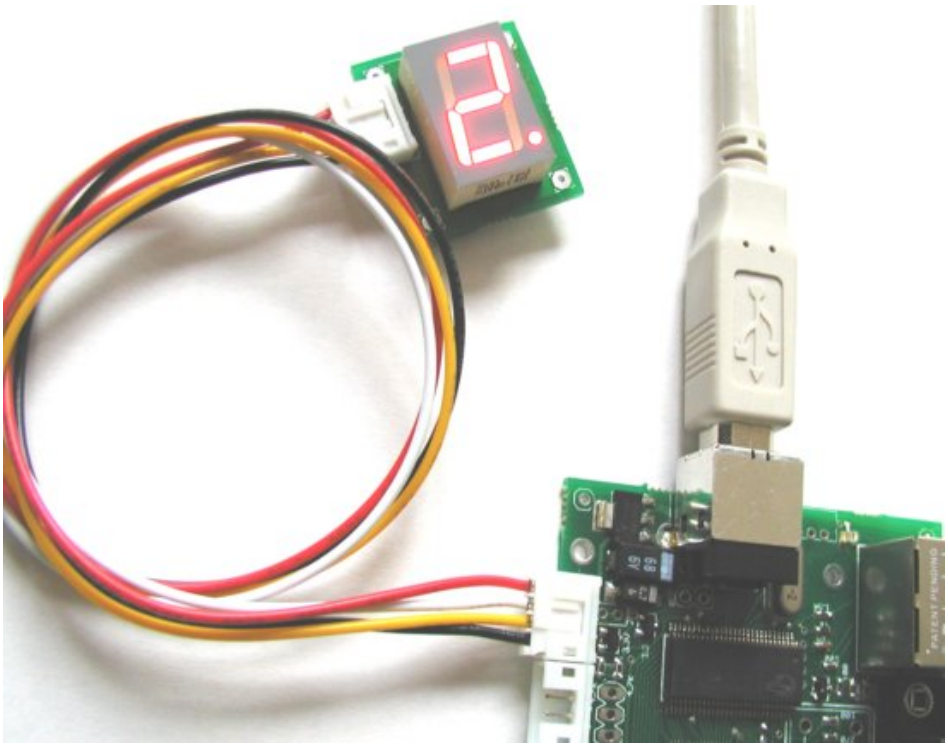
The board has an integrated I2C bus controller. The controller is a “hard macro” (the hardware is built into the FX2), as opposed to a “soft macro” I2C controller that can be programmed into the FPGA.

16.2 On board devices

The following devices share the I2C bus:

- I2C “hard macro” controller (USB chip).
- I2C connector (to connect to external devices through an I2C cable).
- I2C EEPROM (optional on some boards, but can be added if missing – see chapter 17).
- The FPGA (see paragraph 11.8 for details).
  Please note that to be able to communicate with the I2C bus, the FPGA needs to be configured with a “soft macro” I2C controller.

For example, here’s Xylo controlling an external 7-segments led display I2C board.



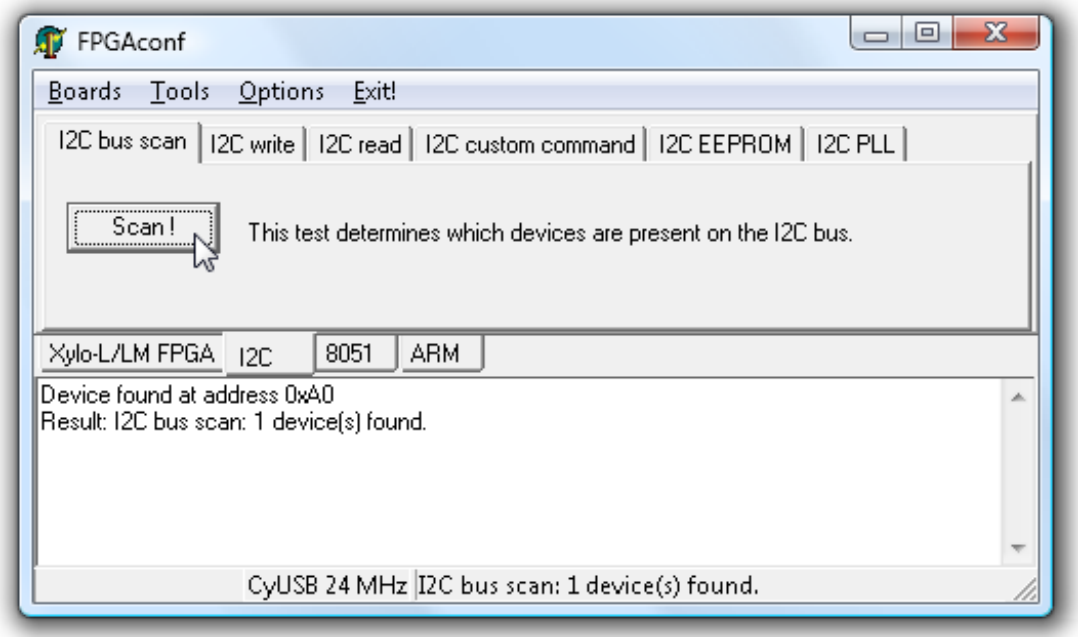
An I2C “control panel” is part of FPGAconf.
It can issue these commands:

- I2C Bus scan
- I2C Write
- I2C Read
- I2C Custom command (multiple reads/writes packet)
- I2C PLL

16.3 Bus scan

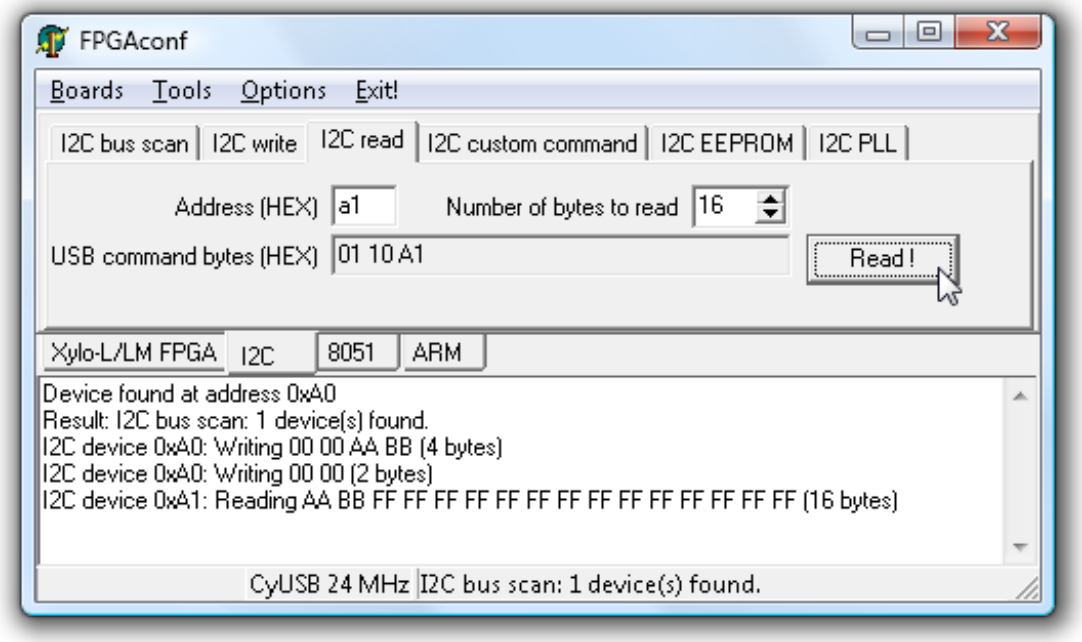
The bus scan is useful to discover what devices are present on the I2C bus.

Note that only devices that respond to I2C write commands are scanned. Most I2C devices respond to both reads and writes, so they can be discovered by the bus scan.



Note: some I2C devices (like the 24LC00 EEPROM) respond at multiple addresses.

16.4 Write & Read



16.5 Custom commands

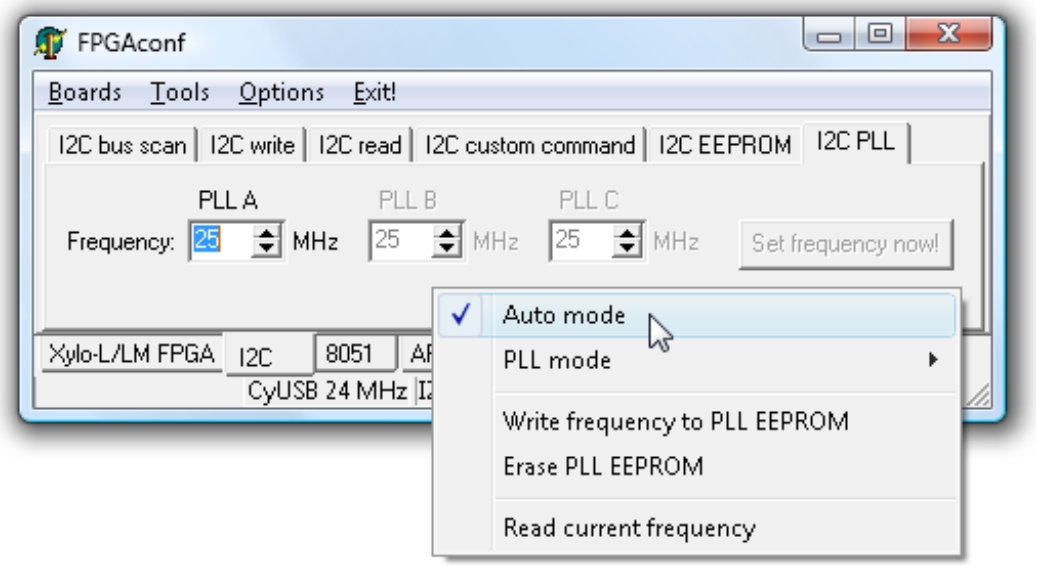
Custom commands allow issuing multiple reads & writes in the same command. It also uses I2C restart signals in between each read and write.

See the paragraph 20.3 for the Command packet required to create such commands.

16.6 PLLs

I2C PLLs can be connected to the I2C bus. The I2C control panel allows direct control of the PLLs (up to three PLLs can be controlled from the control panel).

Additional PLL commands are available from the PLL option menu. Right-click on the PLL window to get the option menu.



Check KNJN's I2C PLLs page for more info.

16.7 EEPROM

FPGAconf can program I2C EEPROMs easily. Go to the “EEPROM” tab, select a “.bin” file and click “Program!”.

See chapter 17 for EEPROM application details.

17 I2C EEPROM

17.1 EEPROM purposes

One characteristic of the USB-2 FX2 chip is that, at power-up, it accesses the I2C bus and looks for an EEPROM.

An EEPROM can be used for three purposes:

1. Change the USB ID
2. Change the USB ID, and load code into the FX2 8051 (chapter 19)
3. Store data

All boards have an EEPROM footprint but the EEPROM component may or may not be present. If it is not, you can either solder one, or use an adapter board (17.2).

The EEPROM on your board is as follows:

Board	EEPROM present by default?	EEPROM style	I2C address	Typical EEPROM used
Xylo-E	No	1-byte address	0xA0	24LC00 SOT23
Xylo-L / Xylo-LM / Dragon-E	Yes	1-byte address	0xA0	24LC00 or 24LC02 SO-8
Saxo / Saxo-L / Xylo / Xylo-EM	No	2-bytes address	0xA2	24LC32 or 24LC64 SO-8
Saxo-Q	Yes	2-bytes address	0xA2	24LC32 or 24LC64 SO-8

An EEPROM using the I2C addresses shown above allows changing the FX2 USB ID (chapter 18). Also some EEPROMs (like the 24LC00) respond to multiple addresses, so will be detected multiple times by an I2C Bus scan.

17.2 EEPROM adapter board

If soldering an EEPROM is not possible or desirable, an EEPROM adapter board can be used, like KNJN's item 1402.

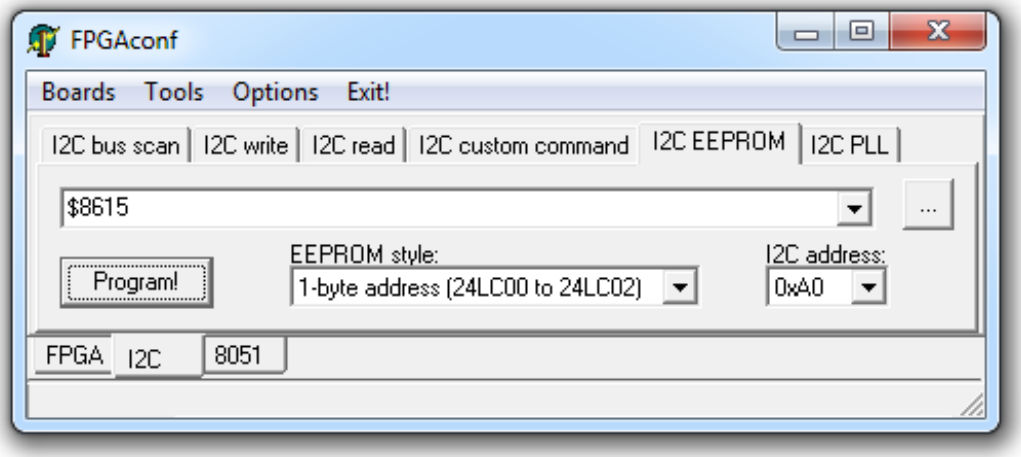
An adapter board allows choosing the EEPROM I2C address, but if you need to adjust the FX2 USB ID (chapter 18) or load code from the EEPROM to the 8051, the address needs to be set to the one shown above. For example, with a 24LC64, be sure to use an I2C address of 0xA2.

If you are only interested in storing data in the EEPROM, other I2C addresses could be used as well.

18 USB IDs

18.1 Custom IDs

If your KNJN FX2 board has an I2C EEPROM (chapter 17), you can change the board's USB ID easily: go to the I2C EEPROM window and enter a dollar sign followed by the desired ID (four hex digits). So for example, the string $8615 means ID 8615.



Then select the right EEPROM style and I2C address, and click “Program!”... Voila, the next time you plug-in the board, Windows will look for a driver with this new USB ID.

> **!** Be careful: if you program a custom ID and you don't have a matching driver, you can render your board inaccessible.

What happened? FPGAconf programmed eight specially crafted bytes into the EEPROM. Then at power-up, the FX2 reads the EEPROM to decide what USB ID to use.

For example, for ID 8615, the sequence C0 B4 04 15 86 00 00 04 is programmed (see FX2 documentation for the explanation of each byte).

18.2 Default IDs

By default (i.e. if the EEPROM is missing or empty), the USB IDs of the FX2 chip are:

- Vendor ID: 04B4 (Cypress Semiconductor)
- Device ID: 8613 (no EEPROM default ID for EZ-USB FX2)

If the EEPROM is present, it is either empty (the board uses use ID 8613) or factory programmed to use Device ID 8614.

As you can see, a USB ID consists of two numbers: a Vendor ID and a Device ID. But the Vendor ID of all the USB drivers provided by KNJN is always the same (04B4), while the Device ID is variable. So when we mention the USB ID in the rest of this document, we are in fact referring only to the USB Device ID.

18.3 hex2bix

Cypress provides a utility named “hex2bix” (in the FX2 development kit) that can be used to generate EEPROM contents. For example, use the following command to generate a file “id.bin” with Vendor ID 04B4 and Device ID 8615 :

```
hex2bix.exe -i -f 0xC0 -v 0x04B4 -p 0x8615 -o id.bin
```

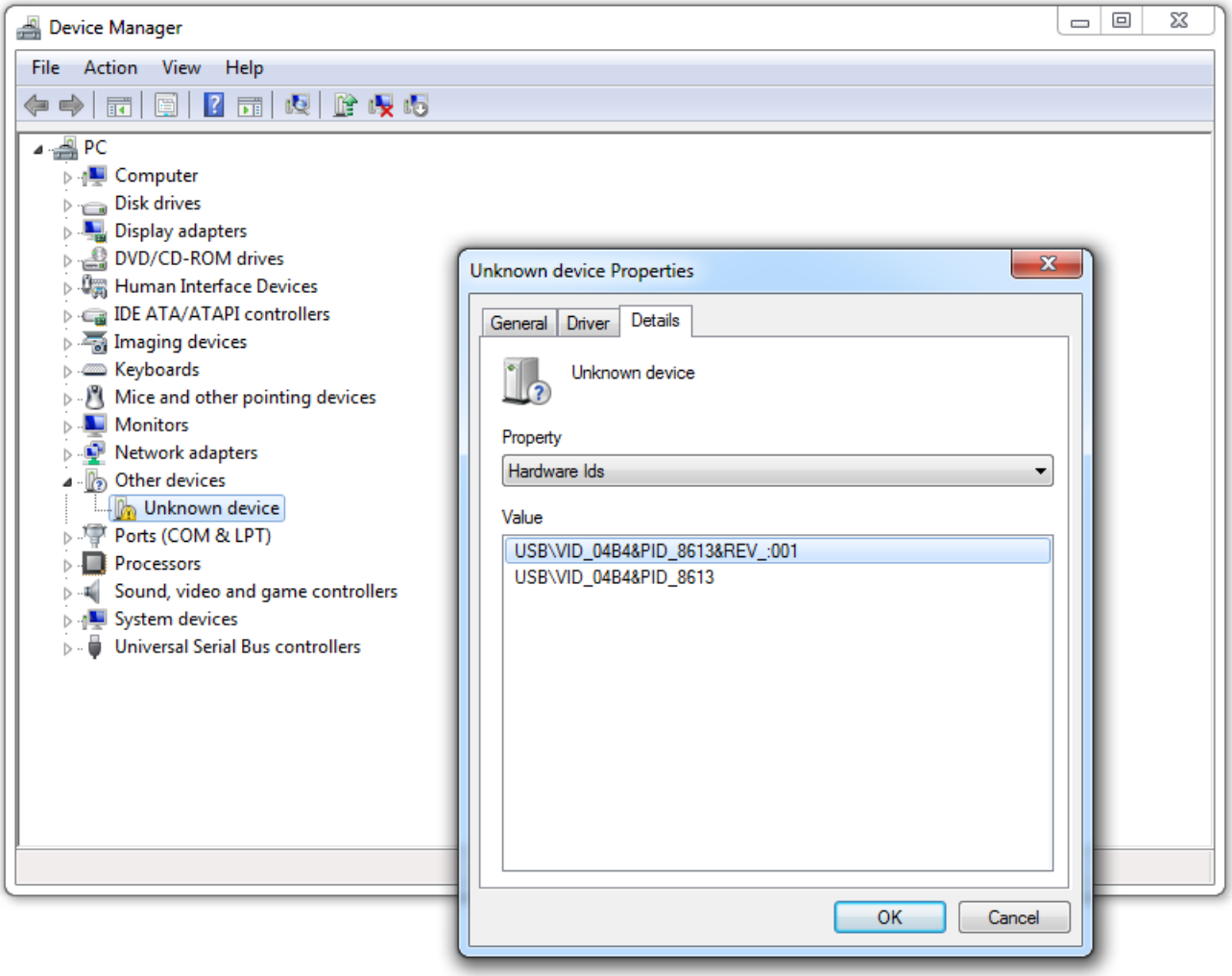
Then program “id.bin” into the EEPROM using FPGAconf: go to the “EEPROM” tab, select “id.bin” and click “Program!”.

