Using the Zybo's XADC in Hardware

Introduction

ADC's are handy for using analog signals in digital contexts, such as FPGAs. However, if you tried to use the XADC IP block in Vivado, you noticed it automatically added it as an AXI output, disallowing you from connecting it to the Zybo's FPGA. This guide will allow you to fix that.

Background

First, just where is that sneaky little ADC? The ADC is connected to the JA PMOD header, where there are a grand total of 2 ADC's connected via MUXes to all the data pins, and to internal temperature and voltage sensors. The pins themselves are grouped in pairs, and the ADC's can be used to measure differential voltages, but that is beyond the scope of this tutorial. Also important to note is that the ADC input range is stated to be 0 to 1V in the manual, so be prepared to either use an external chip with low output voltage or use additional circuitry to shift the output voltage around. How that happens is up to you. Additionally, the output of the ADC is stated to be 12-bits, however the IP actually outputs it as a 16-bit number, filling the top 4 bits with zeros. How you handle that is up to you.

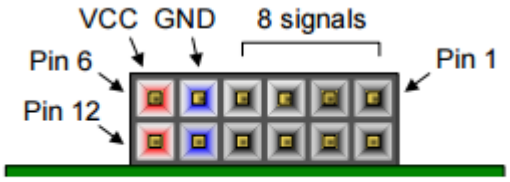

Figure 16. Pmod diagram.

Now, because life is pain, there are several different designations for each pin. There is the PMOD pin number, the internal pin designation, and how XADC refers to the pin. The pins are connected thusly:

Pin Number	Pin Designation	XADC Pin Designation
JA1	N15	AD14_P
JA2	L14	AD7_P
JA3	K16	AD15_P
JA4	K14	AD6_P
JA7	N16	AD14_N
JA8	L15	AD7_N
JA9	J16	AD15_N
JA10	J14	AD6_N

Notice that pins that share a column(JA1 and JA7 for example) are the positive and negative pairs. If you are using a single chip on the PMOD, you don't need to worry about it. The ADC outputs the analog value as a the difference between the P and N pairs. For example, if you plug in a chip into AD14_P, but leave AD14_N open, it reads the difference between the pins, namely whatever your input is and 0V. This tutorial assume you only use the P inputs, if you need to use two chips on the ADC, its up to you to figure that out.

Usage

Now lets get down to the nitty gritty.

1. Set up your basic block design, adding the Zynq processor to the block design, whole shebang. Be sure to enable FCLK_0. Add a clock port (here called D_CLK) as an 1 bit output from the processor, connect it to the FCLK_0 output on the processor block. Generate the HDL wrapper, we'll futz with that later.

2. Add the XADC from the IP catalog into your project. If you right-clicked the block design and added the IP there, you did it wrong. On the left hand side of the window, under Project Management, you should see a button that says IP Catalog, click it, search for “xadc”. Click “Customize IP”, NOT “Add to Block Design”.

3. You should now be at the window called “XADC Wizard”, this allows you to customize the IP block to your liking. You can change the component name as you like, go wild. We'll go through this section by section.

Under Basic:

> Make sure DRP is selected under Interface, Single Channel under Startup Channel, deselect reset_in under Control/Status Ports, and Continuous Mode under Timing Mode.

Under ADC Setup:

> Leave it as-is, unless you have to use the VP/VN MUX or have an external MUX. You're on your own for that. Under Alarms:
>
> XADC is how you can monitor all sorts of neat internal temperature and voltage sensors. Deselect all of 'em, you should see a lot of ports drop off the little block design windows.

Under Single Channel:

> Select whichever pin you want to measure. For example, if you wanted to read pin JA1, it is designated AD14_P. Look for VAUXP14 VAUXN14.

When you finish customizing it, it will ask you if you want to synthesize output products for the black now or later. Chose later, but it doesn't particularly matter if you synthesize it now.

Congratulations, you now have an ADC! Now to instantiate it.

