Implementing Digital Finite Impulse Response Filter Using FPGA

Abdul Hadi Abdul Razak[1], Muhamad Iqbal Abu Zaharin[2] and Nor Zaidi Haron[3]

[1]Faculty of Electrical Engineering,
Universiti Teknologi MARA,
40450 Shah Alam, Selangor.
[2]Faculty of Electronics and Computer Engineering, Universiti Teknikal Malaysia Melaka, Karung Berkunci 1200, Hang Tuah Jaya, 75450, Ayer Keroh, Melaka, Malaysia.
adi3443@salam.uitm.edu.my, iqbal007@salam.uitm.edu.my, zaidi@utem.edy.my

Abstract **-** This paper describes the design of Transposed Form FIR filter implemented in the Spartan-II and Virtex-E family of FPGAs. The design is an 8-tap filter based on 16-bit input samples and 14-bit signed coefficients. The basic building blocks of the filter are KCMs, Adders, Registers, and a delay-locked loop. All the 14-bit coefficient factors are stored with an 18-bit word size in the ROM. The program is written in VHDL source code based on application Xilinx notes [1] that describe the design of an FIR filter. The software tools have been used are Xilinx ISE Webpack 8.1, ModelSim 6.1e and Matlab 7.0.

Keywords - FIR Filter, FPGA, Matlab, VHDL, Xilinx.

I. INTRODUCTION

THIS paper briefly introduces the general aspects of designing FIR filters. The purpose of this project is to study what is FIR filter and how the FIR filter can be designed and implemented by using Xilinx's FPGA. The structure and function for FIR filters are reviewed. Following this general review, the result and discussion of the simulation will be discussed.

Most of the traditional filters are implemented using DSP processors. However, they have bandwidth limitations because only a fixed number of operations can be performed. By using FPGA, the parallel-pipelined architecture is able to enhance the overall performance. The filter is implemented in the Spartan-II (XC2S100) and Virtex-E (XCV100E) device to study their performance. The reprogrammability of FPGAs enables tuning of the filter at any time.

II. SCOPE OF WORK

The main scope of work involves behavioural simulation and verifying the functionality of the filter in programmable device of FPGAs. In order to design a specific digital block of integrated circuit requires a team of engineers working in different level of design stages. One of the design stages is verification stage.

Verification is a predictive analysis to ensure that the synthesized design will perform the desired function. The stage verifies the correctness of the design. It can be performed by simulation and it is responsible for quality of the design.

In this paper, VHDL language is used, compiled and synthesized where the netlist will be generated by using Xilinx ISE Webpack and ModelSim. For comparison purposes, Matlab is used to compare the output results with the one specified in the application notes [1].

III. METHODOLOGY

The design of a digital filter involves four steps [2];

1) Filter specification - This may include stating the type of filter, for example lowpass filter, the desired amplitude and/or phase responses and the tolerances we are prepared to accept, the sampling frequency, and the wordlength of the input data.
2) Coefficient calculation - Determine the coefficients of a transfer function, which will satisfy the specifications given in (1). Our choice of coefficient calculation method will be influenced by several factors, the most important of which are the critical requirements in step (1). By using Matlab's Filter Design Toolbox (FDA tool), the coefficient can be computed based on the design specifications.
3) Realization - This involves converting the transfer function obtained in (2) into a suitable network or structure.
4) Implementation. This involves producing the software code and/or hardware and performing the actual filtering.

By using Xilinx Synthesis tool, the design flow comprises the following steps: design entry, design synthesis and design implementation [3,4].

1) Design entry is the first step in the ISE design flow. During design entry, source file is created using VHDL language.
2) After design entry, the synthesis test is run. During the synthesis process, a netlist file known as NGC will be created.
3) The design implementation is run which comprises *translate* (merges all of the input netlist and design constraints and outputs a NGC file which describes the logical design reduced to Xilinx primitives), *map* (maps the logic defined by an NGD file into FPGA elements, such as CLBs and IOBs), *place and route* (takes a mapped NCD file, places and routes the design, and produces NCD file that is used as input for bitstream generation.

The output results of the simulation are then, studied and compared with the theory of FIR filter to conform the design specifications and functionality.

IV. FIR FILTER

FIR filter is a type of digital filter, in which every sample of output is the weighted sum of past and current samples of input, using only some finite number of past samples. Beside the finite-duration impulse response, another important characteristic of a FIR filter is the linear phase property [7].

When the signal is composed of many frequencies we need to be concerned with the phase response of the filter. If different frequencies take different amounts of time to pass through the filter, the output of the filter will become distorted. The only way to avoid this distortion is to use a phase linear filter [2].