18.4 USB-IF

Official USB IDs are maintained by the USB-IF. To get a new official ID, see http://www.usb.org/developers/usbfaq/

18.5 Checking the USB ID of a plugged board

The device manager shows the ID used by any plugged board. The board doesn't even need to be recognized (driver loaded), see for example an unknown board using the device ID 8613.



19 8051

19.1 8051 processor

The FX2 chip (CY7C68013) has an 8051 processor used for initialization and housekeeping.

Of most interest:

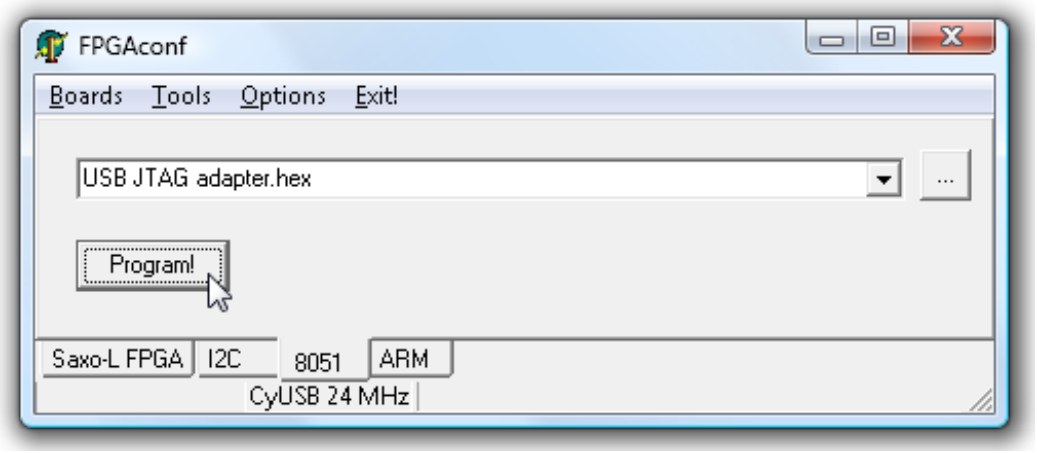
- The 8051 can force the FX2 to re-enumerate with different USB-IDs. See for example paragraph 23.4.
- The FX2 is connected to the FPGA, so the 8051 can be programmed to communicate with the FPGA.

19.2 8051 programming

The 8051 CPU is attached to an 8KB RAM (or 16KB RAM with the more recent CY7C68013A) where the 8051 code is stored.

The 8051 control panel is used to load code into the RAM.

1. Select a ".hex" file (this is generated by an 8051 assembler, see next paragraph)
2. Click on "Program!"



Under Linux, you can also use `fxload`:

1. Locate your FX2 board using
   ```
   lsusb
   ```
2. Locate the device. Depending on your Linux distribution, it is in
   ```
   /proc/bus/usb/00x/00x
   ```
   or
   ```
   /dev/bus/usb/00x/00x
   ```
3. Load the code in the FX2 using something like
   ```
   fxload -I firmware.hex -D /proc/bus/usb/00x/00x -t fx2
   ```
   Note that you may need administrator privileges.

19.3 HEX files

To create hex files for the FX2, you need an 8051 compiler/assembler, like:

- The Cypress FX2 development kit (see 33.2).
- The SDCC cross-C compiler.

19.4 Power-up

The I2C EEPROM (chapter 17) can be programmed with code that runs automatically in the 8051 at power-up. The first byte of the EEPROM is 0xC2 in this case.

As an example, try the file "Sample files - I2C EEPROM\EEPROM FX2 CPU 24MHz.bin". It makes the 8051 default to 24MHz instead of 12MHz. Check the FX2 development kit for more details on how to create ".bin" files.

20 I2C-over-USB protocol

20.1 Background

The I2C bus is controlled from the PC through the USB link.

While you issue I2C reads and writes with the I2C control panel, the PC sends and receives some special USB I2C command and response bytes. They are normally hidden, but FPGAconf can display them on demand (right-click in the I2C panel to see the “I2C LOG options”).

You can also write a user application that sends the commands directly. That allows writing custom I2C applications. A description of the protocol is given below, as well as in the startup kit (“Sample files - I2C control”).

20.2 Protocol

The protocol is packet based. To issue an I2C action, you need to:

1. Send an I2C command packet.
2. Read the I2C response packet.

Command packets are sent to USB-2 pipe 0, and response packets are read from USB-2 pipe 1.

See the “Sample files - I2C control” files for more information.

20.3 Command packet

A command is defined as a series of blocks, each block being a read or write request.

First byte: number of blocks

Then for each block, a few bytes:

- If I2C read: number of bytes to read, plus I2C address (with LSB=1 to indicate a read).
  On return, we get one status byte plus the data read.
- If I2C write: number of bytes to write (including the I2C address), followed by the I2C address (with LSB=0), plus the bytes to write.
  On return, we get one status byte.

20.4 Response packet

The length of the response usually equals the number of command blocks sent (you get one status byte for each block in the command), in addition to the data returned for each read block.

Each status byte is:

- bit 7: BERR (bus error), indicates a bus contention (another I2C master took control of the bus).
- bit 6: ACK, set if the device is present, cleared if the device didn’t respond, or asked to stop.
- bit 5..0: byte count (number of bytes written or read).

During processing, if any block results in an error (BERR or no ACK), the I2C controller stops at this particular block and doesn't process any additional block in the packet. So in this case, the length of the response may be lower than expected, and the last response status byte shows the error.

20.5 Restrictions

- A command can consist of as many read and write blocks as you wish, as long as the command packet total length doesn't exceed 64 bytes.
- A response packet total length can't exceed 64 bytes either. So if the command has reads, make sure you don't read too much because the response packet length must fit in 64 bytes.

20.6 I2C start/restart/stop

A command is initiated with an I2C “start” sequence. Then if more than one block is used, I2C “restart” sequences are used between blocks. Finally a single I2C “stop” sequence is sent at the end of the command.

21 JTAG FPGA configuration

21.1 Configuration files

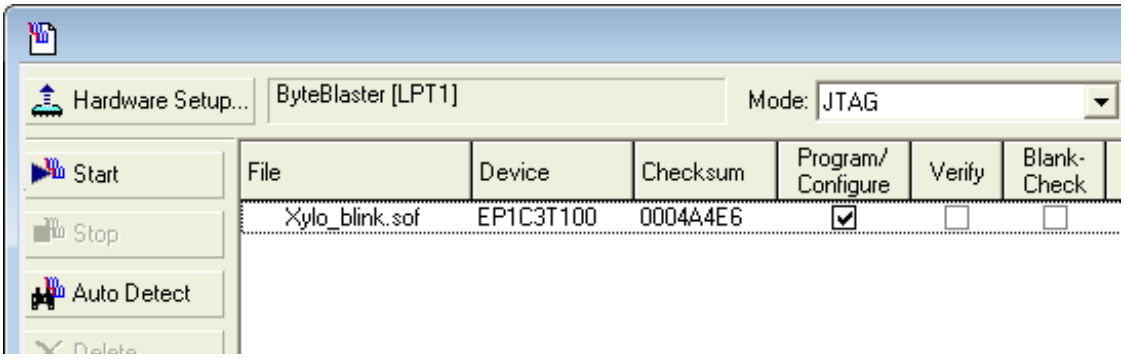
Depending on the FPGA configuration interface used, different file types are required.

FPGA	Interface	Preferred software	File to use
Altera	USB or boot-PROM	FPGAconf	".rbf" (RBF)
Altera	JTAG	Quartus-II	".sof" (SOF)
Xilinx	USB or boot-PROM	FPGAconf	".bit" (BIT)
Xilinx	JTAG	iMPACT	".bit" (BIT)

Refer to chapters 9 and 10 to learn how to generate the configuration files.

21.2 JTAG FPGA configuration with Altera's Quartus-II

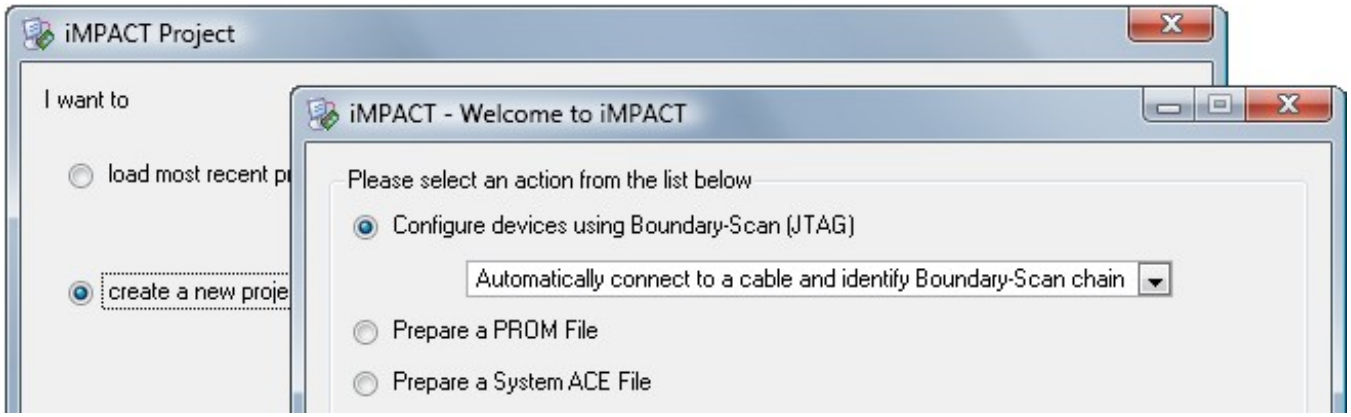
1. Make sure the JTAG connection is activated with a JTAG cable or JTAG-over-USB (paragraph 22.2).
2. In Quartus-II, open the "Programmer" window (in menu → Tools/Programmer).



3. Click on the "Hardware Setup" button and select the JTAG cable you're using (usually a ByteBlaster/-II or USB-Blaster).
4. Select "JTAG" in the "Mode" drop-down list.
5. Load the ".sof" file.
6. Check the "Program/Configure" check-box and click "Start".

21.3 JTAG FPGA configuration with Xilinx's ISE

1. Check chapter 38 to find the location of the FPGA JTAG header on your board. Connect a JTAG cable to the JTAG header (suitable JTAG cables include the USB or Parallel Xilinx cables, and KNJN JTAG cables).
2. Run the iMPACT software (which is part of Xilinx's ISE). Select "create a new project" and then automatic JTAG discovery. If your JTAG cable is connected correctly, iMPACT detects your JTAG chain automatically and allows you to configure the FPGA.



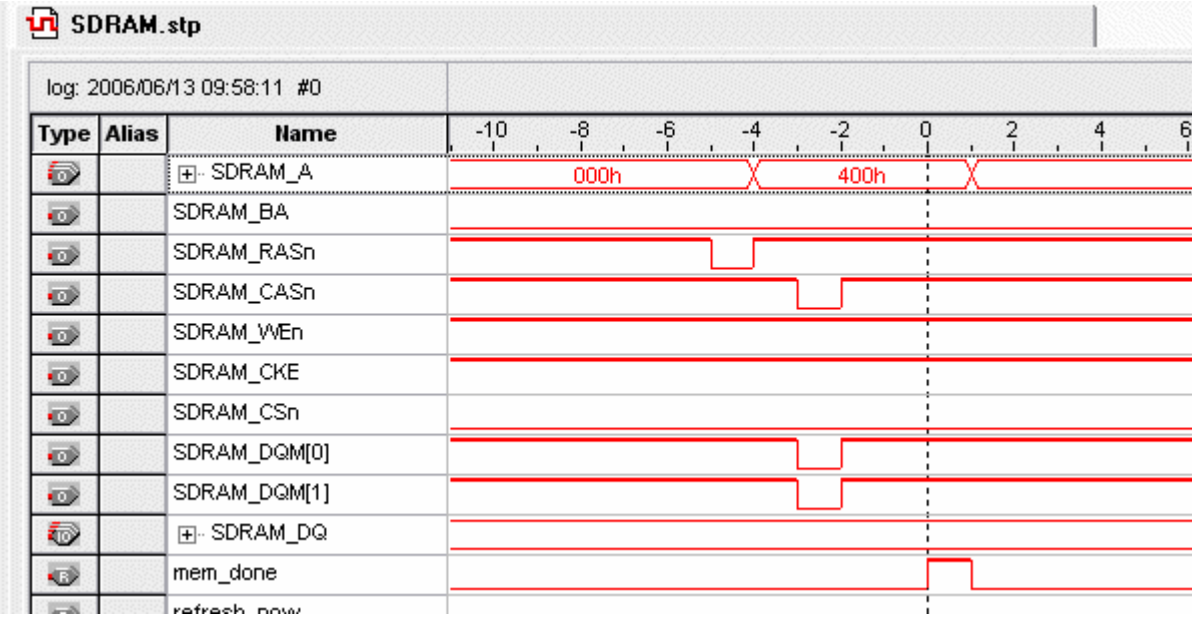
22 JTAG FPGA support in Altera's Quartus-II

22.1 JTAG

In addition to FPGA configuration (chapter 21), the FPGA JTAG port allows you to take advantage of Quartus-II's advanced JTAG features, like

- Boot-PROM programming (see chapter 24)
- SignalTap® II logic analyzer

SignalTap is particularly interesting as it brings a free logic analyzer to the FPGA.



A SignalTap window.

22.2 JTAG connection

There are two ways to use JTAG with Altera's FPGAs:

1. JTAG-over-USB (see chapter 23).
2. JTAG connector (with an external JTAG cable, see 22.3).

The advantage of using JTAG-over-USB is that you don't need a separate JTAG cable. The disadvantage is that it ties down the USB port. So if you want to take advantage of the high-speed FX2 USB-2 port, and still have JTAG access, you need to use the JTAG connector with an external JTAG cable.

22.3 JTAG connector

The JTAG connector is used with an external JTAG cable (like a ByteBlaster II or USB-Blaster).

For Altera boards, the JTAG connector is a 10 pin (2x5) shrouded connector (on Xylo-EM, use a right-angle connector to avoid mechanical troubles if a FlashyD board is mounted).

23 JTAG-over-USB for Altera FPGAs

23.1 Protocol emulation

With JTAG-over-USB, you can emulate an Altera JTAG cable like a USB-Blaster, or any other JTAG protocol you might have in mind.

The KNJN boards JTAG-over-USB support is based on the USB JTAG adapter open source project published at http://www.ixo.de/info/usb_jtag/

23.2 Board support

Saxo-L/-Q and Xylo-L/-LM/-EM are pre-wired for the USB JTAG open source project, so they can use JTAG-over-USB without modification (go to 23.4).

Saxo and Xylo require a preparation first (23.3).

23.3 Non pre-wired boards

With Saxo and Xylo, JTAG-over-USB can be used once four wires have been soldered on the board. These four wires are used to connect the FX2 (port D) to the JTAG connector.

	JTAG function	**JTAG connector pin**	**FX2 pin**
Wire 1	TCK	1	1 (PD.5)
Wire 2	TDO	3	2 (PD.6)
Wire 3	TMS	5	3 (PD.7)
Wire 4	TDI	9	56 (PD.4)

Notes:

- For the JTAG connector, check the Saxo/Xylo layout drawing (paragraph 38.1).
- The wires may be hard to solder directly on the FX2 (the FX2 package pin pitch is small). On the latest Saxo & Xylo board revisions (rev. F and later), the pin signals are also available on the bottom of the board, below the FX2, which makes the connections much easier to make.
- The FX2 WAKEUP pin (pin 51) may need to be high for the FX2 to re-enumerate reliably in JTAG-over-USB mode, but the boards default to low. A resistor jumper on the bottom of the boards allows the WAKEUP level to be changed.

23.4 Switch to JTAG-over-USB mode

Switching to JTAG-over-USB mode is just a matter of loading a file into the 8051. Compiled files are provided in the “Sample files – 8051” directory. Loading a file into the 8051 FX2 is simple, see paragraph 19.2. Once loaded, the PC beeps (re-enumerates the USB device) and you are ready to use Quartus-II built-in JTAG support.

Two versions of the JTAG-over-USB files are provided: full-speed (12mbps) or high-speed (480mbps). Try the high-speed version first, and switch to the other one in case you run into trouble.

Note that the first time the PC sees a device in JTAG-over-USB mode, it asks for the USB-Blaster driver. The driver can be found in your Quartus installation directory (usually something like C:\altera\xx\quartus\drivers\usb-blaster). More info is available on this page.

24 JTAG boot-PROM programming with Altera's Quartus-II

24.1 Create the JTAG Indirect Configuration File

The JTAG indirect mode allows programming the boot-PROM through JTAG.

