

### \*\*\* MICROBLAZE USERS READ ME \*\*\*

This method is very difficult to implement if you are using microblaze in your design. This is because microblaze requires a MIG component to operate, however, the MIG component instantiates the XADC IP by default and there is only one analog to digital converter on the Nexys Video board. Thus, the only way to use XADC and microblaze is to use the MIG wizard to set up the MIG component by hand and disable the XADC inside the MIG while doing so. This is extremely confusing to do on your own and I could not find any documentation on how to do it. If you do manage to figure this out, you'll need to visit the link below for more details on how to use the XADC to supply the MIG with temperature readings. <https://www.xilinx.com/support/answers/51687.html>

I do not believe microblaze has a problem with digital GPIO ports so an external ADC can probably still be used to supply digital signals through the digital GPIO ports (you could also use the XADC IP on another board then wire the two boards together).

### **How to use single channel XADC IP in VHDL**

This write-up will discuss how to use the XADC IP to convert a single analog input into a digital signal for VHDL to use. The XADC IP has more options that are not explained here but are described in the two videos linked below. This write-up is based primarily on the first video but translated from Verilog to VHDL at the end.

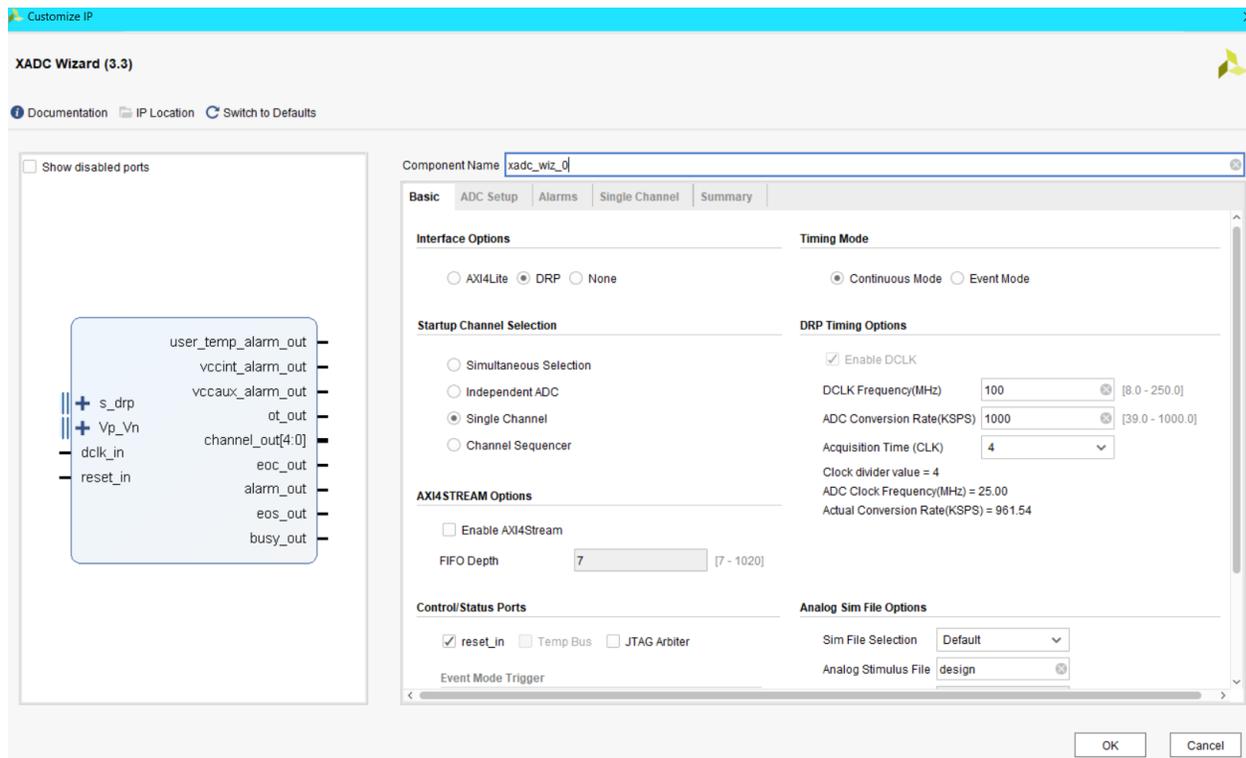
<https://www.youtube.com/watch?v=TFrwwk0VYjc&t=2100s>

<https://www.youtube.com/watch?v=2j4UHLYqBDI>

IP instantiation will be covered first, then how to wire the ports, then how to create/use testbench, then finally how to connect the sensor to the Nexys board.

