Notes about handling data from the A/D converter

rev 5/13/21

Power connections to the A/D converter are zero and +Vdd volts. Vdd is used as the reference voltage and thus the range of analog voltages that can be measured is zero to Vdd. However, we desire to sample an AC voltage which swings plus/minus about what we generally consider to be "zero" volts. To work with our converter an AC "zero" voltage of 1/2 Vdd is created with a voltage divider and capacitive coupling of the input signal to the opamp circuit ahead of the A/D converter on the "AC" input of the A/D board. If you use the "DC" input you will need to add 1/2 Vdd DC to the AC signal (which is easily done if you are creating a test signal using the Digilent Discovery unit's Wavegen function) to keep the voltage greater than zero. In either case, the unsigned value produced by the A/D converter will then be about 2048_{10} if the AC signal has a magnitude of zero, 4095_{10} if the AC signal is at Vdd/2, and 0 if the AC signal is at -Vdd/2.

(Note: The resistors used to create the 1/2 Vdd offset for the AC input are high valued and thus may create a long time constant for input to achieve 1/2 Vdd. If that is a problem use the DC input and add DC offset to the AC in the signal generator setup)

The FIR algorithm assumes that we are multiplying and adding signed numbers. Thus the unsigned 12-bit value created by the A/D converter must be changed to a signed number. And we need 2's complement format. To do that subtract 2048 from the A/D value. Recall that subtraction can be implemented by adding -2048 to the A/D value assuming 2's complement format. In binary, 2048 is 100000000000 which is -2048 for a 12-bit number. Thus add 2048 to the raw A/D value.

Using IEEE standard VHDL, the steps you need to properly do arithmetic for the FIR filter circuit follows. (Reference: section 3.5.4 on pages 60 to 64 of the text). Note, in the comments below that signals considered to be signed numbers have names starting with val.

1) At the top of your VHDL source file include the IEEE numeric library:

```
use ieee.numeric_std.all;
```

2) Define signals that will be numeric and used to hold the incoming data and signals that are intermediate results:

```
signal adc_12bit_vector : std_logic_vector(11 downto 0);  - - data coming from the A/D
signal val_adc_raw : signed(11 downto 0);           - - A/D data converted to signed type
signal val_adc_no_offset : signed(11 downto 0);       - - data with offset removed
```

3) Use statements that will create a signed number and remove the offset thereby creating a value that can then be written to RAM. Note that conversion functions *signed* and *to_signed* are being used. In *to-signed*, the first parameter is an integer to be converted and the second parameter specifies the bit length of the resulting binary integer.

```
val_adc_raw <=  signed(adc_12bit_vector);
val_adc_no_offset <= val_adc_raw + to_signed(2048,12);
```

When doing addition and multiplication you should use signals that have been defined as signed or unsigned rather than std_logic_vector. For example, assume you have library and signals defined as follows:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
signal sigA, sigB, sigC : std_logic_vector(11 downto 0);
```

If you try to use this statement:

```
sigC <= sigA + sigB;
```

or this one:

```
sigC <= sigA + 1;
```

a Synthesis error will occur. You cannot add two std_logic signals.
Now you say, I googled and found that if the following library set is used than I can add std_logic_vectors:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

signal sigA, sigB, sigC : std_logic_vector(11 downto 0);
sigC <= sigA + sigB;
        or
sigC <= sigA + 1;
```

BUT, only if just unsigned or just signed numbers are used in your design, not both.

It turns out that IEEE.STD_LOGIC_ARITH.ALL is not an IEEE standard library but rather is a non-official extension of IEEE.STD_LOGIC_1164. A problem occurs if you need to have both signed and unsigned signal vectors because of the overloading used.

For this lab the official IEEE.NUMERIC_STD.ALL library must be used along with proper data type conversion functions and overloading listed in tables 3.7 and 3.8 on page 62-63 of the textbook By doing so you will explicitly state the type conversions you wish to occur and will likely be more able to track down numeric problems when they occur.

The IEEE_STD_LOGIC_ARITH_ALL and IEEE_STD_LOGIC_UNSIGNED.ALL libraries shall not be used for this lab.