A. FIR Filter Structure

The N-tap FIR digital filter is normally described by the equation:

$$y(n) = \sum_{i=0}^{N-1} H_i X_{n-i} \qquad (1)$$

The FIR filter transfer function is realized by a shifting register and a loop in which the filter coefficients are multiplied by the shifting register values [2, 8]. The sums of these multiplications will determine the output value. In equation (1), H_i refers to the coefficients and X_{n-i} is input samples.

The advantage of the transposed structure is that we do not need an extra shift register and there is no need for an extra pipeline stage for the adder (tree) of the products to achieve high throughput. Transposed Form FIR structure has a lower input to output latency when the adder tree must be pipelined for performance [1].

V. FILTER IMPLEMENTATION IN XILINX DEVICE

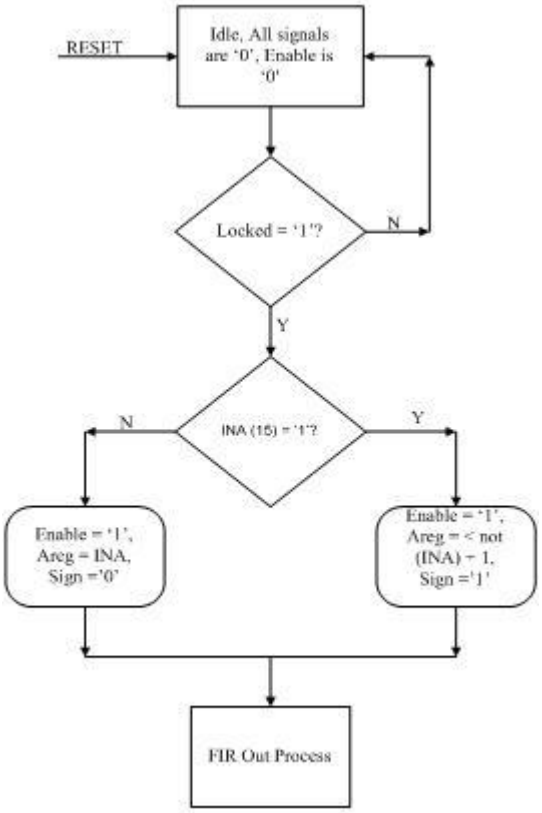


Fig. 1. The flow chart of the main building block of the filter.

Fig. 1 shows the main flow chart of the program. The 'reset' signal plays an important role in order to enable the 'locked' signal. When the 'reset' signal is zero, the clock DLL locks. The code will check whether the bit number 15 of the input is one. If it is one, the 'Areg' will be two's complement of the input and the sign is one. Otherwise the 'Areg' will be the same as input and the sign is zero.

The "FIR Out Process" occurs in the Constant Coefficient Multiplier (KCM). The KCM block waits until the clock DLL locks and able to generate output when enable is one.

Fig. 2. The top level module of FIR filter showing the input and output synthesized in the Xilinx synthesis tool.

A. Constant Coefficient Multiplier (KCM)

The 16-bit input sample is separated into 4-bit nibbles. Each nibble acts as an input to the ROM in different cycles. These ROMs store the product of the constant coefficient k. The word size in the ROM is 14-bit coefficient x 4-bit input nibble = 18-bit ROM word size.

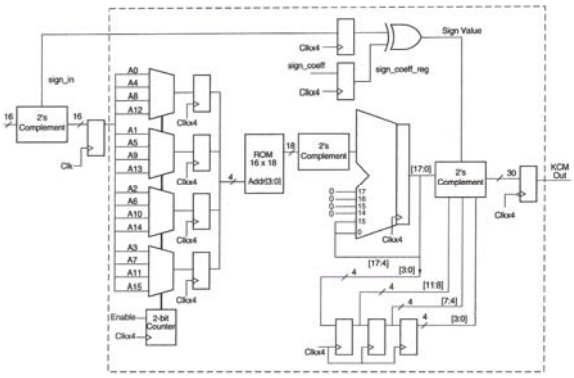


Fig. 3. Constant Coefficient Multiplier

This ROMs functions as a times table of the constant coefficient, k. The value read from this ROM based on its 4-bit input is added to another partial product stored in adjacent ROM. There are three 2's complement modules to handle both signed inputs and coefficients. This is used to avoid signed multiplication and addition. The input sample arrives at clock frequency f_1, while all the internal operations of the KCM can be performed at much higher frequency of f_2 (4 x f_1).