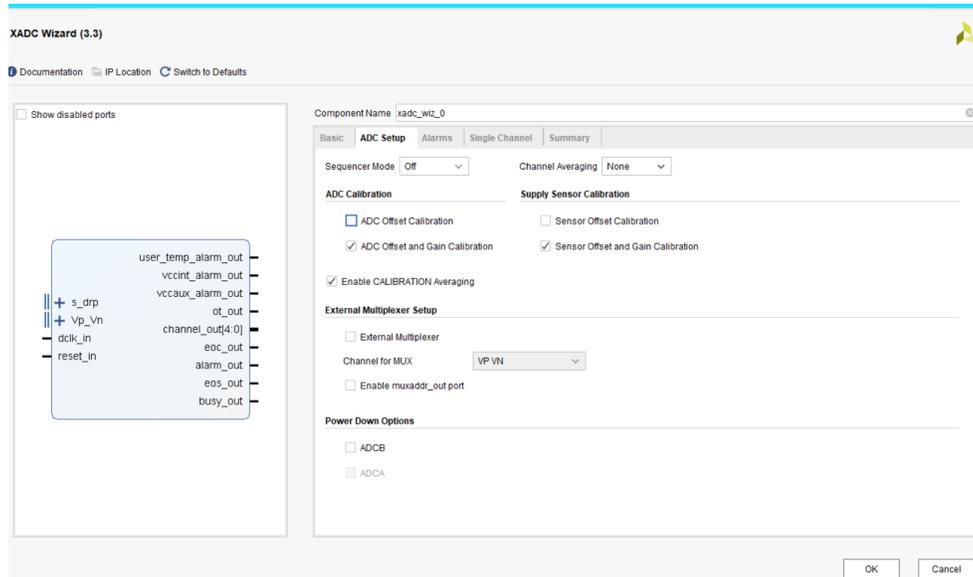
## IP Instantiation Tutorial

- 1) Go to IP catalogue, search for XADC and double click the XADC wiz.
- 2) The wiz should open to the 'Basic' tab as shown below.
  - a) Interface Options: We will use DRP for simplicity. AXI4Lite can be used with microblaze (if you can figure out the problem above).
  - b) Startup Channel Selection: We will be using a single channel for this design. The channel sequencer can also be used if you need more than one analog signal converted. You may need to do additional research to use the sequencer as it will not be totally discussed here.
  - c) Timing Mode: We will use continuous so the signal is always being updated. Event Mode will only cause a signal to be generated if an event triggers the XADC.
  - d) Everything else should be left to default for this design.



