# Pixie-Net Xillinux *userdev* Firmware

2/3/2017

## Conditions of Use

Copyright (c) 2017 XIA LLC

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
* Neither the name of XIA LLC nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF

MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.

IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,

PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

## Vivado Setup

The Pixie-Net *userdev* Vivado firmware project is based on the Xillinux demo project for MicroZed, an Ubuntu Linux distribution with easy access to FPGA resources. This project contains the “always free” xillybus lite, which is used in the Pixie-Net, and the “free for evaluation” full xillybus, which provides higher rate DMA transfers, but is not used in the Pixie-Net. The steps below outline how to set up the Vivado project. Direct copy from one PC or folder to another will likely not work due to Vivado’s handling of internal data caches. Rather, the Vivado project has to be generated by a script. See also xillybus\_getting\_started\_zynq.pdf. Some of the files mentioned are also included in the “userdev” package provided by XIA. Steps involving svn are optional. Developed on Vivado 2015.4.

1. Get xillinux demo package for zynq from xillybus website http://xillybus.com/xillinux   
Use version 1.3 (xillinux-eval-microzed-1.3c.zip), not the just recently released version 2.0a

2. Unzip to working folder, e.g. C:\XIA\PixieDesktopZynq\firmware\userdev

3. Add to svn

3. Open Vivado. Use *Tools > Run Tcl Script* to run project generating script …/verilog/xillydemo-vivado.tcl. The resulting project file is in ...\verilog\vivado  
There have been cases where the script crashes Vivado, and then the compile has ~100 pin property critical warnings. In such cases, start over.

5. Compile demo project (generate bitstream). Ignore warnings and critical warnings.

6. Add...\verilog\vivado\xillydemo.runs\impl\_1\xillydemo.bit to svn and commit

7. Create folders for xia processing code: …\userdev\xpp\verilog and …\userdev\xpp\cores. Copy XIA source files into these folders. Add to svn and commit

8. Copy the XIA-modified xillydemo.v into ..\verilog\src. This is the top level file in Vivado and the first of the two files from the xillinux firmware modified by XIA.

9. Replace ...\vivado-essentials\showstopper.tcl with a comment only file (else prohibits compiles with timing errors!). This is the second of the two files from the xillinux firmware modified by XIA.

10. In Vivado, open xillydemo.v. This is the top level module that instantiates a xillybus module with connections to the PS. XPP.v contains the XIA pulse processing and is called out as a submodule in the XIA-modified xillydemo.v.

11. Add Verilog sources for XPP.v and its submodules:   
Right click on XPP.v and select "add sources", choose "Add or create design sources > Add Direcories" and select XPP/verilog folder.  
Make sure "add sources from subdirectories" is checked  
Make sure "copy sources into project" is NOT checked, though Vivado may do it anyway. If it does, make sure to find those .v files (possibly in …/Verilog/vivado/xillydemo.srcs/sources\_1/imports/…) and add them to svn or apply any modifications to /XPP/Verilog

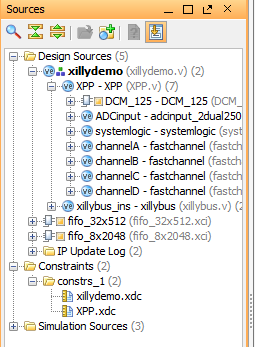
12. Add IP cores for XPP.v and its submodules:   
Expand XPP.v, right click on DCM core and select "add existing IP", choose "Add Direcories" and select XPP/cores folder. See notes in previous item.

13. Add constraints: Right click on constraints, add XPP/verilog/XPP.xdc

14. Compile (generate bitstream)  
There should be only 2 critical warnings about 2 fifo cores, which are unused since they are part of the full xillybus design.   
15. copy xillydemo.bit to boot partition of SD card

## Firmware Overview

The structure of the firmware is shown below. Under the xillydemo top project level, the XIA Pulse Processing module (xpp) includes submodules for module-wide logic (systemlogic.v), ADC data capture (adcinput\_2dual250.v) and 4 ADC channels (fastchannel.v).



