NTNU

Norwegian University of
Science and Technology

Low power/high performance dynamic reconfigurable filter-design

Vebjørn Bystrøm

Master of Science in Electronics
Submission date: June 2008
Supervisor: Kjetil Svarstad, IET

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Problem Description

Digital filters contain a greater number of constant coefficient multipliers and the multipliers can be optimized by using CSD-encoding (Canonical Signed Digit). This kind of optimization can be problematic in applications where one changes the filter coefficients when the coefficient-sets not always are of isomorphic CSD encoding. In reconfigurable hardware (FPGA) this can be solved by dynamically reconfigure the FPGA for the different coefficient-sets where each configuration represents a optimized CSD-encoding. The assignment is as follows.

1. Literature study of CSD-encoding and similar techniques

2. Define a case; a filer with coefficient sets that shall be tested

3. Develop CSD-encoding for the coefficient sets, code them in VHDL modules and test them threw simulation.

4. Test the dynamic CSD-encoding on an FPGA

5. Suggest a method on how to use CSD-encoding if one do not know the coefficients values before run-time.

6. If time left code and test the suggested method.

Assignment given: 15. January 2008
Supervisor: Kjetil Svarstad, IET

NTNU
Norwegian University of
Science and Technology

Digital design

Low power/high performance dynamic reconfigurable filter-design

Vebjørn Bystrøm

Master thesis , Spring 2008
Supervisor at NTNU: Kjetil Svarstad

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Definition of the project

Digital filters contain a greater number of constant coefficient multipliers and the multipliers can be optimized by using CSD-encoding (Canonical Signed Digit). This kind of optimization can be problematic in applications where one changes the filter coefficients when the coefficient-sets not always are of isomorphic CSD encoding. In reconfigurable hardware (FPGA) this can be solved by dynamically reconfigure the FPGA for the different coefficient-sets where each configuration represents a optimized CSD-encoding. The assignment is as follows.

1. Literature study of CSD-encoding and similar techniques
2. Define a case; a filer with coefficient sets that shall be tested
3. Develop CSD-encoding for the coefficient sets, code them in VHDL modules and test them threw simulation.
4. Test the dynamic CSD-encoding on an FPGA
5. Suggest a method on how to use CSD-encoding if one do not know the coefficients values before run-time.
6. If time left code and test the suggested method.

Summary

The main idea behind this thesis was to optimize the multipliers in a finite impulse response (FIR) filter. The project was chosen because digital filters are very common in digital signal processing and is an exciting area to work with.

The first part of the text describes some theory behind the digital filter and how to optimize the multipliers that are a part of digital filters. The substantial thing to emphasize here is the use of Canonical Signed Digits (CSD) encoding. CSD representation for FIR filters can reduce the delay and complexity of the hardware implementation. CSD-encoding reduces the amount of non-zero digits and will by this reduce the multiplication process to a few additions/subtractions and shifts.

In this thesis it was designed 4 versions of the same filter, that was implemented on an FPGA, where the substantial and most interesting results were the differences between coefficients that was CSD-encoded and coefficients that was represented with 2's complement. It was shown that the filter version that had CSD-encoded coefficients used almost 20% less area then the filter version with 2's complement coefficients. The CSD-encoded filter could run on a maximum frequency of 504,032 MHz compared the other filter that could run on a maximum frequency of 249,123 MHz. One of the filters that was designed was designed using the * operator in VHDL, that proved to be the most efficient when it came to the use of number of slices and speed. The reason for this was because an FPGA has built-in multipliers, so if one has the opportunity to use the multiplier they will give the best result instead of using logic blocks on the FPGA

It was also discussed a filter that has the ability to change the coefficients at run-time without starting the design from the beginning. This is an advantage because a constant coefficient multiplier requires the FPGA to be reconfigured and the whole design cycle to be re-implemented. The drawback with the dynamic multiplier is that is uses more hardware resources.

Acknowledgments

I would like to thank my supervisor, Kjetil Svarstad from the department of Electronics and Telecommunications, for giving good guidance and showing great interest in my work.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

OP-AMP	Operational Amplifier
LTI	Linear Time-Invariant
ADC	Analog-to-Digital Converter
DAC	Digital-to-Analog Converter
FIR	Finite Impulse Response
IIR	Infinite Impulse Response
CSD	Canonical Signed Digit
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuits
sra	Shift Right Arithmetic
FPGA	Field-Programmable Gate Array
ASIC	Application-Specific Integrated Circuit
LUT	Lookup Table
DKCM	Dynamic Constant Coefficient Multiplier

Tab. 0.1: Abbreviations

1. INTRODUCTION

Multiplication by a constant value is very useful in many DSP cores; an example is Finite impulse response filters. If the constant values are fixed at design time, the multiplicand can be hardwired since it will never change. One can use some shifts and additions/subtractions to perform the operation instead of using a full multiplier. Smaller, faster and less power consuming circuits will come out of this. As mentioned Finite impulse response filters or another example; discrete cosine transform are central operations in many electrical circuits and applications typically use a large amount of such operations.

In this thesis it is designed 4 versions of the same filter to find out which one uses the least resources without losing functionality. Implementing this on an FPGA gives the opportunity to easily change the coefficients due to the FPGAs flexibility. The central problem is the substitution of full multipliers by an optimized sequence of shifts and additions/subtractions.

Two of the designed versions used CSD-encoding represent the coefficients. These two filters are expected to use less hardware resources then the two other filters. Since the filters are implemented on to an FPGA the version that uses a * operator in VHDL will probably use the built-in multipliers and will give some great results which will be shown in section 4, but since it uses the built in multipliers the filter can not be directly compared to the 3 other filters. It is also anticipated that the last designed filter with 2's complement representation of its coefficients will use the most hardware resources and be slow since 2's complement will often represent a binary number with a substantial number of non-zero digits. In other words this will need to do more calculations and will be slower. A comparison between CSD and 2's complement is described in section 4.2.

The filter that is designed is a Direct Form FIR filter and is not the best structure of a filter but is very easy to design and since the focus is on the multipliers in the filter the different structures that can be used is only discussed in the theory part, section 2.

The first part of the report describes the theory of digital filters, FPGAs, a literature study of CSD-encoding and looks at how to reduce the use of hardware resources, section 2. Furthermore, the designed filters with the description of the coefficients sets are described in detail in section 3, both CSD-encoded and 2's complement. In section 4 the results are presented where we look at the CSD-encoded filters on the FPGA and this section

conclude with a suggestion of a method on how to make the filter dynamic or said in another way a method to easily change the coefficients without starting the design process form the beginning. My contribution to the thesis is to prove that CSD-encoding can easily reduce the use of hardware resources by designing different filters to simulate, implement on an FPGA and analyze Due to limited time the last parts of the thesis definition is only discussed in the results and are not designed and simulated together with an analysis of the results.

2. THEORY

2.1 Digital Filters

In a circuit one often wants to remove noise or extract useful parts of a signal, such as components lying within a certain frequency range. To do this one uses filters. There are two kinds of filters, digital and analog. Analog are build up by resistors, capacitors and op-amps. Analog filters are mathematically modeled using ordinary differential equations of Laplace transforms. They are analyzed in the time of Laplace domain. Digital filters perform numerical calculations on sampled values of a signal. Figure 2.1 shows how a digital filter can be used. By using an analog to digital converter (ADC) one can sample and digitized the unfiltered analog signal. The numerical calculations can then be carried out in a processor. Here the sampled signal is typically multiplied with some coefficients and added together. If an analog output is wanted a digital to analog converter (DAC) can be used as shown in figure 2.1.

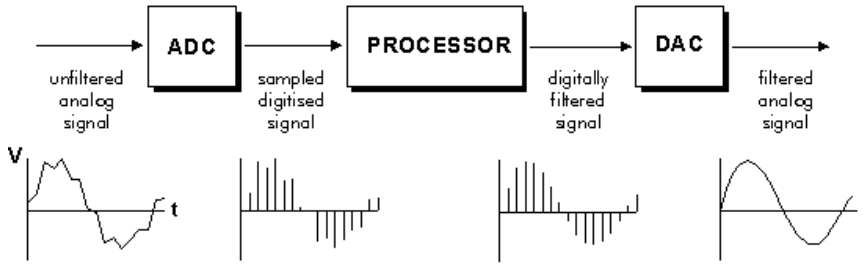

Fig. 2.1: Basic setup of a digital filter

2.1.1 Advantages and disadvantages

+ Digital filters can realize characteristics that is not so easily done by analog filters

+ Digital filters have the potential to attain a much better signal-to-noise ratio.

+ Digital filters are much easier to design, test and implement than an analog filter.

+ Digital filters can handle low frequency signals accurately

- Digital storage and computation limitations will give deterministic quantization errors after the ADC stage.

2.1.2 FIR

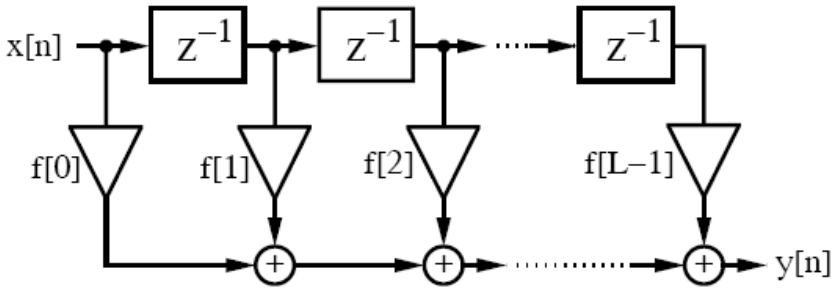

Fig. 2.2: Direct form FIR filter

The most used digital filter is the LTI-filter (linear time-invariant filter). LTI digital filters are generally classified as being finite impulse response (FIR)(shown in figure 2.2) or infinite impulse response (IIR). FIR consists of a finite number of sample values, giving the following definition [17]:

$$y[n] = x[n] * f[n] = \sum_{k}^{L-1} f[k]x[n-k] \tag{2.1}$$

where $f[n]$ is the filter's impulse response, $x[n]$ is the input signal and $y[n]$ is the convolved output. $f[0] \neq 0$ through $f[L-1] \neq 0$ are the filter's L coefficients. y[n] expressed in the z-domain is shown in equation 2.2

$$Y(z) = F(z)X(z) \tag{2.2}$$

where $F(z)$ is the FIR's transfer function defined in the z-domain by

$$F(z) = \sum_{k=0}^{L-1} f[k]z^{-k} \tag{2.3}$$

Figure 2.2 shows a typical FIR filter that we just have described. The "taps" consists of adders, delay module and a multiplier. Each multiplier multiplies a coefficient and the input signal $x[n-L]$

2.1.3 Filter structures

In figure 2.2, [17], one can see a Direct Form FIR filter. The direct form FIR filter are a structure where the multiplier coefficients are precisely the

coefficients of the transfer function. The filter is characterized by $L-1$ coefficients and require $L+1$ multipliers and L two input adders. The Direct Form FIR filter is said to be canonic because the number of delays in the block diagram representation is equal to the order of the transfer function. A noncanonic filter is shown in figure 2.3, [18]

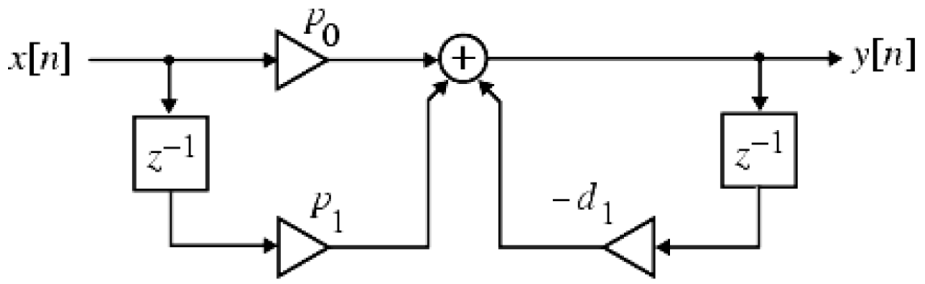

Fig. 2.3: Noncanonic digital filter

By doing the following one can get the transposed structure from the direct form

- Exchanging the input and output
- Inverting the direction of signal flow
- Substituting an adder by a fork and vice versa

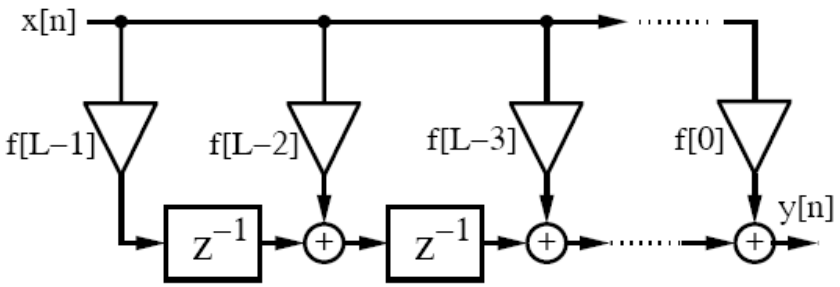

Fig. 2.4: Transposed structure of a FIR filter

A transposed version of the direct form structure is shown in figure 2.4, [18].Both the direct form and the transposed structure are canonic with respect to delays.

