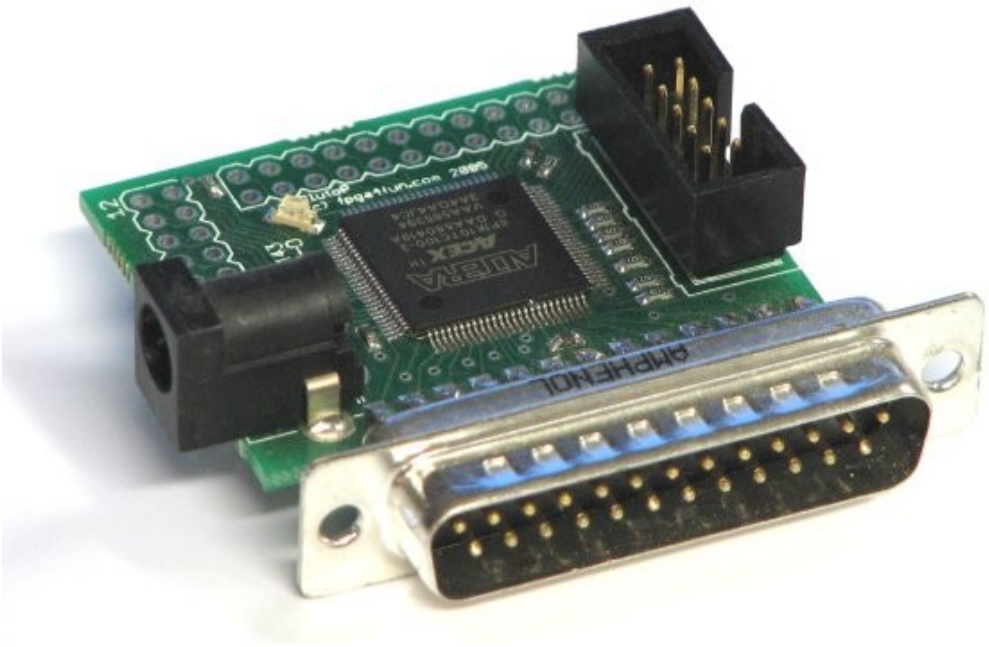
KNJN FPGA Pluto-P parallel development board

http://www.knjn.com/

This document applies to the following board.

- Pluto-P (revision B)



Document last revision on **March 6, 2024**

Table of Contents

1 Board installation

1.1 Software tools

Many files included with the Pluto-P board are in source-code form.

You need the following tools to compile them:

1. FPGA: get the free Quartus II Web Edition 9.0 Service Pack 2
2. C compiler: get Microsoft Visual C++ or Digital Mars C compiler

1.2 Driver installation

Pluto-P connects to the parallel port of your PC and can be used without any driver.

A Windows driver can be used for enhanced access to the parallel port (SPP & EPP IO modes). See chapters 6 & 7 for more info.

1.3 Power connection

The board can be easily powered from a DC-adapter.

- Use a DC-adapter that provides between +5V and +10V.
- Make sure the DC-adapter provides the positive voltage in the center of its plug.

Note: when used as a “JTAG cable”, you do not need a DC-adapter as Pluto-P borrows power from its host JTAG board.

2 FPGA

2.1 FPGA configuration

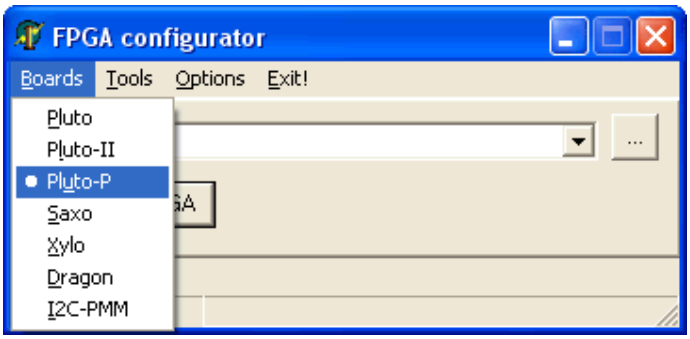
You configure Pluto-P's through the parallel port of your PC by loading an ".rbf" file (called RBF in this document) into the FPGA.

Configuration works as follow:

1. Connect Pluto-P to the parallel port on your PC.
2. Apply power to Pluto-P. Pluto-P's LED glows slightly.
3. Run "FPGAconf", select an RBF file, and click "Configure".