3. In the sources pane, where you have all your Verilog and wrappers and all that jazz, click IP sources, you should your brand new xadc module. Open it up, there should be a folder called “Instantiation Template”. Open up the .veo file, it will have a handy “CUT HERE” comment for you. Copy it into

your wrapper.

4. Now you have your instantiation in the wrapper, what do all those input and outputs mean? Unlike if you chose to add your XADC to the block design, you have to pull the data out of the registers manually. If you have just one channel, its fairly simple, and what we'll do in the rest of the tutorial. If you need more channels, you can create a Verilog wrapper for your XADC instantiation to handle that.

daddr_in: 7-bit wide input: This is the address you want to read/write from. There are separate addresses for each input, including the internal sensors. You can dig through the manual for the internal sensors, but the external pin's addresses follow the pattern h10 through h1F. For example, if you wanted the address for AD14, JA1 and JA7, you'd want address h1E. JA2 and JA8, with AD7, are in address h17. To fill the whole 7 bits, insert it as “7'h1E” in the wrapper (Tip: you can use constants for instantiations in the wrapper).

dclk_in: 1-bit input: clock input. Put D_CLK or whatever you named your clock output.

den_in: 1-bit input: The enable input. Wire the output from eoc_out (more on that later) to this input to get it to operate it continuously. If you want to control when the ADC reads, wire it to something else.

di_in: 16-bit input: Data input. If you really want to, you can change whats in the data registers. Set it 0, we don't need to do that.

dwe_in: 1-bit input: write enable in. If you really want to futz with the data registers, use this to do it. We don't, set it 0.

busy_out: 1-bit output: If you want to tell when the ADC is busy, you can use this wire. We don't care, so we set it 0.

vauxp* and vauxn*: 1-bit inputs: these are the actual inputs from the pins. They will change with whatever you selected the input as. For JA1 and JA7, they will be vauxp14 and vauxn14.

vp_in and vn_in: 1-bit inputs: Dedicated analog input pair for differential analog input. We don't care, set it 0.

alarm_out: 1-bit output: Tells you if there is an alarm. Since we disabled all the alarms, we don't care, set it 0.

do_out: 16-bit output: The actual data output. Since the actual output of the ADC is 12-bits wide, it fills the top 4 bits of the actual output with 0's, handle that accordingly.

eoc_out: 1-bit output: end of conversion signal, goes high when the ADC finishes converting an analog input. Wire this into de_in for continuous operation.

eos_out: 1-bit output: End of sequence signal, goes high when data from the channel sequencer is finished being written. Since we don't use the channel sequencer, set it 0.

channel_out: 4-bit output: channel selector output. Since we are using single channel, we don't care, set it 0.

drdy_out: 1-bit output: tells you when data is ready on the do_out bus. Wire it up to something, you'll need it if you want to get valid data from the register.

4. Comment D_CLK from the data direction, processor declaration, and processor instantiation in the wrapper. Add whatever wire you're using for the ADC input to those things, labeling it as an output. Add whatever intermediate wires you need.

5. In the constraints file (add it if you haven't already), search for “XADC”, the section header should say “PMOD Header JA (XADC)”. Replace the “ja_n[whatever]” of your selected pin with the name of your ADC input wire. For example, I named the input wire Vaux14p, and I want to replace ja_n[0] with that on both lines. Uncomment them. The lines should now read:

set_property PACKAGE_PIN N16 [get_ports {Vaux14p}]

set_property IOSTANDARD LVCMOS33 [get_ports {Vaux14p}]

And you're done! You should now be able to read analog inputs using the ADC, and use them FPGA-land!

References:

Zybo reference manual: http://www.xilinx.com/support/documentation/university/XUP%20Boards/XUPZYBO/documentation/ZYBO_RM_B_V6.pdf

XADC Reference Manual: http://www.xilinx.com/support/documentation/user_guides/ug480_7Series_XADC.pdf