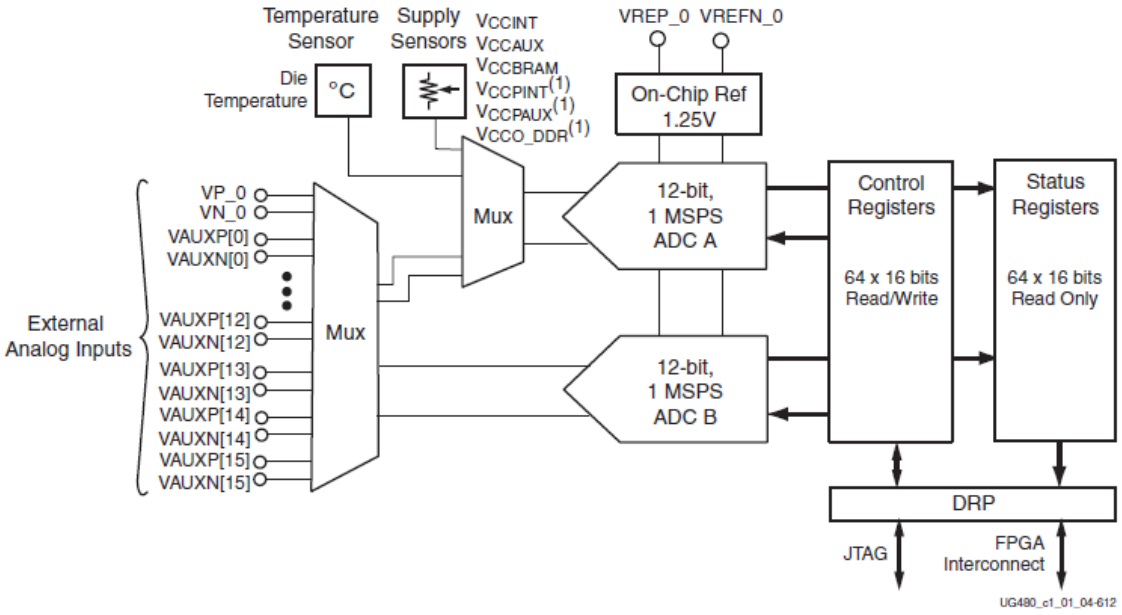
Niels Smidth
CPE439
Spring 2015

Getting Started with the Zybo's XADC

Introduction:

ADC's are useful for sampling all kinds of analog signals. The ZYNQ has the ability to use its logic or DSP capabilities to perform filtering or other processing on the signals sampled likely much faster than a standard microcontroller. Some applications of this are performing DSP on sampled audio or implementing a digital oscilloscope.

This is a document to help you get started using the ZYNQ XC7Z010's onboard ADC, the "XADC". This tutorial was performed on the Zybo board using Vivado 2014.4, using FreeRTOS in the Processing System (PS).



XADC Info:

Table 1: **Zynq-7000 All Programmable SoC *(Cont'd)***

	Zynq-7000 All Programmable SoC							
	Device Name	**Z-7010**	**Z-7015**	**Z-7020**	**Z-7030**	**Z-7035**	**Z-7045**	**Z-7100**
	Part Number	**XC7Z010**	**XC7Z015**	**XC7Z020**	**XC7Z030**	**XC7Z035**	**XC7Z045**	**XC7Z100**
Programmable Logic	Xilinx 7 Series Programmable Logic Equivalent	Artix®-7 FPGA	Artix-7 FPGA	Artix-7 FPGA	Kintex®-7 FPGA	Kintex-7 FPGA	Kintex-7 FPGA	Kintex-7 FPGA
	Programmable Logic Cells (Approximate ASIC Gates)(3)	28K Logic Cells (~430K)	74K Logic Cells (~1.1M)	85K Logic Cells (~1.3M)	125K Logic Cells (~1.9M)	275K Logic Cells (~4.1M)	350K Logic Cells (~5.2M)	444K Logic Cells (~6.6M)
	Look-Up Tables (LUTs)	17,600	46,200	53,200	78,600	171,900	218,600	277,400
	Flip-Flops	35,200	92,400	106,400	157,200	343,800	437,200	554,800
	Extensible Block RAM (# 36 Kb Blocks)	240 KB (60)	380 KB (95)	560 KB (140)	1,060 KB (265)	2,000 KB (500)	2,180 KB (545)	3,020 KB (755)
	Programmable DSP Slices (18x25 MACCs)	80	160	220	400	900	900	2,020
	Peak DSP Performance (Symmetric FIR)	100 GMACs	200 GMACs	276 GMACs	593 GMACs	1,334 GMACs	1,334 GMACs	2,622 GMACs
	PCI Express® (Root Complex or Endpoint)	—	Gen2 x4	—	Gen2 x4	Gen2 x8	Gen2 x8	Gen2 x8
	Analog Mixed Signal (AMS) / XADC	2x 12 bit, MSPS ADCs with up to 17 Differential Inputs						
	Security(2)	AES and SHA 256b for Boot Code and Programmable Logic Configuration, Decryption, and Authentication						