For your convenience, a sample "ledblink.rbf" file is provided in Pluto-P's "StartupKit\Sample files" directory.

Note that FPGAconf is a generic application that can be used with other FPGA boards. Make sure that Pluto-P is selected in the "Boards" menu.



2.2 FPGAconf parallel modes

FPGAconf can be used in three modes:

1. Printer mode
2. IO SPP mode
3. IO EPP mode

The mode can be changed in the "Options/Printer port" menu.

2.3 Printer mode

- Configuration takes about 2 seconds.
- No Windows driver is required.
- The printer port is usually spooled by Windows, so that the FPGA data is actually stored by Windows and then sent when Pluto-P is present. If Pluto-P is not connected, the data is stored for the next time Pluto-P is connected. If configuration doesn't work, investigate the issue but don't press "Configure" multiple times as each request is spooled and may cause confusion later. If multiple requests are spooled, you can erase them in Window's printer spooler window.

2.4 IO mode (SPP or EPP)

- Configuration takes about 1 second.
- You need to install a driver, see paragraph 7.1.
- There is no spooling as FPGAconf takes controls of the parallel port (over Windows).

3 Pins, connectors and headers

3.1 FPGA pins

Pluto-P's most important board pins are:

Pin name	FPGA pin number	Comment
CLK	91	40 MHz crystal oscillator clock
LED	50	LED (active high)
nConfig	49	Pull high for normal operation, pull low to un-configure FPGA
nPaperOut	6	Pull high if the FPGA is ever configured in printer mode

IO headers:

Two headers are available to connect other peripherals to Pluto-P.

1. CONIO1: 16 FPGA IOs + 1 clock + 2 dedicated inputs + 3.3V power.
   This header is compatible with FlashyD. Note that FPGA pin 72 can also be used as clock.
2. CONIO2: 12 FPGA IOs

See the FPGA pin assignments drawing (chapter 9) for more information.

3.2 Parallel port

The FPGA is connected to all the parallel port signals.

Parallel port pin number	SPP port pin name	EPP port pin name	FPGA pin number	FPGA polarity	Comment
1	nStrobe	nWrite	90		
2	Data 0	Data 0	77		
3	Data 1	Data 1	79		
4	Data 2	Data 2	82		
5	Data 3	Data 3	85		
6	Data 4	Data 4	86		
7	Data 5	Data 5	94		
8	Data 6	Data 6	96		
9	Data 7	Data 7	97		
10	nACQ	Intr	98		
11	Busy	nWait	87	Inverted	
12	nPaperOut		6	Inverted	Pull high if the FPGA is ever configured in printer mode
13	Select		99		
14	AutoLF	nDataStr	80		
15	nError		78		
16	nInit		81		
17	nSelect	nAddrStr	84		
18-25	Ground	Ground			

