VHDL description of a simple FIR-filter

Christian Söderbacka
s92696
List of contents

1. PDSP vs. FPGA
2. Some common implementations
3. A (very) simple example in VHDL
4. Another example
5. Using software to generate the coefficients

Appendices:
- VHDL code used for simulation
- VHDL testbench code
DSP vs. FPGA in digital signal processing

The good question: why use FPGA:s instead of DSP:s which are optimized for digital signal processing?

Important to remember: most signal processing algorithms are Multiply and Accumulate intensive, MAC

(Meyer-Baese, p.12-13)
PDSP – Programmable digital signal processor

- Has been dominating for decades, inexpensive
- Based on RISC (Reduced Instruction Set Computer)
- Has at least one fast array multiplier of size 16x16 bit to 24x24 bit of fixed point or 32-bit floating point
- Extended wordwidth accumulator
- Multistage pipeline architecture => MAC rates limited only by the speed of the array multiplier
- Advantageous in complex algorithms requiring several if-then-else constructs
FPGA

- FPGA:s have massive parallell computing capabilities that can be exploited by optimizing the architecture
- Multiple MAC cells can be implemented on one chip to provide extended bandwidth => wireless communication, satellite transmission, multimedia
- Several algorithms such as CORDIC, NTT and error-correction algorithms are more efficient when implemented on FPGA:s
- Beneficial when implementing FIR filters and FFT:s
FIR-filtering is basically all about convolution. Convolution is the most important concept in dsp.

\[ y[n] = x[n] * f[n] = \sum_{k=0}^{L-1} f[k] x[n - k] \]
Some common implementations

- Direct form FIR filter (inefficient)
- Transposed structure FIR (better)
package eight_bit_int is
    subtype byte is integer range -128 to 127;
    type byte_array is array(0 to 3) of byte;
end eight_bit_int;

signal tap: byte_array := (0,0,0,0);
The convolution process

- The coefficients are \([-1 \ 3.75 \ 3.75 \ -1]\)

\[
y \leq 2 \times \text{tap}(1) + \text{tap}(1) + \text{tap}(1)/2 + \text{tap}(1)/4 \\
+ 2 \times \text{tap}(2) + \text{tap}(2) + \text{tap}(2)/2 + \text{tap}(2)/4 \\
- \text{tap}(3) - \text{tap}(0);
\]

- \(\text{tap}(0) \times (-1)\)
- \(\text{tap}(1) \times (2+1+0.5+0.25)\)
- \(\text{tap}(2) \times (2+1+0.5+0.25)\)
- \(\text{tap}(3) \times (-1)\)
Tapped delay line: shift one left

for i in 3 downto 1 loop
    tap(i) <= tap(i-1);
end loop;
tap(0) <= x; -- read in new integer
The impulse response is the same thing as the filter coefficients in FIR filters.
ModelSim simulation of the impulse response
Example from Lyons, p.216-222
Convolution with $n = 0$

\[ y(0) = h(0)x(0) + h(1)x(-1) + h(2)x(-2) + h(3)x(-3) = 1 + 0 + 0 + 0 = 1 \]

Convolution with $n = 1$

\[ y(1) = h(0)x(1) + h(1)x(0) + h(2)x(-1) + h(3)x(-2) = 2 + 1 + 0 + 0 = 3 \]

Convolution with $n = 2$

\[ y(2) = h(0)x(2) + h(1)x(1) + h(2)x(0) + h(3)x(-1) = 3 + 2 + 1 + 0 = 6 \]

Convolution with $n = 3$

\[ y(3) = h(0)x(3) + h(1)x(2) + h(2)x(1) + h(3)x(0) = 0 + 3 + 2 + 1 = 6 \]

Convolution with $n = 4$

\[ y(4) = h(0)x(4) + h(1)x(3) + h(2)x(2) + h(3)x(1) = 0 + 0 + 3 + 2 = 5 \]

Convolution with $n = 5$

\[ y(5) = h(0)x(5) + h(1)x(4) + h(2)x(3) + h(3)x(2) = 0 + 0 + 0 + 3 = 3 \]
The result of the convolution sequence

<table>
<thead>
<tr>
<th>Index k or m</th>
<th>h(k)</th>
<th>x(k)</th>
<th>h(k) * x(k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Simulation results
Moving from time domain to frequency domain

What does the filter response look like in the frequency domain?

Fourier transform pairs

\[ h(k) \ast x(k) \xrightarrow{\text{DFT}} H(m) \cdot X(m) \]

\[ h(k) \cdot x(k) \xrightarrow{\text{DFT}} H(m) \ast X(m) \]
Example:

- convolve the filter coefficients of a 32-tap FIR with the input signal
- append (zero-pad) 480 zeros
- apply a 512-point DFT

The FFT performs faster when the number of points is an integer power of 2 \( (2^9 = 512) \)
Example magnitude response of a lowpass filter in the frequency domain
Parks-McClellan exchange FIR filter design method (Matlab)
We define the wanted frequency response and let the software calculate the coefficients.
### Filter information