### Xillydemo

The Xillybus module defines the I/O registers for communication with the Linux software. See the Pixie-Net software manual for register definitions. In the *userdev* variant, many of these registers are unconnected. They are organized into 4 groups:

* Registers  
  Registers are used to set DAQ parameters such as gain and offset, with the possibility to be read back. The FW does not change these values. For most of the pulse processing operation, these parameters are considered constants, i.e. they are set at the setup stage and don’t change during an acquisition.  
  The 5x 16 32bit registers visible to Linux are mapped into 5x 511bit wires for passing into the submodules, and extracted there into 2D arrays again. (Verilog does not support 2D arrays in ports.)
* Event data  
  The event data group contains pulse processing output data (and other information), generally on a per-pulse basis. These are intended for current output value, e.g. the pulse height from the most recent detected pulse. The 5x 16 32bit registers visible to Linux are defined in XPP.v and below as 2D register arrays, but brought up as 5x 511bit wires. Event data is read only.
* Run Statistics  
  The run statistics group contains cumulative values such as the run time, number of events processed, etc. The data from various counters is latched into intermediate registers in systemlogic.v and fastchannel.v. A counter *parnum* cycles through these intermediate registers and writes their values to a top level 2D runstats array which is visible to the Linux side. These registers are sequentially latched into a *runstats* array. Run statistics data is read only.

Writing or reading certain registers creates 1-cycle pulses (strobes), for example after writing a DAC value to set the offset, a strobe starts the serial I/O from FPGA to DAC.

### XPP

XPP.v instantiates a DCM to derive the following clocks from the 100 MHz user\_clk:

* dspclk, a 125 MHz clock for the pulse processing
* adcclk, a 250 MHz clock for the ADC clocking and data capture
* clk25cc, a 25 MHz clock that serves as a common denominator for dspclk and user\_clk in clock domain crossings
* clk8, an 8 MHz clock for DAC operations
* clk200idy, a 200 MHz clock for input delay control

It also instantiates the ADC input module, which decodes the 250 MHz DDR inputs from the two dual ADCs into 4 2-word wide 125 MHz data streams ADCdata[A..D][a,b][0..13]. XPP.v further instantiates all input and output buffers.

### Systemlogic

Systemlogic.v decodes module-wide input and output registers, and fills some of them with run statistics values, i.e. the system time and the run time (time during which the RunEnable bit was high). Other bits are used to control an I2C bus via register r/w from the Linux C code, and to swap ch.0/1 and ch.2/3 data streams (since the DDR data for each dual ADC is undetermined regarding which clock edge is linked to which channel) It further instantiates two DAC control blocks for the front panel HV control DAC and a serial port included in the GPIO connector.

### Fastchannel

Fastchannel.v instantiates a DAC control block for the offset DAC and packs and unpacks I/O registers. The ADC data stream is reorganized, optionally inverted, and made available to the Linux side through an event register (reading the momentary value of the ADC). As the ADC data comes in faster than (and not synchronized with) Linux reads, there are two output registers – one for receiving fresh data and one for reading by the Linux side – which toggle after every read.

### Constraints

XPP.xdc contains the pin location and input standard constraints, clock definitions and 2 max\_delay constraints to relay timing on non-critical nets. We find it pointless to specify input and output delay constraints.

### Debug

The MicroZed has a JTAG connector that allows debugging with Chipscope (or the Vivado equivalent). This is not used in the current design. The ADC carrier board has 6 test points (T0..T5) which can be assigned with test nets in xpp.v and probed on the board. Further debug nets can be brought out through the GPIO connector.