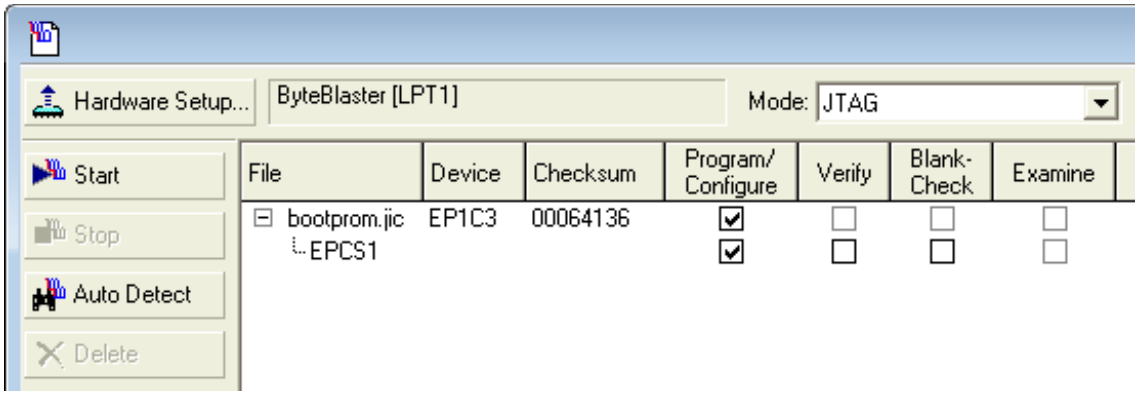
1. In Quartus-II, go to “File/Convert programming file”
2. In the “Output programming file” panel, select
   a) Programming file Type = “JTAG Indirect Configuration File (.jic)”
   b) Configuration device = “EPCS1“ (Saxo/Xylo) or “EPCS4” (Saxo-Q/Xylo-EM) for the boot-PROM 1 or 4Mbit
   c) File-name = a “.jic” file name of your choice
3. In “Input file to convert”, select
   a) Flash loader: Cyclone (Saxo/Xylo) or Cyclone-II EP2C5 (Saxo-Q/Xylo-EM)
   b) SOF Data: SOF file that you want to use for the boot-PROM
4. Click “OK”

This creates the “.jic” file.

24.2 Program the boot-PROM

Now we can use the “.jic” file to program the boot-PROM.

1. Load the “Programmer” window (in Tools/Programmer)
2. Open the hardware setup window (“Hardware Setup” button), select the JTAG cable you are using, and close the hardware setup window.
3. Select JTAG as “Mode”
4. Load the “.jic” file
5. Select configure (for both FPGA and EPCS devices)
6. Click the “Start” button
7. You can also verify and erase the EPCS if you want

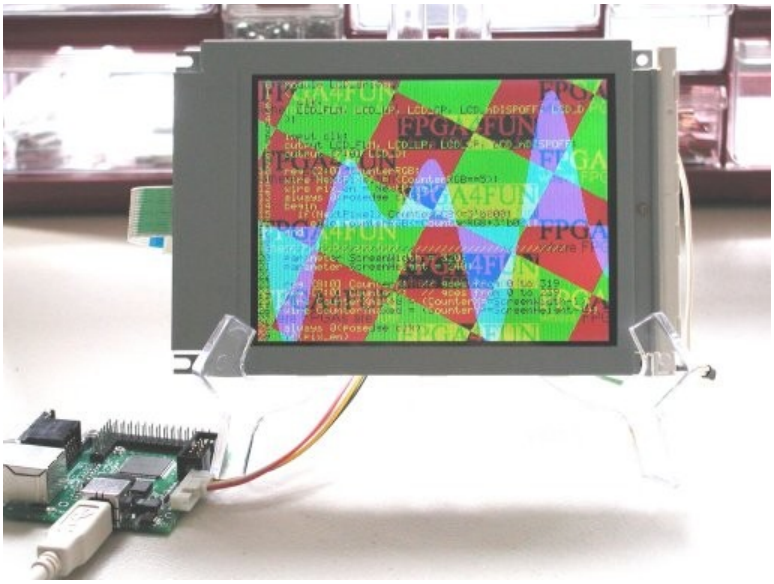


For more information, check http://www.altera.com/literature/an/an370.pdf

25 Graphic LCD

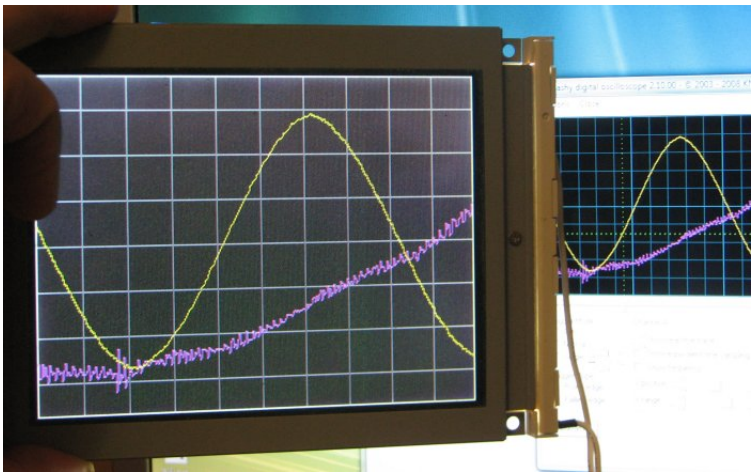
25.1 KNJN graphic LCDs (all boards)

KNJN graphic LCDs connect to the secondary connector of your board (see 11.6), using an adapter module (provided with the LCD) that makes the connection very easy.



These LCDs use a video-like interface and so are well adapted to be driven from an FPGA. See here for the list of available LCDs and here for an example of use.

Some LCDs and FPGA boards can be used with a Flashy board to create an oscilloscope with external display. See FlashyDemo in chapter 27.2 for more information.



25.2 LVDS LCD panel (Dragon-E only)

Dragon-E features a special LCD connector to connect an LVDS TFT 1024x600 color LCD panel with CCFL backlight. The LCD is available as a Dragon-E purchase option and it comes with the matching FFC cable.

The LVDS LCD requires both 3.3V and 5V powers – see chapter 32 for power options.

25.3 Other graphic LCDs

Other LCDs can be used. Please note that they may require a special connector, many IOs and non-standard voltages.

26 Flashy

26.1 What is Flashy

Flashy is a high-speed acquisition board. When used in conjunction with a KNJN FPGA board, the system becomes a digital oscilloscope.

The KNJN FPGA boards include two Flashy designs:

1. The FlashyMini design (27.1) comes with HDL and C source codes.
2. The Flashy demo design (chapter 27) comes with a GUI to use Flashy as an oscilloscope.

26.2 Flashy connection

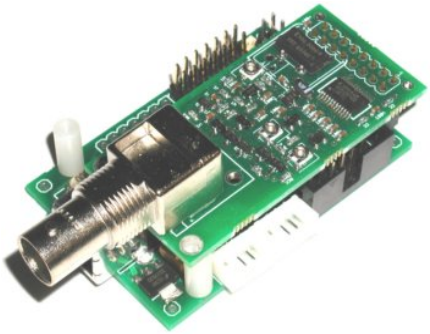
Before using Flashy, you have to connect it to your FPGA board.

Proceed as follow:

1. Make sure the FPGA board is NOT powered (unplug it from USB).
2. Plug Flashy/FlashyD into the FPGA board.

For example with Saxo:

Saxo



Saxo with Flashy

Saxo with FlashyD

Now you are ready to run FlashyMini or FlashyDemo (next chapter).

27 Flashy designs

Two Flashy designs are provided in the startup kit: FlashyMini and FlashyDemo.

27.1 FlashyMini

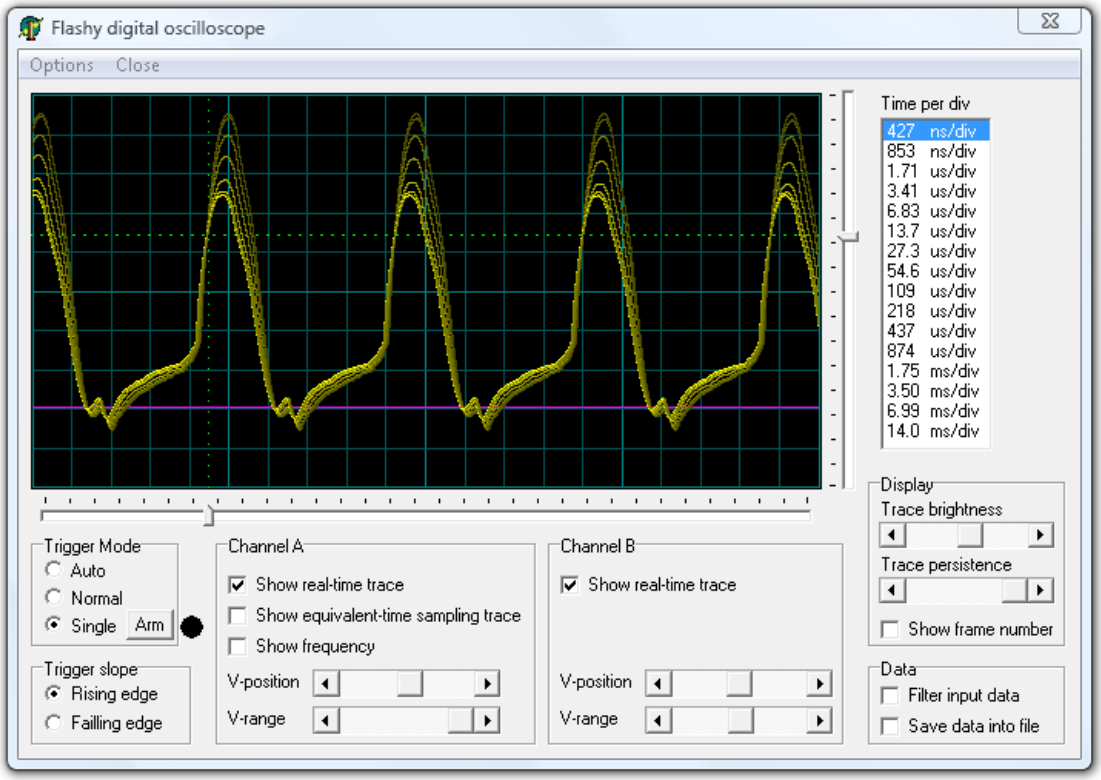
FlashyMini is a simple design that displays the acquired data in text form. FlashyMini is provided with source code and can be used as a skeleton to develop your own acquisition project.

27.2 FlashyDemo

FlashyDemo is a showcase of Flashy possibilities, implementing features found in regular oscilloscopes, like pre-triggering and equivalent-time-sampling.

To run FlashyDemo:

1. Open FPGAconf and configure the FPGA with the FlashyDemo bitfile (find it in the *FPGA Project – FlashyDemo* directory).
2. Go to Menu → Tools → Flashy Oscilloscope (or press CTRL-F) to bring-up the oscilloscope window.



FlashyDemo works equally well with the EzUSB and CyUSB drivers (chapter 33) and support Flashy (one channel), FlashyD (two channels) and Saxo-Q (four channels). FlashyDemo also works with some KNJN external LCDs. For more information, check the latest Flashy documentation available here.

FlashyDemo is not available in source code form.

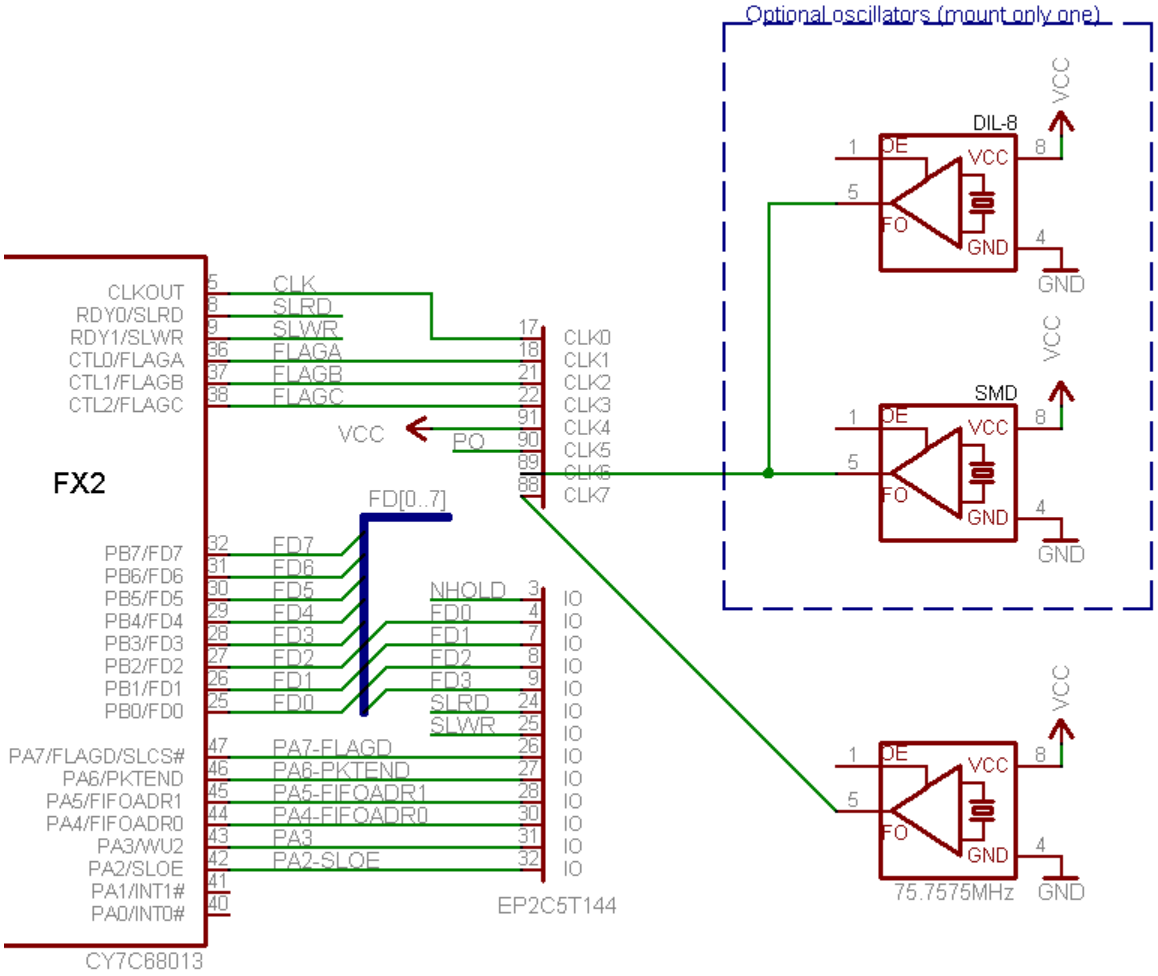
27.3 Flashy vs. Widy

The Widy boards feature higher-resolution ADCs than the Flashy boards. FlashyDemo has been ported to support Widy boards with a select number of FPGA boards as a debug tool (but the higher-resolution is not supported).

28 Saxo-Q

28.1 FX2 and clock connections

Saxo-Q has the regular FX2 high-speed USB-2 connection (see “USB/FX2 interface“ for more info) and specific oscillator options.

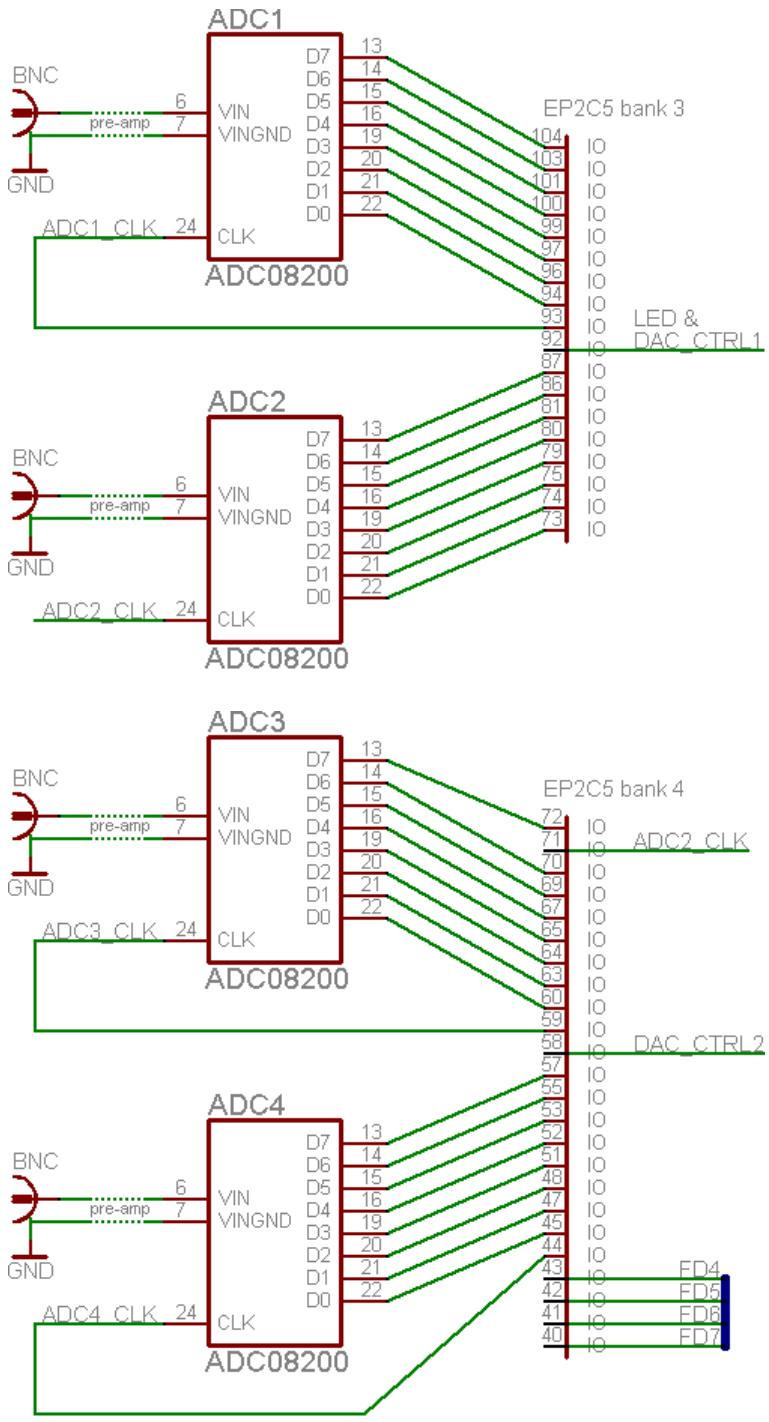


Since Saxo-Q has four 8-bit 200MSPS ADC (see 28.2), it needs to generate up to four sampling frequencies. A 75.7575MHz crystal oscillator is provided on CLK7 as a good candidate to be the base for the ADC sampling frequencies. That's because this odd frequency is guaranteed to be asynchronous to any periodic sampled signals (a condition required in equivalent time sampling mode). Typically Saxo-Q uses one of the FPGA PLL to generate a single sampling frequency fed to the four ADCs at a multiple of 75.7575MHz (FlashyDemo uses a PLL x 2 to get 151.51MHz).