Figure 1: Xilinx DS190 Zynq Overview

The ZYNQ's overview states that the Zynq contains two 12-Bit, 1MSPS ADCs and is capable of operating with up to 17 differential inputs. On the Zybo board, the JA PMOD Connector gives access to some of these inputs. Figure 2 below shows the connection of the PMOD connector to the ZYNQ's pins :

Pmod JA (XADC)
JA1: N15
JA2: L14
JA3: K16
JA4: K14
JA7: N16
JA8: L15
JA9: J16
JA10: J14

Table 9. Pmod pinout.

Figure 16. Pmod diagram.

Figure 2: PMOD Connections from ZYBO Ref Manual

Because these ADC inputs are differential, they are grouped into pairs : Pins 1&7, 2&8, 3&9, and 4&10. These pins are connected to the ZYNQ on the ZYBO board through a partially complete anti-alias filter, shown below in Figure 3. The capacitors are not installed on the board, so if this anti-aliasing behavior is desired, they will have to be soldered on, or it might be easier to just make an external network (because they picked very inconvenient soldering locations for those caps).

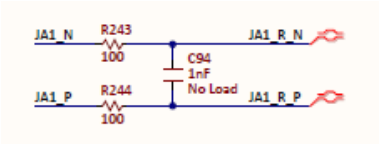


Figure 3: Incomplete Anti-Alias Filters from p10 of ZYBO Schematic

These inputs can be run in Unipolar or Bipolar mode, check P31 of UH480 : XADC User guide for more info on exactly what this means. Don't put any signals smaller than 0V or larger than 1.8V (voltage supply level to XADC) into either of the ADC pins, or you'll risk damage to the XADC. If you want to stick in a signal centered at zero (ex : audio from mp3 player), you'll need to add external circuitry to bias it at 0.5V.

Tutorial:

- In this tutorial, we're going to be setting up the XADC to be read via AXI using the XADC wizard Xilinx IP. I'll be using the AUX14 input to the ADC. I'd suggest before you get started, to read some of the following resources to get to know the XADC you'll be playing with :
    - XADC User GuideUG480
    - LogiCORE IP AXI XADC Product Guide PG019
    - ZYNQ Preliminary Product Specification DS190 (the XADC section)
    - ZYNQ Technical Reference Manual UG585 (the XADC section)
    - Zybo Manual and Schematic (how is XADC connected?)
- This tutorial assumes you've got a fresh new project with only a ZYNQ block added (though I'm sure it would work fine with other stuff too.
    - My ZYNQ block has M AXI GP0 interface, Timer 0, Watchdog, and FCLK_CLK0 enabled and had block automation run. See Figure 4 below for my block diagram's starting point.

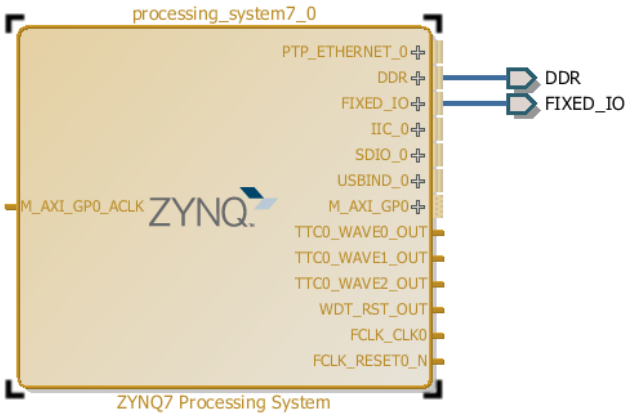


Figure 4: Starting Block Design

- Now that we have our basic processor block, we want to add the actual XADC. Right click on some empty space in the block diagram and enter "Add IP…". Select the "XADC Wizard" as shown below:

