Notes regarding FPGA circuit debugging
With some particulars regarding an FIR circuit

There is no particular order to the comments below.

1) Pay attention to not just error messages when synthesizing and building a circuit (errors have to be fixed) but also to warnings. I have experienced situations where a warning about some aspect of a circuit is a cause of the circuit not working. In general, you don't want any warnings but if you retain warnings you need to determine what they mean and decide that they will not affect circuit operation. (see item 2 below)

2) The signed output of the multiplier is 24 bits (12 x 2). The accumulator input is 29 bits. You must sign extend the multiplier's 24 to 29. This can be done with the resize function

```
multiplier_extended <= resize(multiplier_out, 29);
```

where both signal vectors here are signed data type.

3) When using a ROM, if in the ROM initialization data there is a particular bit that is zero in all words stored in the ROM (such as when you want one coefficient to have a value of one, i.e. x"001", and all the rest to be zero), output flip-flop(s) for bits that are zero in all words will be removed and likely other downstream logic. That will cause the circuit to be different than you intended and hence not work. Not good.

4) The example ROM avoids the "all zeros" problem by having 4 sets of coefficients:
   a - The first word x"001" and the remaining 31 words x"000"
   b - All 32 words x"001"
   c - A low pass filter coefficient set
   d - The first word x"FFF" and 31 words are x"000" This guarantees that no bit position is all zeros

   The ROM thus has 128 12-bit words but can be viewed as 4 pages of memory with 32 words (coefficients) per page or one set of coefficients per page. The ROM address is then 7 bits long where bits 4 downto 0 select one of 32 coefficients and the upper bits 6 and 5 select one of the 4 pages. Then I used two switches, for example sw1 and sw0, to drive address bits 6 and 5 respectively. This really helps debugging because you can quickly select different coefficient sets or ultimately you can have multiple filter definitions and switch between them.

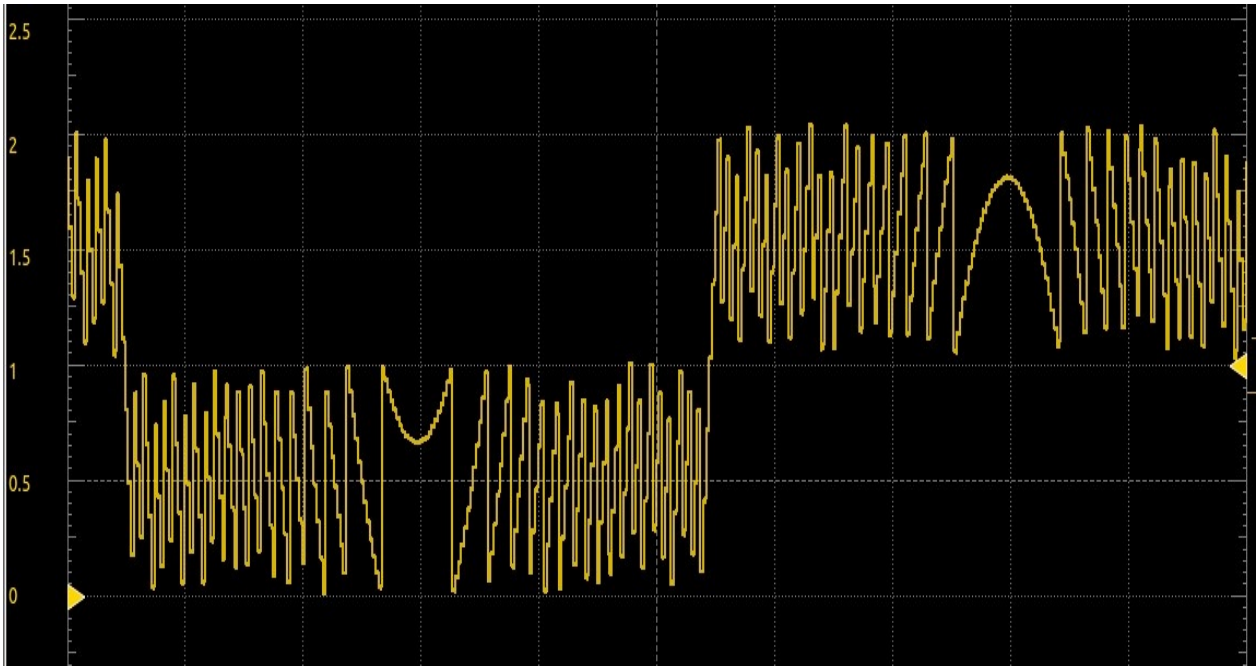
5) When testing with 32 coefficients all set to x"001" note that this is in fact a low pass filter. This filter averages 32 data samples. Now if those 32 samples come from exactly one cycle of a sine wave, the average (integral) over one cycle is zero. For example, if sample rate is 20Khz and one cycle is 32 samples long, that frequency is 20Khz/32 = 625 Hz. If you input a 625 Hz sine wave and observe the DAC output it is flat line with little or no AC component. This "filter" has removed that 625 Hz signal.

