30 minutes VHDL design for very fast FIR Filter on low cost FPGA

By
Edgard GARCIA
MVD

edgard.garcia@mvd-fpga.com
Introduction

• In this example we will demonstrate the ability of the low cost FPGA family to implement high performance DSP functions – even without using dedicated multipliers.

• Such implementation requires a good understanding of the algorithm

• It also requires a good knowledge of the FPGA architectures

• Finally, using the right coding style with the right synthesis tool such as Synplify-Pro/Synplify-Premier and powerful optimization features, very high performance can be achieved.
FIR example

60 tap Bandpass
16 bit data
16 bit coefficients
38 bit output
Fs = 120 Mhz

To be implemented in a low cost FPGA without using dedicated multipliers (3S700A-4 FG400 in this example)
FIR example
Multiplier/Accumulator

- Using Multiplier Accumulator structure would require working at 10 GHz for a 100-tap, 100 Mhz !!!
**Notes:**

- This structure requires N multipliers for a N x Taps FIR
- Same comment for data registers
- Summation implemented with an adder tree
Adder tree
Using pipeline registers to improve multipliers performance

- Very easy for the multiplications
- Pipelining multipliers for “free” (embedded registers)
Using pipeline registers to improve adder tree performance

A little bit more tedious and expensive for adder tree
Resources and performances

- A 60 tap FIR filter based on this architecture would require the following resources (for 16 bits data and coefficients)
  - 60 x 16 bit registers (for data pipeline)
  - 60 Multipliers (dedicated or not)
  - 59 adders (with registers)

- Routing delay will adversely affect the performance (due to the nets length in the adder tree)

- Using symmetry would save 30 multipliers (at the expense of 30 adders) and reduce the size of the adder tree by 50%
  - 60 x 16 bit registers (for data pipeline) = 480 slices
  - 30 x 17 bit pre-adders = 270 slices
  - 30 Multipliers (dedicated or not) > 2000 slices (if slices based)
  - 29 adders (with registers) = 600 slices

- Total: > 3,300 slices and a tremendous routing congestion (preventing probably meeting timing constraints)
The Transpose FIR structure

- Excellent architecture for medium and high speed filters
- The adder chain can be easily placed and routed due to the structure of the Xilinx FPGA (arithmetic functions are placed in columns)
- Multipliers can be pipelined to improve performance
- The same data being multiplied by all the coefficients, partial results can be used for many multiplications (regardless of symmetry)
- Very low latency, excellent performance even without using dedicated multipliers
Resource estimation

60 Taps, 16 bit data, 16 bit coefficients

- Even considering that the output could need up to 38 bits (typically no more than 34) the adder chain will require 19 slices x 60 = 1140 slices

- Multipliers can be (and will be) implemented in slices (pipeline registers can be used at no cost)

- Targeted FPGA family : Spartan3A –4 (lowest speed grade)
VHDL source code

60 Taps, 16 bit data, 16 bit coefficients

Coefficients, multiplication results as well as adders will be defined as arrays
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;

entity FIR_60_TAPs is
  Port ( CK : in  STD_LOGIC;
         DIN : in  STD_LOGIC_VECTOR (15 downto 0);
         DOUT : out  STD_LOGIC_VECTOR (37 downto 0));

attribute syn_useioff : boolean;
attribute syn_useioff of DIN : signal is true;
attribute syn_useioff of DOUT : signal is true;

end FIR_60_TAPs;
VHDL source code
Architecture declarations

architecture Behavioral of FIR_60_TAPs is

signal DINR : std_logic_vector(DIN'range);

type COEFS_TYPE is array(59 downto 0) of std_logic_vector(15 downto 0);
constant COEFS : COEFS_TYPE := (x"ffff",x"ffb3",x"00d6",x"0063",x"fdb1",x"01d9",x"025f",x"fab4",
x"0141",x"063b",x"f92d",x"fe5a",x"0825",x"fbe0",x"fccd",x"031e",
x"0011",x"0410",x"f766",x"fcc2",x"1730",x"eeb3",x"ea68",x"2dac",
x"f57a",x"cd9b",x"3731",x"0dd7",x"b74e",x"2a15",x"2a15",x"b74e",
x"0dd7",x"3731",x"cd9b",x"f57a",x"2dac",x"ea68",x"eeb3",x"1730",
x"fcc2",x"f766",x"0410",x"0011",x"031e",x"fccd",x"fbe0",x"0825",
x"fe5a",x"f92d",x"063b",x"0141",x"fab4",x"025f",x"01d9",x"f92d",
x"0063",x"00d6",x"ffbb",x"ffff")