Other structures can be Cascade Form, Linear-Phase and Polyphase structure. We will take a short look at there structures.

A cascade form FIR structures is shown in figure 2.5, [18]. The higher-order FIR transfer function is realized as a cascade of second-order FIR sections. $H(z)$ is expressed as shown below and in [18]

$$H(z) = h[0] \prod_{k=1}^{K} 1 + \beta_{1k} z^{-1} + \beta_{2k} z^{-2} \tag{2.4}$$

where $K = L/2$ if L is even and $K = (L+1)/2$ if L is odd with $\beta_{2K} = 2$

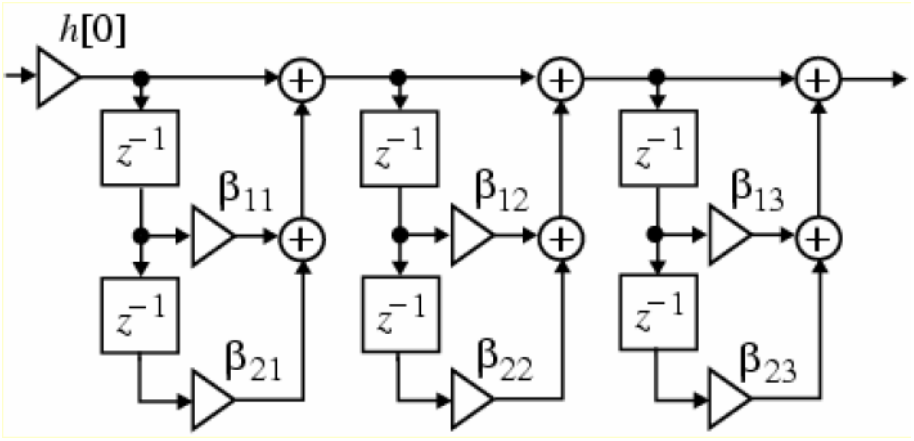

Fig. 2.5: Cascade Form FIR structure

The linear-Phase FIR structure is used to reduce the number of multipliers to almost half compared with the direct form. Take a look at the following transfer function.

$$H(z) = h[0]+h[1]z^{-1}+h[2]z^{-2}+h[3]z^{-3}+h[2]z^{-4}+h[1]z^{-5}+h[0]z^{-6} \tag{2.5}$$

Rewriting the transfer function $H(z)$ gives the following $H(z)$

$$H(z) = h[0](1+z^{-6}) + h[1](z^{-1}+z^{-5}) + h[2](z^{-2}+z^{-4}) + h[3]z^{-3} \tag{2.6}$$

this will give the circuit shown in figure 2.6, [18]

This linear-phase structure requires 4 multipliers whereas a direct form realization requires 7 multipliers.

To show how the Polyphase FIR structures is build up we will take a look at the transfer function shown in equation 2.7. One takes this function and decompose it so it leads to a parallel form structure.

$$\begin{aligned} H(z) = h[0] + h[1]z^{-1} + h[2]z^{-2} + h[3]z^{-3} + h[4]z^{-4} \\ + h[5]z^{-5} + h[6]z^{-6} + h[7]z^{-7} + h[8]z^{-8} \end{aligned} \tag{2.7}$$

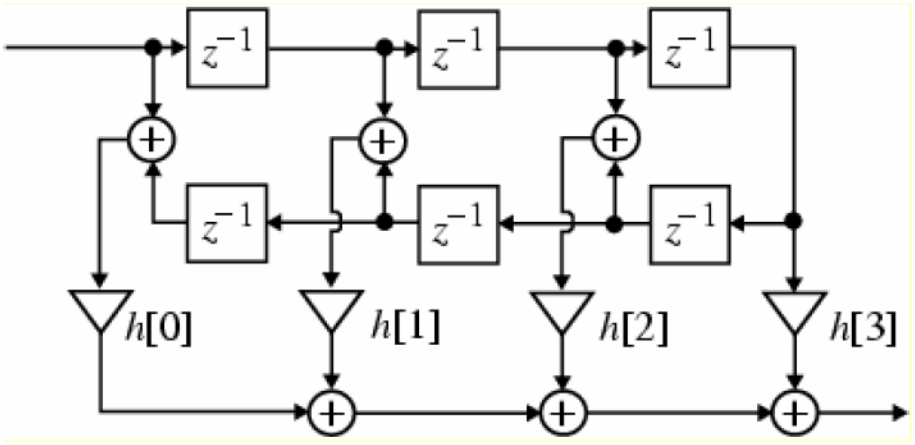

Fig. 2.6: Linear-Phase FIR structure

Odd and even indexed coefficients are split up in a sum of two terms as shown in equation 2.8

$$\begin{aligned} H(z) &= (h[0] + h[2]z^{-2} + h[4]z^{-4} + h[6]z^{-6} + h[8]z^{-8}) \\ &\quad + (h[1]z^{-1} + h[3]z^{-3} + h[5]z^{-5} + h[7]z^{-7}) \\ &= (h[0] + h[2]z^{-2} + h[4]z^{-4} + h[6]z^{-6} + h[8]z^{-8}) \\ &\quad + z^{-1}(h[1] + h[3]z^{-2} + h[5]z^{-4} + h[7]z^{-6}) \end{aligned} \tag{2.8}$$

The polyphase decomposition of $H[z]$ will then become

$$H(z) = E_0(z^2) + z^{-1}E_1(z^2) \tag{2.9}$$

where

$$E_0(z) = h[0] + h[2]z^{-1} + h[4]z^{-2} + h[6]z^{-3} + h[8]z^{-4} \tag{2.10}$$

$$E_1(z) = h[1] + h[3]z^{-1} + h[5]z^{-2} + h[7]z^{-3} \tag{2.11}$$

Figure 2.7, [18], shows the polyphase realization of a transfer function $H(z)$

2.1.4 Bit-serial vs. bit-parallel architectures

The bit-serial architecture is area-efficient but has a disadvantage in time efficiency. It may be desirable to combine the area efficiency of a bit-serial architecture with the time efficiency of a corresponding bit-parallel architecture into a single area-efficient and time efficient digit serial architecture.

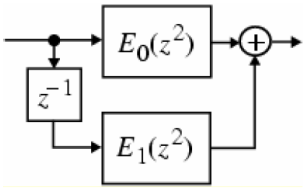

Fig. 2.7: Polyphase FIR Structure

In a digit serial arithmetic implementation, the W bits of a data word are processed in units of the digit size N in W/N clock cycles. This leads to arithmetic operators that have smaller area than equivalent parallel arithmetic designs and have a lager througput than equivalent parallel arithmetic designs.

Architectures based on the digit-serial approach may offer the best overall trade-off between speed, efficient area utilization, throughput, I/O pin limitations and power consumption. In other words this will be an effective implementation style for FPGAs.

2.2 Optimizations to reduce the use of hardware resources

When implementing a FIR filter on an FPGA or such one wants to reduce the total number of 1's required in the coefficient's two's complement representation. There are two methods that we will take a look at, Booth encoding and Canonic Signed Digit (CSD). There are also some algorithms that is used to reduce the hardware which can be studied in detail in [1], [2], [3] and [4]

2.2.1 Booth encoding

Booth multiplication algorithm multiplies two signed binary numbers in two's complement notation. By using the Booth algorithm it is possible to reduce the number of partial products by half. Instead of shifting and adding for every column of the multiplier term and multiplying by 1 or 0, one can take every second column and multiply by ± 1, ± 2 or 0. The Booth algorithms works as shown below and described in [5]

Booth's algorithm involves repeatedly adding one of two predetermined values A and S to a product P, then performing a rightward arithmetic shift on P. Let x and y be the multiplicand and multiplier, respectively; and let x and y represent the number of bits in $\mathbf{x}$ and $\mathbf{y}$.

1. Determine the values of A and S, and the initial value of P. All of these numbers should have a length equal to $(x + y + 1)$.
   (a) A: Fill the most significant (leftmost) bits with the value of x. Fill the remaining $(y + 1)$ bits with zeros.

(b) S: Fill the most significant bits with the value of $(-x)$ in two's complement notation. Fill the remaining $(y+1)$ bits with zeros.

(c) P: Fill the most significant x bits with zeros. To the right of this, append the value of $\mathbf{y}$. Fill the least significant (rightmost) bit with a zero.

2. Determine the two least significant (rightmost) bits of P.

   (a) If they are 01, find the value of $P + A$. Ignore any overflow.

   (b) If they are 10, find the value of $P + S$. Ignore any overflow.

   (c) If they are 00 or 11, do nothing. Use P directly in the next step.

3. Arithmetically shift the value obtained in the previous step by a single place to the right. Let P now equal this new value.

4. Repeat steps 2 and 3 until they have been done y times.

5. Drop the least significant (rightmost) bit from P. This is the product of $\mathbf{x}$ and $\mathbf{y}$.

We can now look at an example to show that we use Booth to multiply two numbers and still reduce the number of partial products compared with regular adding and shifting [5].

We shall find $3 * -4$, $x = 3$ and $y = -4$ [5]

$$A = 001100000 \tag{2.12}$$
$$S = 110100000 \tag{2.13}$$
$$P = 000011000 \tag{2.14}$$

We perform the loop four times:

1. $P = 000011000$. The last two bits are 00.
   - $P = 000001100$. Arithmetic right shift.
2. $P = 000001100$. The last two bits are 00.
   - $P = 000000110$. Arithmetic right shift.
3. $P = 000000110$. The last two bits are 10
   - $P = 110100110$. $P = P + S$.
   - $P = 111010011$. Arithmetic right shift.
4. $P = 111010011$. The last two bits are 11.
   - $P = 111101001$. Arithmetic right shift

The product is 11110100, which is -12

2.2.2 Canonical Signed Digit

The Canonical Signed Digit (CSD) number system is a signed digit number system that minimize the number of non-zero digits. This can be used to reduce the number of adds in a logic circuit. The digit set is ternary and each digit can be either. -1, 0, or $+1$. CSD digits that are beside each other are never both $= 1$. This implies that for an n-bit number, there are at most $[n/2]$ non-zero digits. For a 2's complement number, all the digits can be *ones* which is not hardware friendly when it comes to area and power consumption. It can also be shown that the probability of a digit being zero is roughly 2/3 for CSD and exactly 1/2 for 2´s complement [14]. The following table shows an example for $n = 3$ where $\bar{1}$ represents -1. As can be seen in Table 2.1, for negative numbers the number of non-zero digits is less for the CSD representation than the 2´s complement representation, 9 versus 12. [14]

Number	*2´s Complement*	*Canonical Signed Digit*
3	011	$10\bar{1}$
2	010	010
1	001	001
0	000	000
-1	111	$00\bar{1}$
-2	110	$0\bar{1}0$
-3	101	$\bar{1}01$
-4	100	$\bar{1}00$

Tab. 2.1: Three Digit Canonical Signed Digit Numbers.