Another oscillator can also be mounted on the bottom of the board (optional oscillator above, either in a DIL-8 or SMD5x7 package) if a different frequency base is desired.

28.2 ADC inputs

Saxo-Q incorporates the equivalent of four Flashy boards (based on four NS ADC08200), with V-pos/V-range controls on each channel, and PO (period output) on channel one. The ADC clocks are under the FPGA control (typically generated from the FPGA PLL).

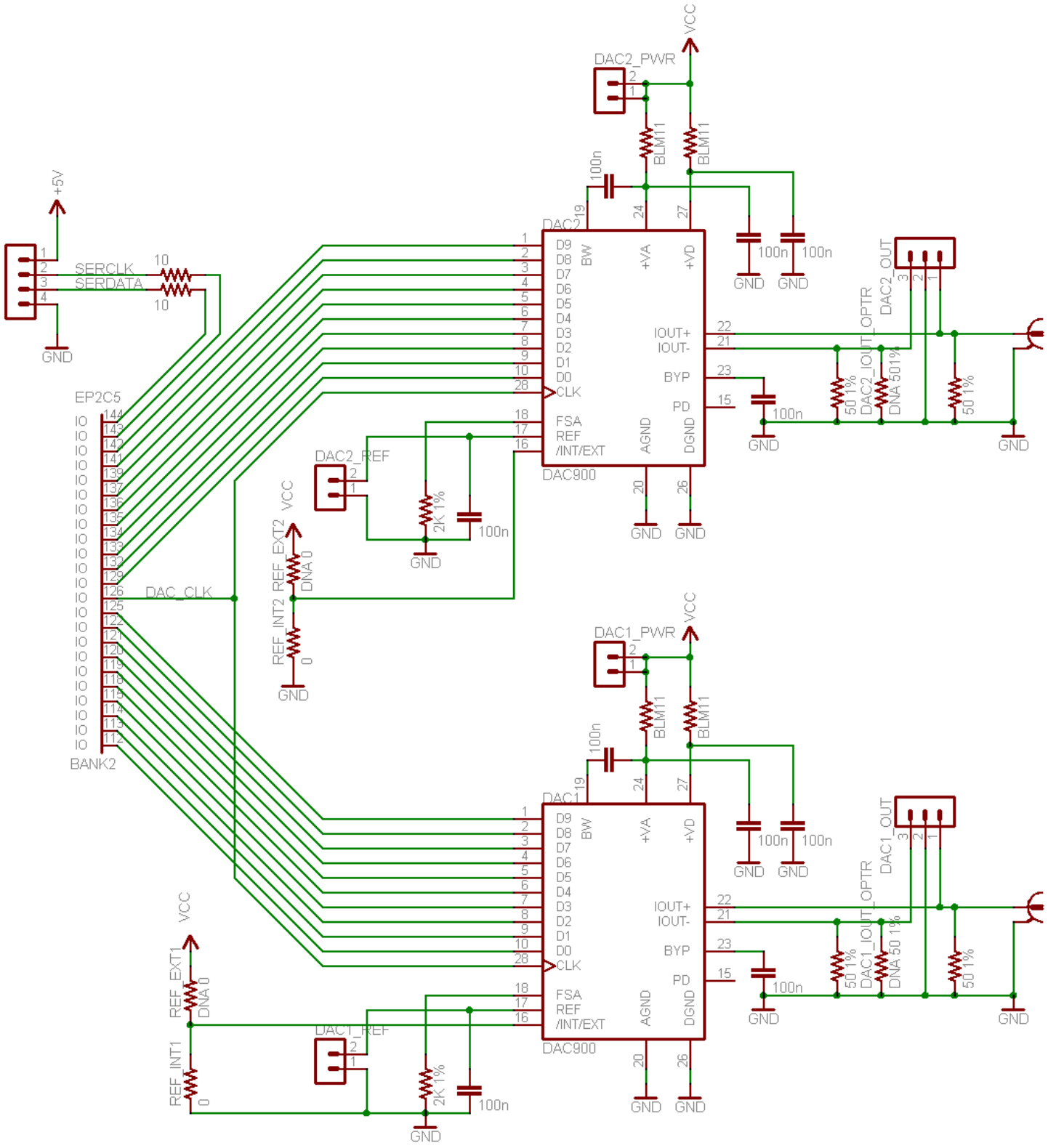


The two DAC_CTRL signals control the V-pos/V-range on each pair of channels through low-speed DACs (see the Flashy documentation for details).

28.3 DAC outputs

Saxo-Q has two 10-bit 165MSPS high-speed DAC outputs based on two TI DAC900. The DACs are powered by 3.3V, with an option to raise the DAC900 “+VA” analog power to 5V. The outputs are differentials 50 ohms, with the option to use output transformers on DAC1/2_OUT connectors (see 38.3 for the locations).

The FPGA bank 2 is used to drive the two DACs. A serial 2-wire output (named the Secondary connector) is also driven from the same FPGA bank, which can be used to connect to a TXDI/MAX232 board or an external LCD for example.



29 Serial interfaces

29.1 RS-232 with the FPGA

Your FPGA board doesn't have a native RS-232 port, but its secondary connector can be used in conjunction with a TXDI/MAX232 board to create an RS-232 port.

Examples:

- HDL: see the startupkit “Projects\SerialRxTx”
- C code: see chapter 36 - RS-232 Win32 send & receive sample C code
- fpga4fun's RS-232 project

29.2 Serial interfaces with the ARM (Saxo-L & Xylo-L/-LM)

Check the KNJN ARM board documentation (see 1.3).

30 Text LCD

Your board can easily drive a text LCD based on the HD44780 controller. With Saxo/-L & Xylo, you can attach a text LCD directly (see 30.1). With other boards, you can wire a text LCD manually (see 30.2).

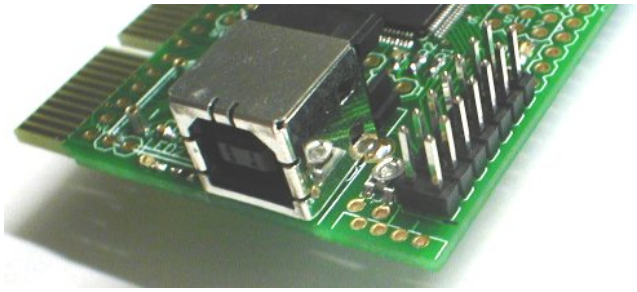
30.1 Text LCD connector (Saxo/-L & Xylo)

The Saxo/-L & Xylo boards accommodate text LCD modules on CONIO2 (find it on your board layout - chapter 38).

A small potentiometer is available next to CONIO2 to adjust the LCD contrast (LCD_VO pin of your text LCD). Note that the LCD you plan to use needs to be able to display a picture with a positive voltage on LCD_VO (some LCDs require a negative voltage on LCD_VO, and so can't be used without a separate negative supply). Suitable LCDs (with positive LCD_VO) are available here.

Now, there are two ways to connect a text LCD on CONIO2: either directly or through a flat cable.

- Direct connection (Saxo/-L & Xylo)
  It is possible to connect the LCD directly on the CONIO2 connector. Typically, you use a male connector (facing up) on CONIO2, and a female connector (facing down) on the LCD.
  For example, see here a male connector, facing up (Dragon board on the picture).



  On Xylo, the VGA connector may get in the way of the LCD. If so, use a flat cable (next paragraph).

- Flat cable (Saxo/-L & Xylo)
  It is also possible to use a flat cable to connect the LCD. You use a male connector on CONIO2 and the LCD. A 2x7 flat cable (female connectors) mates them easily.

  But care needs to be taken not to connect the LCD with the pins reversed. That's because there are 2 ways to solder the connector on the LCD module, facing down or facing up.

  Use the same on CONIO2, facing down or facing up - or the connections through the flat cable will be inverted. See more details here (the page uses a Dragon board, but the principle applies to any board).

30.2 Manual wiring (Xylo-L/-LM/-EM)

If your board doesn't have a native text LCD connector, you can wire it manually to the LCD. That typically requires soldering 13 wires to the board, plus an external potentiometer or supply on LCD_VO (pin 3 of the LCD).

For more information on text LCDs, see here.

30.3 LCD code example

See paragraph 14.2

31 Board power (all boards but Dragon-E)

31.1 USB power

Your FPGA board is USB-powered. It is recommended to connect the board directly to a PC's USB port (not through a USB hub).

On some boards. a three pin power header gives access to the GND, +3.3V and +5V signals. The +3.3V and +5V can typically provide 300mA to power outside boards. The +5V comes directly from the USB connector. The +3.3V is generated from the +5V.

31.2 Current limit

The board should not draw a total of more than 500mA from the PC. Typically a PC (or USB hub) can limit the current above that.

The board itself won't limit the current consumption. If you load it (by connecting peripherals), make sure the total current drawn from the PC is reasonable. Otherwise, power the board externally.

31.3 External power

You can power the board by providing +3.3V or +5V directly on the power header (if present) or other connector.

- If you power the board using 5V (recommended), the 3.3V is also present (generated by the on-board voltage regulator).
- If you power the board using 3.3V, the +5V should not be used.

You may want to cut a trace before powering the board externally, see the next paragraph 31.4

31.4 Current measurement

On some boards, a little PCB bridge (copper trace) is present next or below the USB connector. You can cut the trace to disconnect the +5V USB power from the +5V board power rail. That can be useful if you want to power the board externally while USB is still in use, or want to measure the board current consumption.

32 Board power (Dragon-E only)

32.1 Power rails

Dragon-E has three main power rails: +3.3V, +5V and +12V. Each power rail has an orange status LED (top left of the board).

- The +3.3V is required while the +5V and +12V are optional.
- If +5V is provided, +3.3V is automatically created on board using a linear regulator.
- The +12V isn't used but can power external peripherals.

32.2 Power sources

Dragon-E has multiple possible sources of power.

- PCI Express: +3.3V and +12V.
- USB: +5V.
- Optional power connectors (compatible with common PC cases / ATX power supplies):
  - SATA: +3.3V, +5V and +12V.
  - Disk drive (AKA four pins peripheral): +5V and +12V.
  - 3½” floppy drive: +5V and +12V.

A jumper named “PCI-E power” is present on the board to decouple the 3.3V power from the PCI Express power. This jumper should be present by default, but can be removed if you use both the PCI Express and USB connector, each from a different PC (it removes the possibility to power the PCI Express PC from the USB PC).

32.3 Possible board uses

Dragon-E can be used in four combinations:

1. Without PCI Express nor USB port. The board is powered though another connector (like SATA power) and the FPGA is configured from the FPGA boot-PROM or from JTAG.
2. With USB (no PCI Express). The board is powered through the USB connector (+5V) and the +3.3V rail is created from the +5V rail. The FPGA is configured through USB, the boot-PROM, or JTAG. The +12V rail is not powered.
3. With PCI Express (no USB). The board is powered though the PCI Express connector (+3.3V and +12V) and the FPGA is configured from the FPGA boot-PROM, or from JTAG. The +5V rail is not powered.
4. With both PCI Express and USB. The board's +3.3V and +12V rails are powered through PCI Express and the +5V rail is powered through USB. The FPGA is configured through USB, the boot-PROM, or JTAG.
   The USB and PCI Express PCs can be the same or different PCs (so for example you could create bitfiles in one PC, load them though the USB and use them in the PCI Express PC). In the later case, it is recommended to remove the “PCI-E power” jumper to avoid the possibility of powering the PCI Express PC motherboard through the USB.

In all configurations, the optional power connectors can provide additional power (not required unless you connect power hungry peripherals to the board, like an LCD).

33 FX2 USB driver

33.1 USB drivers

Different USB drivers are usable with your KNJN FPGA board.

- EzUSB
- CyUSB

CyUSB is usually a better choice, especially if you plan to develop a USB application in C++.
Cypress provides C++/.NET libraries that simplify programming.

33.2 EzUSB (32bit Windows only)

EzUSB is a legacy driver that supports blocking USB transactions only. It is provided with a development kit called the Cypress FX2 development kit

- Go to CY3681 page and get the file “EZ-USB_devtools_version_261700.zip”
- or CY3684 page and get the file “SETUP_FX2LP_DVK_1004.exe”

33.3 CyUSB (32bit and 64bit Windows)

CyUSB supports blocking and non-blocking USB transactions. It is controlled from C or C++/.NET libraries. The CyUSB C++ development kit is available from this Cypress SuiteUSB.NET page.

33.4 CyUSB driver signature

Some CyUSB drivers are not signed. They work fine with 32bit Windows but are a tricky to use with 64bit Windows. For example, for Windows 7 x64:

1. When Windows starts to boot, press F8
2. Choose "Disable Driver Signature Enforcement".
3. Once Windows is booted, the unsigned 64bit driver should work.

33.5 CyUSB USB ID and GUID

CyUSB drivers support different USB IDs. KNJN FX2 boards with an EEPROM may be factory programmed to use USB ID 8614 (can be changed, see chapter 18), while boards without EEPROM default to USB ID 8613.

So when choosing a CyUSB driver, you have to pick one that supports the USB ID of your board. The driver then selects the GUID used to access it. Two different GUIDs are available:

KNJN GUID	{0EFA2C93-0C7B-454F-9403-D638F6C37E65}
Cypress GUID	{AE18AA60-7F6A-11d4-97DD-00010229B959}

33.6 Multiple instances

It is possible to connect multiple FX2 boards to the same PC, and to use multiple instances of the USB driver to control them. The best method consists in programming a different USB ID for each board (see chapter 18) and to use the CyUSB driver. Then a different USB ID/GUID pair can be used for each board.

You can force FPGAconf to use a specific GUID with the cyusb_guid parameter. For example:

```
FPGAconf.exe cyusb_guid={AE18AA60-7F6A-11d4-97DD-00010229B959}
```

If the same USB ID is used (because there is no EEPROM connected to the FX2 for example), it is still possible to use multiple instances of the same driver, but the order in which you plug the boards (or the PC USB locations used) may affect the numbering of the boards. If that's not a problem and you are using CyUSB, a call to `SetupDiEnumDeviceInterfaces` allows to specify a `deviceNumber` parameter (more info is available in the CyUSB documentation). With EzUSB, it is even simpler as the device number is specified in the `CreateFile` device name string.

34 Changing the USB driver

You can use EzUSB and CyUSB on the same PC.

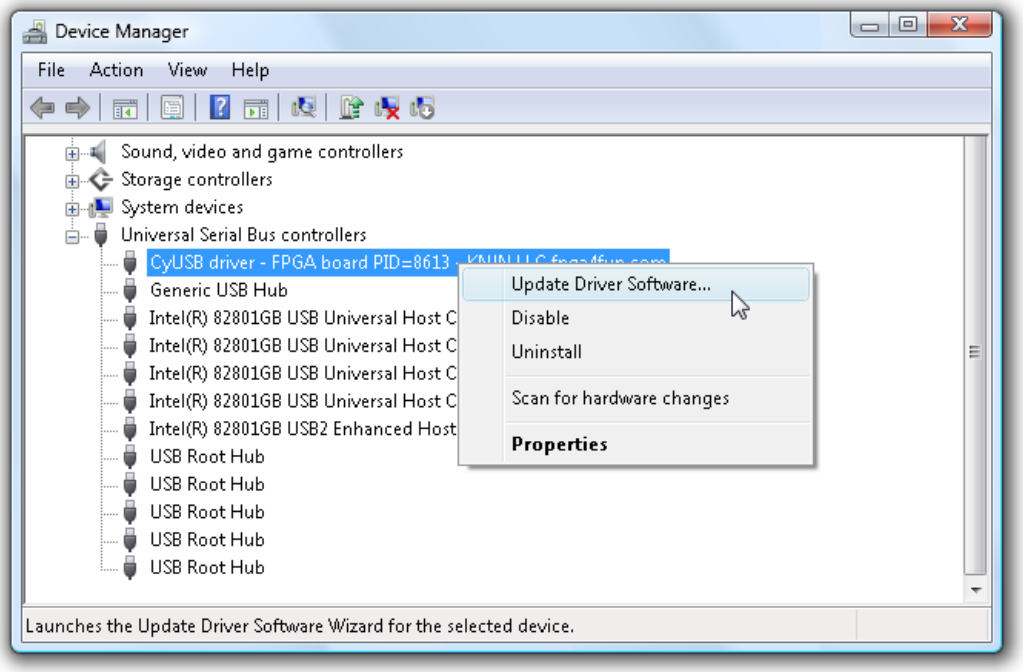
Remember to select the driver in FPGAconf's menu → Options → USB driver.

34.1 USB port

Windows allows selecting a different USB driver for each USB port on your PC. So you can have one USB port with EzUSB and another with CyUSB.

34.2 Driver swap using the Windows Device Manager

If you want to swap between EzUSB and CyUSB on a particular port, choose “Update Driver...” in the device manager.



If Windows doesn't let you change the driver, either use devcon (see 34.3) or change the board USB ID and select a different USB driver for this ID (chapter 18).

34.3 Driver swap using devcon

Devcon is a utility from Microsoft used as an alternative to the device manager.