The operation of a KCM multiplier implemented using a ROM is explained with the following example:

16-bit input: 0001 0010 0000 0100 (Decimal equivalent 4612)
14-bit coefficient: 00 0000 0000 0010 (Decimal equivalent 2)

The 16-bit input is separated into 4-bit nibbles: "0001", "0010", "0000", and "0100". All fifteen coefficient factors, 0 x 2, 1 x 2, 2 x 2, …15 x 2 are stored with an 18-bit (14-bit x 4-bit) word size in the ROM. Each 4-bit nibble of the 16-bit input acts as an address to the ROM. The corresponding ROM content at this address is read.

First partial product =
00 0000 0000 0000 1000 (ROM contents at address "1000")
Second partial product =
00 0000 0000 0000 0000 (ROM contents at address "0000")
Third partial product =
00 0000 0000 0000 0100 (ROM contents at address "0100")
Fourth partial product =
00 0000 0000 0000 0010 (ROM contents at address "0010")

All the partial products are then added after shifting them appropriately (shown below):

First partial product = 00 0000 0000 0000 1000
Second partial product = +00 0000 0000 0000 0000 0000
Third partial product = +00 0000 0000 0000 0100 0000 0000
Fourth partial product = +00 0000 0000 0000 0010 0000 0000 0000

Decimal Equivalent, 9224 = 00 0000 0000 0000 0010 0100 0000 1000

B. Delay Locked Loop (DLL)

Two basics types of circuits remove clock delay and eliminate clock skew between external clock input and on-chip clock: Phase Locked Loops (PLLs) and Delay Locked Loops (DLLs) [5]. In addition to the primary function of removing clock distribution delay, DLLs and PLLs typically provide some additional functionality such as frequency synthesis (clock multiplication and clock division) and clock conditioning (duty cycle correction and phase shifting) [11].

The Virtex-E and Spartan-II devices have Delay Locked Loop. Since the KCM uses two clocks of frequency f_1 and f_2 where $f_1 = f_2/4$ or $f_2 = 4 \times f_1$, one DLL is used with an input frequency of f_1 to generate the frequency $f_1 = f_2/4$ using the clock division capability of the DLL.

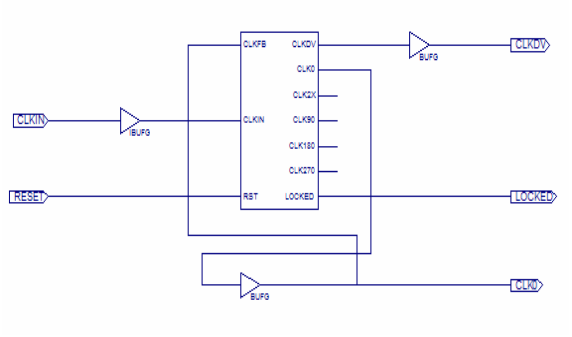


Fig. 4. The delay locked loop synthesized from Xilinx synthesis tool.

VI. Filter Design Using Matlab

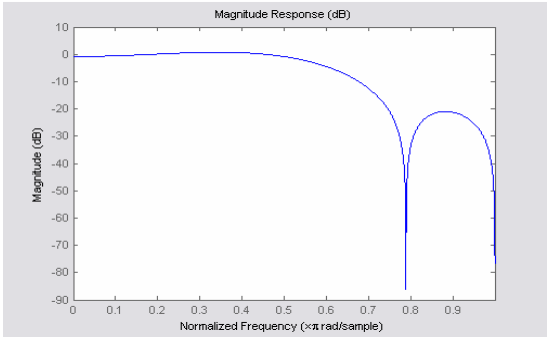


Fig. 5. A lowpass filter frequency response.

Fig. 5 shows a lowpass filter with 16-bit input and 14-bit coefficients. The specifications of the filter are:

- Sampling Frequency, Fs = 16MHz
- Passband Frequency = 4MHz
- Stopband Frequency = 6MHz
- Passband edge, $\omega_p = 0.6$
- Stopband edge, $\omega_s = 0.8$
- Passband ripple, $\alpha_p = 3$dB
- Stopband ripple, $\alpha_s = 20$dB

For comparison purposes, Matlab is used to calculate the ideal coefficients and then round these coefficients to 14-bit to be used in the hardware specified in the application notes [1]. The ideal coefficients as obtained in the Matlab are:
H = [0.0239, -0.1269, 0.0326, 0.5256, 0.5256, 0.3258, -0.1269, 0.0239]

When H is rounded to 14-bit (and scaled to integers) the result is:
H_r = [196, -15345, 267, 4306, 4306, 267, -15345, 196]. This was then used to implement the FIR filter using the approach outlined in the application notes [1].