CSD encoding is similar to Booth encoding and can be accomplished by analyzing pairs of adjacent digits as shown in 2.2.1. If the number is negative, the MSB+1 (x_n^*) is 1, otherwise, it is a 0. This can be implemented as a copy of the MSB. Table 2.2 and the flow chart of Figure 2.8 provide a method to convert 2´s complement numbers to CSD numbers [14]. This method was used to create Table 1. The digits x_i and x_{i+l} are adjacent digits of the 2´s complement number and the digits, c_i, are the CSD digits. The flow chart shows the algorithm to convert 2´s complement filter coefficients, X, to CSD representation of the filter coefficient, C, where $X = x *_n x_{n-1}x_{n-2}x_{n-3}\ldots x_3x_2x_1x_0$ and $C = c_{n-1}c_{n-2}c_{n-3}\ldots c_3c_2c_1c_0$. [14]

The following are the properties of CSD numbers:

- No 2 consecutive bits in a CSD number are non-zero.
- The CSD representation of a number contains the minimum possible number of non-zero bits, thus the name canonic.

Carry-in	x_{i+1}	x_i	*carry-out*	c_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	-1
1	0	0	0	1
1	0	1	1	0
1	1	0	1	-1
1	1	1	1	0

Tab. 2.2: Three Digit Canonical Signed Digit Numbers.

- The CSD representation of a number is unique.
- CSD numbers cover the range (-4/3,4/3), out of which the values in the range [-1,1) are of greatest interest.
- Among the W-bit CSD numbers in the range [-1,1), the average number of non-zero bits is W/3 + 1/9 + O(2-W). Hence, on average, CSD numbers contains about 33 % fewer non-zero bits than two´s complement numbers.

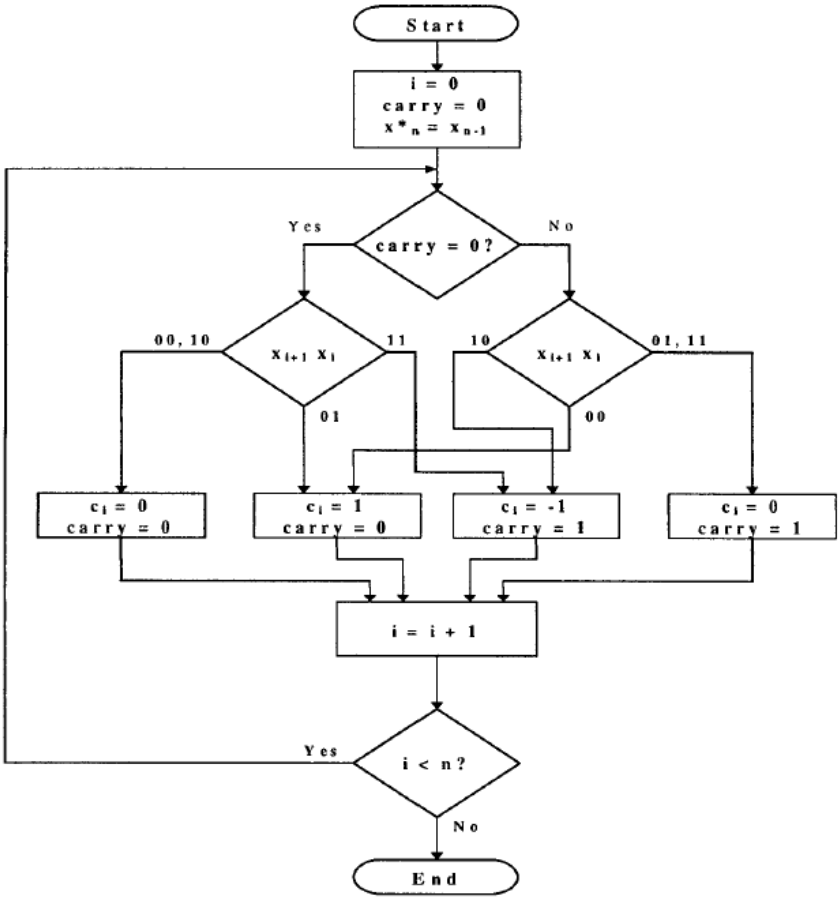

Fig. 2.8: Flow chart of conversion of two's complement to CSD

CSD in filters

Problems with high speed filters:

- Digital Finite Impulse Response (FIR) filters are widely used in digital processing
- Multipliers in filter are the most complex hardware realization which increases the cost of filters
- Multiplication is the most time and power consuming process limits varity of high speed applications.

The solution to the problems mentioned over is to eliminate the multipliers from filter by employing CSD represented filter coefficients to increase the speed and reduce hardware complexity. CSD representation can help high speed FIR digital filter design because there will become less nonzero digits in a CSD form cuts number of addition stages and thereby also reduce both hardware complexity and power consumption. The number of adder/-subtracter required to realize a CSD coefficient is one less then the number of nonzero digits in the code.

For example consider a filter coefficient in 8-bit with value $= 0,9921875$

$$0.9921875_{10} = 0.1111111_2 = 1.000000\bar{1}_{csd} = 2^0 - 2^{-7} \tag{2.15}$$

One can observe above that one uses only one subtracter as opposed to seven adders

2.3 FPGA

An FPGA (field-programmable gate array) is a semiconductor device containing programmable logic components called logic blocks, and programmable interconnects. Figure 2.9 shows the architecture of an FPGA [16], [9]. Logic blocks can be programmed to perform the function of basic logic gates, mathematical functions or decoders. FPGAs were introduced as an alternative to custom ICs. With the help of computer aided design tools circuits can be implemented in a short amount of time. The benefits with FPGAs are no physical layout process, no mask making and no IC manufacturing.

Figure 2.10 shows how FPGA is more flexible compared to ASICs but not so flexible compared to processors [9].

There are some different vendors that deliver FPGAs, Xilinx and Altera is two of the vendors. Families of FPGAs differ in:

- physical means of implementing user programmability
- arrangement of interconnections wires

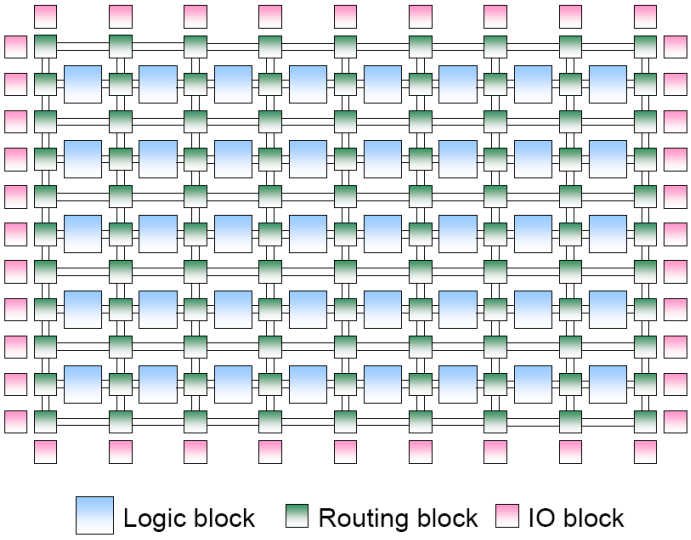

Fig. 2.9: FPGA Architecture

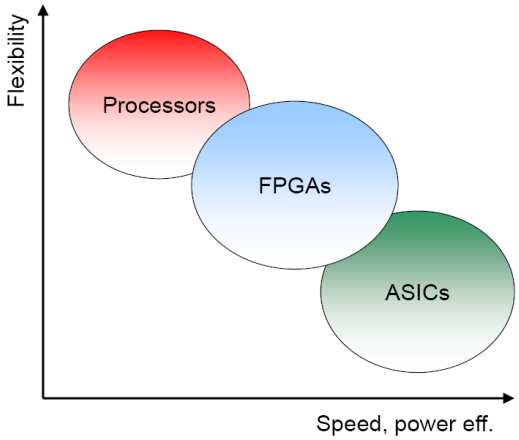

Fig. 2.10: FPGA vs. ASIC vs. Processors

- the basic functionality of the logic blocks
- flexibility of the logic blocks and connections

Figure 2.11 shows how a logic block is built up. It contains of a LUT (look up table) which implements combinational logic functions, a register that stores output of LUT (optional), [9].

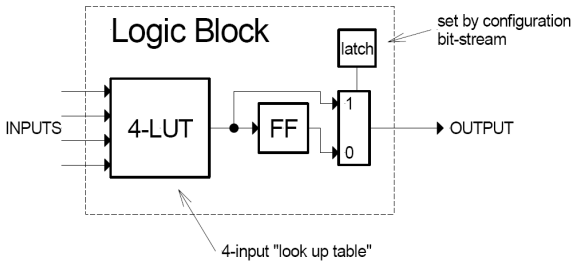

Fig. 2.11: Logic Block

Figure 2.12 and 2.3 shows how equation 2.16 can be implemented in a LUT, [8].

$$q = (a_0 AND a_1) OR (a_2 AND a_3) \tag{2.16}$$

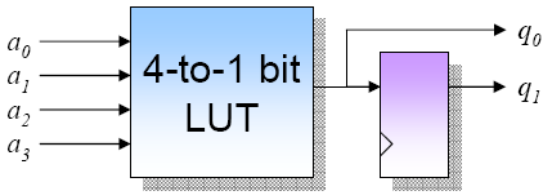

Fig. 2.12: Logic Block example

a_0	a_1	a_2	a_3	q
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Tab. 2.3: LUT implementation, truth tabel

The design flow on an FPGA is as follows[9]:

- Design Entry:
  - Create your design files using schematic editor or hardware description language (Verilog or VHDL
- Design Implementation
  - partition, place and route to create bit-stream file
- Design Verification
  - Use simulator to check function
  - Load onto FPGA device via data cable from computer
  - check operation at full speed in real environment

The design flow is shown graphically in figure 2.13, [9]

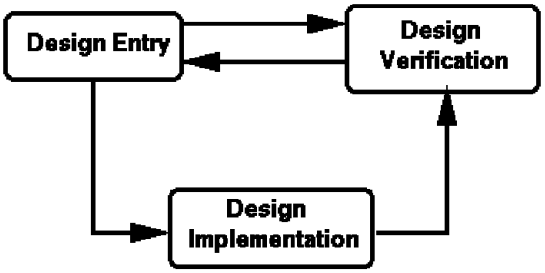

Fig. 2.13: Design Flow to an FPGA

3. MODELING AND SIMULATION

As shown in the section 2 digital filters consists of a lot of constant multipliers. Multiplying two numbers uses a lot of hardware resources. The central problem then becomes substituting the full multipliers by an optimized sequence of shift, additions and subtractions. In this thesis I have designed 4 version of a digital filter. Each version has its own method to multiply the coefficient with the input signal. The version is described in details in section 3.1. The whole filter is shown in figure 3.1 The digital filters characteristics is shown in table 3.1, the frequency response is shown in figure 3.2 and the filters coefficients is shown in table 3.2 [6].

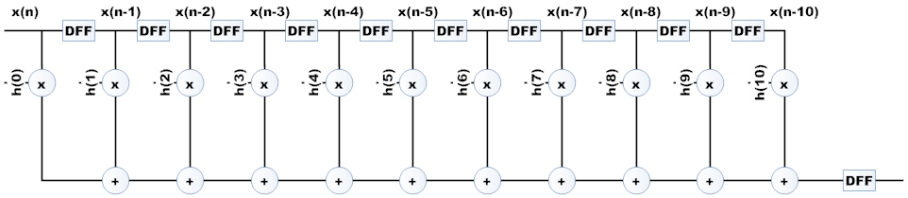

Fig. 3.1: The designed digital FIR filter

Filter type	Low pass
Window type	Hanning
Filter order	10
Passband	0 to 1000 Hz
Stoppband attenuation	44 dB
Transiton bandwidth	2488 Hz

Tab. 3.1: Digital Filter characteristics

k	$h(k)$
0	0,002437094
1	0,013715162
2	-0,044250023
3	-0,044364337
4	0,28976238
5	0,5575594
6	0,28976238
7	-0,044364337
8	-0,044250023
9	0,013715162
10	0,002437094

Tab. 3.2: Digital Filter coefficients

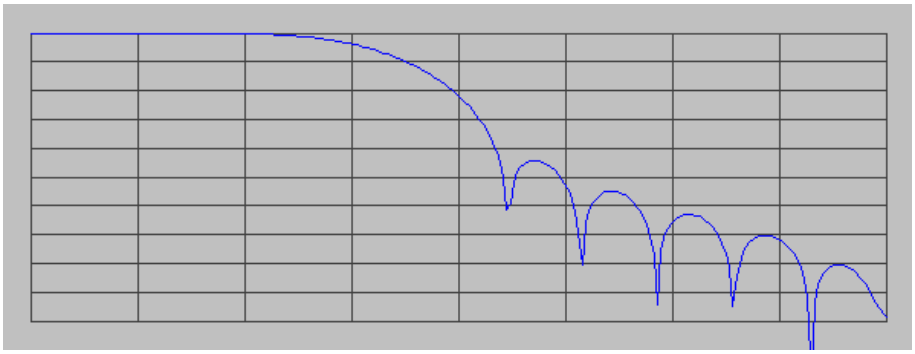

Fig. 3.2: Frequency response of the designed filter

3.1 Filter Versions

Like mentioned above I have designed 4 versions of the same filter by using VHDL. All the filters are Direct Form FIR filter as explained in section 2.1.3 and in [7]. The differences of the filters are described below.