To try it, first open an administrator command prompt.
Then locate all the FX2 boards connected to your PC:

```
devcon find *USB\VID_04B4*
```

To update a driver, try this (using PID 8613 in this example):

```
devcon update driverincludingpath.inf "USB\Vid_04b4&Pid_8613&Rev_0000"
```

Follow this link for more information on getting and using devcon:
http://support.microsoft.com/kb/311272

34.4 Removing a driver from DriverStore repository

Windows keeps a copy of the drivers it installed, so that it doesn't have to ask the user if a driver needs to be installed again.

If for any reason you want to erase one driver from Windows repository, erase it from the device manager, or use pnputil (Vista and above). More info on http://technet.microsoft.com/en-us/library/cc730875.aspx

35 Other OSes support

Although KNJN boards are only officially supported on Windows, here are tips to use other OSes like Linux and Mac OS.

35.1 JTAG support

JTAG is your best bet to use the board with any OS. You can either use a JTAG cable, or JTAG-over-USB with Linux and Mac OS.

- A JTAG cable works with Xilinx and Altera FPGAs.
- JTAG-over-USB works with Altera FPGAs. The “USB JTAG adapter.Full-speed.hex” files have been reported to give the best results on Linux and Mac OS.

35.2 Windows emulators

Another solution is to use a Windows emulator. VMware within Linux, and Parallels virtual machine under MacBook PRO / Leopard have been reported to support FPGAconf and Quartus for example.

The free VMware player is available at http://www.vmware.com/products/player/

36 RS-232 Win32 send & receive sample C code

Refer to chapter 29 for information on the RS-232 port.
Also http://www.robbayer.com/files/serial-win.pdf for more information on Windows serial port programming.

```
#include <windows.h>
HANDLE hCom;

void OpenCom()
{
    DCB dcb;
    COMMTIMEOUTS ct;

    hCom = CreateFile("COM1:", GENERIC_READ | GENERIC_WRITE, 0, NULL,
                OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
    if(hCom==INVALID_HANDLE_VALUE) exit(1);
    if(!SetupComm(hCom, 4096, 4096)) exit(1);

    if(!GetCommState(hCom, &dcb)) exit(1);
    dcb.BaudRate = 115200;
    ((DWORD*)(&dcb))[2] = 0x1001;  // set port properties for TXDI + no flow-control
    dcb.ByteSize = 8;
    dcb.Parity = NOPARITY;
    dcb.StopBits = 2;
    if(!SetCommState(hCom, &dcb)) exit(1);

    // set the timeouts to 0
    ct.ReadIntervalTimeout = MAXDWORD;
    ct.ReadTotalTimeoutMultiplier = 0;
    ct.ReadTotalTimeoutConstant = 0;
    ct.WriteTotalTimeoutMultiplier = 0;
    ct.WriteTotalTimeoutConstant = 0;
    if(!SetCommTimeouts(hCom, &ct)) exit(1);
}

void CloseCom()
{
    CloseHandle(hCom);
}

DWORD WriteCom(char* buf, int len)
{
    DWORD nSend;
    if(!WriteFile(hCom, buf, len, &nSend, NULL)) exit(1);

    return nSend;
}

void WriteComChar(char b)
{
    WriteCom(&b, 1);
}

int ReadCom(char *buf, int len)
{
    DWORD nRec;
    if(!ReadFile(hCom, buf, len, &nRec, NULL)) exit(1);

    return (int)nRec;
}

char ReadComChar()
{
    DWORD nRec;
    char c;
    if(!ReadFile(hCom, &c, 1, &nRec, NULL)) exit(1);

    return nRec ? c : 0;
}
```

37 Ethernet UDP sample C code

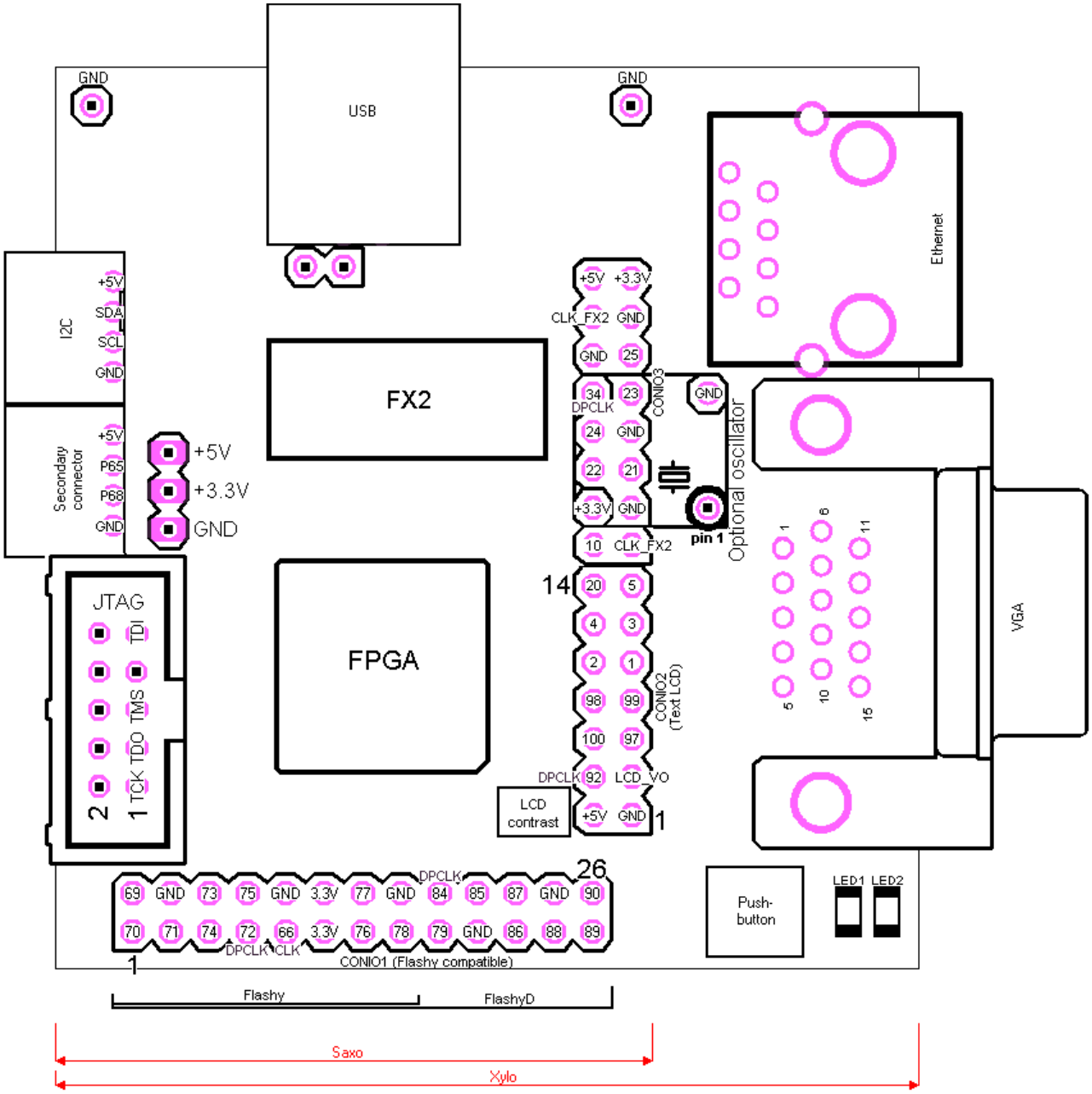
Here's a simple example of UDP transmit.
Check also the startup kit for more examples.

```
#include <winsock.h>
#include <stdio.h>
#pragma comment (lib, "wsock32.lib")

char* szUDPAddress = "192.168.0.44";
u_short UDPPort = 1024;
char szMessage[1500] = "Whoa!";

int main(int argc, int **argv)
{
 WSADATA wsda;
 int ret;

 SOCKET s;
 SOCKADDR_IN addr;

 WSAStartup(MAKEWORD(1,1), &wsda);
 s = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
 if(s==SOCKET_ERROR) { printf("Failed to create socket, error %d\n", WSAGetLastError()); exit(1); }

 addr.sin_family = AF_INET;
 addr.sin_port = htons(UDPPort);
 addr.sin_addr.s_addr = inet_addr(szUDPAddress);

 ret = sendto(s, szMessage, strlen(szMessage), 0, (struct sockaddr *) &addr, sizeof(addr));
 if(ret == SOCKET_ERROR) { printf("Failed to send, error %d\n", WSAGetLastError()); exit(1); }

 closesocket(s);
 WSACleanup();
 return 0;
}
```

38 Board layouts and pin assignments

The following drawings show the board layout for the FPGA and ARM IO pin assignments (numbers drawn on the header pins).

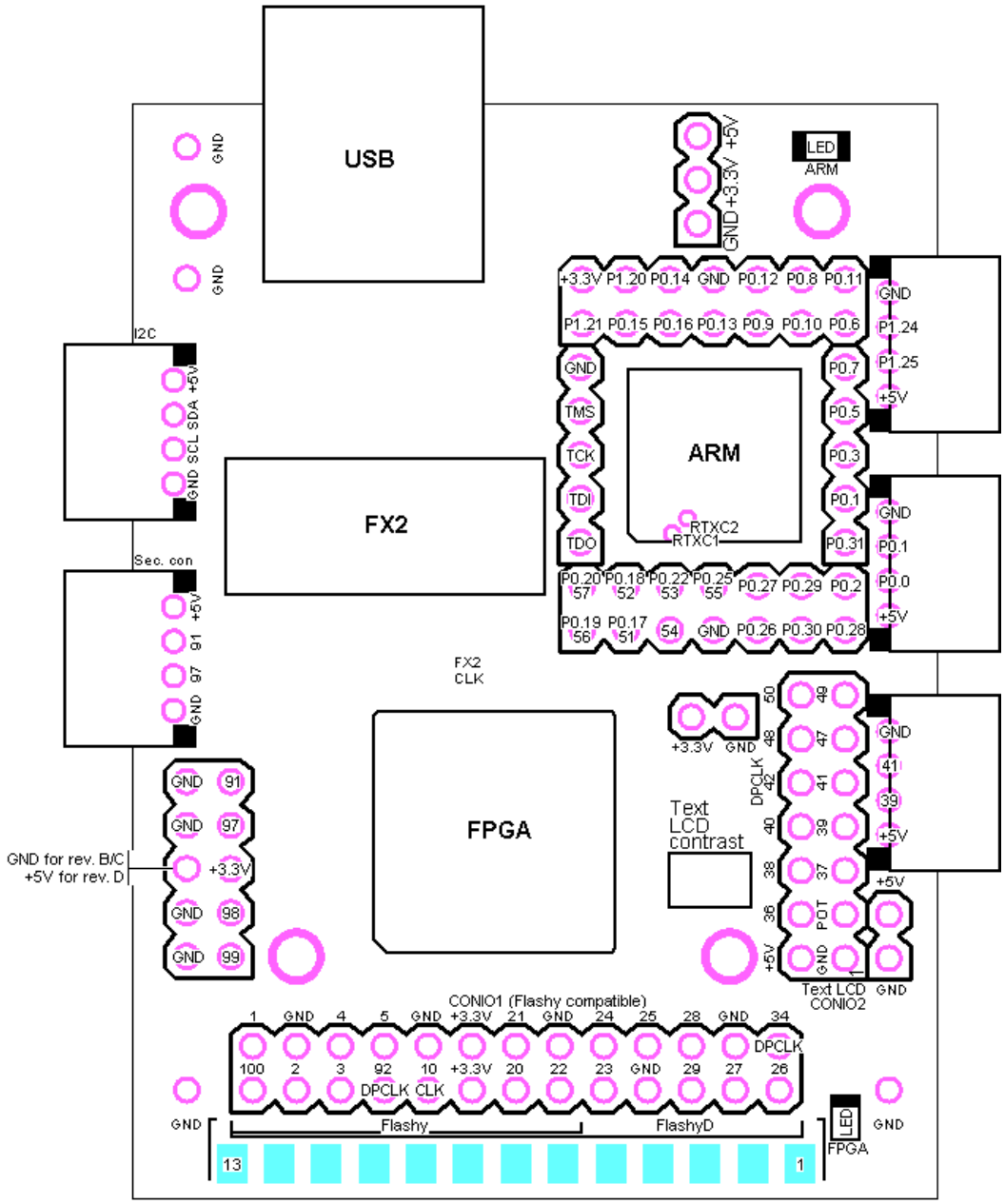
38.1 Saxo and Xylo



Notes:

- Saxo is just a small version of Xylo.
- On Xylo, a DIL-8 oscillator can be added on the board (marked “optional oscillator”).

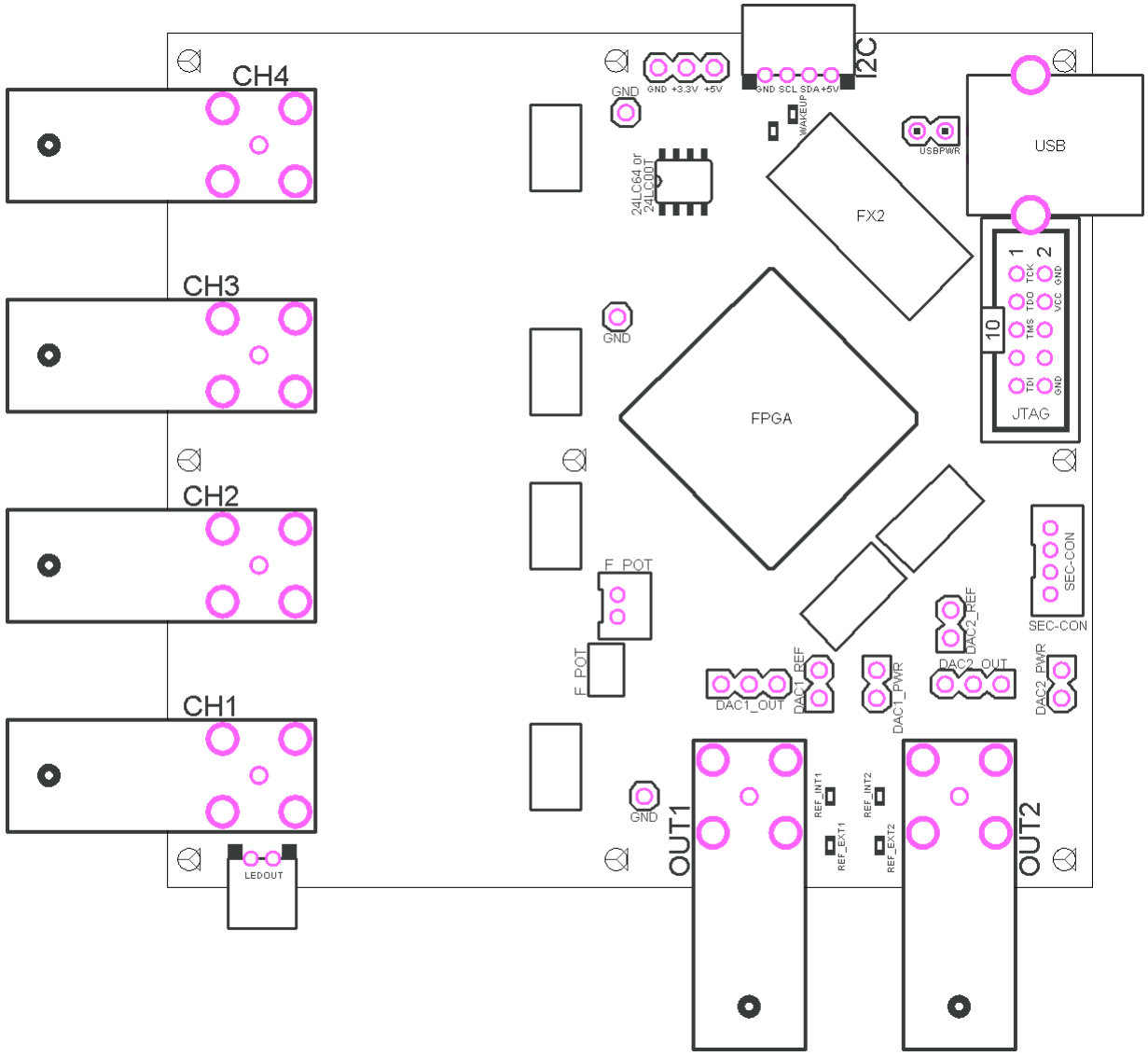
38.2 Saxo-L



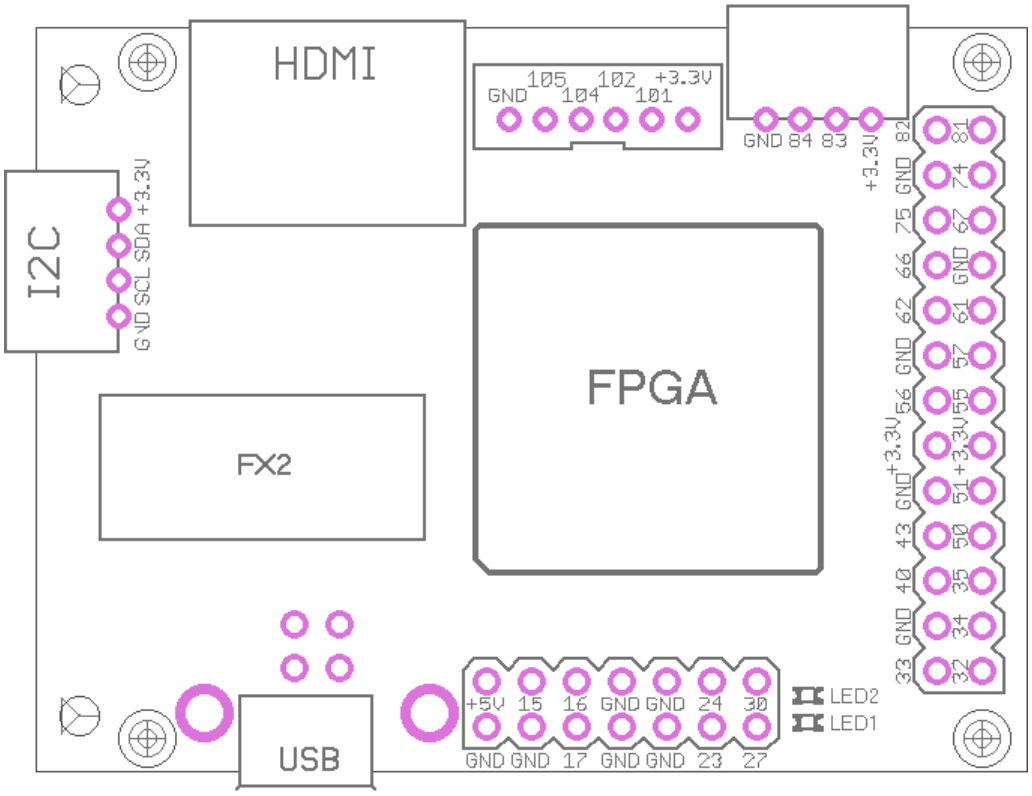
Notes:

- Some pins have two numbers (connected to both the FPGA and the ARM).
- The header pins on the left of the ARM are the ARM JTAG pins.
- The ARM pins RTXC1 and RTXC2 are accessible below the ARM (bottom of the board).
- The connector below CONIO1 (light blue on the drawing) can be soldered to a DB-25 female (solder cups type, KNJN item#1004), so that the board can easily be used as a CNC stepper motor controller with DB-25 output.