6) Note that if you use the DC input of the ADC and look at the DC output of the DAC, any DC component in the input signal will appear in the output scaled by the ratio of maximum

output voltage of the DAC (2.048 volts) to maximum input voltage of the ADC (~3.3V) or 2.048/3.3 (Vout = 0.621 x Vin).

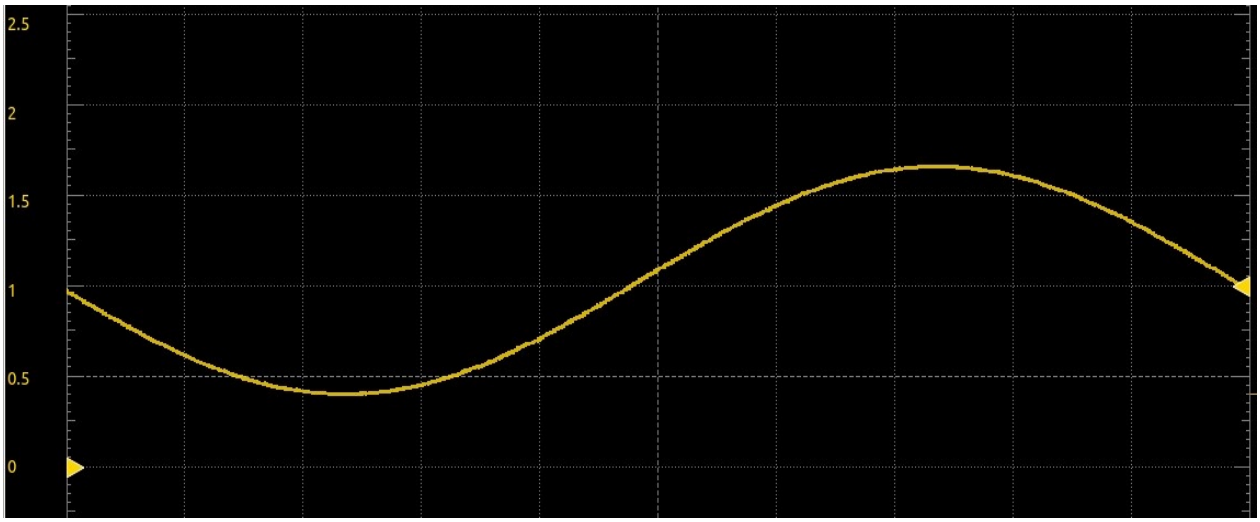
7) If using the DC input terminal to the ADC, made sure you have a DC component with the AC signal going + and - around that. When generating an AC signal, use a DC offset value of 1.62V

8) It is hard to guess what the output magnitude of the accumulator will be. It depends on the magnitude of the coefficients as well as the incoming waveform. And if the magnitude of the accumulated result becomes greater than the 12-bits chosen out of the possible 29 then you will see weird waveforms at the output of the DAC. For example, below is the output of the DAC for an input to the ADC of a 50Hz, 1 Vp-p, 1.92Vdc offset with 32 coefficients all with a value of one and the input data to the DAC taken as the lower 11 bits of the accumulator and the MSB bit 28 of the accumulator (the sign bit). Note that there is symmetry around a 1volt level which is correct, i.e. 2.048v/2 is the AC zero of the output. (time scale is about 20 ms from left edge to right edge or one cycle of 50hz)



If however accumulator bits 15 downto 5 are taken as the output with again accumulator bit 28 used as the MSB then the waveform from DAC is a sinewave (see next page)

Choosing various ranges of output bits will produce a variety of output waveform shapes. If the 11 highest bits (plus sign) of the accumulator output are chosen for output the accumulated value might be small enough so that all of the 11 highest bits are zero and thus the DAC output will be a straight line.



9) The bits used for output can be selected using the toggle switches. For example to choose one of four ranges you could use a circuit description like this:

```
acc_subset  <=  (acc_reg(28) & acc_reg(10 downto 0)) when (sw76="00") else
                (acc_reg(28) & acc_reg(15 downto 5)) when (sw76="01") else
                (acc_reg(28) & acc_reg(20 downto 10)) when (sw76="10") else
                (acc_reg(28) & acc_reg(25 downto 15));
```