VII. Result And Discussion

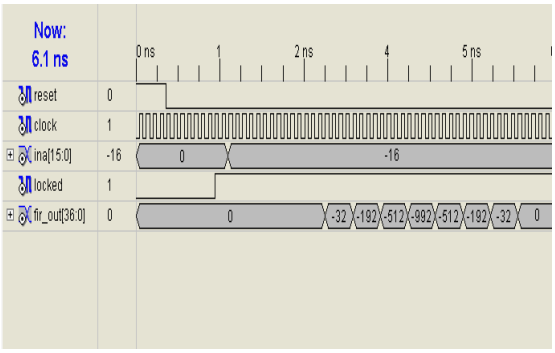


Fig. 6. Simulation output when input is -16 (signed number) simulated by using Xilinx ISE Webpack 8.1.

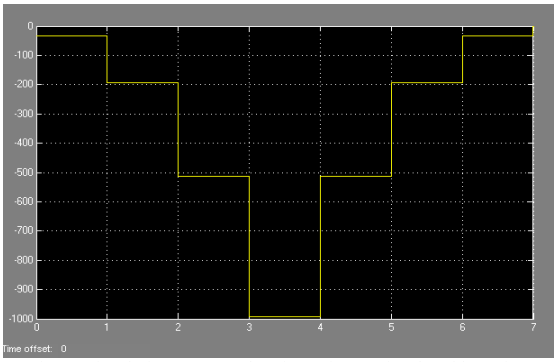


Fig. 7. Simulation output when input is -16 simulated by using MATLAB 7.0.

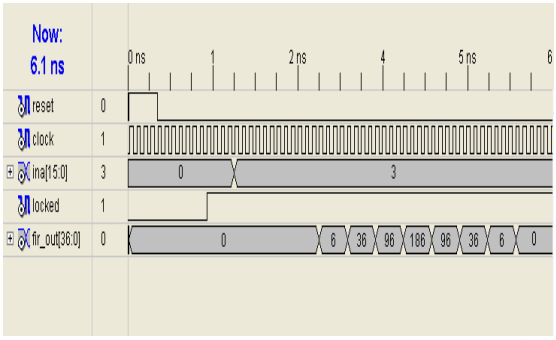


Fig. 8. Simulation output when input is 3 simulated by using Xilinx ISE Webpack 8.1.

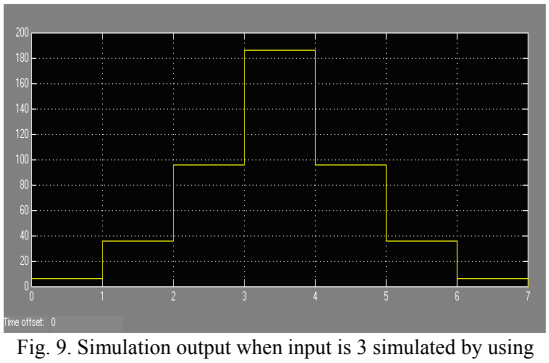


Fig. 9. Simulation output when input is 3 simulated by using MATLAB 7.0.

TABLE I
COMPARISON RESULT BETWEEN MATLAB AND FPGA OUTPUT.

Sequence	1	2	3	4	5	6	7	8
Input Signal	3	3	3	3	3	3	3	3
Matlab Output	6	36	96	186	96	36	6	0
FPGA Output	6	36	96	186	96	36	6	0

From the simulation results, every sample of output is the weighted sum of past and current samples of input, using only some finite number of past samples. Referring to table I, it can be observed that both Xilinx and Matlab yield a same result. How the outputs are calculated:

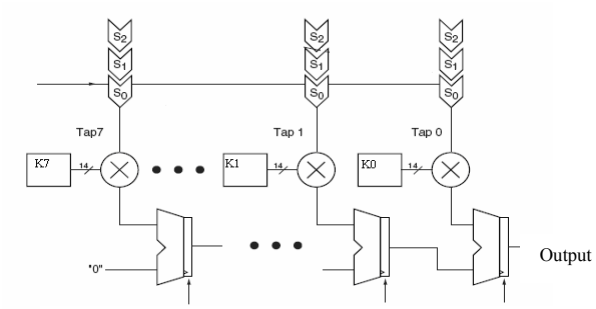
Cycle 1: $(0) + S_0k_0 = 3(2) = \underline{\mathbf{6}}$
Cycle 2: $S_0k_1 + S_1k_0 = 3(10) + 3(2) = \underline{\mathbf{36}}$
Cycle 3: $S_0k_2 + S_1k_1 + S_2k_0 = 3(20) + 3(10) + 3(2) = \underline{\mathbf{96}}$
Cycle 4: $S_0k_3 + S_1k_2 + S_2k_1 + S_3k_0 = 3(30) + 3(20) + 3(10) + 3(2) = \underline{\mathbf{186}}$
Cycle 5: $S_0k_4 + S_1k_3 + S_2k_2 + S_3k_1 + S_4k_0 = 3(-30) + 3(30) + 3(20) + 3(10) + 3(2) = \underline{\mathbf{96}}$
Cycle 6: $S_0k_5 + S_1k_4 + S_2k_3 + S_3k_2 + S_4k_1 + S_5k_0 = 3(-20) + 3(-30) + 3(30) + 3(20) + 3(10) + 3(2) = \underline{\mathbf{36}}$
Cycle 7: $S_0k_6 + S_1k_5 + S_2k_4 + S_3k_3 + S_4k_2 + S_5k_1 + S_6k_0 = 3(-10) + 3(-20) + 3(-30) + 3(30) + 3(20) + 3(10) + 3(2) = \underline{\mathbf{6}}$
Cycle 8: $S_0k_7 + S_1k_6 + S_2k_5 + S_3k_4 + S_4k_3 + S_5k_2 + S_6k_1 + S_7k_0 = 3(-2) + 3(-10) + 3(-20) + 3(-30) + 3(30) + 3(20) + 3(10) + 3(2) = \underline{\mathbf{0}}$

For both of the simulations above, the coefficients used are: 2, 10, 20, 30, -30, -20, -10 and -2.

TABLE II
COMPARISON RESULT USING DIFFERENT DEVICES.

Xilinx ISE Webpack 8.1 Synthesis Tool			
Device	**Frequency (MHz)**	**No. of slices**	**Est. Power Consumption**
Spartan-II	58.65	599	11mW
Virtex-E	66.59	598	7mW

Table II shows comparison using different kind of devices. In order to meet the specifications both devices are using almost the same number of resources. The frequency achieved by using the Virtex E device is higher than using Spartan-II device. The estimated power consumption for Virtex-E is lower than Spartan-II.



System level delay implemented by registered adders.

Fig. 10. Transposed Form FIR Filter Block Diagram.

VIII. CONCLUSION

This paper has studied about FIR filter and how it can be implemented by using FPGA. In a typical digital design, VHDL is a language for describing specifications of designs, simulation of performance and interface to hardware. Then, it is synthesized to create netlist and implemented into FPGA.

The simulation results achieved from Xilinx and Matlab simulation are equal. The FIR filter calculates the output by weighing sum of past and current samples of input, using only some finite number of past samples.

Both device (Spartan and Virtex) produce the same output result. In terms of performance and power consumption, there are slight differences between them. The Virtex-E uses lower power and achieved higher frequency than Spartan-II. Therefore, the FIR filter is better to be implemented on the Virtex-E device because of the performance.

REFERENCES

[1] V. Pasham, A. Miller and K.Chapman, *Transposed Form FIR Filters*, Xilinx Application Note, XAP219, October 25, 2001. (http://www.xilinx.com/xapp/xapp219.pdf)

[2] Leland B. Jackson, *Digital Filters and Signal Processing with MATLAB exercises*, Toppan Kluwer, 1998.

[3] Sudhakar Yalamanchili, *VHDL- A Starter's Guide*, Pearson Prentice Hall, 2005.

[4] Douglas E.Ott, Thomas J.Widerotter, *A Designer's Guide to VHDL Synthesis*, Kluwer Academic Publishers, 1994.

[5] M. Morris Mano, *Digital Design*, Prentice Hall, Third Edition, 2002.

[6] Mark Zwolinski, *Digital System Design with VHDL*, Pearson Prentice Hall, Second Edition, 2004.

[7] Emmanuel C. Ifeachor, Barrie W.Jervis *Digital Signal Processing a Practical Approach*, Addison Wesley, 1993.

[8] Andreas Antoniou, *Digital Filters Analysis, Design and Applications*, Second Edition, Mc Graw Hill, 1996.

[9] Duane Hanselman, Bruce Littlefield, *Mastering MATLAB* 7, Pierson Prentice Hall, 2005.

[10] Kah-Howe Tan, Wen Fung Leong, Kadambari Kaluri, M.A. Soderstand and Louis G. Johnson, *FIR Filter Design Program that Matches Specifications Rather than Filter Coefficients Results in Large Savings in FPGA Resources*, 2001. (http://www.ieeexplore.org)

[11] *Using the Virtex Delay Locked Loop*, Xilinx Application Note, XAP132, January 5, 2006. (http://www.xilinx.com/xapp/xapp132.pdf)