38.3 Saxo-Q


CH4
CH3
CH2
CH1
LEDOUT
GND
GND +3.3V +5V
GND SCL SDA +5V
I2C
WAKEUP
USBPWR
USB
24LC64 or 24LC01
FX2
FPGA
JTAG
TDI
TMS
TDO
TCK
VCC
GND
F_POT
DAC1_REF
DAC1_OUT
DAC1_PWR
DAC2_REF
DAC2_OUT
DAC2_PWR
SEC-CON
REF_INT1
REF_EXT1
REF_INT2
REF_EXT2
OUT1
OUT2

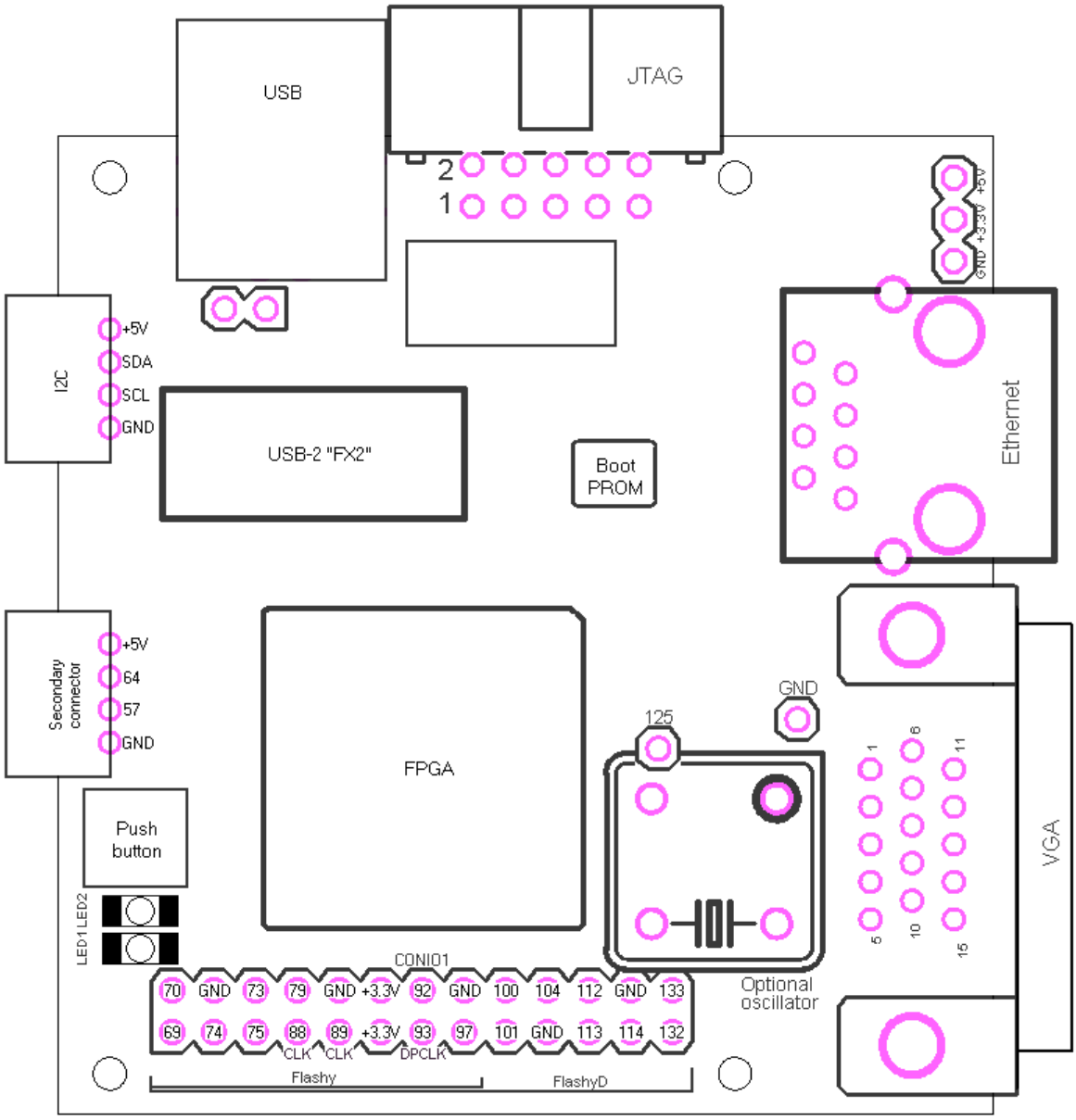
38.4 Xylo-E



Notes:

- Pins 50, 51, 55 and 56 can be used a GCLK.

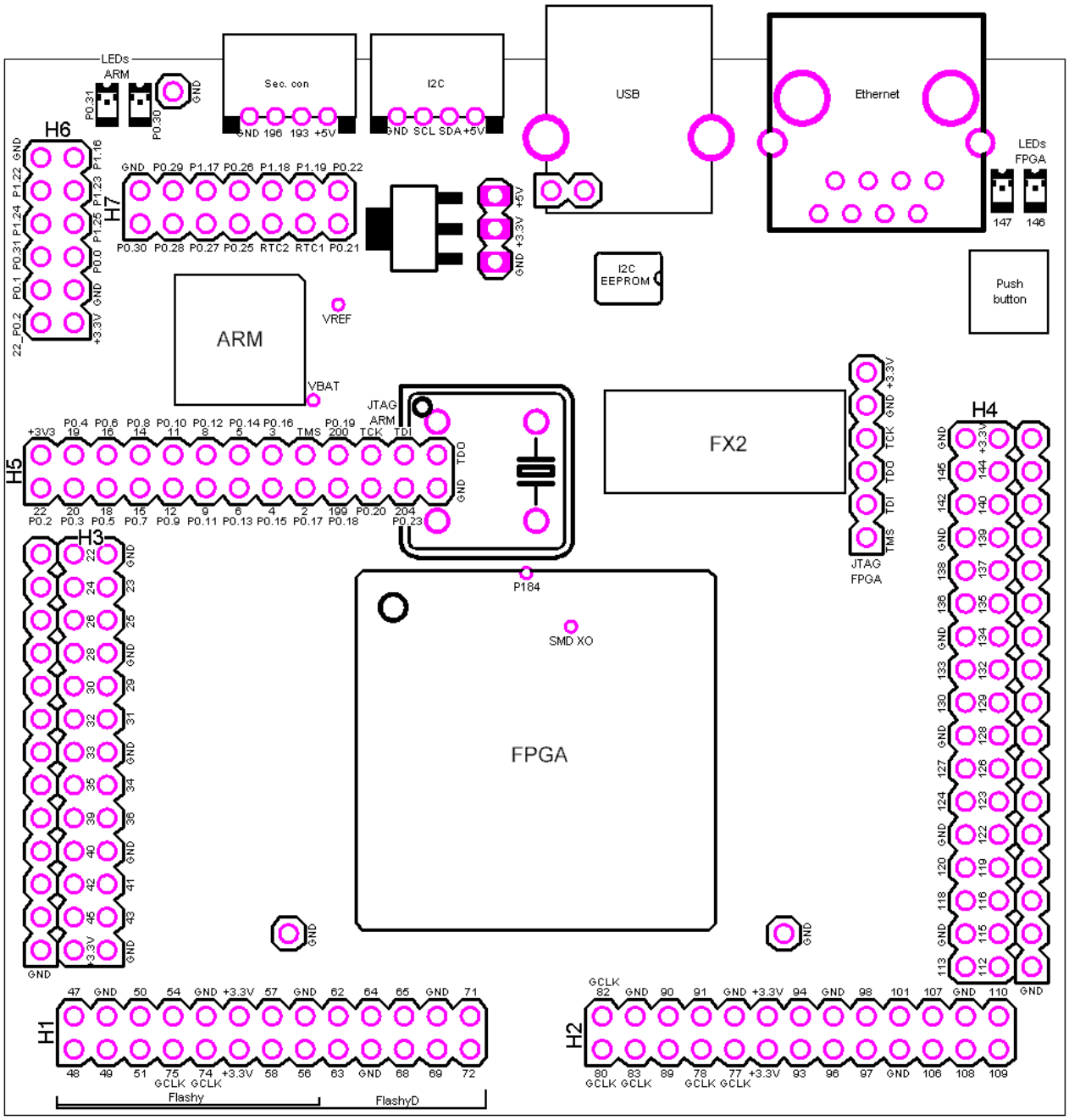
38.5 Xylo-EM



Notes:

- An extra IO (pin 125) is available on a single header pin in the middle of the board.
- A DIL-8 oscillator can be added on the board (marked “optional oscillator”).

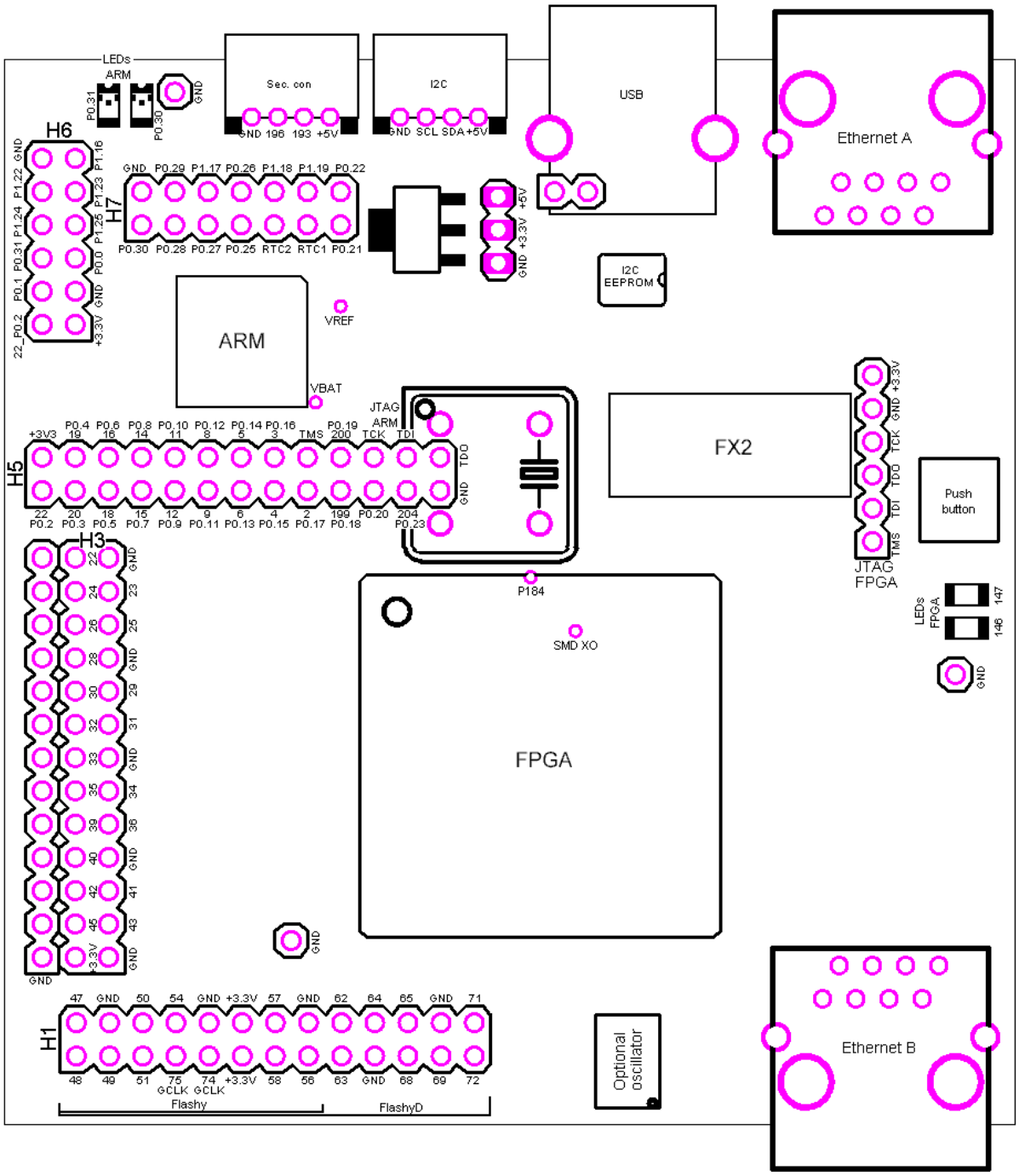
38.6 Xylo-L



Notes:

- Some pins have two numbers on header H5 (connected to both the FPGA and the ARM).
- Headers H1 and H2 are FlashyD compatible (but the FlashyD demo design supports only H1).
- Headers H3 and H4 have a third row of ground pads, so that high-speed connections that require an alternate pattern signal/ground can be made by using rows 2 (signal) and 3 (ground).
- A DIL-8 oscillator can be added on the board (partially covers H5).
- An SMD oscillator can be soldered on the bottom of the board, below the FPGA (note: a wire is required).
- FPGA pin 56 is dual purpose (IO and INIT_B). Pulling it low prevents FPGA configuration.

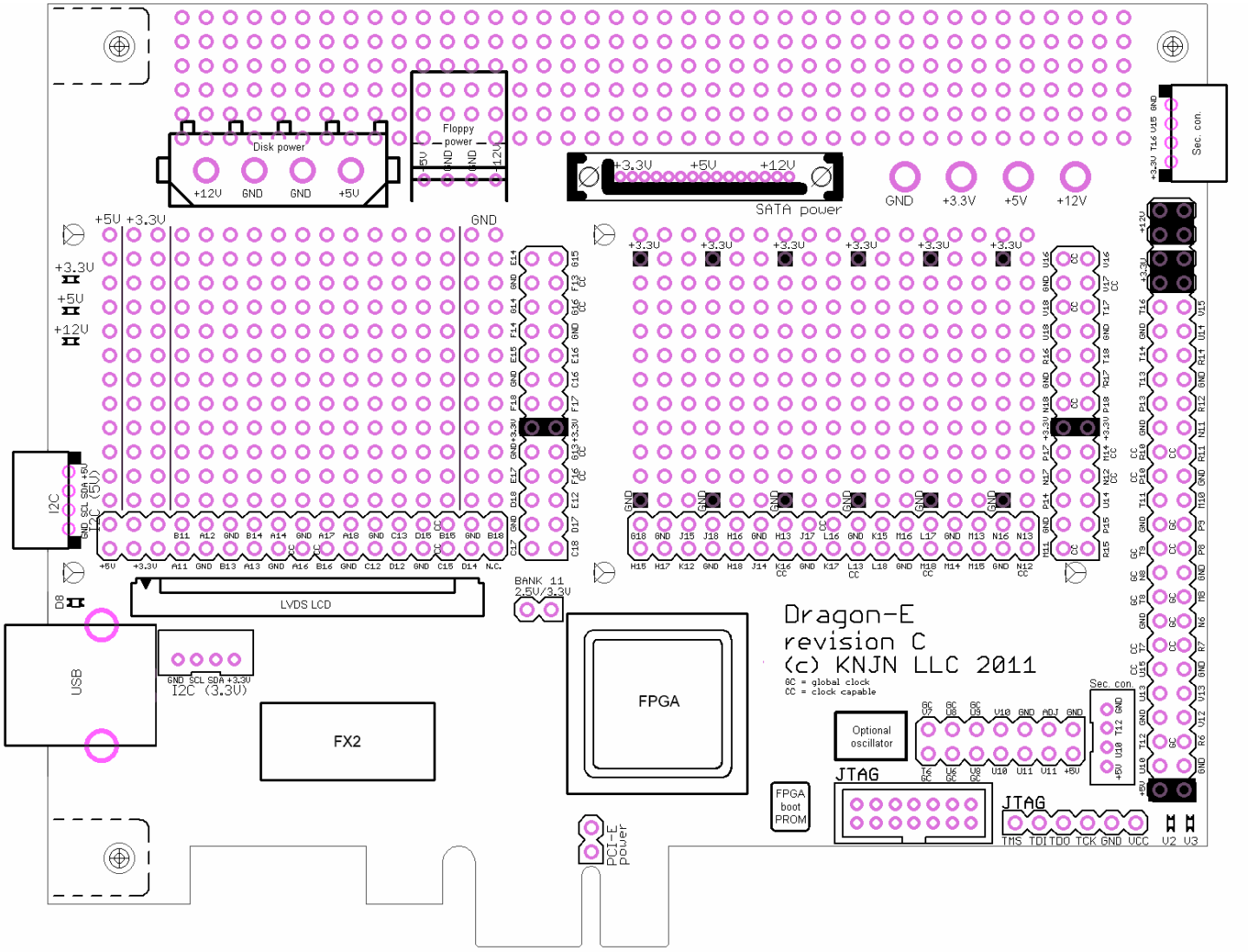
38.7 Xylo-LM



Notes:

- Some pins have two numbers on header H5 (connected to both the FPGA and the ARM).
- Header H1 is FlashyD compatible.
- Header H3 has a third row of ground pads, so that high-speed connections that require an alternate pattern signal/ground can be made by using rows 2 (signal) and 3 (ground).
- A DIL-8 oscillator can be added on the board (partially covers H5).
- One SMD oscillator can be soldered on the top of the board (to the left of the Ethernet B connector).
- Another SMD oscillator can be soldered on the bottom of the board, below the FPGA (note: a wire is required).
- FPGA pin 56 is dual purpose (IO and INIT_B). Pulling it low prevents FPGA configuration.