4 Pluto-P as a JTAG cable

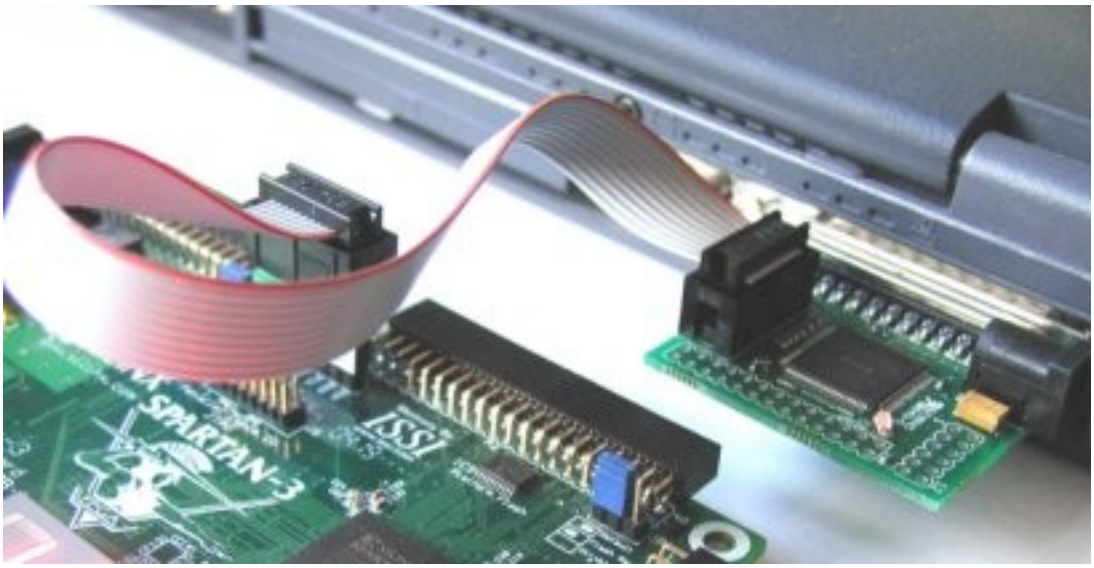
4.1 Pluto-P modes

Pluto-P can be used in two modes:

- Standalone mode
- "JTAG cable" mode

A “JTAG cable” is an active cable (with electronics inside) allowing a PC to connect to an external JTAG port. Pluto-P can to be configured to behave like a JTAG cable, and so can control another board JTAG port.

See here Pluto-P connected to a Xilinx board:



The following RBF files are provided to emulate different JTAG cables:

RBF files	*Emulation*
Pluto-P Xilinx Parallel.rbf	Xilinx JTAG parallel-III
Pluto-P Altera ByteBlaster II.rbf	Altera JTAG ByteBlaster II

4.2 Using Pluto-P as a JTAG cable

1. Configure the Pluto-P FPGA using one of the previous RBF files. Pluto-P enters the “JTAG mode”.
2. Use one the following software as JTAG programmer:
    - Xilinx ISE WebPack and Xilinx ISE iMPACT
    - Altera Quartus-II Web Edition and Altera Quartus-II Programmer

Note: when Pluto-P enters “JTAG mode”, it stays in this mode until it is powered down.

5 Pluto-P as an FPGA board

Pluto-P can to be used as a regular FPGA board.

There are a couple of things you want to enable while using Quartus-II with your Pluto-P board.

1. While in your Quartus-II project, go to “Assignments/Device”
2. Choose Family “ACEX1K” and device “EP1K10TC100-3”
3. Click on “Device & Pin Options…”
    a. Go to the “Programming Files” tab, select “Raw Binary File (.rbf)”
       Otherwise, only SOF files are created. See Chapter 2 for more information.
    b. Go to Unused Pins”, select “As inputs, tri-stated”
    c. Click “OK”
4. Click “OK”

Option 3.b is just a convenience. It prevents the FPGA from driving pins that are not used in your project. Otherwise, Quartus-II drives ground on 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.

See http://www.fpga4fun.com/QuartusQuickStart.html for step-by-step instructions on how creating your own project.

6 Pluto-P in SPP mode

SPP is the standard parallel mode.

In SPP mode, Pluto-P is treated as a printer.

6.1 LEDblink

One simple Verilog HDL file that can be used to make an LED blink is:

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

reg [31:0] cnt;
always @(posedge clk) cnt <= cnt + 32'h1;

assign LED = cnt[23];
endmodule
```

This file won't work in Pluto-P as-is. As shown in paragraph 3.1, the pin nConfig and nPaperOut need to be assigned for normal operation.

Here's our updated file:

```
module ledblink(clk, LED, nConfig, nPaperOut);
input clk;
output LED;
output nConfig, nPaperOut;

assign nConfig = 1'b1;        // nConfig high for normal operation
assign nPaperOut = 1'b1;      // nPaperOut high for printer mode configuration

////////////////////////////////////
reg [31:0] cnt;
always @(posedge clk) cnt<=cnt+32'h1;

assign LED = cnt[23];
endmodule
```

Before compiling it in Quartus-II, don't forget to make the correct pin assignments (in the "Assignments/Pins" menu) with clk = pin 91, …

See the complete project in Pluto-P's startup-kit "Projects\SPP\SPP1".

If you want to un-configure the FPGA (i.e. to try another RBF file), power-cycle Pluto-P. Or read the next paragraph.

6.2 nConfig pin

Pluto-P FPGA has one particular feature. It can un-configure itself. This is done by asserting the nConfig pin low.

In the previous example (LED1), the only option to un-configure the FPGA was to power-cycle the board.

Now, let's modify the example to let the LED blink 8 times, and then un-configure the FPGA.