All the filters are built up by two main files; one main VHDL file (my_fir.vhd) that describes the whole filter and one file (my_functions.vhd) that returns the multiplication between the input signal and the coefficients. The whole code is attached in appendix A to D

The difference between the versions are in the my_functions.vhd files where the multiplications are done. Section 3.1.1 to 3.1.4 describes the differences.

The filter coefficients, $h(k)$, are shown in tables 3.3 and 3.4. All the coefficients are scaled by 10^4 and rouned to the nearest integrer. This scaling is done so that the largest coefficient is less then the largest, 16, bit, 2's complement number.

3.1.1 Filter version 1

In filter version 1 it is used CSD encoding to represent the coefficients. This technique is used to reduce the number of 1's. The CSD representation is a signed power-of-two representation where each of the digits is in the set $1, 0, \bar{1}$. $1-$addition, $0-$no operation and $\bar{1}-$subtraction. The theory behind CSD is described in section 2.2.2. Using CSD representation for the coefficients implies that the multiplication can be conducted in a shift, subtract and add fashion using the lowest number of operations. The coefficients are shown in table 3.3

k	*h(k)*	*h(k) scaled*	*h(k)rounded*	*h(k) 16-bit, CSD encoded*
0	0,002437094	24,37094	24	0000000000101̄000
1	0,013715162	137,15162	137	0000000010001001
2	-0,044250023	-442,50023	-443	000000 1̄001000101
3	-0,044364337	-443,64337	-444	000000 1̄001000100
4	0,28976238	2897,6238	2898	00010 1̄0 1̄01010010
5	0,5575594	5575,594	5576	00101 1̄0 1̄00 1̄001000
6	0,28976238	2897,6238	2898	00010 1̄0 1̄01010010
7	-0,044364337	-443,64337	-444	000000 1̄001000100
8	-0,044250023	-442,50023	-443	000000 1̄001000101
9	0,013715162	137,15162	137	0000000010001001
10	0,002437094	24,37094	24	0000000000101̄000

Tab. 3.3: CSD representation of the coefficients

Each function that returns the multiplication between the coefficient and the input signal is described in VHDL as show below.

```
FUNCTION coeff_5 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE temp : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift2 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift3 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift4 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift5 :std_logic_vector (23 DOWNTO 0);

        BEGIN
                temp := c & "0000000000000000";
                shift1 := (temp sra 2);
                shift2 := (temp sra 4);
                shift3 := (temp sra 6);
                shift4 := (temp sra 9);
        shift5 := (temp sra 12);

                return (shift1 - shift2 - shift3 - shift4 + shift5);
END coeff_5;
```

Here c (the input signal) is shifted the number of times depending on the coefficient and then all the shifts are added/subtracted together. One can see that the **sra**(shift right arithmetic) opertator is used. The VHDL code for the shift operators can be found in Appendix E. In the example above coefficient 5 is used. The multiplication can be described as shown in equation 3.1

$$return = (c << 2) - (c << 4) - (c << 6) - (c << 9) + (c << 12) \quad (3.1)$$

All the coefficients are carried out the same way as coefficient 5 and the whole code for version 1 can be seen in appendix A

3.1.2 Filter version 2

Filter version 2 is nearly identical to version 1. Version 2 uses also CSD encoding which will give the lowest number of operations. The differences is that in version 2 no shift operations are used. This is done to find out if this filter structure will be more area efficient. The function of coefficient 5 will then be coded in VHDL as shown below

```
FUNCTION coeff_5 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE shift1 : std_logic_vector (17 DOWNTO 0);
                VARIABLE shift2 :std_logic_vector (17 DOWNTO 0);
                VARIABLE shift3 :std_logic_vector (17 DOWNTO 0);
                VARIABLE shift4 :std_logic_vector (17 DOWNTO 0);

        BEGIN
                shift1 := (c(7)&c(7 downto 0)
                                &"000000000");

                shift2 := (c(7)&c(7)&c(7)&c(7)
                                &c(7 downto 0)&"000000");

                shift3 := (c(7)&c(7)&c(7)&c(7)
                                &c(7)&c(7)&c(7)&c(7 downto 0)&"000");

                shift4 := (c(7)&c(7)&c(7)&c(7)
                                &c(7)&c(7)&c(7)&c(7)&c(7)&c(7 downto 0)&"0");

                return (shift1 + shift2 - shift3 - shift4);

        END coeff_5;
```

The whole code can be seen in appendix B

3.1.3 Filter version 3

Filter version 3 uses the multiplication operator * in VHDL. The reason for designing with this operator is to find out how much more resources are used on the FPGA compared to only shifting and adding. Again we look at the function that returns the multiplication of the input signal and coefficient 5:

```
FUNCTION coeff_5 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE temp : std_logic_vector (23 DOWNTO 0);


        BEGIN
                temp := c * "0001010111001000";


                return temp;

        END coeff_5;
```

Here we just mutiply c with coefficient 5 by using the * operator. The whole code can be seen in appendix C

3.1.4 Filter version 4

Filter version 4 is implemented with 2's complement number representatsion. The binary representation is shown in figure 3.4. Similar to filter version 1 the multiplication can be conducted in a shift, subtract and add fashion. The multiply functions are carried out in the same way as version 1 using shift operators as shown in the VHDL code below

```
        FUNCTION coeff_5 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE temp : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift2 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift3 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift4 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift5 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift6 :std_logic_vector (23 DOWNTO 0);

        BEGIN
                temp := c & "0000000000000000";
                temp := (temp sra 16);
                shift1 := (temp srl 3);
                shift2 := (temp srl 6);
                shift3 := (temp srl 7);
                shift4 := (temp srl 8);
                shift5 := (temp srl 10);
                shift6 := (temp srl 12);

                return (shift1 + shift2 + shift3
                        + shift4 + shift5 + shift6);

        END coeff_5;
```

The multiplication can be described as shown in equation 3.2

$$\begin{aligned} return = (c << 3) + (c << 6) + (c << 7) \\ + (c << 8) + (c << 10) + (c << 12) \end{aligned} \tag{3.2}$$

The whole code can be seen in appendix D

k	*h(k)*	*h(k) scaled*	*h(k)rounded*	*h(k) 16-bit, 2's complement*
0	0,002437094	24,37094	24	0000000000011000
1	0,013715162	137,15162	137	0000000010001001
2	-0,044250023	-442,50023	-443	1111111001000101
3	-0,044364337	-443,64337	-444	1111111001000100
4	0,28976238	2897,6238	2898	0000101101010010
5	0,5575594	5575,594	5576	0001010111001000
6	0,28976238	2897,6238	2898	0000101101010010
7	-0,044364337	-443,64337	-444	1111111001000100
8	-0,044250023	-442,50023	-443	1111111001000101
9	0,013715162	137,15162	137	0000000010001001
10	0,002437094	24,37094	24	0000000000011000

Tab. 3.4: 2's complement representation of the coefficients

4. RESULTS AND COMPARISONS

Simulation, testing of the code and implementation was done in ModelSim [10] and Xilinx ISE [11]. The device properties is shown in table 4.1. The area is measured in number of slices. The required area compared to the 4656 available slices in the XC3S500E device. The delays are expressed in nanoseconds and the number of additions/subtractions is computed by Xilinx Project Navigator [11].

Property name	*Value*
Product Category	All
Family	Spartan 3E
Device	XC3S500E
Package	FG320
Speed	-4

Tab. 4.1: Device Properties

4.1 Results from simulation of the filter versions

Table 4.2 and 4.3 shows the utilizations summary of the different versions of the designed filter. We can clearly observe that version 3 uses the least number of slices but since the FPGA has built-in multipliers so this will not give a good picture of the used number of slices. So the interesting versions to look at are version 1, 2 and 4. Here we can see that version 1 and version 2 are nearly identical since they both use CSD-encoding. Version 2 has an improvement of 22% when it comes to number of slices compared to version 1. Version 3 that uses 2's complement uses a lot more of area compared to the CSD-encoded versions. Version 2 has an improvment of 36% then version 4.

Table 4.4 shows HDL Synthesis Report from Xilinx Project Navigator [11]. Again we can observe that version 3 of the designed filter uses the built-in multipliers so the number of adders and subtractors are low so version 3 can not be directly compared to the other versions. Version 1 and version 2 use CSD-encoding and have a low number of adders and subtractors compared to version 4 that uses 2's complement representation of the coefficients. Version 2 has a 52% improvement compared to version 4. One

can also observe that version 4 do not use subtractors, this because 2's complement represents each digit with 0 or 1 while version 1 and 2 uses encoding is ternary and each digit can be either -1, 0 or 1.

	Version 1	*Version 2*
Number of Slices	343 out of 4656 (7,4%)	267 out of 4656 (5,7%)
Number of Slice Flip Flops	99 out of 9312 (1,1%)	92 out of 9312 (0.98%)
Number of 4 input LUTs	628 out of 9312 (6,7%)	484 out of 9312 (5,2%)
Number of IOs	34	34
Number of bonded IOBs	34 out of 232	34 out of 232
Number of MULT18X18SIOs	-	-
Number of GCLKs:	1 out of 24	1 out of 24

Tab. 4.2: Device utilization summary, Version 1 and Version 2

	Version 3	*Version 4*
Number of Slices	130 out of 4656 (2,7%)	418 out of 4656 (8,9%)
Number of Slice Flip Flops	88 out of 9312 (0,95%)	121 out of 9312 (1,3%)
Number of 4 input LUTs	227 out of 9312 (2,4%)	728 out of 9312 (7,8%)
Number of IOs	34	34
Number of bonded IOBs	34 out of 232	34 out of 232
Number of MULT18X18SIOs	11 out of 20	-
Number of GCLKs:	1 out of 24	1 out of 24

Tab. 4.3: Device utilization summary, Version 3 and Version 4

Filter Version	*Adders/subtractors*	*Adders*	*Subtractors*	*Registers*	*Multipliers*
1	40	27	13	88	-
2	31	19	12	88	-
3	10	10	-	88	11
4	65	65	-	88	-

Tab. 4.4: HDL Synthesis Report

Table 4.5 and 4.6 shows the timing summary of the filters versions. Before comparing the filters the expression will shortly explained and can be found in detail at [11].

- Minimum Period - The maximum delay from any synchronous element to another.
- Maximum Period - The maximum delay from any synchronous element to another; displayed in MHz. These paths can run at this frequency.

- Minimum input arrival time before clock - The minimum global OFFSET IN BEFORE.
- Maximum output required time after clock - The maximum global OFFSET IN AFTER.

In table 4.5 and 4.6 one can see that version 3 of the of filters has the highest frequency. This again is because the FPGA has built-in multipliers. Comparing version 1, 2 and 4 one can identify that version 1 and 2, which uses CSD encoding, has a much higher maximum frequency then version 4 that represent the coefficients with 2's complement. This is because version 4 has lot more logic and will be a litter slower then version 1 and 2. It should be mentioned that if one had used another filter structure one had gotten a lot better timing number. This because the Direct Form that is used, adds all taps together at the end and takes a lot of time.