**Discrete-Time FIR Filter (real)**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter Structure</td>
<td>Direct-Form FIR</td>
</tr>
<tr>
<td>Filter Length</td>
<td>27</td>
</tr>
<tr>
<td>Stable</td>
<td>Yes</td>
</tr>
<tr>
<td>Linear Phase</td>
<td>Yes (Type I)</td>
</tr>
<tr>
<td>Design Method Information</td>
<td></td>
</tr>
<tr>
<td>Design Algorithm</td>
<td>equiripple</td>
</tr>
<tr>
<td>Design Options</td>
<td></td>
</tr>
<tr>
<td>Density Factor</td>
<td>1.0</td>
</tr>
<tr>
<td>Maximum Phase</td>
<td>false</td>
</tr>
<tr>
<td>Minimum Order</td>
<td>any</td>
</tr>
<tr>
<td>Minimum Phase</td>
<td>false</td>
</tr>
<tr>
<td>Stopband Decay</td>
<td>0</td>
</tr>
<tr>
<td>Stopband Shape</td>
<td>flat</td>
</tr>
<tr>
<td>Uniform Grid</td>
<td>true</td>
</tr>
<tr>
<td>Sampling Frequency</td>
<td>44.1 kHz</td>
</tr>
<tr>
<td>Response</td>
<td>Lowpass</td>
</tr>
<tr>
<td>Specification</td>
<td>$f_p, f_s, A_p, A_s$</td>
</tr>
<tr>
<td>Passband Edge</td>
<td>5 kHz</td>
</tr>
<tr>
<td>Stopband Edge</td>
<td>8 kHz</td>
</tr>
<tr>
<td>Passband Ripple</td>
<td>3 dB</td>
</tr>
<tr>
<td>Stopband Atten</td>
<td>60 dB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Measurement</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sampling Frequency</td>
<td>44.1 kHz</td>
</tr>
<tr>
<td>Passband Edge</td>
<td>5 kHz</td>
</tr>
<tr>
<td>3-dB Point</td>
<td>5.3279 kHz</td>
</tr>
<tr>
<td>6-dB Point</td>
<td>5.7454 kHz</td>
</tr>
<tr>
<td>Stopband Edge</td>
<td>8 kHz</td>
</tr>
<tr>
<td>Passband Ripple</td>
<td>2.3965 dB</td>
</tr>
<tr>
<td>Stopband Atten</td>
<td>61.5164 dB</td>
</tr>
<tr>
<td>Transition Width</td>
<td>3 kHz</td>
</tr>
</tbody>
</table>

**Implementation Cost**

- Number of Multipliers: 27
- Number of Adders: 26
- Number of Stages: 26
- Multiplications per Input Sample: 27
- Additions per Input Sample: 26
### Generated filter coefficients

| Numerator: | 0.003078415471677433 0.0102368833437561411 0.0204970626778980588 0.0307536156576460546 0.0307536156576460546 0.0417631698869623 0.0541845471066649082 0.044099882066787503 0.05023672251877177 0.01099105669111201 0.0655984188574793 0.1563853067148521 0.23603759242687232 0.2674111705917903 0.23603759242687333 0.16583853066148531 0.0858984198574793 0.01599105669111201 0.05023672251877177 0.044099882066787503 0.0154145471066649082 0.0417631698869623 0.0307536156576460546 0.0205970626778980588 0.0102368833437561411 0.003078415471677433 |
Most softwares can also generate VHDL-code.
References

- Meyer-Baese, Uwe, "Digital signal processing with field programmable gate arrays", 3rd edition, 2007

package eight_bit_int is
    subtype byte is integer range -128 to 127;
    type byte_array is array(0 to 3) of byte;
end eight_bit_int;

library work;
use work.eight_bit_int.all;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
-- Define a "Blackbox"

entity fir_srg is
port( clk: in std_logic;
    x: in byte;
    y: out byte);
end fir_srg;

-- Chosen architecture modeling type: behavioural

architecture flex of fir_srg is
signal tap: byte_array := (0,0,0,0);
begin
p1: process
begin
wait until clk = '1'; -- wait for rising edge of clock
-- Compute output y with the filter coefficients weight
-- The coefficients are [-1 3.75 3.75 -1]
-- Division only allowed for powers-of-two values!
y <= 2 * tap(1) + tap(1) + tap(1)/2 + tap(1)/4
+ 2 * tap(2) + tap(2) + tap(2)/2 + tap(2)/4
- tap(3) - tap(0);
-- Tapped delay line: shift one left
for i in 3 downto 1 loop
  tap(i) <= tap(i-1);
end loop;
tap(0) <= x; -- Input inserted to register, overwrites old value
end process p1;
end flex;
fir_test.vhd Testbench for 4-tap fir-filter

library work; -- include the user-defined types
use work.eight_bit_int.all;
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity fir_test is
end fir_test;

architecture testbench of fir_test is
signal Xin: byte := 0;
signal Yout: byte := 0;
signal clock: std_logic := '0';
constant T: time := 20 ns;
begin
fir: entity work.fir_srg(flex)
port map(x => Xin, y => Yout, clk => clock);
clocksim: process  -- simulate clock
begin
  clock <= '0';
  wait for T/2;
  clock <= '1';
  wait for T/2;
end process clocksim;
input_test: process
begin
wait for 10 ns;
Xin <= 1;
wait for 20 ns;
Xin <= 2;
wait for 20 ns;
Xin <= 3;
wait for 20 ns;
Xin <= 0;
wait;
end process input_test;
end testbench;