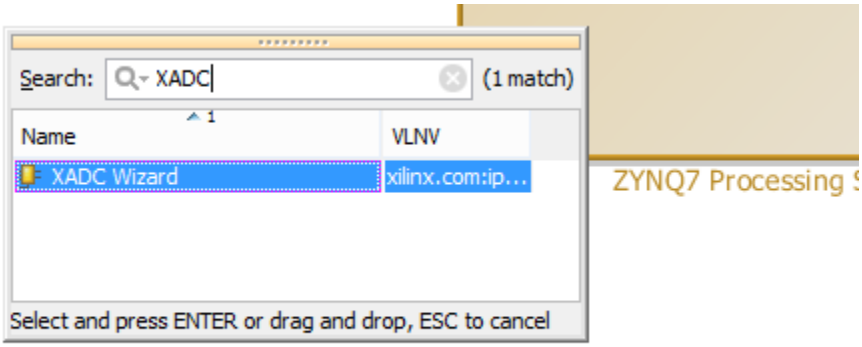


Figure 5: Adding XADC Wizard

- You've now added your XADC Wizard IP block. Double click on it to change its settings to better suit our uses.
    - Under the Basic tab:
        - We want to read via AXI, so select "AXI4Lite"
        - We only need one channel for this demo, so select "Single Channel"
        - We want the XADC to continuously read, so keep it in "Continuous Mode"
        - Don't mess with the clocks for now. I've got the DCLK at 100MHz, ADC conversion rate at 1000KSPS, and acquisition time at 4 CLKs

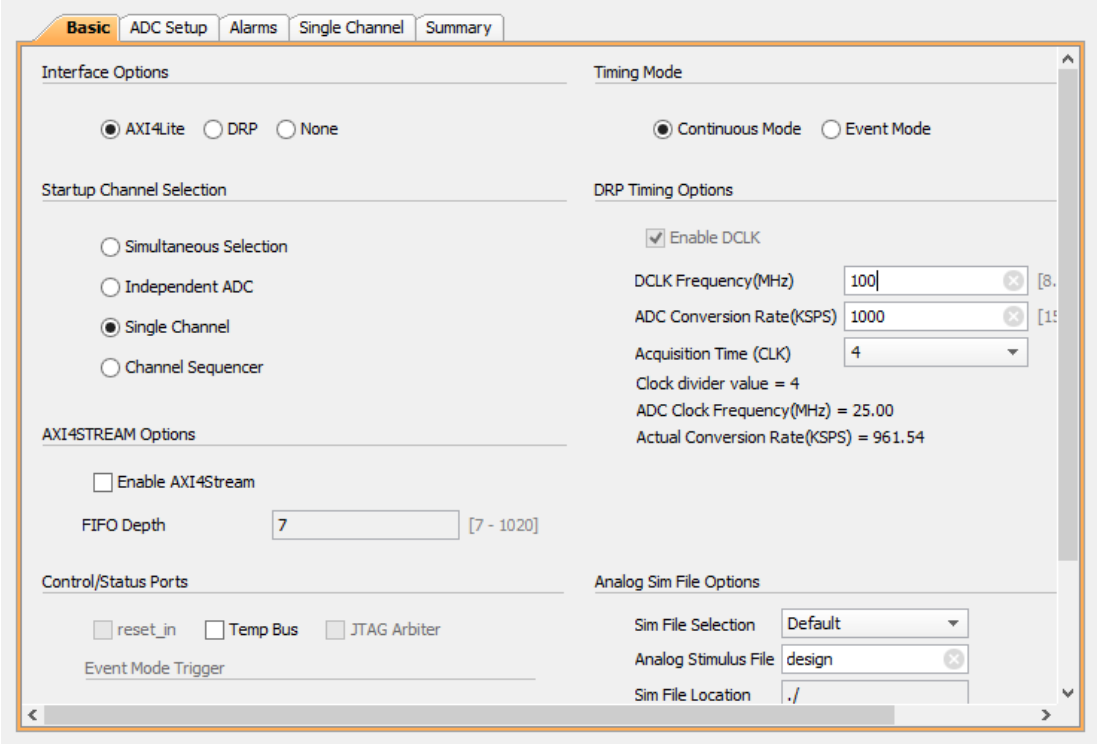


Figure 6: XADC Wizard Setup Basic Tab

- Under the "ADC Setup" tab, I've not touched anything. I keep Channel Averaging to 0 because I want my samples fast. I don't know enough about the calibrations to know how important they are, I will ignore it for now to get you started. You should probably read about that to see if it is good or not for your application.
- Under the "Alarms" tab, I turned off all alarms for this application. You can use these to make sure your various voltage rails or on chip temperatures are within limits. But we don't need that for this simple tutorial.
- Under the "Single Channel" tab, you'll have the option to select if you want Bipolar mode (where Unipolar is default, this is what I used). This will depend on your application, I'll leave it up to you to find that out. You do want to select your desired channel, which is "VAUXP14 VAUXN14".