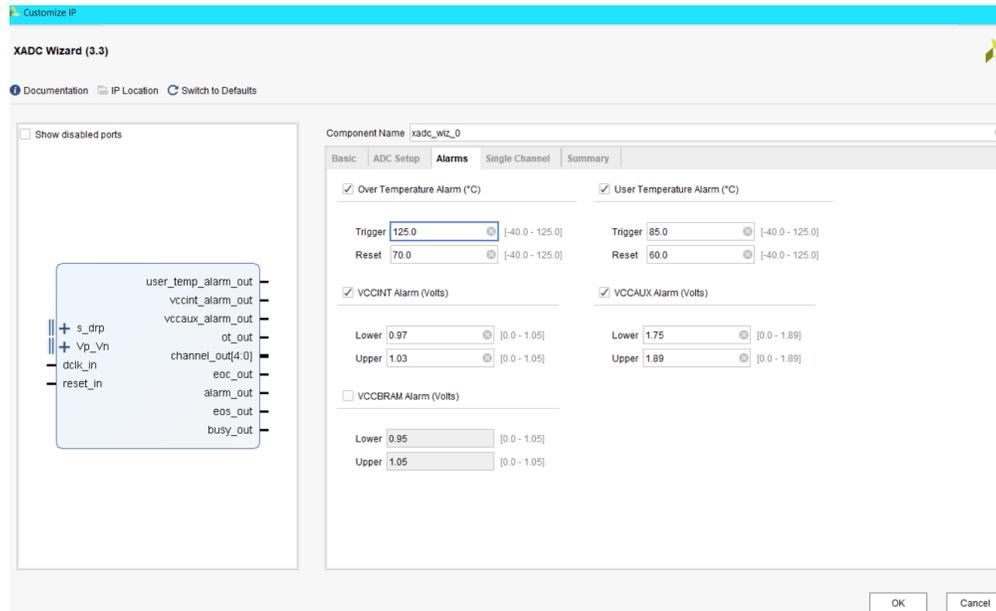
### 3) ADC Setup tab

- a) Sequencer Mode: Only used for channel sequencer
- b) Channel Averaging: Averages the signal so there are less extreme outliers. Isn't required for this design but can be used.
- c) Leave everything else set to default values.



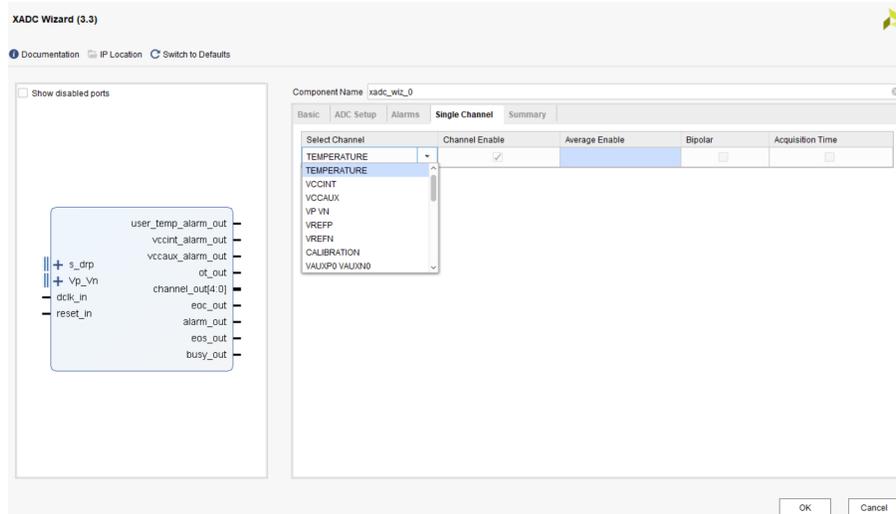
### 4) Alarms Tab

- a) For this design we will **uncheck** all alarms. If you need their values you may need to utilize the channel sequencing option in the Basic tab.