# Xillinux MicroZed SD Card and Boot Files

The xillinux MicroZed SD card has a small FAT partition with

Devicetree.dtb specifies processor and peripherals for Linux  
 Create dts source with SDK or manual edit, compile with **Zynq** Linux dtc

Boot.bin Zynq boot image with FSBL and uboot  
 Create with SDK using Vivado/ISE generated “hardware platform” and  
 u-boot-xillinux-1.3.elf downloaded from xillinux

Xillydemo.bit PL bitstream created by Vivado

uImage Linux boot image   
 Part of xillinux SD card download

A larger Linux partition contains the Linux file system

# Reconfiguring MIO pins (for PMOD connector)

## Specifications/Definitions

Sandia wants the following functionality

PMOD-1: I2C SDA (I2C used to communicate with about a dozen devices)

PMOD-7: 1-Wire (used for a few DS18B20 temperature probes)

PMOD-2: I2C SCL

PMOD-8: GPIO (currently use to output a non-time-critical pulse)

PMOD-3: GPIO (future use, maybe detect state of a switch)

PMOD-9: Pulse Per Second (ex. see here for linux config \*1\*)

PMOD-4: (Prefer UART TX, will maybe use for GPS, not critical if it introduces difficulties)

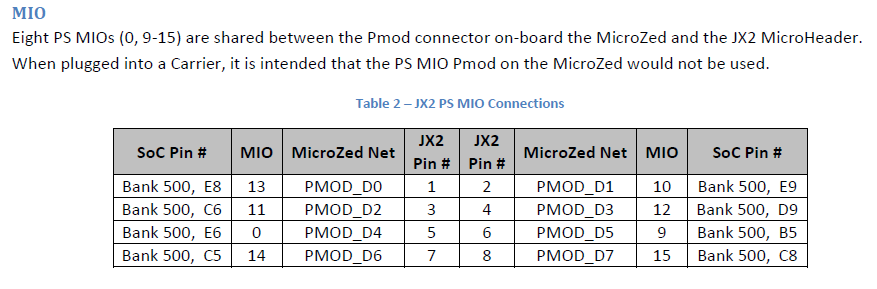
PMOD-10: (Prefer UART RX, will maybe use for GPS, not critical if it introduces difficulties)

PMOD-5, PMOD-11: ground

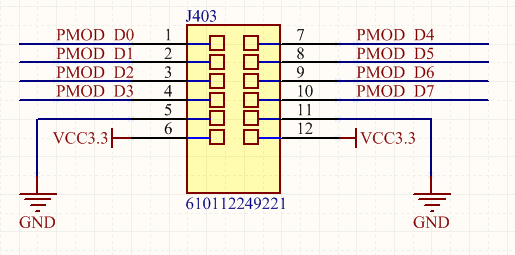
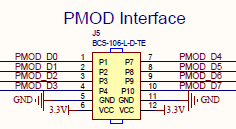
PMOD-6, PMOD-12: +3.3V

\*1\* https://github.com/Xilinx/linux-xlnx/blob/master/Documentation/devicetree/bindings/pps/pps-gpio.txt

MicroZed PMOD pins are mapped to Zynq pins as follows:



And brought out to Pixie-Net’s (left) and MicroZed’s (right) PMOD connector as follows:

So

Desired pinout

SNL function PMOD pin (1-12) MZ Net MIO

I2C SDA 1 D0 13

I2C SCL 2 D1 10

UART TX 4 D3 12

UART RX 10 D7 15

1-wire 7 D4 0

GPIO 8 D5 9

GPIO 3 D2 11

PPS 9 D6 14

Actual pinout

SNL function MIO MZ Net PMOD pin [1-12] Notes

I2C SDA 11 D2 3 I2C0 can use MIO 10/11 or 14/15, I2C1 can use 12/13

I2C SCL 10 D1 2

UART TX 15 D7 10 UART1 can use MIO 10/11 or 14/15

UART RX 14 D6 9

GPIO 0 D4 7 no build-in 1-wire peripheral in Zynq, possibly use SW

I2C interrupt 9 D5 8

GPIO 12 D3 4

GPIO 13 D0 1 no build-in PPS output in Zynq, possibly use SW

## Potentially Useful Forum Posts

http://henryomd.blogspot.com/2014/11/understanding-linux-serial-device.html

not so much: http://www.hivmr.com/db/sp1mc89x91jxffkjcm1zp7acpmakmk81

**http://zedboard.org/content/ps-i2c-xillinux**

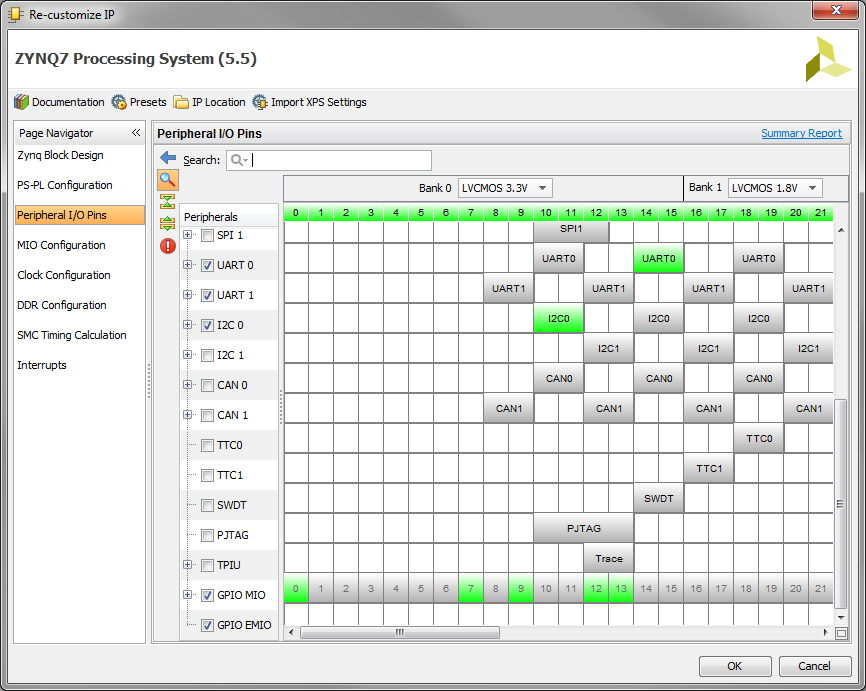
broken -- <http://www.arbot.cz/post/2012/11/14/PS-I2C-in-Xillinux.aspx>

**http://forum.xillybus.com/viewtopic.php?f=4&t=319**

**http://xillybus.com/tutorials/device-tree-zynq-1**

## Vivado:

1. Create Separate Vivado project, without XPP, as described above.   
   May compile with lots of critical warnings due to missing I/O pin configurations.
2. Open xillydemo > xillybus\_ins > system\_i > vivado\_system\_i (the block diagram)
3. Open (double click) block for Zynq processing\_system7 (opens recustomize IP)
4. Under Peripheral I/O Pins:  
   - Enable I2C0 for pins MIO 10/11 and UART0 for MIO 14/15  
   - Enable I2C interrupt, MIO 9
5. Under MIO Configuration:  
   - I/O Peripherals > GPIO > USB Reset: check USB0 Reset and assign MIO7
6. Click ok