- Your XADC IP block / summary should look something like Figure 7

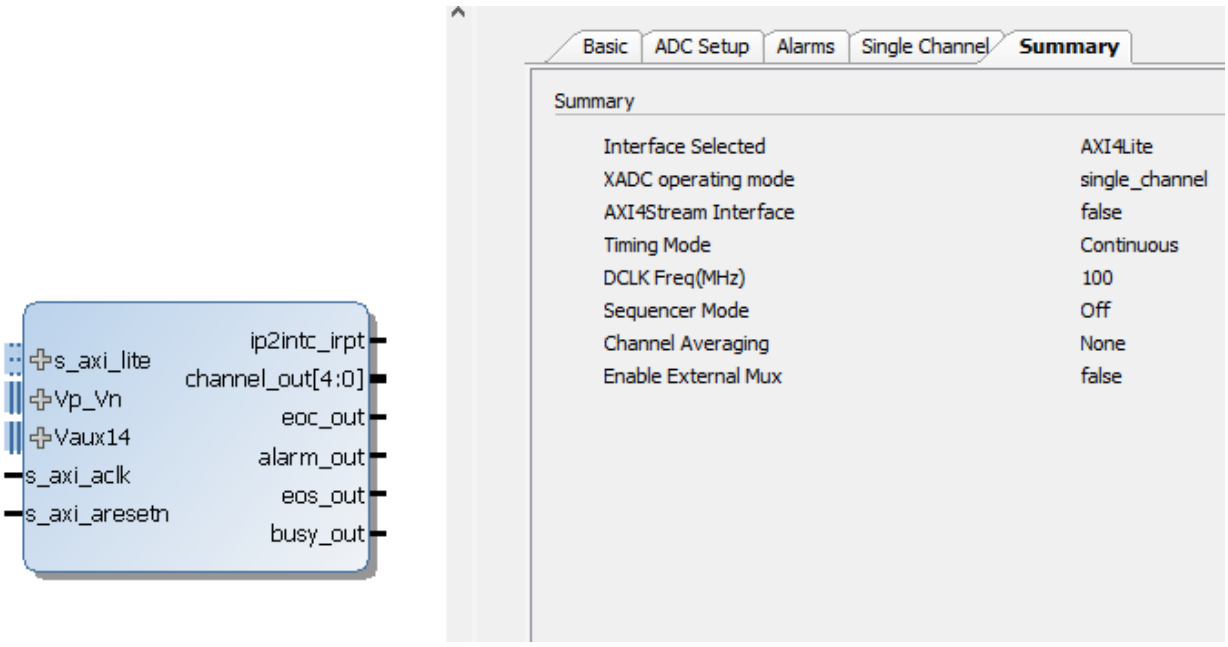


Figure 7: Complete XADC Wizard

- You might be tempted now to click the tempting "Run Connection Automation" box now to automatically hook up an AXI interface. This is what I tried first and for some strange reason, never could get it to work this way (likely because the Connection Automation also adds a "Processor System Reset" block).

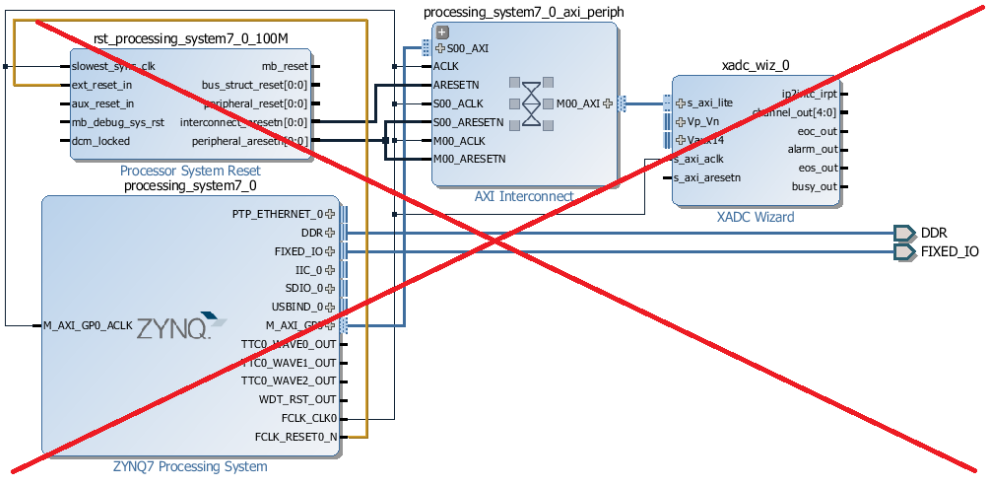


Figure 8: Wrong Hookup Procedure

- From my reading of other tutorials on the internet (namely this one), I tried deleting the “Processor System Reset” block and hooking up the RST and remaining CLK wires by hand. This did end up working for me, so I’d suggest you try it this way first. You’ll also want to right click on the “Vaux14” port of the XADC Wizard and make it an external input.