```
module ledblink(clk, LED, nConfig, nPaperOut);
input clk;
output LED;
output nConfig, nPaperOut;

assign nPaperOut = 1'b1;

////////////////////////////////////
reg [31:0] cnt;
always @(posedge clk) cnt<=cnt+32'h1;

assign LED = ~cnt[23];
assign nConfig = ~cnt[27];    // un-configure the board once cnt[27] becomes high
endmodule
```

Try this example for the SPP\SPP2 directory. See how the LED blinks 8 times and then the FPGA "shuts down".

6.3 Un-configure the FPGA on command

We read the parallel port data from the PC, and un-configure the FPGA once we receive a specific data. The value chosen is 8'b01010101 = 0x55 = 85.

```
module ledblink(clk, LED, nConfig, nPaperOut, PPi);
```

```
input clk;
output LED;
output nConfig, nPaperOut;
input [7:0] PPi;

assign nPaperOut = 1'b1;

////////////////////////////////////
reg [31:0] cnt;
always @(posedge clk) cnt<=cnt+32'h1;

assign LED = ~cnt[23] & ~cnt[21];
assign nConfig = ~(PPi==8'b01010101);       // un-configure the board when 01010101 is received
endmodule
```

We call PPi the 8-bits port that receives the parallel data from the PC (named “data” on paragraph 3.2).

Try this example in the SPP\SPP3 directory:

1. Configure Pluto-P
2. Switch to printer mode (see paragraph 2.2)
3. Open the FPGAconf scroll-bar window (CTRL-S) and move the bar to the value 85 (= 0x55). As soon as you reach 85, Pluto-P FPGA un-configures.

This trick can be used in all your Pluto-P projects because FPGAconf sends an “85” every time it wants to re-configure the FPGA. This way, you don't need to power-cycle Pluto-P to re-configure it.

6.4 Get data from the PC

Now we would like Pluto-P to receive data from the PC.

The parallel port specifies that the nStrobe signal toggles for each byte received. We could use nStrobe directly as a clock signal, but a more practical thing to do is to sample nStrobe using the internal Pluto-P clock.

Get the complete project in the SPP\SPP4 directory.

Again, you can use FPGAconf's scroll-bar to send data to Pluto-P.

6.5 C printer code

Here's an example C code:

```
char* LPTname = "LPT1";                     // parallel port used

void trythis()
{
    FILE* F = fopen(LPTname, "wb");
    fputc(0x55, F);           // send one character to the parallel port
    fclose(F);
}
```

Pluto-P is treated as a printer. No driver is required.

7 Pluto-P in EPP mode

EPP allows fast and easy bi-directional communication with the PC.

EPP requires IO access to the parallel port IOs.

7.1 IO access

With Windows XP, the PC requires a driver to allow access to the PC's IO ports.

We recommend using a driver like UserPort.

7.2 EPP tutorial

EPP provides bi-directional communication over a parallel port, i.e. a way to read-from and write-to a peripheral attached to a PC's parallel port.

Check http://www.fpga4fun.com/EPP.html for a tutorial.

7.3 EPP HDL files

Pluto-P's startup-kit has example files in Projects\EPP\EPP1 and EPP2.

7.4 EPP C code

Pluto-P's startup-kit has an example C file in Projects\EPP.

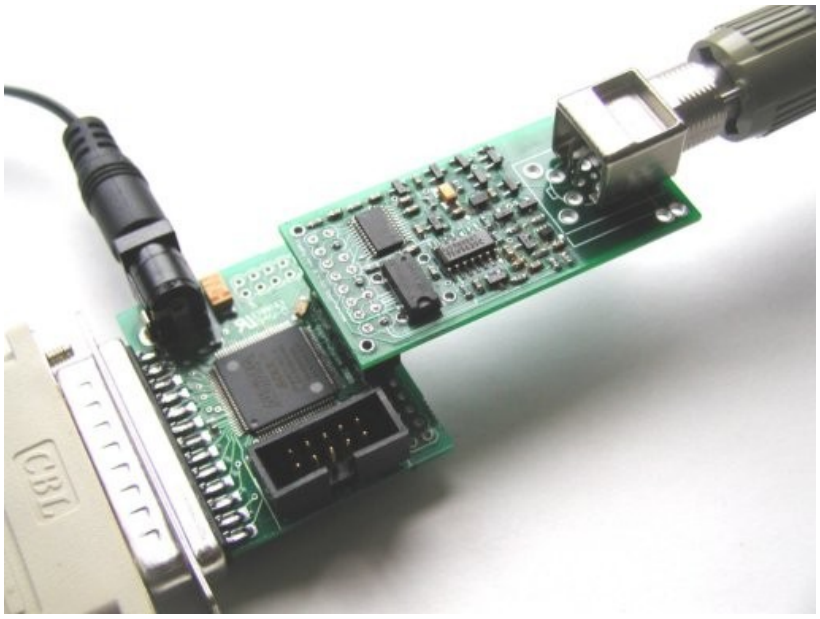
8 Flashy

Pluto-P output connector is compatible with Flashy and FlashyD.

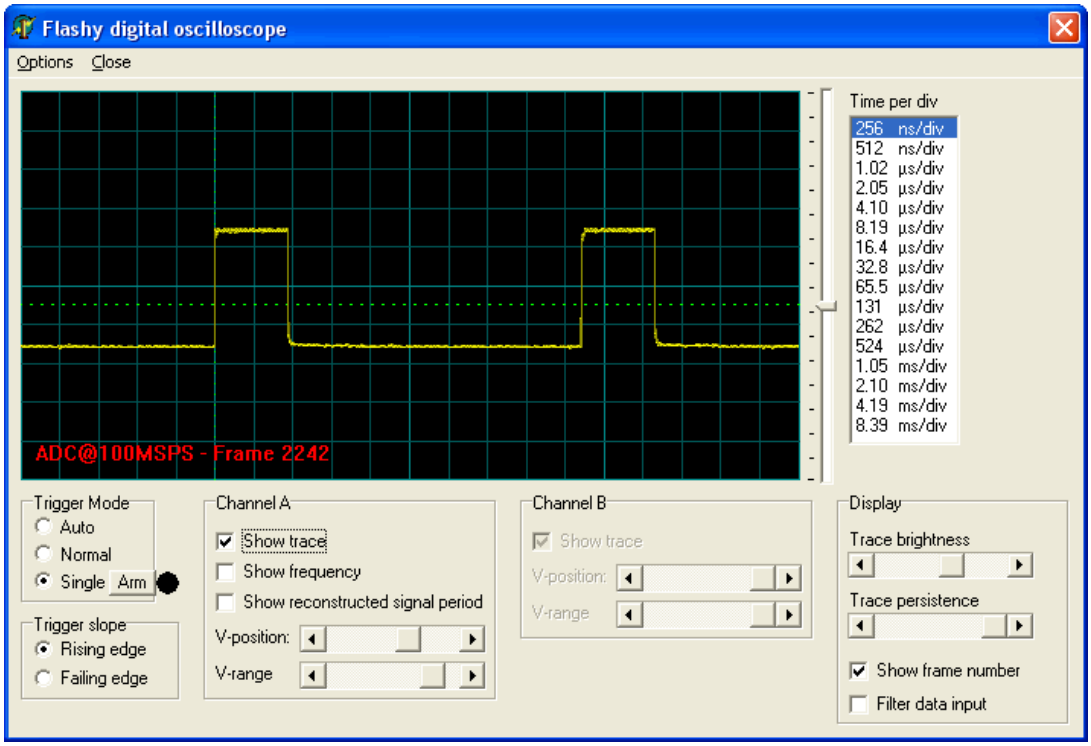
8.1 Reference design

A reference design for Flashy (1 channel) is provided (the FPGA is too small for FlashyD reference design).

1. Make sure Pluto-P is not powered.
2. Plug Flashy on Pluto-P (see paragraph 9). Connect an oscilloscope probe on Flashy.
3. Power-up Pluto-P.