where

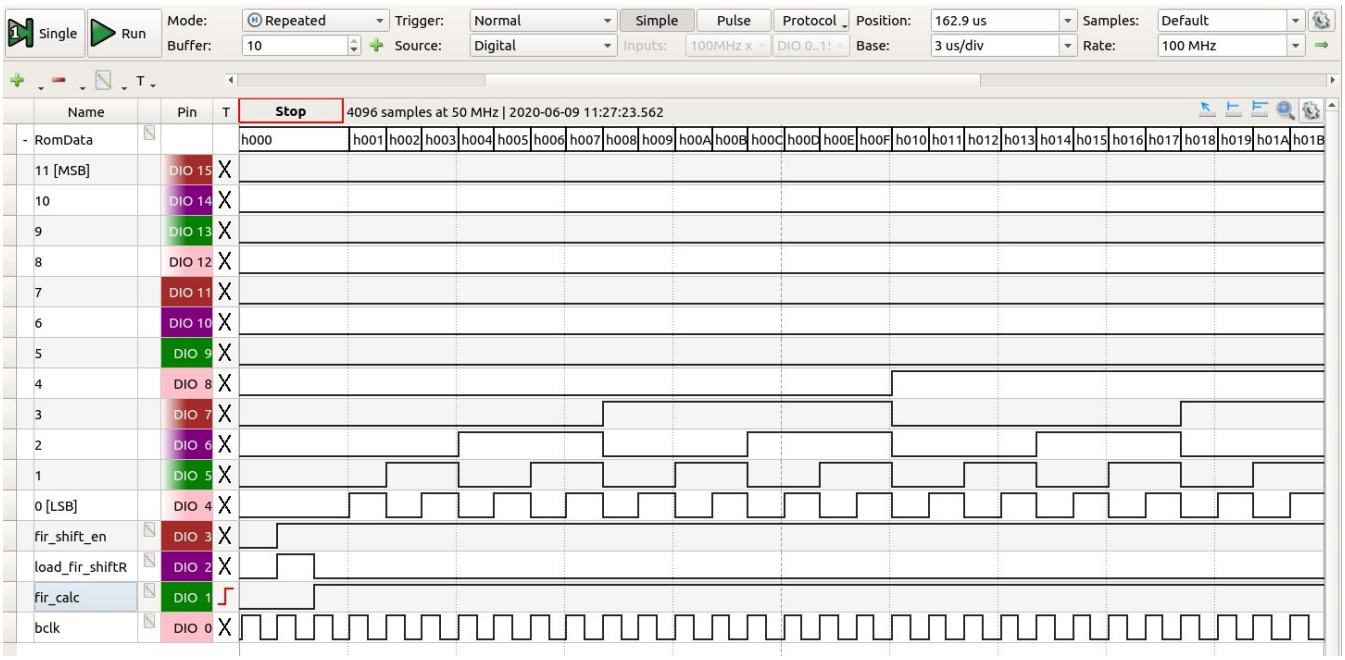
```
signal acc_subset : signed(11 downto 0);       – 12 bit subset of the accumulator
signal acc_reg : signed(28 downto 0);          – the accumulator register
signal sw76 : std_logic_vector(1 downto 0); – sw7 and sw6 used as a selector
```

10) Since we are dealing with both signed and unsigned numbers in this design use only the IEEE.NUMERIC_STD.ALL library for numeric data types and then the appropriate type conversion functions such as signed, unsigned, to_signed, std_logic_vector, etc. Use the to_signed(X,Y) function for using a constant where X is a numeric value such as 2047 and Y is the number of bits that Y should be encoded in, such as 12.

11) For debugging data such as numbers coming out of the data memory (RAM), or ROM, or the ADC, etc. send out up to 16 bits to the TEK1 and TEK2 ports on the FPGA board to which you have connected the 16 digital inputs to the Digilent Discovery module. Then in the Waveforms software use the logic analyzer mode to display the data. If you select key control signals to display along with the data it can help with debugging.

    As an example, with the ROM coefficients filled with sequential numbers 0 to 31 and 12 bits of ROM data output plus significant control signals such as the clock, a signal that asserts when calculation starts and the ROM coefficient data should sequence out to the multiplier, capturing all of that can help confirm operation. Note that the ROM circuit is registered, i.e. the data goes through an edge triggered register and thus ROM output is