	Version 1	*Version 2*
Minimum period	1,984 ns	1,950 ns
Maximum Frequency	504,032 MHz	512,821 MHz
Minimum input arrival time before clock	1,946 ns	1,973 ns
Maximum output required time after clock	40,720 ns	32,072 ns

Tab. 4.5: Timing Summary, Version 1 and Version 2

	Version 3	*Version 4*
Minimum period	1,346 ns	4,014 ns
Maximum Frequency	742,942 MHz	249,128 Mhz
Minimum input arrival time before clock	1,946 ns	1,946 ns
Maximum output required time after clock	28,509 ns	49,355 ns

Tab. 4.6: Timing Summary, Version 3 and Version 4

The RTL schematic of version 2 is shown in figure 4.1 and in more depth in figure 4.2 and 4.3

4.2 2's complement vs. CSD representation

Tables 4.7 and 4.8 shows the total number of adds and shifts. As one can observe the former drops from 55 to 30 adds/subtracts and the latter drops from 62 to 41 that is an improvement of 46% when it comes to adds and subtracts and 34% when it comes to shifts. This will in other word be a substantial improvement when it comes to the use of hardware resources. It should be mentioned that the add and the subtract are considered as the same cost operation. The improvement is very clearly at coefficient number 2 ($k = 2$). If one look at the equations 4.1 and 4.2 one can observe that 2's

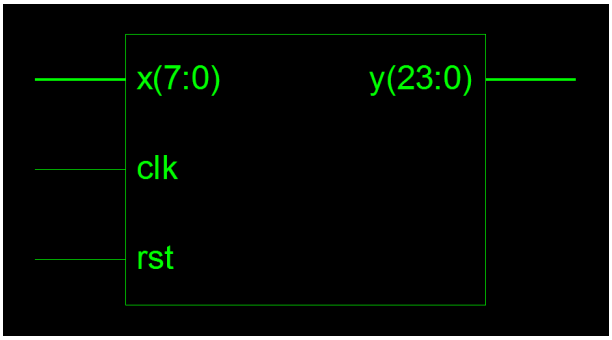

Fig. 4.1: Schematic of FIR filter version 2

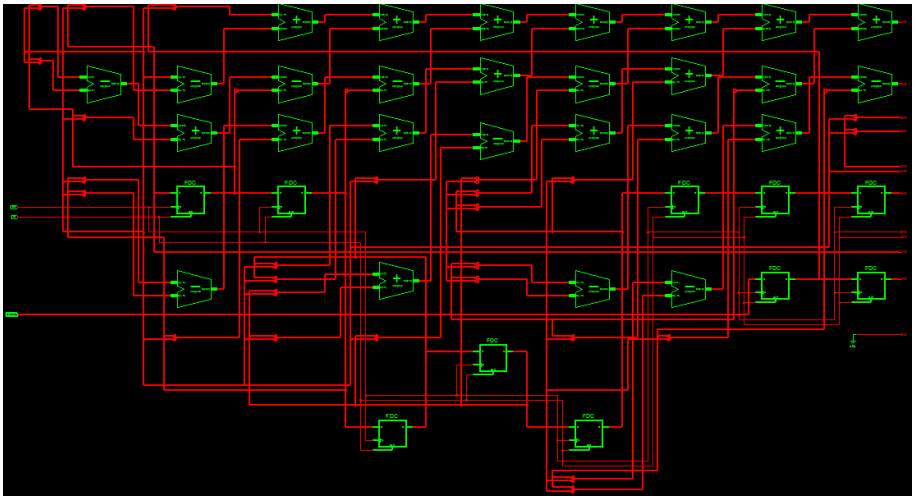

Fig. 4.2: RTL schematic of FIR filter version 2, 1/2

k	*h(k)*	*h(k) 16-bit, 2's complement*	*Total Adds*	*Total Shifts*
0	0,002437094	0000000000011000	1	2
1	0,013715162	0000000010001001	2	2
2	-0,044250023	1111111001000101	9	9
3	-0,044364337	1111111001000100	8	9
4	0,28976238	0000101101010010	5	6
5	0,5575594	0001010111001000	5	6
6	0,28976238	0000101101010010	5	6
7	-0,044364337	1111111001000100	8	9
8	-0,044250023	1111111001000101	9	9
9	0,013715162	0000000010001001	2	2
10	0,002437094	0000000000011000	1	2
		Total	**55**	**62**

Tab. 4.7: Total adds and shifts when using 2's complement representation of the coefficients

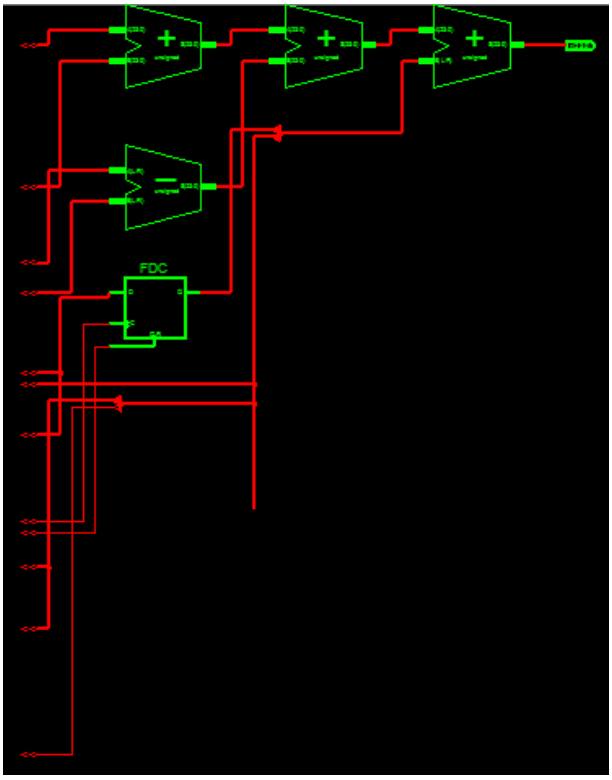

Fig. 4.3: RTL schematic of FIR filter version 2, 2/2

k	*h(k)*	*h(k) 16-bit, CSD encoded*	*Total Adds/Subtracts*	*Total Shifts*
0	0,002437094	0000000000101$\bar{1}$000	1	2
1	0,013715162	0000000010001001	2	3
2	-0,044250023	000000$\bar{1}$001000101	3	4
3	-0,044364337	000000$\bar{1}$001000100	2	3
4	0,28976238	00010$\bar{1}$0$\bar{1}$01010010	5	6
5	0,5575594	0010$\bar{1}$0$\bar{1}$00$\bar{1}$001000	4	5
6	0,28976238	00010$\bar{1}$0$\bar{1}$01010010	5	6
7	-0,044364337	000000$\bar{1}$001000100	2	3
8	-0,044250023	000000$\bar{1}$001000101	3	4
9	0,013715162	0000000010001001	2	3
10	0,002437094	0000000000101$\bar{1}$000	1	2
		Total	**30**	**41**

Tab. 4.8: Total adds and shifts when using CSD representation of the coefficients

complement representation (equation 4.2) uses a lot more adds and shifts to multiply the input signal and the coefficient then CSD representation (equation 4.1). One can also clearly see this in figure 4.4 and 4.5 which shows the RTL schematics of the coefficient 2 when using CSD and 2's complement representation.

$$y_{binary} = -(c << 6) + (c << 9) + (c << 13) + (c << 15) \tag{4.1}$$

$$\begin{aligned} y_{CSD} = {} & (c << 2) + (c << 6) + (c << 9) + (c << 10) + (c << 11) \\ & + (c << 12) + (c << 13) + (c << 14) + (c << 15) \end{aligned} \tag{4.2}$$

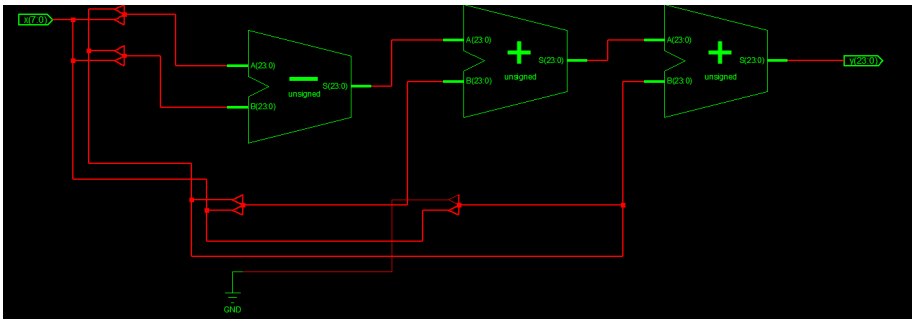

Fig. 4.4: RTL schematic of multiplication with coefficient 2, CSD representation

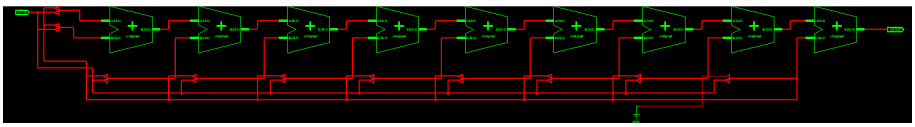

Fig. 4.5: RTL schematic of multiplication with coefficient 2, 2's complement representation

One can also see from the synthesis report that using CSD representation on coefficient 2 reduces the number of slices with 62% and the number of LUTs with 60% as shown in tabel 4.9

	CSD representation	*2's complement representation*
Number of Slices	19 out of 4656	49 out of 4656
Number of 4 input LUTs	36 out of 9312	90 out of 9312
Number of IOs	26	32
Number of bonded IOBs:	26 out of 232	32 out of 232

Tab. 4.9: Device utilization summary, coefficient 2

4.3 Dynamic Constant Coefficient Multiplier

Until now we have only discussed Constant Coefficient Filters, or in other words we have looked at how to optimize a multiplier when we know the coefficients. Some times one need to change the coefficients and then one will meet a challenge. The change of a coefficient value requires not only the FPGA to be reconfigured but also the whole design cycle to start over. This causes the change of a coefficient to take time and this can not be performed while the circuit is running.

A solution is to make a Dynamic Constant Coefficient Multiplier (DKCM), [15]. The DKCM implements LUT based Multiplications (figure 4.6 shows a LUT based multiplier). If one wants to change the coefficients one can change the LUT memory contents. The solution can implement in-circuit coefficient reconfiguration. By doing this one do not need to reimply the design fitting into the FPGA structure. The drawback is that this will use more area on the chip then a constant coefficient multiplier, which is a substantial drawback. As mentioned the idea behind the dynamic change of a coefficient value is to properly change the contents of the memories. To do this one needs an extra RAM programming interface[15]. The RAM programming interface has two functions as described below.

1. Allows the RAMs to be programmed.

2. The RAM programming unit which produces proper data sequences and control signals for RAM programming

Figure 4.7 shown how a DKCM circuit is built up. One can see that this is an expansion of figure 4.6.

An alternative solution is programming RAMs using an FPGA partial reconfiguration instead of the RAM programming unit but this may not be accepted by a given FPGA. This also requires longer reconfiguration time and pre-calculated RAM contents.