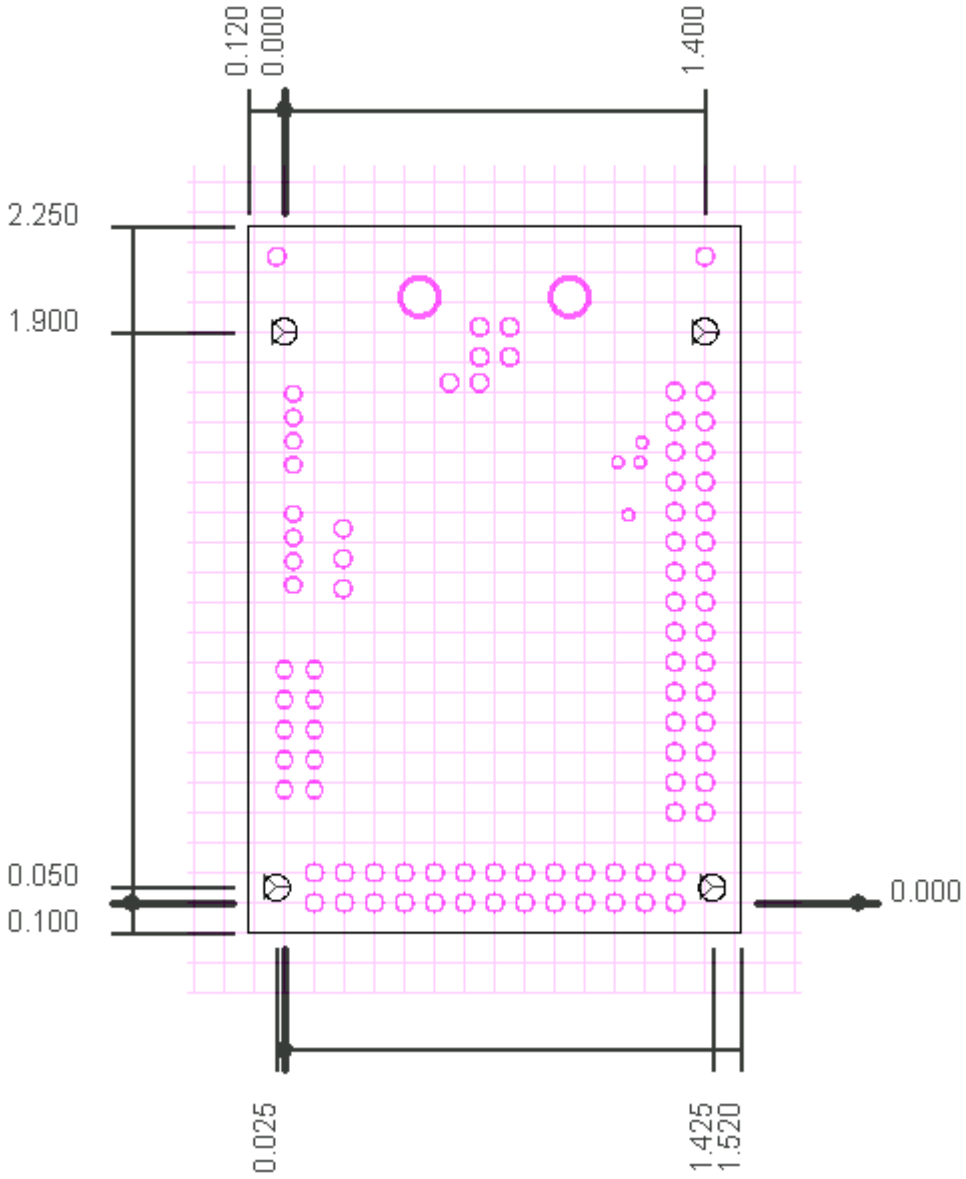
38.8 Dragon-E


Disk power
+12V GND GND +5V
Floppy power
+5V GND GND +12V
+3.3V +5V +12V
SATA power
GND +3.3V +5V +12V
Sec. con.
+5V +3.3V
GND
+3.3V
+5V
+12V
I2C
GND SCL SDA +5V (5V)
LVDS LCD
BANK 11
2.5V/3.3V
USB
GND SCL SDA +3.3V
I2C (3.3V)
FX2
FPGA
Dragon-E
revision C
(c) KNJN LLC 2011
GC = global clock
CC = clock capable
Optional oscillator
FPGA boot PROM
JTAG
TMS TDI TDO TCK GND VCC
V2 V3
PCI-E power

39 Mechanical drawings

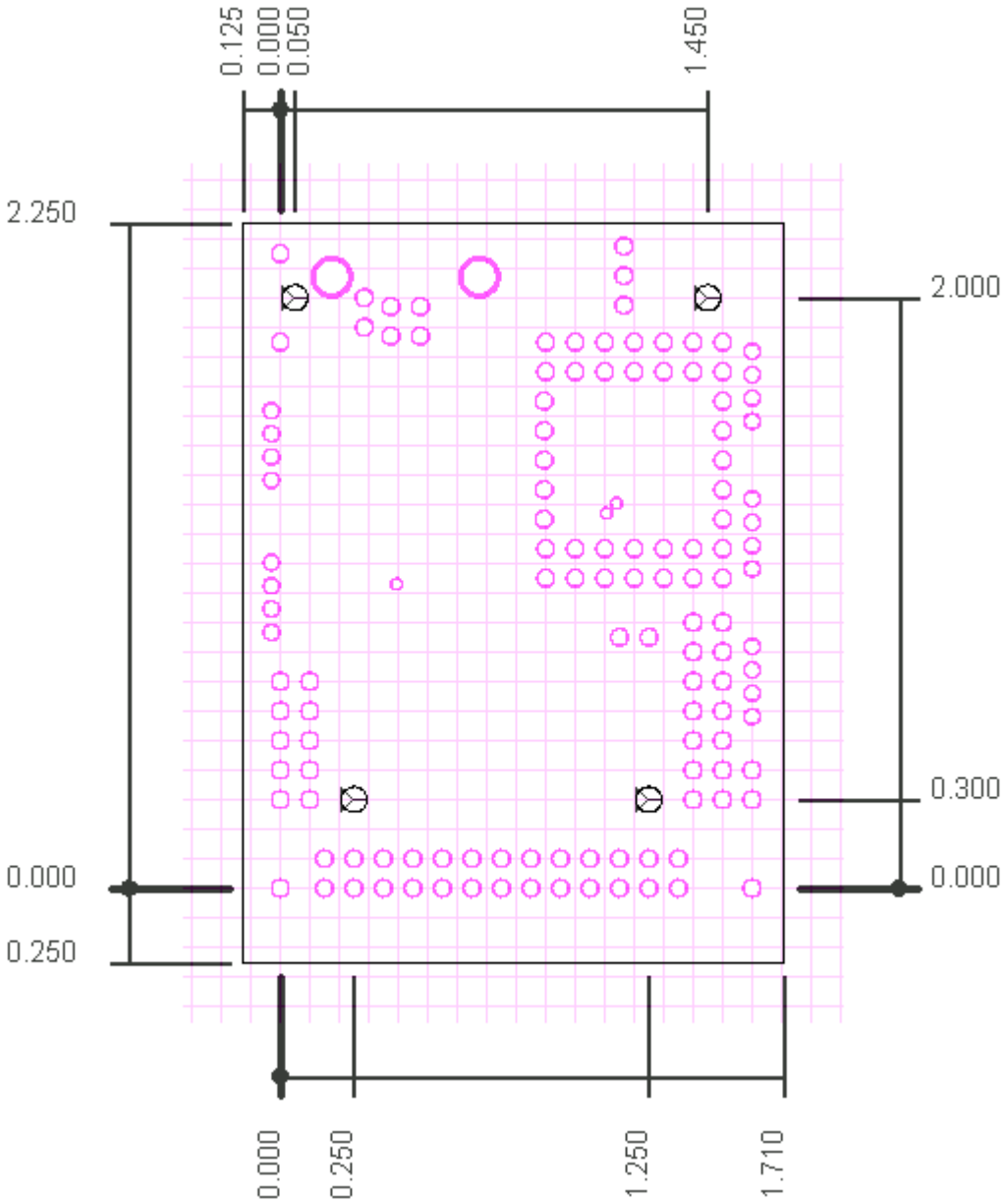
All dimensions are given in inches (1 inch = 25.4mm).

39.1 Saxo

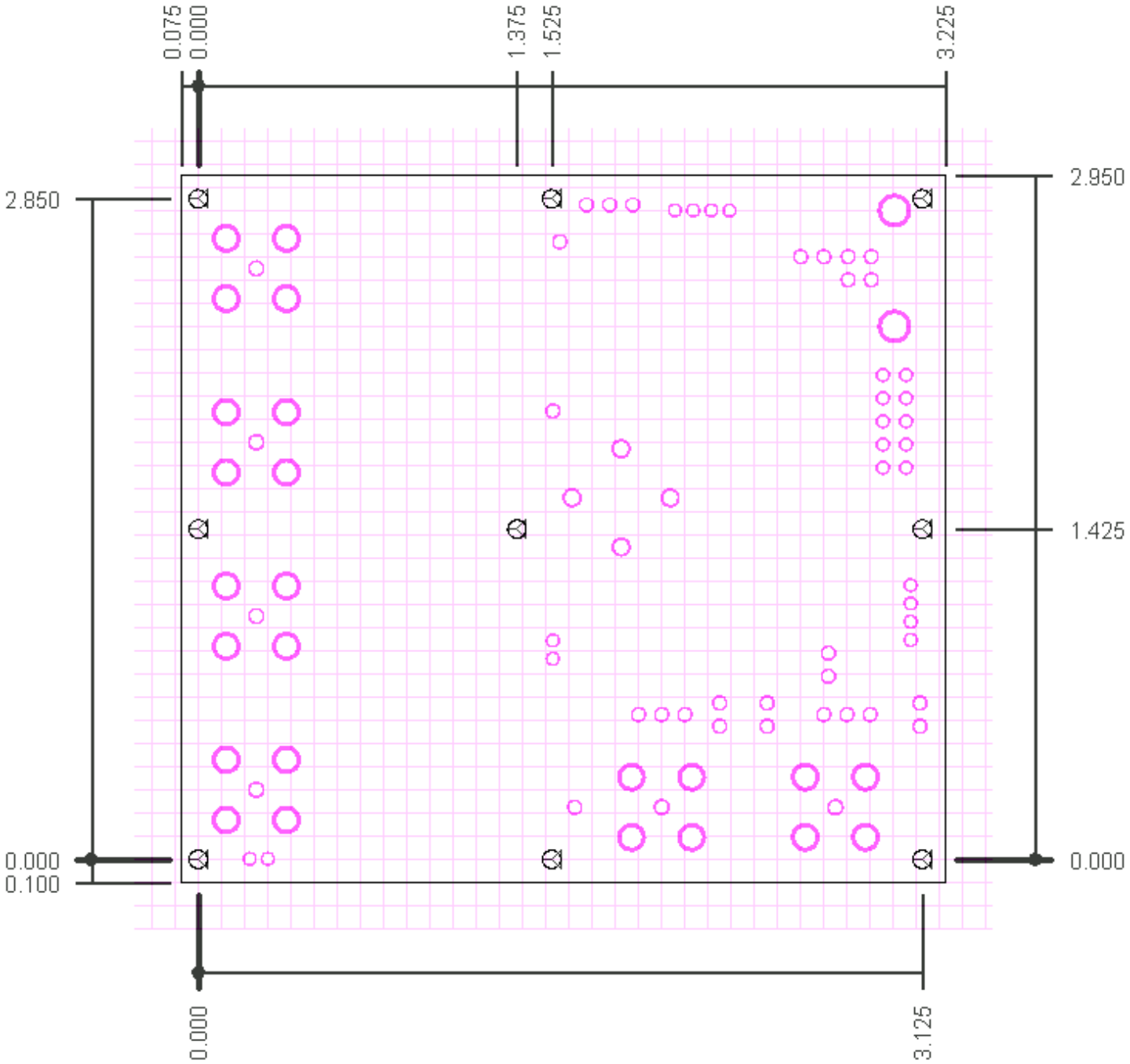


Note: Saxo is a smaller version of Xylo.

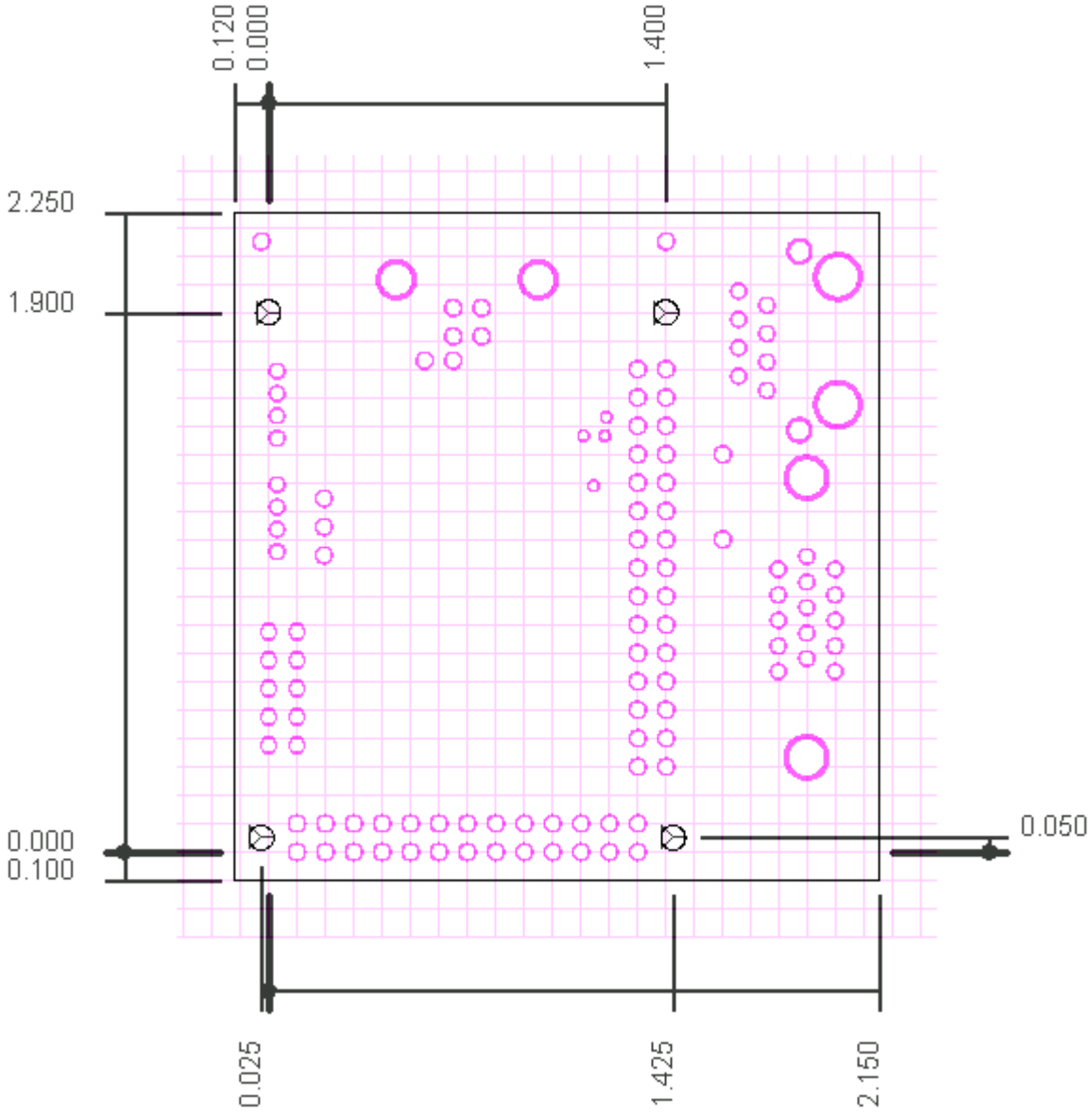
39.2 Saxo-L


0.125
0.000
0.050
1.450
2.250
2.000
0.300
0.000
0.000
0.250
0.000
0.250
1.250
1.710