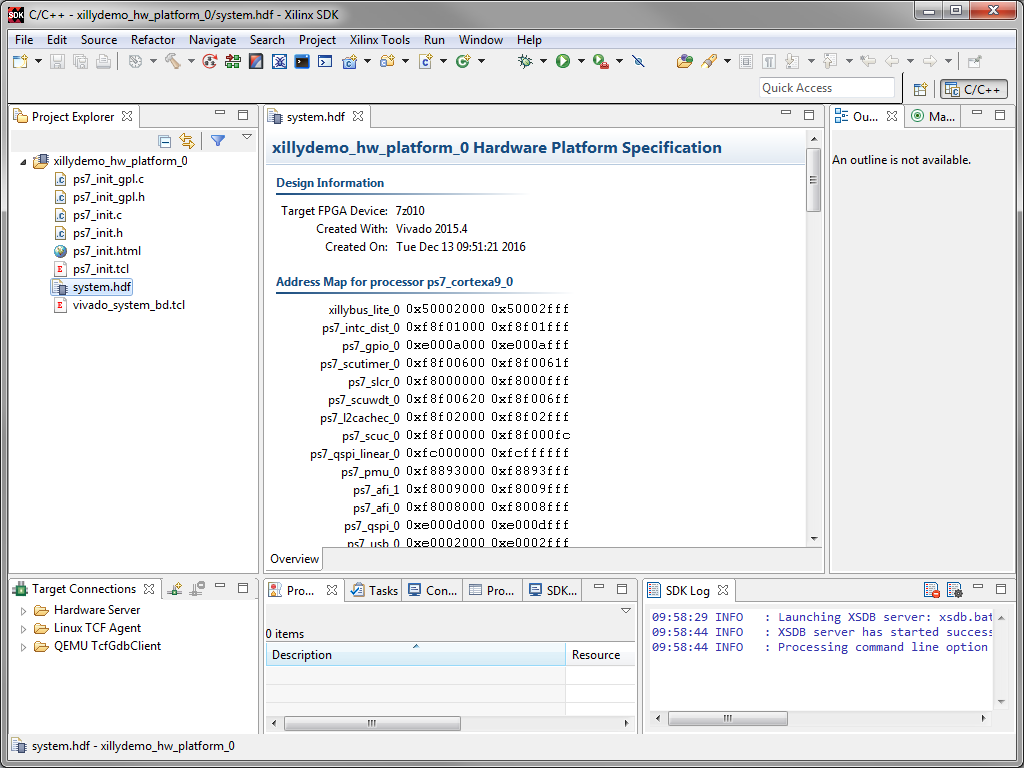


1. Recompile Vivado project (generate bitstream)
2. File > Export > Hardware to a local folder. Do not include bitstream  
   for example, to …/SDK/xillydemo.hdf
3. File > Launch SDK

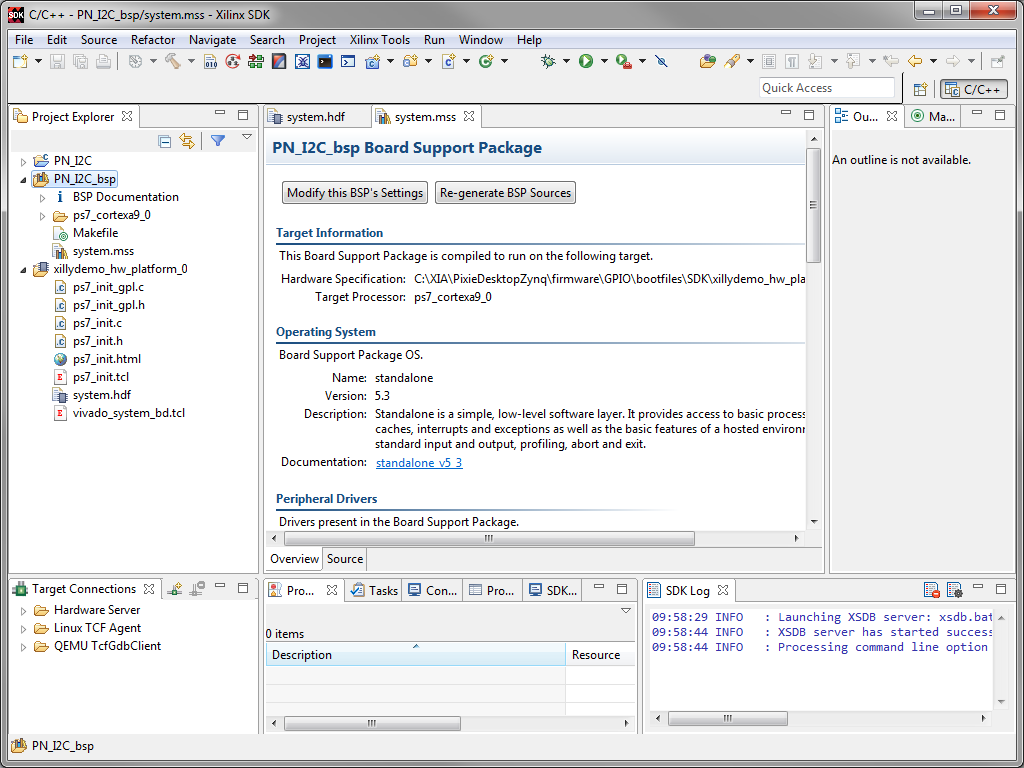
## SDK:

Following the 4 steps in http://forum.xillybus.com/viewtopic.php?f=4&t=319

SDK opens with hardware platform just created by Vivado.



**Step 1/2): Application project**  
- File > New > application project  
- Page 1, specify project name = “PN\_I2C”,   
 OS = “standalone”,   
 hardware platform = “xillydemo\_hw\_platform” as just exported   
 Processor = “ps7\_cortexa9\_0” (not sure if \_1 makes a difference)   
- Next > Page 2, specify template = “Zynq FSBL”  
- Finish   
 => Creates PN\_I2C and PN\_I2C\_bsp



**Step 3/4): Boot Image**  
- right click on PN\_I2C > create boot image.   
- In dialog: new bif file  
 boot image partitions i) PN\_I2C.elf (edit > type: bootloader)  
 ii) add > u-boot-xillinux-1.3.elf (type datafile)  
 create Image   
- creates boot.bin

## Device Tree:

### Edit existing dts file

1. Go to /usr/src/kernels/3.?.?-xillinux-?.?/ dtc compiler below
2. Original dts file in /boot
3. Add entries for I2C0 and UART0
4. Compile to dtb   
   $ scripts/dtc/dtc -I dts -O dtb -o /path/to/my-tree.dtb /path/to/my-tree.dts

Working I2C entry:

i2c0: i2c@e0004000 {

compatible = "xlnx,ps7-i2c-1.00.a"; // calls out a xlnx driver hopefully present in xillinux

reg = <0xE0004000 0x1000>; // memory address and size, fixed by Zynq HW for I2C0

bus-id = <0>;

clocks = <&clkc 38>; // ref to sys clk #38 = I2C clock. Simply <4000000> didn’t work

i2c-clk = <100000>; // 100kHz standard slow I2C

interrupt-parent = <&ps7\_scugic\_0>; // see note

interrupts = <0 25 4>;

#address-cells = <1>; //default

#size-cells = <0>; // default

};

Notes re Interrupt: In other dts files, <1> or <&intc> is often used. This did not work; Linux boot complained about interrupt error. Other xillinux dts entries use the Use <&ps7..> in apparent equivalence to <&intc>, so tried it here also and it worked.

### Create dts file from scratch

Follow <http://www.wiki.xilinx.com/Build+Device+Tree+Blob>

Creates various dts and dtsi files:

1. Pl.dtsi specific from PL interconnect (here: xillybus)
2. Skeleton.dtsi “bare minimum”
3. Zynq-7000.dtsi generic Zynq, all peripherals?
4. System.dts top level, /incuding zynq and pl

### Don’t forget to copy back to /boot

## Linux:

Checking the kernel config file (per /boot/ config-3.12.0-xillinux-1.3):

CONFIG\_I2C=y

CONFIG\_I2C\_BOARDINFO=y

CONFIG\_I2C\_COMPAT=y

CONFIG\_I2C\_CHARDEV=y

CONFIG\_I2C\_MUX=y

So this seems to be already addressing the “final” issue in

<http://zedboard.org/content/ps-i2c-xillinux>

there is an entry in /sys/class/i2c-adapter and /sys/class/i2c-dev, but not in /dev

i2cdetect not found in original xillinux

* Install from Ubuntu: apt-get install i2c-tools
* Also recommended apt-get install libi2c-dev

I2c tools provides I2cset, i2cget, i2cdetect

I2cdetect finds Xilinx I2C. i2cget toggles MZ PMOD pins 2 (clk) and 3 (data)

## To Do:

Check UART