type MULT_TYPE is array(59 downto 0) of std_logic_vector(31 downto 0);
signal MULT : MULT_TYPE;
attribute syn_multstyle : string;
attribute syn_multstyle of MULT : signal is "logic";

type ADD_TYPE is array(59 downto 0) of std_logic_vector(37 downto 0);
signal ADD : ADD_TYPE;
constant ZERO : std_logic_vector(37 downto 0) := (others => '0');
begin

process(CK) begin
  if CK'event and CK = '1' then
    DINR <= DIN;
    for I in 59 downto 0 loop
      MULT(I) <= DINR * COEFS(59-I);
      if I = 0 then
        ADD(I) <= ZERO + MULT(0);
      else
        ADD(I) <= MULT(I) + ADD(I-1);
      end if;
    end loop;
    DOUT <= ADD(59);
  end if;
end process;
end Behavioral;
Implementation results

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Check</th>
<th>Worst Case</th>
<th>Best Case</th>
<th>Timing</th>
</tr>
</thead>
<tbody>
<tr>
<td>NET &quot;CK&quot; PERIOD = 8.333 ns</td>
<td>-3.630ns</td>
<td>11.963ns</td>
<td>2563</td>
<td></td>
</tr>
</tbody>
</table>

Total REAL time to PAR completion: 1 mins 52 secs

Timing errors !!!
Timing analysis for a 38 bit adder

Timing model for slice based arithmetic functions

Total FF to FF delay:
- Tco = 650 ps
- Tnet = 2400 ps (assumed)
- T(first LUT + carry) = 1300 ps
- Tcarry x 37 = 2960 ps
- Tsu = 1000 ps

Total = 8310 ps

The adder chain must be able to work at 120 Mhz even with a net delay of 2400 ps
Multipliers implementation

• Multipliers can be implemented on slices by using adders

• Adders can be pipelined at no FPGA resources cost

• “Manual” optimization of the source code would be a time consuming task.

• Instead, we can use the “RETIMING” option of Synplify-Pro to do the job, and push the registers where needed to meet performance.

• Let’s modify the source code to add some levels of pipeline on the data bus
Adding pipeline registers

3 levels of pipeline (in this example)

By selecting the “RETIMING” option in Synplify-Pro, we expect the
registers to be pushed inside the multiplier logic in order to improve
performance
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;

entity FIR_60_TAPs is
  Generic (N : integer := 3);  -- Number of pipe cycles
  Port ( CK : in  STD_LOGIC;
        DIN : in  STD_LOGIC_VECTOR (15 downto 0);
        DOUT : out  STD_LOGIC_VECTOR (37 downto 0));

  attribute syn_useioff : boolean;
  attribute syn_useioff of DIN : signal is true;
  attribute syn_useioff of DOUT : signal is true;

end FIR_60_TAPs;
Modifying the VHDL source code

Architecture declarations

architecture Behavioral of FIR_60_TAPs is

signal DINR : std_logic_vector(DIN'range);

type DINS_TYPE is array(N-1 downto 0) of std_logic_vector(DIN'range);
signal DINS : DINS_TYPE;

type COEFS_TYPE is array(59 downto 0) of std_logic_vector(15 downto 0);
constant COEFS : COEFS_TYPE := (
    x"ffff", x"ffb3", x"00d6", x"0063", x"fdb1", x"01d9", x"025f", x"fab4",
    x"0141", x"0613", x"f92d", x"fe5a", x"0825", x"fbe0", x"fccd", x"031e",
    x"0011", x"0410", x"f766", x"fcc2", x"1730", x"eeb3", x"ea68", x"2dac",
    x"f57a", x"cd9b", x"3731", x"0dd7", x"b74e", x"2a15", x"2a15", x"b74e",
    x"0dd7", x"3731", x"cd9b", x"f57a", x"2dac", x"ea68", x"eeb3", x"1730",
    x"f766", x"0410", x"0011", x"031e", x"fccd", x"fbe0", x"0825",
    x"fe5a", x"f92d", x"0613", x"0141", x"fab4", x"025f", x"01d9", x"fdb1",
    x"0063", x"00d6", x"ffb3", x"ffff");