39.3 Saxo-Q

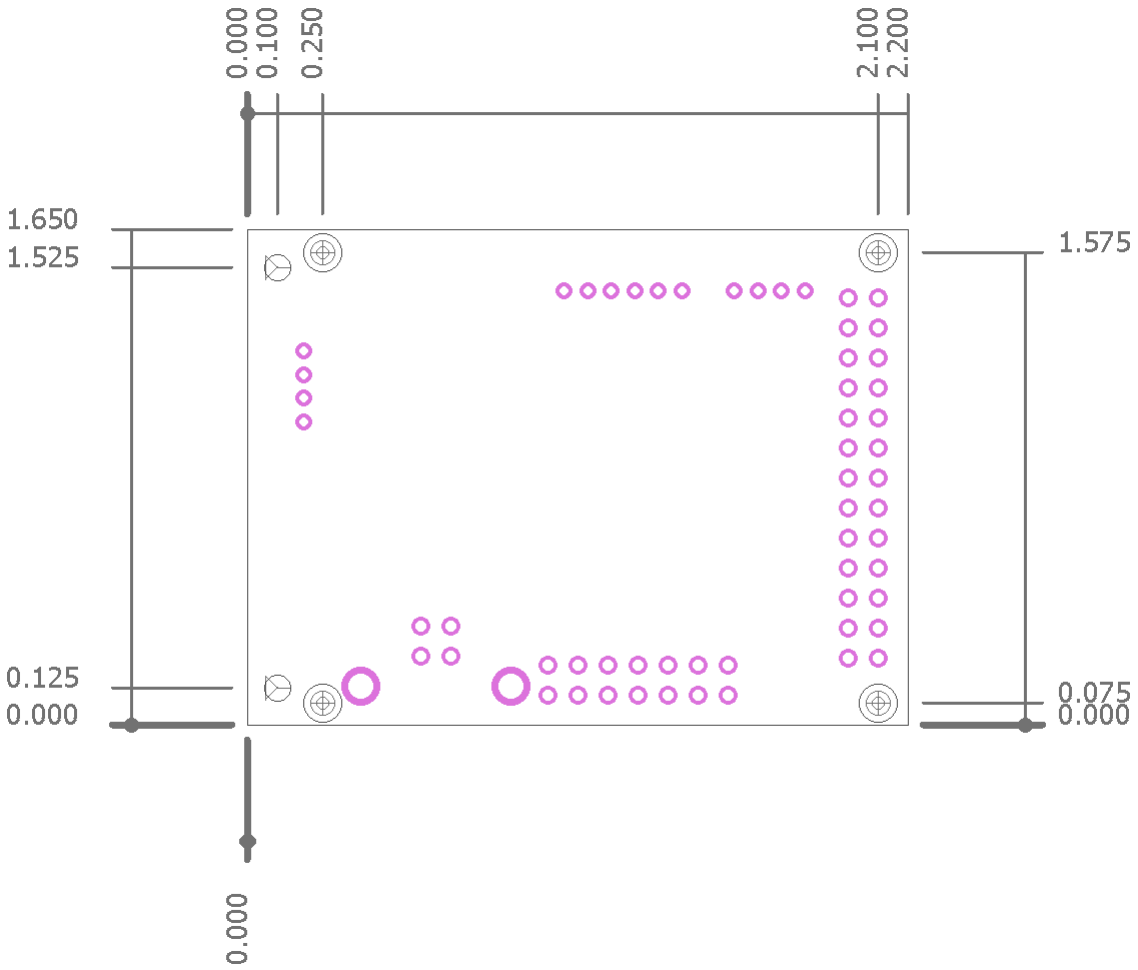

0.075
0.000
1.375
1.525
3.225
2.850
2.950
1.425
0.000
0.100
0.000
0.000
3.125

39.4 Xylo

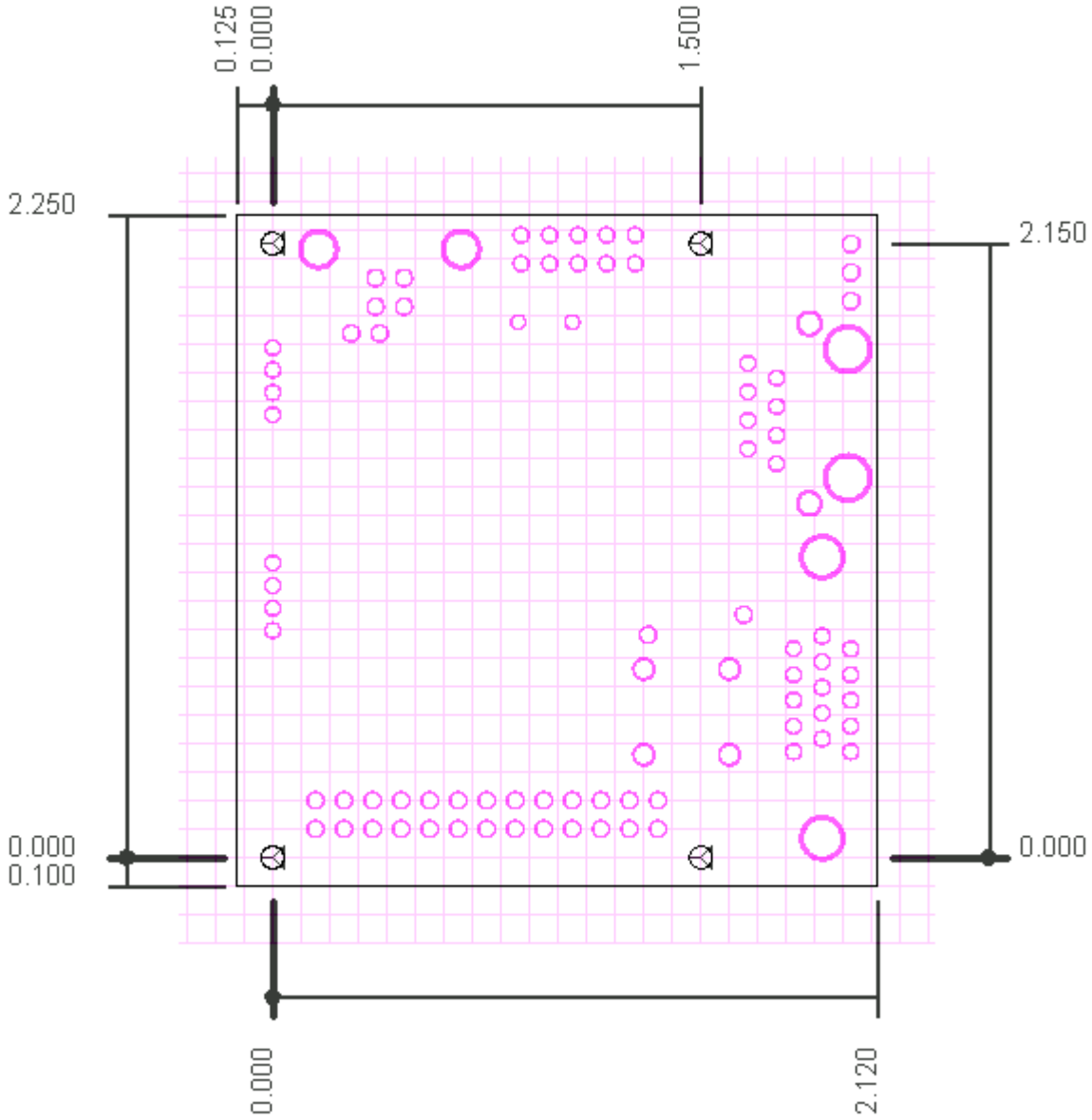


Note: Xylo is a bigger version of Saxo.

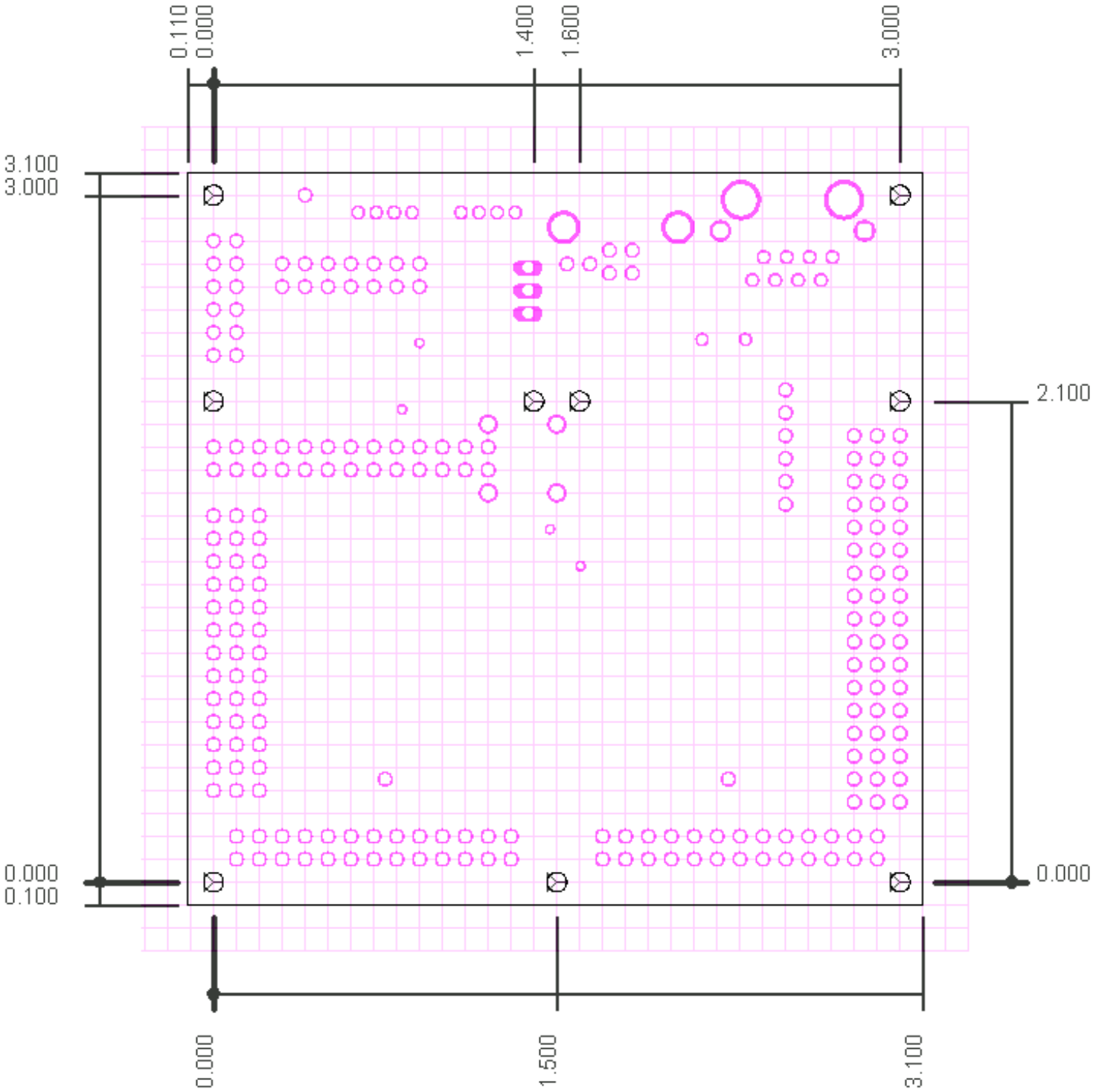
39.5 Xylo-E


0.000
0.100
0.250
2.100
2.200
1.650
1.525
1.575
0.125
0.000
0.075
0.000
0.000

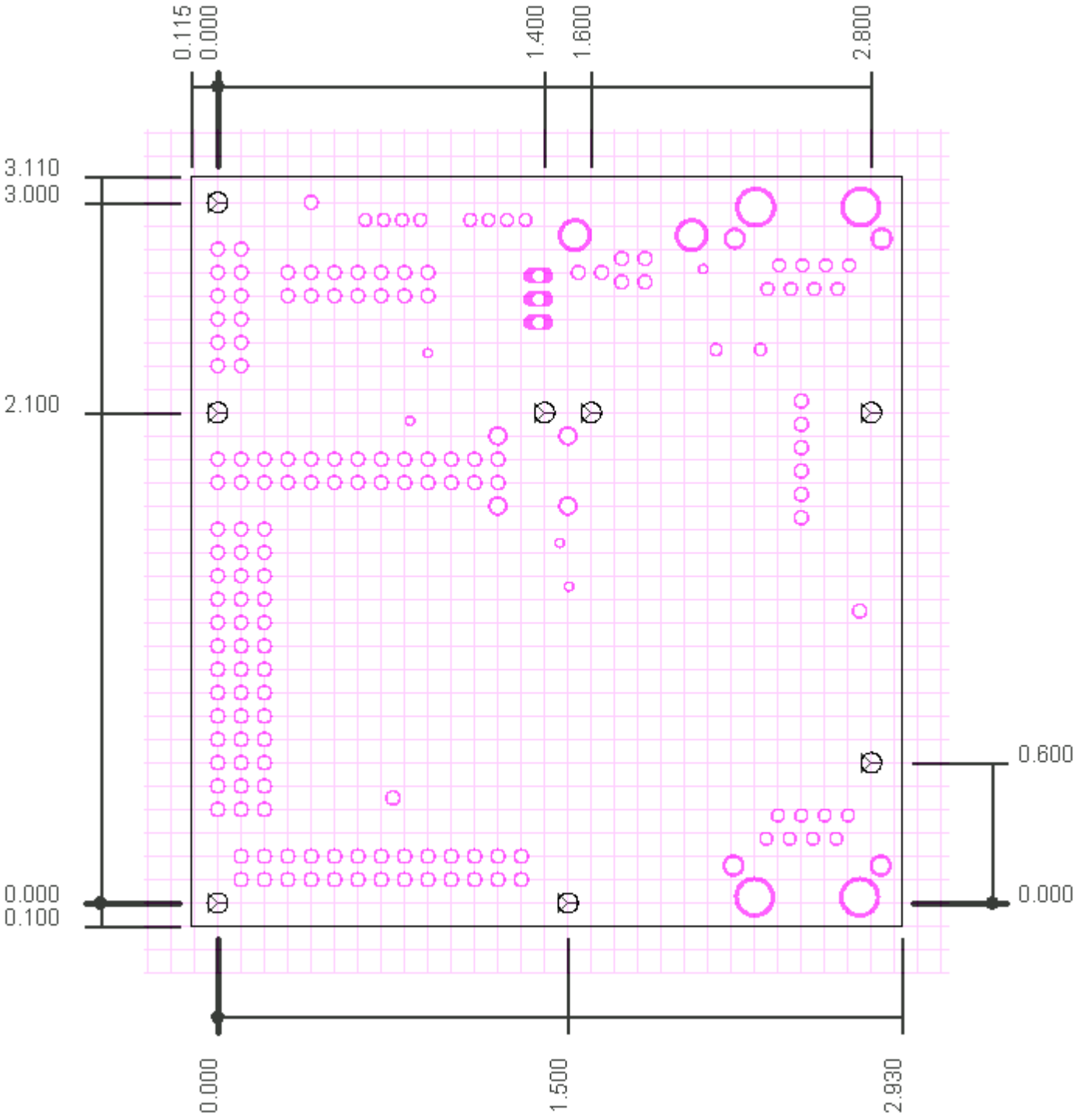
39.6 Xylo-EM



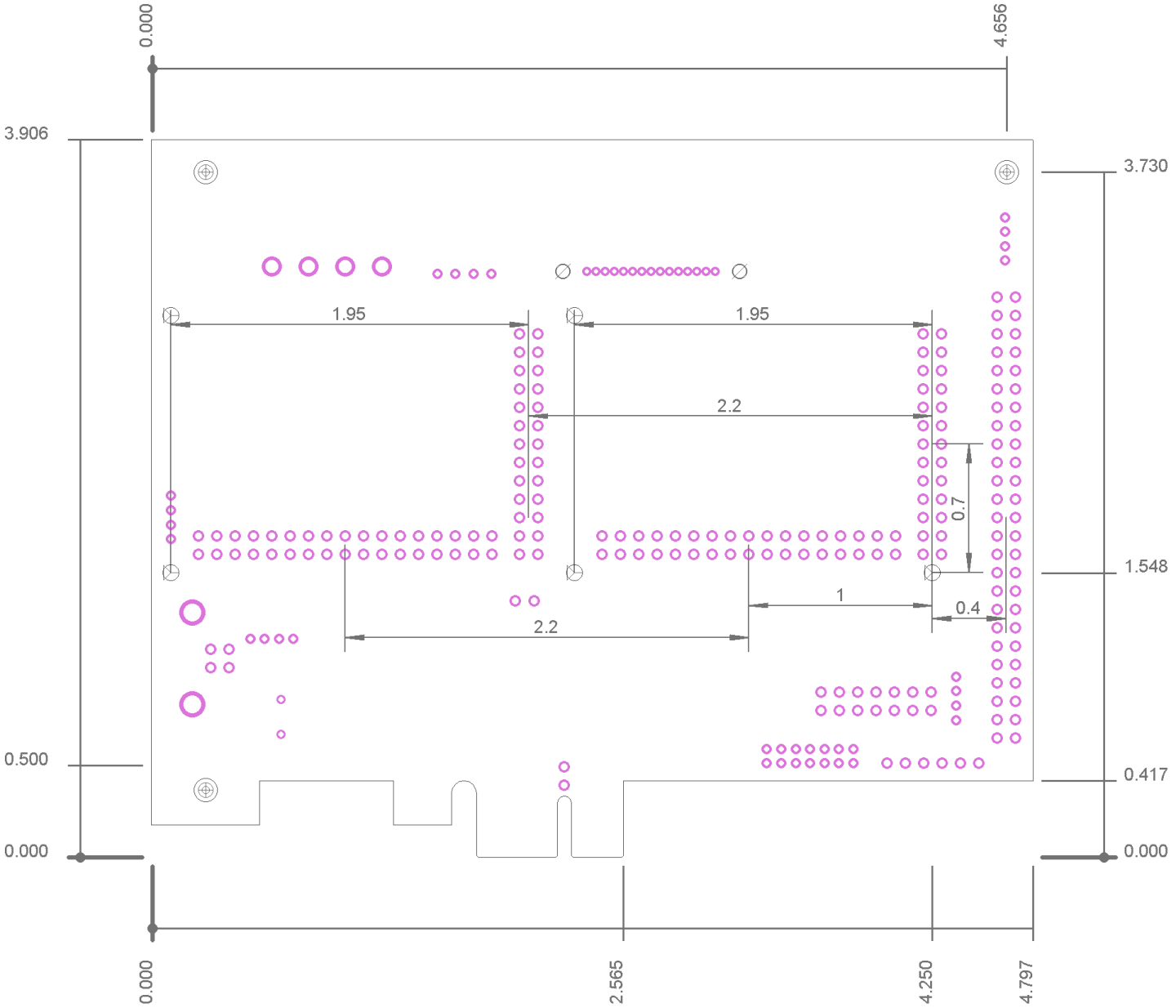
39.7 Xylo-L


0.110
0.000
1.400
1.600
3.000
3.100
3.000
2.100
0.000
0.100
0.000
0.000
1.500
3.100

39.8 Xylo-LM


0.115
0.000
1.400
1.600
2.800
3.110
3.000
2.100
0.600
0.000
0.000
0.100
0.000
1.500
2.930

39.9 Dragon-E


0.000
4.656
3.906
3.730
1.95
1.95
2.2
0.7
1.548
1
0.4
2.2
0.500
0.417
0.000
0.000
0.000
2.565
4.250
4.797

40 Errata

- On Saxo and Xylo rev. G, the JTAG connector signal names shown on the bottom silkscreen are not correct. Please use the top silkscreen.
- On Dragon-E rev. B, the 5V secondary connector is mirrored so it should be soldered on the back of the board.