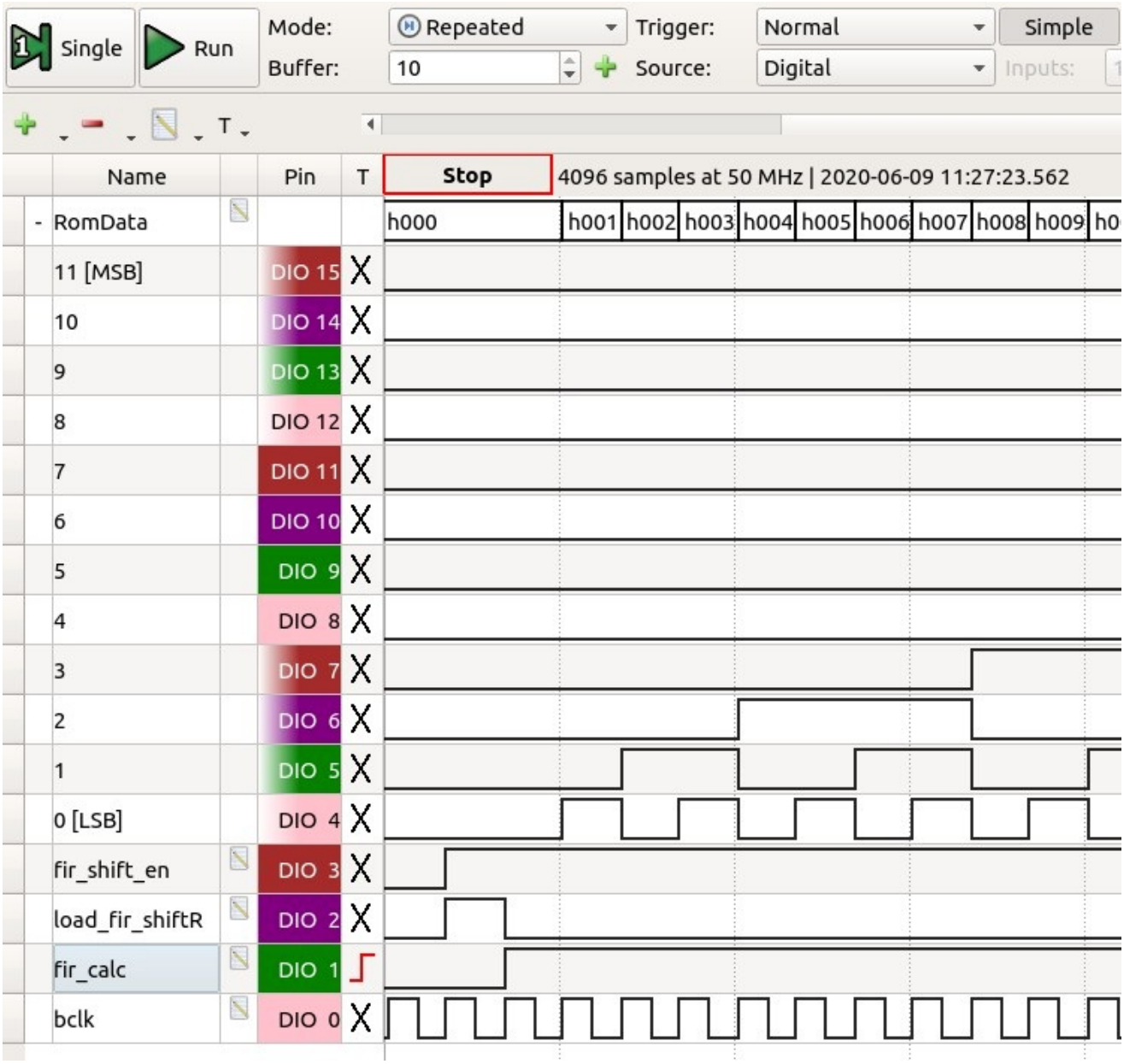
updated one clock cycle after a new address is presented to the ROM. Thus it is useful to check that data is coming out of the ROM at the proper time. Here is an example waveform:



This is hard to read. On the next page is shown only the first part. Note that inputs DIO4 up to DIO15 were defined as a bus and thus the top line of data is the 12 bits coded into 3 hex digits. Very helpful.

(NOTE: some signal names and their function shown above may not be relevant to your design)

Note the value of the 12 bits starts at zero and increases by one each clock tick tracking the coefficient values in the ROM. What is significant in this test is that during the first clock cycle after the signal named fir_calc asserts that the ROM output is zero which represents the first coefficient (likely non-zero in an actual filter set).



I have placed the Waveforms workspace file used in this example on the web page if you wish to use it and save time setting up the BUS etc.

12) If a DC offset needs to be removed from data coming from the A/D converter, the value 2048 needs to be subtracted from the ADC data. Subtraction can be done by adding a negative value. Here are statements that will do that:

```
val_adc_raw <=  signed(adc_12bit_vector);          – convert from std_logic_vector
val_adc_no_offset <= val_adc_raw + to_signed(2048,12);
```

No negative sign is needed on the 2048 because in 12 bits it is 1000 0000 0000 in binary, i.e. the most significant bit is a one which means negative in 2's complement.

13) Before the output of the FIR circuit can be sent to the DAC, the waveform effectively needs to be level shifted so that the negative half cycles of a sine wave (or other waveform) coming out are always positive numbers. To do this, add 2047 (the largest positive 2's compliment format 12-bit number) to the 12-bit signed number selected from the accumulator output. Then likely you need to convert from signed to std_logic_vector, at least if you are using the supplied DAC driver component.