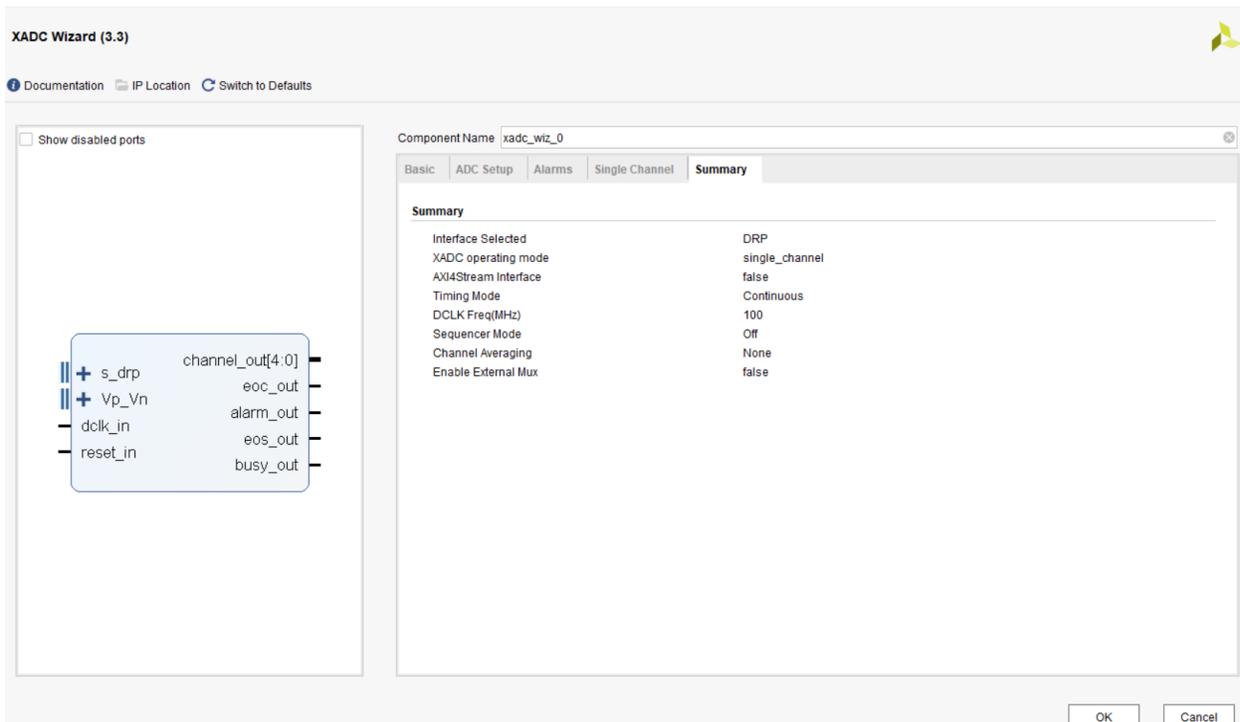
## 5) Single Channel Tab

- Select Channel:** We will use the 'VP VN' pin option found in the drop down menu. If using a channel sequencer, more rows will be available and you can set them to the VAUXP/N pins.
- Channel Enable:** Only toggleable for the channel sequencing mode. Can be used to disable specific channels.
- Bipolar and Acquisition time** disabled for this design. Reference XADC documentation in top left of wizard as shown below for their uses.



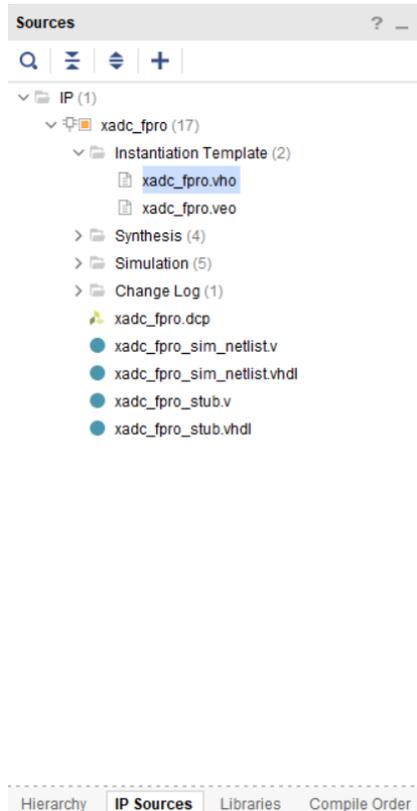
## 6) Summary Tab

- My design summary is shown below. Press OK to complete the design.