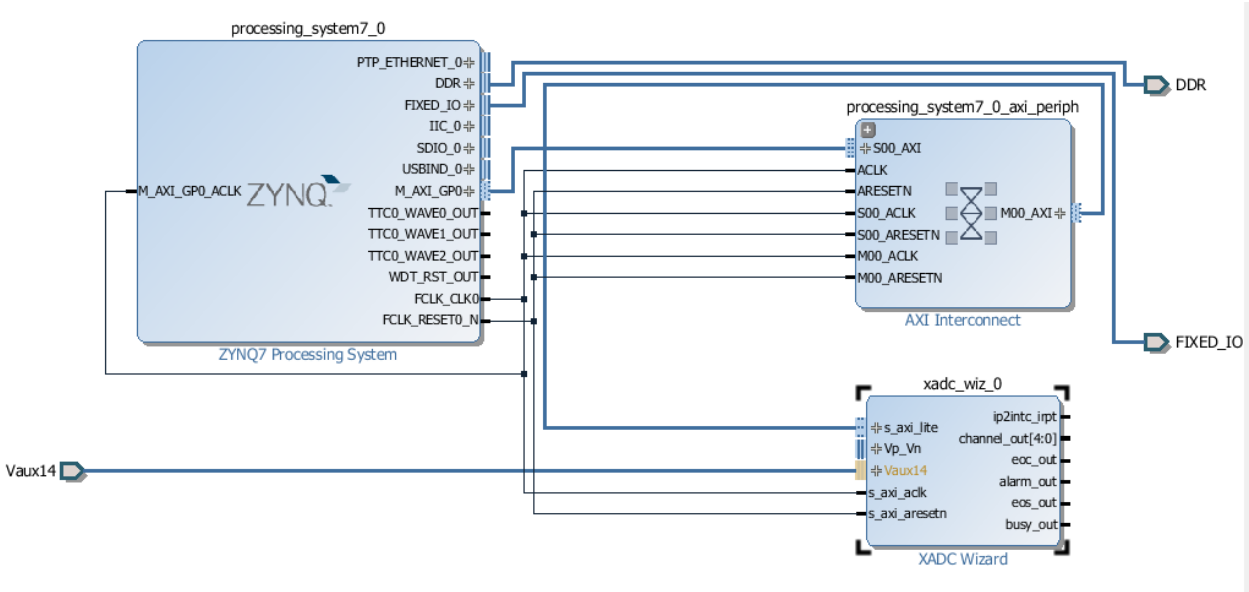


Figure 9: Correct Hookup

- Now that you’ve got your block diagram set up, you’re all ready to generate your HDL wrapper, Copying generated wrapper to alow user edits as usual. I do receive the following two warnings :

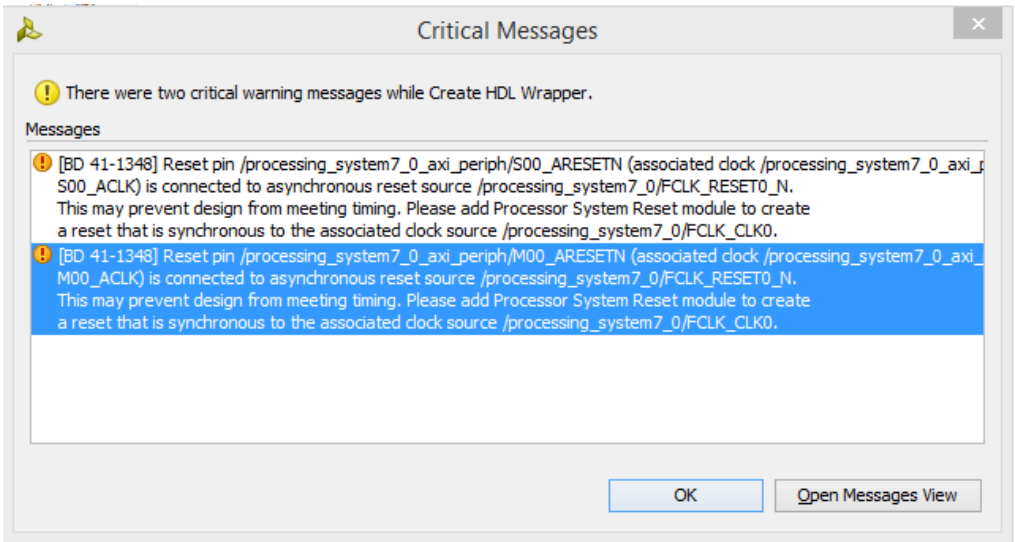


Figure 10: Warning Generated from HDL Wrapper Creation

I’ve yet to have a problem yet and haven’t succesfully used the System Reset module, so for now I’ve ignored it. But unlike me, you good students should go back and figure out how to make this warning not appear.