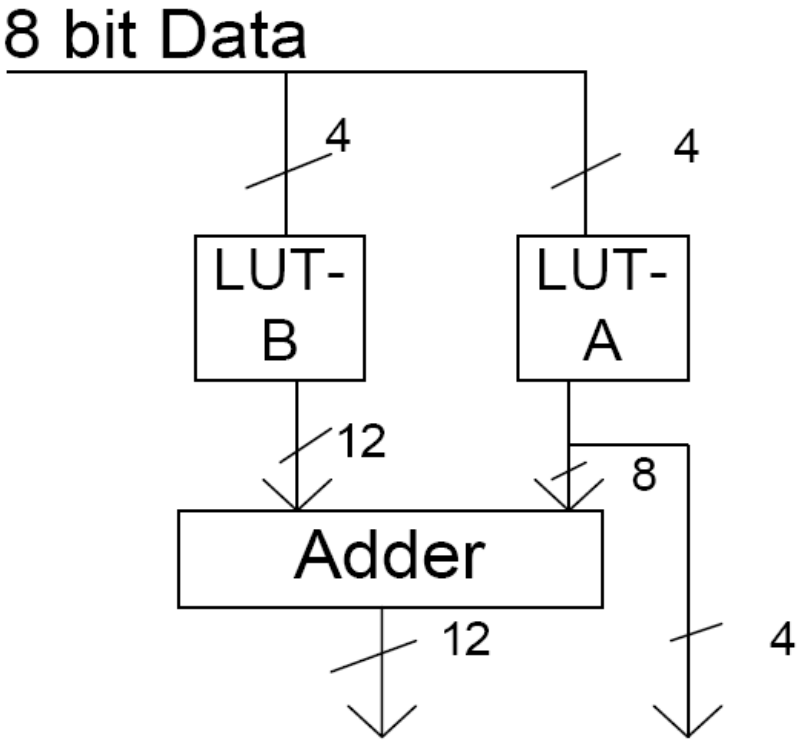

Fig. 4.6: LUT based multiplier

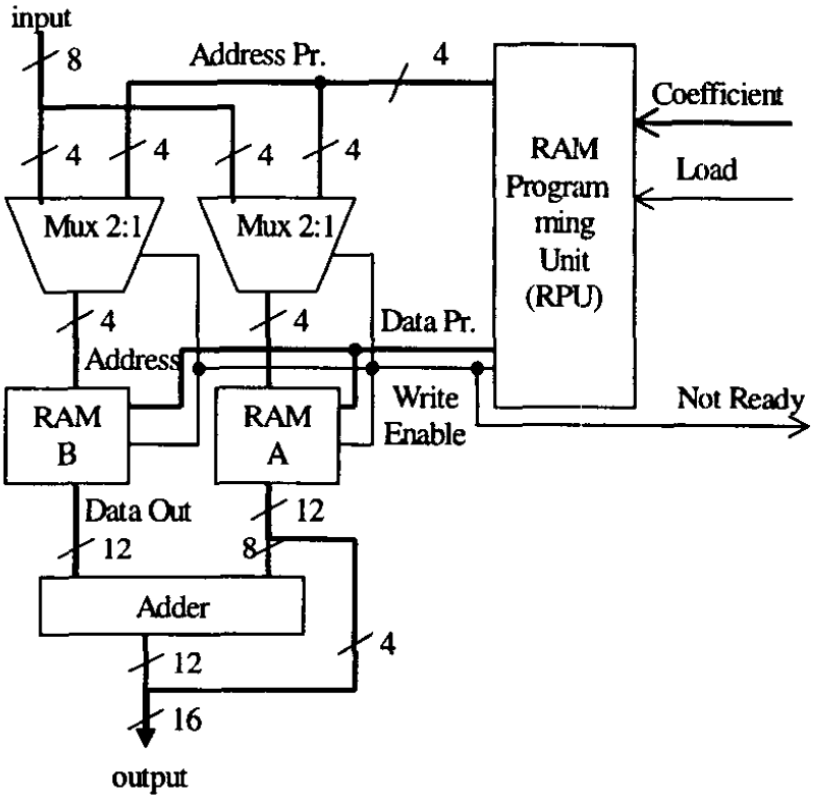

Fig. 4.7: Dynamic Constant Coefficient Multiplication

5. CONCLUSIONS AND FUTURE WORK

The theory part, section 2, in the text gives a short description of digital filters. It describes advantages and disadvantages of digital filters among other factors digital filters are much easier to design, test and implement than an analog filter.

The primary emphasis of digital filters in this paper is finite impulse response (FIR) filters. FIR consists of a finite number of sample values, as shown in equation 2.1. Section 2.1.3 explain different filter structures that can be used to design a filter. These structures are Direct Form, Transposed Direct Form, Cascade Form, Linear-Phase and Polyphase. Each structure has its benefits and drawbacks. Direct Form structure that is used is easy to design but slow and need more multipliers then for instance the Linear-Phase structure, so in other words the designed filter can much better but the assignement was to look at how to improve the multipliers in the filter.

The theory part also explains a little bit about bit-serial architecture vs. bit-parallel architecture. In section 2.2 we have described two methods to reduce hardware; booth encoding and canonical signed digit (CSD) representation. Booth is used to easier multiply two binary numbers that has a two's complement notation. CSD number system is a signed digit number system that minimizes the number of non-zero digits. CSD encoding is ternary so each digit can be -1, 0 or 1, where as regular digital representation us only 0 and 1. The basic idea is that no non-zero digits are adjacent. This reduces the hardware since a regular binary number can have a lots of adjacent non-zero digits. This section describes also a little bit what CSD encoding can help to reduce hardware in filter. The final section in the theory chapter it is explained a little bit how an FPGA is built up and how the design flow is carried out.

In chapter 3 and 4 the four versions of the filter that is design is explained. The filter that is designed is a low pass filter with filter order equals 10 and the pass band is 0 to 1000 Hz. The four versions that are designed of the same filter are used to compare what kind of filter that is most efficient. Version 1 and 2 has CSD encoded coefficients and are with out doubt the most efficient when it comes to area and speed, which was also assumed in advance. The differences between version 1 and 2 are that version 1 uses shift operators in VHDL code and the other one do not. Version 1 uses some more area and are a little slower then version 2.

The third version uses the multiplication operator *. When using this

operator on the FPGA it will use the built-in multipliers which will result in a low number of slices and a high speed, so if the multipliers not are used in the FPGA it is an advantage to use them instead of logic blocks. So version 3 can not be directly compared to the other versions of the filter since it uses the multipliers and not logic blocks.

The most interesting comparison is between version 1 and 4 where the first one uses CSD encoding and the latter uses 2's complement representation. It is proven in the thesis that version 1 uses almost 20% less slices of the FPGA then version 4 and version 1 has a maximum frequency of 504,032 MHz and version 4 only has a maximum frequency of 249,128 MHz. This shows as expected that CSD-encoding of the coefficients really can improve digital applications.

Finally in the paper a Dynamic Constant Coefficient Multiplier (DKCM) is discussed. The DKCM is used if one wants to change the coefficients of the filter. The DKCM implements LUT based multiplications. The DKCM offers much quicker reconfigurations but occupies more area in comparison with a constant coefficient multiplier that is used in the designed filter in this thesis. If one wants to change the coefficients in a constant coefficient filter that is designed in this thesis one has to design the circuit from the beginning.

5.1 Future work

Future work my include:

- The filter that was designed in this thesis was a Direct Form FIR filter, this as shown in the paper is not the most efficient structure so future work could also redesign the filter to another structure and identify the improvements of the new filter.
- Since the time ran out on this master thesis and no Dynamic Constant Coefficient Multiplier (DKCM) was designed and implemented it could be interesting in future work to design a DKCM and implement this on an FPGA.
- Investigation/research of a coefficient multiplier based re-configuration system with fixed module which is an alternative to DKCM.

BIBLIOGRAPHY

[1] R.Bernstein: *Multiplication by integer constants*, Software - Practice and Experience, July 1986,

[2] N.Homma, T.Aoki and T.Higuchi, *Evolutionary graph generation system with transmigration capability and its application to arithmetic circuit synthesis*, IEEE Proceedings, April 2002

[3] N.Boullis and A.Tisserand, *Some Optimizations of Hardware Multiplication by Constant Matrices*, Computer Arithmetic, 2003. Proceedings. 16th IEEE Symposium

[4] P.Briggs and T.Harvey: *Multiplicaton by integer constants*, Technical report, Rice University, 1994,

[5] Wikipedia, *Booth's multiplication algorithm*, http://en.wikipedia.org/wiki/Booth's_multiplication_algorithm

[6] *FIR Digital Filter Design Applet*, http://www.dsptutor.freeuk.com/FIRFilterDesign/FIRFilterDesign.html

[7] U.Meyer-Baese, *Digital Signal Processing with Field Programmable Gate Arrays*, Third Edition, Springer Berlin Heidelberg, page 165-214

[8] Kimmo Järvinen, *Field Programmable Gate Arrays*, http://www.automationit.hut.fi/file.php?id=787 , TKK - Signal Processing Laboratory, Department of Electrical and Communications Engineering at Helsinki University of Technology , Oct 2007

[9] John Wawrzynek, *Lecture 3 - Field Programmable Gate Arrays (FPGAs)*, http://inst.eecs.berkeley.edu/ cs150/sp03/handouts/2/LectureA/lec03-fpga.pdf , EECS150 - Digital Design, University of California, Berkeley , Jan 2003

[10] Mentor Graphics Corp, *www.modelsim.com*

[11] Xilinx, Inc, *www.xilinx.com*

[12] R.M. Hewlitt, E.S. Swartzlantler, Jr., *Canonical signed digit representation for FIR digital filters*, Signal Processing Systems, 2000. SiPS 2000. 2000 IEEE Workshop

[13] T. Kean, B. New, B. Slous, *A Fast Constant Coefficient Multiplier for the XC6200*, Proceedings of the 6th International Workshop on Field-Programmable Logic, Smart Applications, New Paradigms and Compilers, 1996

[14] K. Wiatr, E. Jamro , *Constant coefficient multiplication in FPGA structures*, Euromicro Conference, 2000. Proceedings of the 26th

[15] K. Wiatr, *Implementation of multipliers in FPGA structures*, Quality Electronic Design, 2001 International Symposium

[16] *Digital Filters: An Introduction*, http://www.dsptutor.freeuk.com/dfilt1.htm

[17] *Finite Impulse Response (FIR) Digital Filters*, Book: Digital Signal Processing with Field Programmable Gate Arrays, pages 165-214

[18] M. Bolic, *FIR filters*, http://web.cecs.pdx.edu/m̃perkows/CAPSTONES/DSP1/ELG6163_FIR.pdf

APPENDIX

A. MY FIR FILTER VERSION 1

A.1 my_fir.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE work.my_package.all;
USE work.my_functions.all;


------------------------------------------------------
-- MY FIR filter
-- Version 1.0
-- 11 coefficients
-- Author: Vebjørn Bystrøm, NTNU 8.may 2008
------------------------------------------------------

---------------- ENTITY ------------------------------
ENTITY my_fir IS
    PORT (
        x: IN std_logic_vector(7 downto 0);
                clk, rst : IN std_logic;
        y: OUT std_logic_vector(23 downto 0)
        );
END my_fir;
------------------------------------------------------

---------------- ARCHITECTURE ------------------------

ARCHITECTURE rtl of my_fir IS

TYPE registers IS ARRAY (10 downto 0)
OF std_logic_vector(7 downto 0);

SIGNAL reg: registers;


----------------------PROCESS-------------------------
BEGIN


        process(clk, rst, reg)

        VARIABLE coeff0: std_logic_vector(23 downto 0) ;
        VARIABLE coeff1: std_logic_vector(23 downto 0) ;
        VARIABLE coeff2: std_logic_vector(23 downto 0) ;
        VARIABLE coeff3: std_logic_vector(23 downto 0) ;
        VARIABLE coeff4: std_logic_vector(23 downto 0) ;
        VARIABLE coeff5: std_logic_vector(23 downto 0) ;
        VARIABLE coeff6: std_logic_vector(23 downto 0) ;
        VARIABLE coeff7: std_logic_vector(23 downto 0) ;
        VARIABLE coeff8: std_logic_vector(23 downto 0) ;
        VARIABLE coeff9: std_logic_vector(23 downto 0) ;
        VARIABLE coeff10: std_logic_vector(23 downto 0);


        BEGIN
        --- reset registers ---
        if (rst = '1') then
                for i in 10 downto 0 loop
                        for j in 7 downto 0 loop
                                reg(i)(j) <= '0';
                        end loop;
                end loop;
        --- update registers ---
        elsif (clk'EVENT and clk ='1') then
                for k in 10 downto 1 loop
                        reg(k) <= reg(k-1);
```

```
                end loop;
                reg(0) <= x;

        end if;