## How to wire XADC Ports

- 1) The component and instantiation declarations can be copy/pasted from Sources  
-> IP Sources -> IP -> XADC -> Instantiation Template -> xadc.vho.
  - a) Note: I named my instantiation “xadc\_fpro” as seen in the image below



- 2) I created an XADC entity as shown below (left) so I could simply supply a clk, reset, and voltage then read the data coming out. (This XADC\_TEST file is provided along with my testbench)
  - a) Looking at the code to the right, signals were created for all of the ports not declared in the entity port map.
  - b) Note: eoc is connected to den\_in. EOC goes high when the converted value is ready to be read so I connected this to the read enable so the data is read as soon as it is ready. Also, dwe\_in is set to 0 because we are not writing anything to the XADC in this design. Finally, daddr\_in allows you to select which channel in the XADC you read the data from. This signal has channel\_out appended to the end so that it always reads from the last channel that was written to. If using a channel sequencer, this may need to be changed so you can manually select the channel.

```

entity XADC_TEST is
  Port (
    do_out : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
    dclk_in : IN STD_LOGIC;
    reset : IN STD_LOGIC;
    vp_in : IN STD_LOGIC;
    vn_in : IN STD_LOGIC);
end XADC_TEST;

```

```

architecture Behavioral of XADC_TEST is

```

```

  COMPONENT xadc_fpro

```

```

  PORT (
    di_in : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
    daddr_in : IN STD_LOGIC_VECTOR(6 DOWNTO 0);
    den_in : IN STD_LOGIC;
    dwe_in : IN STD_LOGIC;
    drdy_out : OUT STD_LOGIC;
    do_out : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
    dclk_in : IN STD_LOGIC;
    reset_in : IN STD_LOGIC;
    vp_in : IN STD_LOGIC;
    vn_in : IN STD_LOGIC;
    channel_out : OUT STD_LOGIC_VECTOR(4 DOWNTO 0);
    eoc_out : OUT STD_LOGIC;
    alarm_out : OUT STD_LOGIC;
    eos_out : OUT STD_LOGIC;
    busy_out : OUT STD_LOGIC
  );
END COMPONENT;

```

```

SIGNAL di_in : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL daddr_in : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL drdy_out : STD_LOGIC;
SIGNAL channel_out : STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL eoc_out : STD_LOGIC;
SIGNAL alarm_out : STD_LOGIC;
SIGNAL eos_out : STD_LOGIC;
SIGNAL busy_out : STD_LOGIC;

```

```

begin

```

```

  daddr_in <= "00" & channel_out;

```

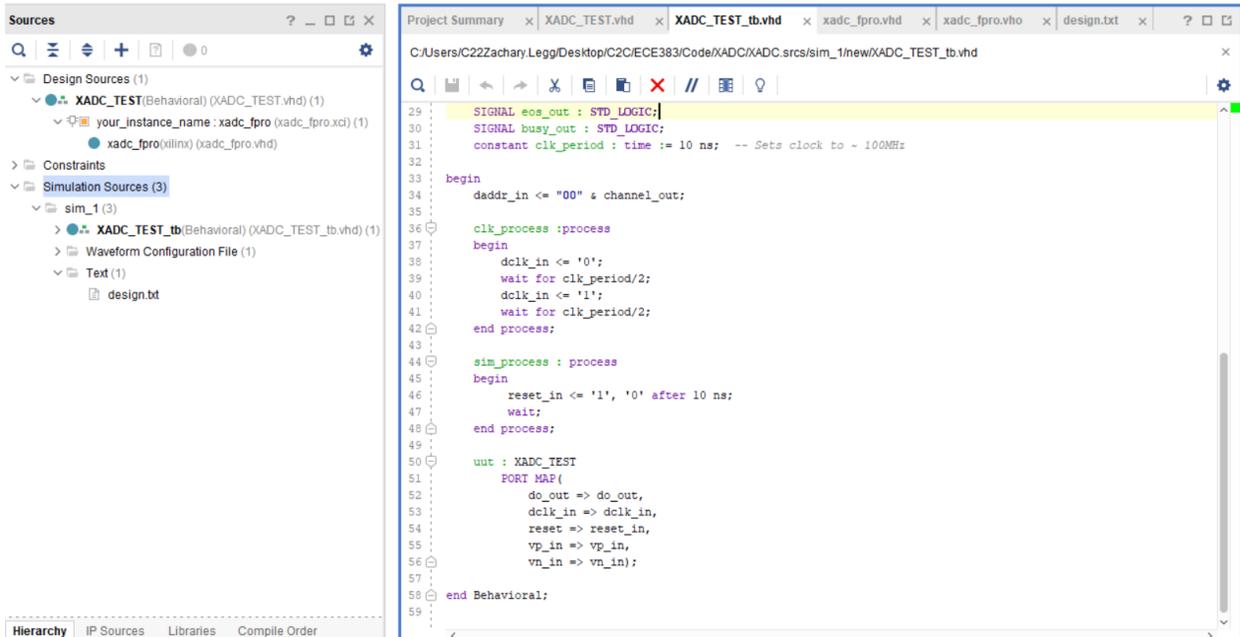
```

  your_instance_name : xadc_fpro
  PORT MAP (
    di_in => x"0000",
    daddr_in => daddr_in,
    den_in => eoc_out,
    dwe_in => '0',
    drdy_out => drdy_out,
    do_out => do_out,
    dclk_in => dclk_in,
    reset_in => reset,
    vp_in => vp_in,
    vn_in => vn_in,
    channel_out => channel_out,
    eoc_out => eoc_out,
    alarm_out => alarm_out,
    eos_out => eos_out,
    busy_out => busy_out
  );