- With your wrapper generated you need to connect your Vaux14 input to the outside world through your constraints file. Go to the “Sources” tab, and right click -> “Add Sources”

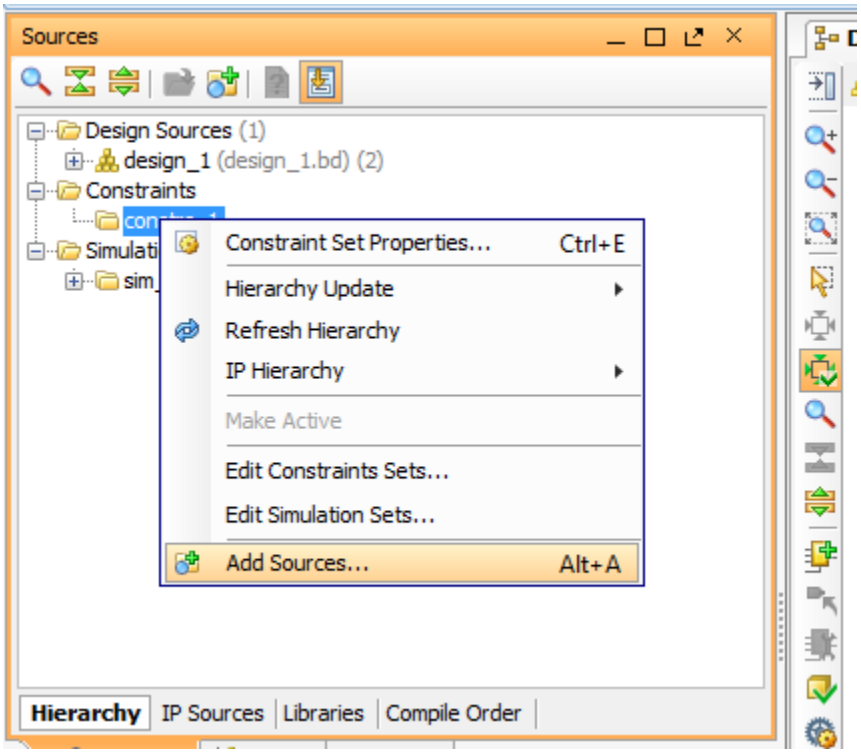


Figure 11: Adding Constraint File

You’ll want to “Add or Create Constraints”. Add the “ZYBO_Master.xdc” file you’ve used for the previous tutorials (which you’ve hopefully copied into the project folder). Now under the constraints folder, open and edit this file.

- This file links the internal signals to the ouside world, so we need to make sure our AUX14 signals we made external are hooked up to the right pins.
- Scroll down to Line 163. Youll see these lines control the connection to the PMOD Header JA (which is the XADC specified connector). Uncomment the set_property commands controlling AD14n and AD14P.
- You now need to connect these to the right internal signal. If you check your wrapper, you’ll see the following signals specified :

```
input Vaux14_v_n;
input Vaux14_v_p;
```

Figure 12: Definition of AUX14 Signals in Wrapper

- Now you know the names of the signals to connect. Modify your constraints file to be read as shown below :

```
163 ##Pmod Header JA (XADC)
164 ##IO_L21N_T3_DQS_AD14N_35
165 set_property PACKAGE_PIN N16 [get_ports {Vaux14_v_n}]
166 set_property IOSTANDARD LVCMOS33 [get_ports {Vaux14_v_n}]
167
168 ##IO_L21P_T3_DQS_AD14P_35
169 set_property PACKAGE_PIN N15 [get_ports {Vaux14_v_p}]
170 set_property IOSTANDARD LVCMOS33 [get_ports {Vaux14_v_p}]
171
172 ##IO_L22N_T3_AD7N_35
```

Figure 13: Correct Constraint of AUX14 Pins

- Make sure you save your .xdc file to save your changes.

- Now we've configured our IP blocks, wrapped them in HDL, and connected it to our PMOD connector. Run the bitstream generation and go make some tea.
- Once that is complete, export your hardware with bitstream and launch the SDK.
- In SDK, you'll want to generate your Board Support Package for the design wrapper you just generated. If you explore the system.mss of the BSP, you'll hopefully see the "xadc_wiz_0" driver. This indicates that your BSP saw that you used the XADC wizard and you'll have access to the commands necessary to read from the XADC (these are used below in example code).

ps7_usb_0	usbps	Documentation	Import Examples
ps7_wdt_0	wdtps	Documentation	Import Examples
ps7_xadc_0	xadcps	Documentation	Import Examples
xadc_wiz_0	sysmon	Documentation	Import Examples