type MULT_TYPE is array(59 downto 0) of std_logic_vector(31 downto 0);
signal MULT : MULT_TYPE;

attribute syn_multstyle : string;
attribute syn_multstyle of MULT : signal is "logic";

type ADD_TYPE is array(59 downto 0) of std_logic_vector(37 downto 0);
Modifying the VHDL source code

RTL code

begin

process(CK) begin
  if CK'event and CK = '1' then
    DINR <= DIN;
    DINS <= DINS(N-2 downto 0) & DINR;
    for I in 59 downto 0 loop
      MULT(I) <= DINS(N-1) * COEFS(59-I);
      if I = 0 then
        ADD(I) <= ZERO + MULT(0);
      else
        ADD(I) <= MULT(I) + ADD(I-1);
      end if;
    end loop;
    DOUT <= ADD(59);
  end if;
end process;
end Behavioral;

MULTI VIDEO DESIGNS
FPGA Experts!
Setting up Synplify-Pro synthesis options (ISE environment)

- 140 Mhz request to have some extra timing margin for routing
- Retiming option to allow Synplify-Pro to rearrange the Flip-Flops where needed to obtain the required performance
Setting up SynplifyPro synthesis options (Synplify-Pro environment)

- An extra routing delay of 1 ns is used to improve accuracy and match post place-and-route results.
- Retiming option to allow Synplify-Pro to re-arrange the Flip-Flops where needed to obtain the required performance.
- Specify I/O locations.

All constraints are forward-annotated to Xilinx (.ncf file)
Implementation results using Synplify-Pro “RETIMING” option

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Check</th>
<th>Worst Case</th>
<th>Best Case</th>
<th>Timing</th>
<th>Timing</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>NET &quot;CK&quot; PERIOD</td>
<td>8.333 ns</td>
<td>0.008ns</td>
<td>8.325ns</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

No more timing errors !!!

120 Mhz worst case for the lowest speed grade

Previous results (no pipe)

- Number of Slice Flip Flops: 4,158 out of 11,776 36%
- Number of 4 input LUTs: 4,289 out of 11,776 36%
- Number of occupied Slices: 2,426 out of 5,888 43%
- Number of bonded IOBs: 55 out of 311 17%
- IOB Flip Flops: 54

Almost 10% improvement in slice utilization

120 Mhz worst case for the lowest speed grade
Conclusion

• Many improvements are still possible to save additional logic resources (more than 10% of slices) and/or improve timing.
  Note that the implementation would give similar results with any other Spartan3 family.
  Much higher performance with Virtex4 or Virtex5 family

• However, we have seen that by combining
  • a good understanding of the Xilinx FPGA architecture
  • an efficient structure for the FIR implementation (transpose in this case)
  • an efficient coding (less than 10 lines for an N tap filter)
  • appropriate features of Synplify-Pro/ Synplify-Premier

  such a design can be done in less than 15 minutes, starting from scratch.
Conclusion

• Xilinx FPGA are very well suited for many kind of applications, particularly for DSP design, even when DSP blocks or dedicated multipliers are not available.

• Synplify-Pro/ Synplify-Premier takes advantage of the powerful Xilinx FPGA architecture

• User must be aware of the architectures and tools features

• MVD offers public and on site trainings.
  • VHDL, Xilinx architectures & tools, design methodology, FPGA based DSP, MicroBlaze, PowerPC…

• PDF and source code available at : www.mvd-fpga.com

  Corporate => publications => press publications and app notes
  Société => publications => publications de presse et notes d’applications

MULTI VIDEO DESIGNS
FPGA Experts!

www.mvd-fpga.com
Thank You!

www.mvd-fpga.com