        --- multiply coeff with x and accumulate ---
        coeff0 := coeff_0(reg(0)(7 downto 0));
        coeff1 := coeff_1(reg(1)(7 downto 0));
        coeff2 := coeff_2(reg(2)(7 downto 0));
        coeff3 := coeff_3(reg(3)(7 downto 0));
        coeff4 := coeff_4(reg(4)(7 downto 0));
        coeff5 := coeff_5(reg(5)(7 downto 0));
        coeff6 := coeff_6(reg(6)(7 downto 0));
        coeff7 := coeff_7(reg(7)(7 downto 0));
        coeff8 := coeff_8(reg(8)(7 downto 0));
        coeff9 := coeff_9(reg(9)(7 downto 0));
        coeff10 := coeff_10(reg(10)(7 downto 0));

        y <= coeff0 + coeff1 + coeff2 + coeff3
        + coeff4 + coeff5 + coeff6 + coeff7
        + coeff8 + coeff9  + coeff10;
        END PROCESS;
END rtl;
-------------------------------------------------
```

A.2 my_functions.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE work.my_package.all;

-------------------------------------------------
-- MY FIR filter
-- Computation of the coefficients
-- Version 1.0
-- 11 coefficients
-- Author: Vebjørn Bystrøm, NTNU 8.may 2008
-------------------------------------------------

PACKAGE my_functions is

FUNCTION coeff_0 (signal c: in std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_1 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_2 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_3 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_4 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_5 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_6 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_7 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_8 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_9 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_10 (signal c : std_logic_vector)
RETURN std_logic_vector;

END my_functions;

PACKAGE BODY my_functions is


----------------  COEFF_0 --------------------------------
        FUNCTION coeff_0 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE temp : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift2 : std_logic_vector (23 DOWNTO 0);

        BEGIN
                temp := c & "0000000000000000";
                shift1 := (temp sra 10);
                shift2 := (temp sra 12);
```

```
                return shift1 - shift2;
        END coeff_0;
-------------------------------------------------------
---------------  COEFF_1 ------------------------------
        FUNCTION coeff_1 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE temp : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift2 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift3 :std_logic_vector (23 DOWNTO 0);

        BEGIN
                temp := c & "0000000000000000";
                shift1 := (temp sra 8);
                shift2 := (temp sra 12);
                shift3 := (temp sra 15);

                return (shift1 + shift2 + shift3);

        END coeff_1;
-------------------------------------------------------
---------------  COEFF_2 ------------------------------
        FUNCTION coeff_2 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE temp : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift2 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift3 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift4 :std_logic_vector (23 DOWNTO 0);

        BEGIN
                temp := c & "0000000000000000";
                shift1 := (temp sra 6);
                shift2 := (temp sra 9);
                shift3 := (temp sra 13);
                shift4 := (temp sra 15);

                return ('0' - shift1 + shift2 + shift3 + shift4);

        END coeff_2;
-------------------------------------------------------

---------------  COEFF_3 ------------------------------
        FUNCTION coeff_3 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE temp : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift2 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift3 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift4 :std_logic_vector (23 DOWNTO 0);

        BEGIN
                temp := c & "0000000000000000";
                shift1 := (temp sra 6);
                shift2 := (temp sra 9);
                shift3 := (temp sra 13);

                return ('0' - shift1 + shift2 + shift3);

        END coeff_3;
-------------------------------------------------------


---------------  COEFF_4 ------------------------------
        FUNCTION coeff_4 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE temp : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift2 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift3 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift4 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift5 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift6 :std_logic_vector (23 DOWNTO 0);

        BEGIN
                temp := c & "0000000000000000";
                shift1 := (temp sra 3);
                shift2 := (temp sra 5);
                shift3 := (temp sra 7);
                shift4 := (temp sra 9);
```

```
                shift5 := (temp sra 11);
                shift6 := (temp sra 14);

                return (shift1 - shift2 - shift3
                + shift4 + shift5 + shift6);

        END coeff_4;
-------------------------------------------------------------

---------------  COEFF_5 ------------------------------
        FUNCTION coeff_5 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE temp : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift2 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift3 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift4 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift5 :std_logic_vector (23 DOWNTO 0);

        BEGIN
                temp := c & "0000000000000000";
                shift1 := (temp sra 2);
                shift2 := (temp sra 4);
                shift3 := (temp sra 6);
                shift4 := (temp sra 9);
        shift5 := (temp sra 12);

                return (shift1 - shift2 - shift3
                - shift4 + shift5);

        END coeff_5;
-------------------------------------------------------------

---------------  COEFF_6 ------------------------------
        FUNCTION coeff_6 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE temp : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift2 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift3 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift4 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift5 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift6 :std_logic_vector (23 DOWNTO 0);

        BEGIN
                temp := c & "0000000000000000";
                shift1 := (temp sra 3);
                shift2 := (temp sra 5);
                shift3 := (temp sra 7);
                shift4 := (temp sra 9);
                shift5 := (temp sra 11);
                shift6 := (temp sra 14);

                return (shift1 - shift2 - shift3
                + shift4 + shift5 + shift6);

        END coeff_6;
-------------------------------------------------------------

---------------  COEFF_7 ------------------------------
        FUNCTION coeff_7 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE temp : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift2 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift3 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift4 :std_logic_vector (23 DOWNTO 0);

        BEGIN
                temp := c & "0000000000000000";
                shift1 := (temp sra 6);
                shift2 := (temp sra 9);
                shift3 := (temp sra 13);

                return ('0' - shift1 + shift2 + shift3);

        END coeff_7;
-------------------------------------------------------------


---------------  COEFF_8 ------------------------------
```

```
        FUNCTION coeff_8 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE temp : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift2 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift3 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift4 :std_logic_vector (23 DOWNTO 0);

        BEGIN
                temp := c & "0000000000000000";
                shift1 := (temp sra 6);
                shift2 := (temp sra 9);
                shift3 := (temp sra 13);
                shift4 := (temp sra 15);

                return ('0' - shift1 + shift2 + shift3 + shift4);

        END coeff_8;
-------------------------------------------------------

---------------  COEFF_9 ------------------------------
        FUNCTION coeff_9 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE temp : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift2 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift3 :std_logic_vector (23 DOWNTO 0);

        BEGIN
                temp := c & "0000000000000000";
                shift1 := (temp sra 8);
                shift2 := (temp sra 12);
                shift3 := (temp sra 15);

                return (shift1 + shift2 + shift3);

        END coeff_9;
-------------------------------------------------------

---------------  COEFF_10 -----------------------------
        FUNCTION coeff_10 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE temp : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift2 : std_logic_vector (23 DOWNTO 0);

        BEGIN
                temp := c & "0000000000000000";
                shift1 := (temp sra 10);
                shift2 := (temp sra 12);

                return shift1 - shift2;

        END coeff_10;
-------------------------------------------------------

END my_functions;
```

A.3 my_fir_tb.vhd

```
library ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

-------------------------------------------------------
-- MY FIR filter
-- Test bench
-- Version 1.0
-- 11 coefficients
-- Author: Vebjørn Bystrøm, NTNU 8.may 2008
-------------------------------------------------------

---------------- ENTITY -------------------------------
entity my_fir_tb is

end my_fir_tb;

-------------------------------------------------------

---------------- ARCHITECTURE -------------------------
```

```
architecture my_fir_tb_arch of my_fir_tb is

---------------- COMPONENTS ------------------------------
component my_fir
        PORT (
                x:              IN std_logic_vector (7 DOWNTO 0);
                clk, rst: IN std_logic;
                y:              OUT std_logic_vector (23 DOWNTO 0)
        );
end component;
----------------------------------------------------------

signal x                : std_logic_vector (7 DOWNTO 0)
:= "00000000";
signal y                : std_logic_vector (23 DOWNTO 0)
:= "000000000000000000000000";
signal clk              : std_logic := '1';
signal rst              : std_logic;


begin

-- device under test --
dut: my_fir
port map (
        x => x,
        y => y,
        clk => clk,
        rst => rst
);

-- clk generator --
clk <= not clk after 1 ms;


-- stimulate divece under test --
wstim: process
        begin
                rst <= '1';
                wait until clk'event and clk = '1';
                rst <= '0';
                x <= "00000000"; --0
                wait until clk'event and clk = '1';
                x <= "00000001"; --1
                wait until clk'event and clk = '1';
                x <= "00000010"; --2
                wait until clk'event and clk = '1';
                x <= "00000011"; --3
                wait until clk'event and clk = '1';
                x <= "00000100"; --4
                wait until clk'event and clk = '1';
                x <= "00000101"; --5
                wait until clk'event and clk = '1';
                x <= "00000110"; --6
                wait until clk'event and clk = '1';
                x <= "00000111"; --7
                wait until clk'event and clk = '1';
                x <= "11111111"; -- -1
                wait until clk'event and clk = '1';
                x <= "11111110"; -- -2
                wait until clk'event and clk = '1';
                x <= "11111101"; -- -3
                wait until clk'event and clk = '1';
                wait;
        end process;
end;
----------------------------------------------------------
```

B. MY FIR FILTER VERSION 2

B.1 my_fir.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE work.my_package.all;
USE work.my_functions.all;


------------------------------------------------------
-- MY FIR filter
-- Version 2.0
-- 11 coefficients
-- Author: Vebjørn Bystrøm, NTNU 8.may 2008
------------------------------------------------------

---------------- ENTITY ------------------------------
ENTITY my_fir IS
    PORT (
        x: IN std_logic_vector(7 downto 0);
                clk, rst : IN std_logic;
        y: OUT std_logic_vector(23 downto 0)
        );
END my_fir;
------------------------------------------------------

---------------- ARCHITECTURE ------------------------

ARCHITECTURE rtl of my_fir IS

TYPE registers IS ARRAY (10 downto 0)
OF std_logic_vector(7 downto 0);

SIGNAL reg: registers;


----------------------PROCESS-----------------------------
BEGIN


        process(clk, rst, reg)

        VARIABLE coeff0: std_logic_vector(23 downto 0) ;
        VARIABLE coeff1: std_logic_vector(23 downto 0) ;
        VARIABLE coeff2: std_logic_vector(23 downto 0) ;
        VARIABLE coeff3: std_logic_vector(23 downto 0) ;
        VARIABLE coeff4: std_logic_vector(23 downto 0) ;
        VARIABLE coeff5: std_logic_vector(23 downto 0) ;
        VARIABLE coeff6: std_logic_vector(23 downto 0) ;
        VARIABLE coeff7: std_logic_vector(23 downto 0) ;
        VARIABLE coeff8: std_logic_vector(23 downto 0) ;
        VARIABLE coeff9: std_logic_vector(23 downto 0) ;
        VARIABLE coeff10: std_logic_vector(23 downto 0);


        BEGIN
        --- reset registers ---
        if (rst = '1') then
                for i in 10 downto 0 loop
                        for j in 7 downto 0 loop
                                reg(i)(j) <= '0';
                        end loop;
                end loop;
        --- update registers ---
        elsif (clk'EVENT and clk ='1') then
                for k in 10 downto 1 loop
                        reg(k) <= reg(k-1);
```

```
                end loop;
                reg(0) <= x;

        end if;