```

## How to create XADC Test Bench

- 1) Declare your XADC entity and any extra signals as usual. (My entity is XADC\_TEST)
- 2) Create a 100MHz clock and a process to start the tb with reset high. This can be seen below.



```
29 SIGNAL eos_out : STD_LOGIC;
30 SIGNAL busy_out : STD_LOGIC;
31 constant clk_period : time := 10 ns; -- Sets clock to ~ 100MHz
32
33 begin
34     daddr_in <= "00" & channel_out;
35
36     clk_process : process
37     begin
38         dclk_in <= '0';
39         wait for clk_period/2;
40         dclk_in <= '1';
41         wait for clk_period/2;
42     end process;
43
44     sim_process : process
45     begin
46         reset_in <= '1', '0' after 10 ns;
47         wait;
48     end process;
49
50     uut : XADC_TEST
51     PORT MAP(
52         do_out => do_out,
53         dclk_in => dclk_in,
54         reset => reset_in,
55         vp_in => vp_in,
56         vn_in => vn_in);
57
58 end Behavioral;
```

- 3) Now you will need to add "design.txt". To do this, select Add Sources -> Add or Create Sim Sources -> Add File -> (Navigate to your design, mine is simply XADC) -> (your design name).ip\_user\_files -> mem\_init\_files -> design.txt
  - a) The testbench will automatically read test values from the design.txt file if you left the Analog\_Sim\_File\_Options set to default in the Basic tab of the XADC wiz.
  - b) The values in the design.txt file can be changed but be careful to change the correct values since the organization of the file is a bit weird.
- 4) Run the simulation and make sure you're displaying the values straight from the XADC on the waveform. The VP/VN values from the design.txt file will not be displayed here but their converted values should be displayed as 16 bit hex values coming out of do\_out.

## How to connect your analog sensor

**Important:** The VP VN ports used in this design only accept up to 1V so make sure your sensor either outputs a max of 1V or you use a voltage divider to reduce the output to a max of 1V. VAUX ports (circle 21 below) may have different tolerances but make sure to read the board's documentation (linked below) before trying higher voltages. <https://reference.digilentinc.com/reference/programmable-logic/nexys-video/reference-manual>

I powered my sensor with the 3.3V VCC and ground ports found in the digital GPIO blocks (denoted below by the 4th circle). My sensor outputs a max of 3.3V so I used a voltage divider to reduce this to 1V max. I then connected VN to ground and VP to the voltage divider. VP and VN pins are denoted below by the 16th circle. My final project routed the VP/VN ports all the way to the top VHDL file and did not need the VP/VN pins to be enabled in the constraint file. This may just be a convenient bug but I'm not sure. I believe the VAUX pins need to be uncommented in the constraints file if you're using them for a channel sequencer.