4. Go into EPP mode (see paragraph 2.2).
5. Configure the FPGA with “FlashyDemo.rbf”
6. Press CTRL-F. Flashy windows appears. You are ready to probe



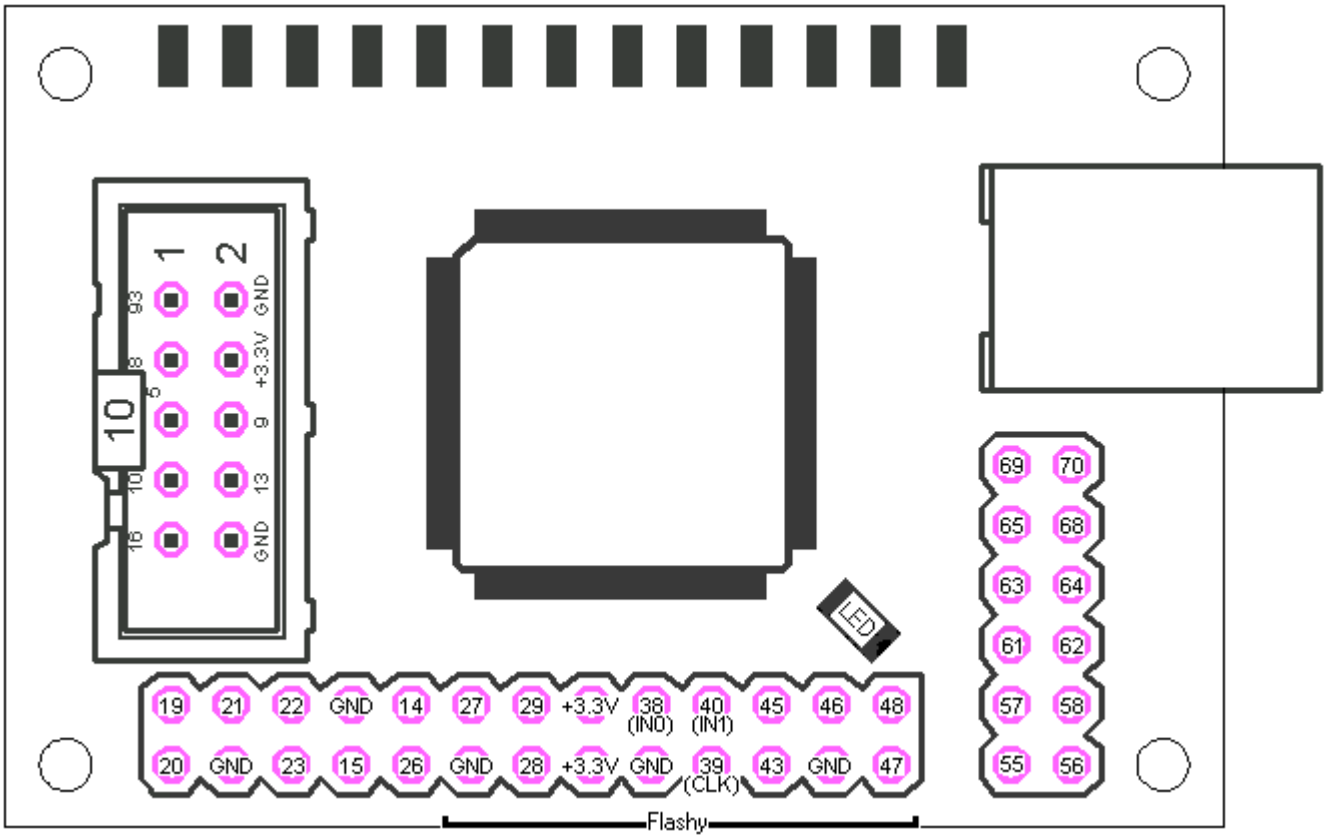
8.2 Acquire data

If you want to write your own code that acquires data to the PC using Flashy's reference design, use the following example code as a startup point.

```
#define nFlashyChannels 1     // Pluto-P supports only one channel

void main()
{
    int i, ch;
    char bufTrigger[3] = {0x00, 0x00, 0x80};
    char bufADC[16+nFlashyChannels*512];
    EPP_init();

    // send trigger information
    EPP_write_addr(0);
    EPP_write_data_block(bufTrigger, sizeof(bufTrigger));

    // wait until scope triggered and data is available
    while(EPP_read_addr()&1) ;

    // get the data
    EPP_read_data_block(bufADC, sizeof(bufADC));

    // display the data
    for(ch=0; ch<nFlashyChannels; ch++)
    {
        printf("Channel %d: ", ch+1);
        for(i=0; i<512; i++) printf("%d ", bufADC[i+512*ch]);
        printf("\n");
    }

    // we could re-trigger to get more data if we wanted
    // but let's just exit now
    printf("Press a key to exit"); getch();
}
```

9 Board connectors and headers, with IO pin assignments

Here's the board layout, with the FPGA IO pin assignments shown on the headers.



10 Mechanical drawing

All units in inches (1 inch = 25.4mm)