Figure 14: Note "xadc_wiz_0" driver

- I am using the same imported FreeRTOS setup that you've used for your previous labs. I won't even use tasks for this tutorial because we're keeping it nice and basic. Using the commands that can be found in the xadc_wiz_0 documentation, the following simple program is written :

```
static XSysMon SysMonInst; //a sysmon instance
#define SYSMON_DEVICE_ID XPAR_XADC_WIZ_0_DEVICE_ID //ID of xadc_wiz_0
#define XSM_CH_AUX_14 30

int main( void )
{
	XSysMon_Config *SysMonConfigPtr;
	XSysMon *SysMonInstPtr = &SysMonInst;
	int xStatus;
	u16 TestData;
```

```
	/* Configure the hardware ready to run the demo. */
	prvSetupHardware();

	// Prepare the Config Pointer
	SysMonConfigPtr = XSysMon_LookupConfig(SYSMON_DEVICE_ID);
	if (SysMonConfigPtr == NULL) printf("LookupConfig FAILURE\n\r");

	// Set up the XADC
	xStatus = XSysMon_CfgInitialize(SysMonInstPtr, SysMonConfigPtr,
SysMonConfigPtr->BaseAddress);
	if(xStatus != XST_SUCCESS) printf("CfgInitialize FAILED\r\n");

	while(1)
	{
		// Wait until XADC is finished with a conversion
		while ((XSysMon_GetStatus(SysMonInstPtr) & XSM_SR_EOC_MASK) !=
XSM_SR_EOC_MASK);

		TestData = XSysMon_GetAdcData(SysMonInstPtr,XSM_CH_AUX_14); //Read the
external Vaux14 Data

	}

	/* Don't expect to reach here. */
	return 0;
}
   /*-----------------------------------------------------------*/
```

The AUX14 channel is defined to be 30 in xsysmon.h :

```
#define XSM_CH_VCCPDRO          0x0F /**< On-chip PS VCCPDRO Channel, Zynq */
#define XSM_CH_AUX_MIN          16   /**< Channel number for 1st Aux Channel */
#define XSM_CH_AUX_MAX          31   /**< Channel number for Last Aux channel */
#define XSM_CH_VUSR0            32  /**< VUSER0 Supply - UltraScale */
```

Since the last AUX channel is AUX15, we know that AUX14 is 30.

I've set a breakpoint on the TestData line so that I can take a new ADC sample each time I press resume. The XADC is a 12 bit ADC, and uses the top 3 bytes of the 4 byte register to hold the data. So when my TestData reads "0x0118", you really are reading the ADC value "0x011".

Now you've got an ADC that works. Play with it some more to get a good feel for the commands that can be used.

If you are wanting to read XADC values from hardware (VHDL that processes your ADC values), this is not the best way to do things. There is another option for communication with the XADC, through the “DRP” (Dynamic Reconfiguration Port). This is a lower level interface allowing writing and reading to the XADC’s internal registers (more like what you may be used to be doing on a microcontroller).

The XADC manual has all the timing info you need to get going on it. For an example of it running in DRP mode (this little trick is covered in the XADC Wizard manual), configure your XADC in DRP mode (double click on its block to make this change). It should now look something like this :

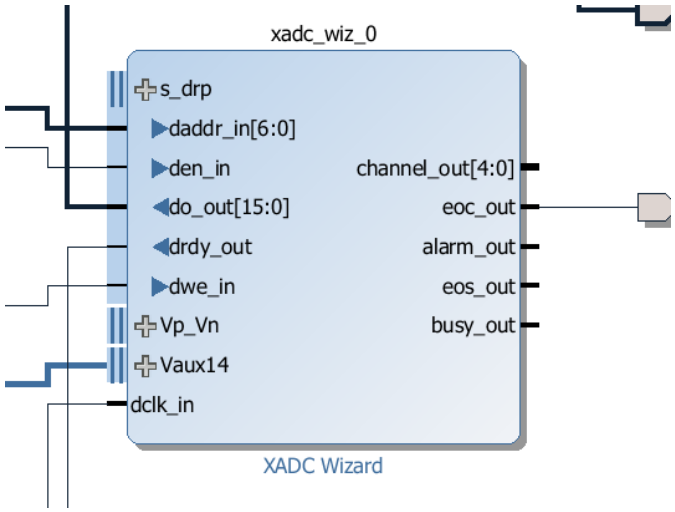


Figure 15: XADC Wizard in DRP Mode

Now the XADC is read to be run by DRP. Now you’ll want to run the following command in the TCL console. Note that in the sources tab, we can see this device’s name is “design_1_xadc_wiz_0_0”.

