Wiki »

EXERCISE: FIR FILTER

Exercise Goals

Implementing a real FIR filter in hardware, simulating it and running it on target gives experience in :

- Estimating basic FIR filter coefficients
- Implementing a pipelined filter structure
- Using arithmetic operators and functions
- Testing a FIR filter in ModelSim
- Stream Bus Interfacing

Prerequisites

Quartus II and ModelSim and GNU/Octave (or Matlab) software must be installed and working.

Exercise Steps

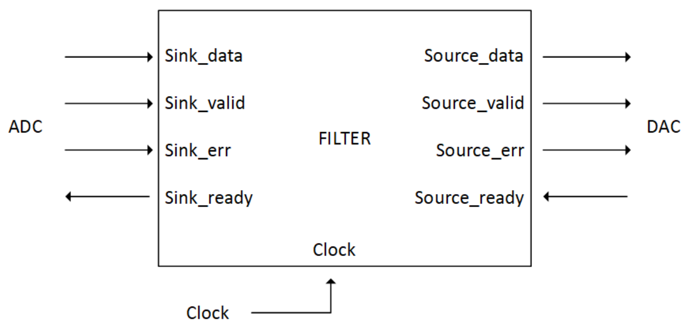
Design a Basic FIR Filter

A) Design 10-order low pass filter with a cut-off frequency of 8KHz using the Equiripple method in GNU/Octave.

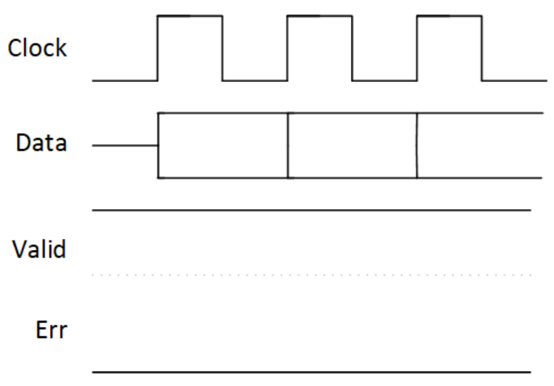
- The Filter input is 24-bit and the coefficients are 8-bit. Only fixed-point values are used.
- Use the function “fir1” in Octave or Matlab to calculate filter coefficients. See the relevant Wiki Guide pages for details.

B) Create the filters interfaces

The filter uses a streaming interface in- and out. The filter has the following interface:



The timing of the signals looks like this:



The error signal is driven by the signal source, here the ADC, whereas the *valid* signal is sent back to the ADC to let it know that we are ready to receive data. In this case we'll just set *valid* to '1' and not use the incomming *err* signal. We'll also set the source_valid signal to '1' to indicate data is always ready on the output side.

The sample frequency is 48KHz

```
entity fir is
 port (
   -- Common --
   clk              : in  std_logic;   -- 48KHz
   reset_n          : in  std_logic;
   -- ST Bus --
   ast_sink_data    : in  std_logic_vector(23 downto 0);
   ast_sink_ready   : out std_logic;
   ast_sink_valid   : in  std_logic;
   ast_sink_error   : in  std_logic_vector(1 downto 0);
   ast_source_data  : out std_logic_vector(23 downto 0);
   ast_source_ready : in  std_logic;
   ast_source_valid : out std_logic;
   ast_source_error : out std_logic_vector(1 downto 0)
   );
end entity fir;
```

C) Implement the filter

- Implement the filter using a method similar to that described in example 3.2 & 3.3 of the text book.
- Simulate the filter using an impulse input. You may use the test bench from previous exercise or create a new.

Exercise Hints!

When designing the filter, remember that a 10th order filter has 11 taps!

The coefficients calculated have a symmetry that can be used by adding the tab values before multilpying. Ex: prod(0) = (tap(0) + tap(10)) * coff(0);

Use arrays of hold coefficients, taps and product values

```
 subtype coeff_type is integer range -128 to 127;
type coeff_array_type is array (0 to filterOrder/2) of coeff_type;

subtype tap_type is signed(inputWidth-1 downto 0);
type tap_array_type is array (0 to filterOrder) of tap_type;

subtype prod_type is signed(inputWidth+coefWidth-1 downto 0);
type prod_array_type is array (0 to filterOrder/2) of prod_type;

constant coeff : coeff_array_type := (-1, -3, 0, 27, 11, 19);
signal tap : tap_array_type;
signal prod : prod_array_type;
```

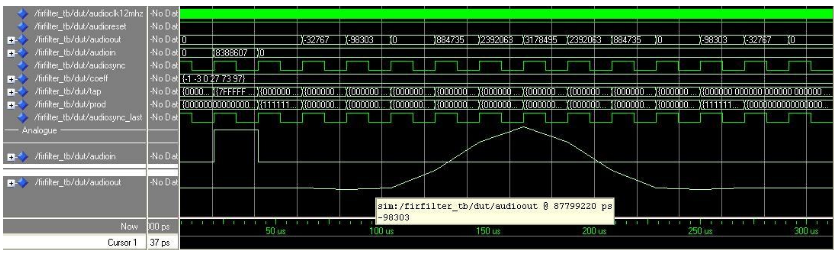
To calculate the products you have to convert the integer to signed:

```
prod(tap_no) <= to_signed(coeff(tap_no),8) * tap(tap_no);
```

Testing the design: A good place to start, is to create a testbench that creates an impulse:

```
...

  -- clock generation
 Clk <= not Clk after period/2;
 Clk48KHz <= not Clk48KHz after period48K/2;

 AudioReset <= '0', '1' after 125 ns;

 -- waveform generation
 WaveGen_Proc: process
 begin
 wait until reset_n = '1';
 wait until clk = '1';

 ast_sink_data <= X"010000";                -- Impulse
 wait until clk = '1';
 ast_sink_data <= X"000000";

 wait;
 end process WaveGen_Proc;

...
```

As we recall from Matlab/Octave we should get a sinc-like output. Using an impulse as imput (value is high for one 48KHz clock period and otherwise zero), we get a value that walks down the delay line with the remaining taps being all zero. This gives an output which is: y[n] = x * tap[n]. The simulation below illustrates this:


/firfilter_tb/dut/audioclk12mhz
/firfilter_tb/dut/audioreset
/firfilter_tb/dut/audioout
/firfilter_tb/dut/audioin
/firfilter_tb/dut/audiosync
/firfilter_tb/dut/coeff
/firfilter_tb/dut/tap
/firfilter_tb/dut/prod
/firfilter_tb/dut/audiosync_last
Analogue
/firfilter_tb/dut/audioin
/firfilter_tb/dut/audioout
Now
Cursor 1
50 us
100 us
150 us
200 us
250 us
300 us

```
audioout: 0 -32767 -98303 0 884735 2392063 3178495 2392063 884735 0 -98303 -32767 0
audioin: 0 8388607 0
coeff: {-1 -3 0 27 73 97}
sim:/firfilter_tb/dut/audioout @ 87799220 ps
-98303
```

The scaled (x8) coefficients are: {-1, -3, 0, 27, 73, 97}.

Using an impulse with amplitude of 0x10000 (=65535d) audioOut becomes {-32768, -98303, 0, 884735 etc}

This corresponds to: { 65535*(-1)/256, 65535*(-3)/256, 65535*0/256, 65535*27/256 etc.}

- fir_timing.png (4,51 KB) Peter Høgh Mikkelsen, 2017-02-14 21:12
- fir_block_signals.png (10,8 KB) Peter Høgh Mikkelsen, 2017-02-14 21:12