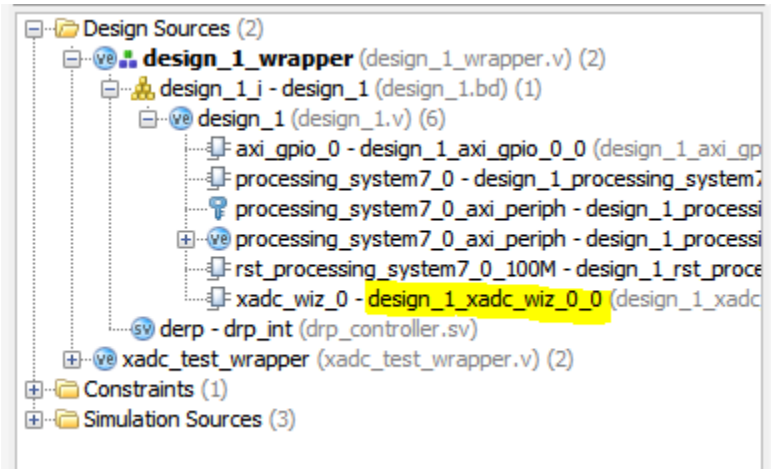


Figure 16: Showing name of XADC Wizard

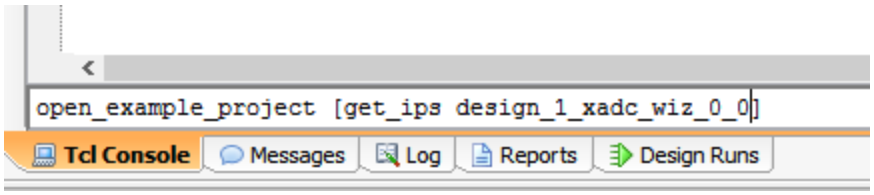


Figure 17: Command to Open Example XADC DRP Project

A new project should open up. It will contain a test bench in the "Simulation Sources" folder. I had to right click on the XADC_wiz_inst instantiation in the sources tab and "Generate Output Products" to get rid of some errors. You can modify the analog input simulation file to the XADC in the XADC configuration window:

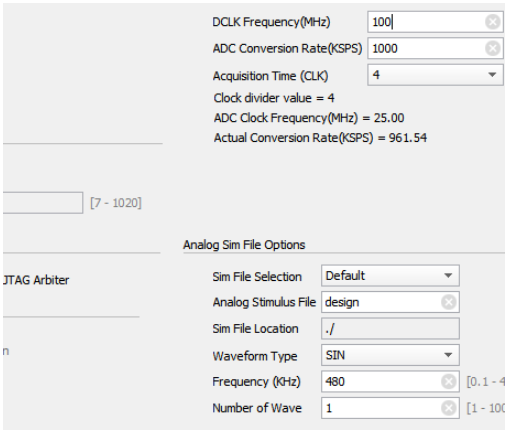


Figure 18: Modifying Analog Stimulus to SIN wave

You might want to "Generate Output Products" again after that is complete if it doesn't prompt you to (which generates the stimulus file), I had some problems when I didn't. Now you can simulate your XADC to see its natural behavior. I expanded on this testbench to make my own "wrapper" for the XADC too!

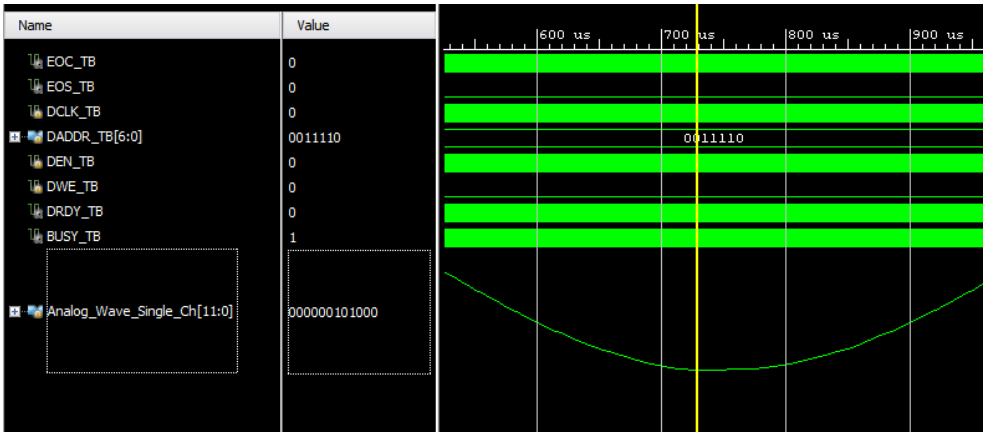


Figure 19: Simulated ADC, showing sampled SIN wave stimulus