        --- multiply coeff with x and accumulate ---
        coeff0 := coeff_0(reg(0)(7 downto 0));
        coeff1 := coeff_1(reg(1)(7 downto 0));
        coeff2 := coeff_2(reg(2)(7 downto 0));
        coeff3 := coeff_3(reg(3)(7 downto 0));
        coeff4 := coeff_4(reg(4)(7 downto 0));
        coeff5 := coeff_5(reg(5)(7 downto 0));
        coeff6 := coeff_6(reg(6)(7 downto 0));
        coeff7 := coeff_7(reg(7)(7 downto 0));
        coeff8 := coeff_8(reg(8)(7 downto 0));
        coeff9 := coeff_9(reg(9)(7 downto 0));
        coeff10 := coeff_10(reg(10)(7 downto 0));

        y <= coeff0 + coeff1 + coeff2 + coeff3
        + coeff4 + coeff5 + coeff6 + coeff7
        + coeff8 + coeff9  + coeff10;
        END PROCESS;
END rtl;
-------------------------------------------------
```

B.2 my_functions.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE work.my_package.all;

-------------------------------------------------
-- MY FIR filter
-- Computation of the coefficients
-- Version 2.0
-- 11 coefficients
-- Author: Vebjørn Bystrøm, NTNU 8.may 2008
-------------------------------------------------

PACKAGE my_functions is

FUNCTION coeff_0 (signal c: in std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_1 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_2 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_3 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_4 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_5 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_6 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_7 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_8 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_9 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_10 (signal c : std_logic_vector)
RETURN std_logic_vector;

END my_functions;

PACKAGE BODY my_functions is


----------------  COEFF_0 --------------------------------
        FUNCTION coeff_0 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);

        BEGIN
                shift1 := (c(7)&c(7)&c(7)&c(7)&
                c(7)&c(7)&c(7)&c(7)&c(7)&c(7)&
                c(7)&c(7)&c(7)&c(7)&c(7)
                &c(7 downto 0)&'0');

                return shift1;
```

```
        END coeff_0;
-------------------------------------------------------------

---------------  COEFF_1 ------------------------------
        FUNCTION coeff_1 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift2 :std_logic_vector (23 DOWNTO 0);

        BEGIN
                shift1 := (c(7)&c(7)&c(7)&c(7)&
                c(7)&c(7)&c(7)&c(7)&c(7)&c(7)&c(7)
                &c(7)&c(7 downto 0)&"0000");
                shift2 := (c(7)&c(7)&c(7)&c(7)&
                c(7)&c(7)&c(7)&c(7)&c(7)&c(7)&c(7)
                &c(7)&c(7)&c(7)&c(7)&c(7 downto 0)&'0');

                return (shift1-shift2);

        END coeff_1;
-------------------------------------------------------------


---------------  COEFF_2 ------------------------------
        FUNCTION coeff_2 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift2 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift3 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift4 :std_logic_vector (23 DOWNTO 0);

        BEGIN
                shift1 := (c(7)&c(7)&c(7)&c(7)&
                c(7)&c(7)&c(7)&c(7)&c(7)
                &c(7 downto 0)&"000000");
                shift2 := (c(7)&c(7)&c(7)&c(7)&
                c(7)&c(7)&c(7)&c(7)&c(7)
                &c(7)&c(7)&c(7 downto 0)&"0000");
                shift3 := (c(7)&c(7)&c(7)&c(7)&
                c(7)&c(7)&c(7)&c(7)&c(7)
                &c(7)&c(7)&c(7)&c(7 downto 0)&"000");
                shift4 := (c(7)&c(7)&c(7)&c(7)&
                c(7)&c(7)&c(7)&c(7)&c(7)
                &c(7)&c(7)&c(7)&c(7)&c(7)&c(7)&c(7 downto 0));

                return ('0' - shift1 + shift2 + shift3 - shift4);

        END coeff_2;
-------------------------------------------------------------

---------------  COEFF_3 ------------------------------
        FUNCTION coeff_3 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift2 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift3 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift4 :std_logic_vector (23 DOWNTO 0);

        BEGIN
                shift1 := (c(7)&c(7)&c(7)&c(7)&
                c(7)&c(7)&c(7)&c(7)&c(7)&c(7)
                &c(7 downto 0)&"000000");
                shift2 := (c(7)&c(7)&c(7)&c(7)&
                c(7)&c(7)&c(7)&c(7)&c(7)&c(7)
                &c(7)&c(7)&c(7 downto 0)&"0000");
                shift3 := (c(7)&c(7)&c(7)&c(7)&
                c(7)&c(7)&c(7)&c(7)&c(7)&c(7)
                &c(7)&c(7)&c(7)&c(7 downto 0)&"000");
                shift4 := (c(7)&c(7)&c(7)&c(7)&
                c(7)&c(7)&c(7)&c(7)&c(7)&c(7)
                &c(7)&c(7)&c(7)&c(7)&c(7)&c(7)&c(7 downto 0));

                return ('0' - shift1 + shift2 + shift3 - shift4);

        END coeff_3;
-------------------------------------------------------------


---------------  COEFF_4 ------------------------------
        FUNCTION coeff_4 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS
```

```
                VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift2 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift3 :std_logic_vector (23 DOWNTO 0);

        BEGIN

                shift1 := (c(7)&c(7)&c(7)&c(7)&
                c(7)&c(7)&c(7)&c(7)&c(7 downto 0)&"00000000");
                shift2 := (c(7)&c(7)&c(7)&c(7)&
                c(7)&c(7)&c(7)&c(7)&c(7)&c(7)
                &c(7)&c(7 downto 0)&"00000");
                shift3 := (c(7)&c(7)&c(7)&c(7)&
                c(7)&c(7)&c(7)&c(7)&c(7)&c(7)
                &c(7)&c(7)&c(7)&c(7 downto 0)&"000");

                return (shift1 + shift2 + shift3);

        END coeff_4;
-------------------------------------------------------

---------------  COEFF_5 -------------------------------

        FUNCTION coeff_5 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift2 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift3 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift4 :std_logic_vector (23 DOWNTO 0);

        BEGIN
                shift1 := (c(7)&c(7)&c(7)&c(7)&
                c(7)&c(7)&c(7)&c(7)&c(7 downto 0)&"000000000");
                shift2 := (c(7)&c(7)&c(7)&c(7)&
                c(7)&c(7)&c(7)&c(7)&c(7)&c(7)
                &c(7 downto 0)&"000000");
                shift3 := (c(7)&c(7)&c(7)&c(7)&
                c(7)&c(7)&c(7)&c(7)&c(7)&c(7)
                &c(7)&c(7)&c(7)&c(7 downto 0)&"000");
                shift4 := (c(7)&c(7)&c(7)&c(7)&
                c(7)&c(7)&c(7)&c(7)&c(7)&c(7)
                &c(7)&c(7)&c(7)&c(7)&c(7)&c(7 downto 0)&"0");

                return (shift1 + shift2 - shift3 - shift4);

        END coeff_5;
-------------------------------------------------------

---------------  COEFF_6 -------------------------------
        FUNCTION coeff_6 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift2 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift3 :std_logic_vector (23 DOWNTO 0);


        BEGIN
                shift1 := (c(7)&c(7)&c(7)&c(7)&
                c(7)&c(7)&c(7)&c(7)&c(7)&c(7 downto 0)&"00000000");
                shift2 := (c(7)&c(7)&c(7)&c(7)&
                c(7)&c(7)&c(7)&c(7)&c(7)
                &c(7)&c(7)&c(7 downto 0)&"00000");
                shift3 := (c(7)&c(7)&c(7)&c(7)&
                c(7)&c(7)&c(7)&c(7)&c(7)
                &c(7)&c(7)&c(7)&c(7)&c(7 downto 0)&"000");

                return (shift1 + shift2 + shift3);

        END coeff_6;
-------------------------------------------------------
---------------  COEFF_7 -------------------------------
        FUNCTION coeff_7 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift2 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift3 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift4 :std_logic_vector (23 DOWNTO 0);

        BEGIN
                shift1 := (c(7)&c(7)&c(7)&c(7)&
                c(7)&c(7)&c(7)&c(7)&c(7)&c(7)
```

```
                &c(7 downto 0)&"000000");
                shift2 := (c(7)&c(7)&c(7)&c(7)&
                c(7)&c(7)&c(7)&c(7)&c(7)
                &c(7)&c(7)&c(7 downto 0)&"0000");
                shift3 := (c(7)&c(7)&c(7)&c(7)&
                c(7)&c(7)&c(7)&c(7)&c(7)
                &c(7)&c(7)&c(7)&c(7 downto 0)&"000");
                shift4 := (c(7)&c(7)&c(7)&c(7)&
                c(7)&c(7)&c(7)&c(7)&c(7)
                &c(7)&c(7)&c(7)&c(7)&c(7)&c(7 downto 0));

                return ('0' - shift1 + shift2 + shift3 - shift4);

        END coeff_7;
-------------------------------------------------------


---------------  COEFF_8 ------------------------------
        FUNCTION coeff_8 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift2 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift3 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift4 :std_logic_vector (23 DOWNTO 0);

        BEGIN
                shift1 := (c(7)&c(7)&c(7)&c(7)&
                c(7)&c(7)&c(7)&c(7)&c(7)&c(7)
                &c(7 downto 0)&"000000");
                shift2 := (c(7)&c(7)&c(7)&c(7)&
                c(7)&c(7)&c(7)&c(7)&c(7)&c(7)
                &c(7)&c(7)&c(7 downto 0)&"0000");
                shift3 := (c(7)&c(7)&c(7)&c(7)&
                c(7)&c(7)&c(7)&c(7)&c(7)&c(7)
                &c(7)&c(7)&c(7)&c(7 downto 0)&"000");
                shift4 := (c(7)&c(7)&c(7)&c(7)&
                c(7)&c(7)&c(7)&c(7)&c(7)&c(7)
                &c(7)&c(7)&c(7)&c(7)&c(7)&c(7)&c(7 downto 0));

                return ('0' - shift1 + shift2 + shift3 - shift4);

        END coeff_8;
-------------------------------------------------------

---------------  COEFF_9 ------------------------------
        FUNCTION coeff_9 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift2 :std_logic_vector (23 DOWNTO 0);

        BEGIN
                shift1 := (c(7)&c(7)&c(7)&c(7)&
                c(7)&c(7)&c(7)&c(7)&c(7)&c(7)&c(7)&c(7)
                &c(7)&c(7 downto 0)&"0000");
                shift2 := (c(7)&c(7)&c(7)&c(7)&
                c(7)&c(7)&c(7)&c(7)&c(7)&c(7)&c(7)&c(7)
                &c(7)&c(7)&c(7)&c(7)&c(7 downto 0)&'0');


                return (shift1 - shift2);

        END coeff_9;
-------------------------------------------------------

---------------  COEFF_10 -----------------------------
        FUNCTION coeff_10 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);

        BEGIN
                shift1 := (c(7)&c(7)&c(7)&c(7)&
                c(7)&c(7)&c(7)&c(7)&c(7)&c(7)&c(7)&c(7)
                &c(7)&c(7)&c(7)&c(7)&c(7 downto 0)&'0');


                return (shift1);

        END coeff_10;
-------------------------------------------------------
```

```
END my_functions;
```

B.3 my_fir_tb.vhd

```
library ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

--------------------------------------------------------
-- MY FIR filter
-- Test bench
-- Version 2.0
-- 11 coefficients
-- Author: Vebjørn Bystrøm, NTNU 8.may 2008
--------------------------------------------------------

---------------- ENTITY --------------------------------
entity my_fir_tb is

end my_fir_tb;

--------------------------------------------------------

---------------- ARCHITECTURE --------------------------
architecture my_fir_tb_arch of my_fir_tb is


---------------- COMPONENTS ----------------------------
component my_fir
        PORT (
                x:              IN std_logic_vector (7 DOWNTO 0);
                clk, rst: IN std_logic;
                y:              OUT std_logic_vector (17 DOWNTO 0)
        );
end component;
--------------------------------------------------------

signal x                : std_logic_vector (7 DOWNTO 0)
:= "00000000";
signal y                : std_logic_vector (17 DOWNTO 0)
:= "00000000000000000";
signal clk              : std_logic := '1';
signal rst              : std_logic;


begin

-- device under test --
dut: my_fir
port map (
        x => x,
        y => y,
        clk => clk,
        rst => rst
);

-- clk generator --
clk <= not clk after 1 ms;


-- stimulate divece under test --
wstim: process
        begin
                rst <= '1';
                wait until clk'event and clk = '1';
                rst <= '0';
                x <= "00000000"; --0
                wait until clk'event and clk = '1';
                x <= "00000001"; --1
                wait until clk'event and clk = '1';
                x <= "00000010"; --2
                wait until clk'event and clk = '1';
                x <= "00000011"; --3
                wait until clk'event and clk = '1';
                x <= "00000100"; --4
                wait until clk'event and clk = '1';
                x <= "00000101"; --5
                wait until clk'event and clk = '1';
                x <= "00000110"; --6
                wait until clk'event and clk = '1';
                x <= "00000111"; --7
                wait until clk'event and clk = '1';
                x <= "11111111"; -- -1
```

```
                wait until clk'event and clk = '1';
                x <= "11111110"; -- -2
                wait until clk'event and clk = '1';
                x <= "11111101"; -- -3
                wait until clk'event and clk = '1';
                wait;
        end process;
end;
```

C. MY FIR FILTER VERSION 3

C.1 my_fir.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE work.my_package.all;
USE work.my_functions.all;


------------------------------------------------------
-- MY FIR filter
-- Version 3.0
-- 11 coefficients
-- Use of multiplication character "*"
-- Author: Vebjørn Bystrøm, NTNU 14.may 2008
------------------------------------------------------

---------------- ENTITY ------------------------------
ENTITY my_fir IS
    PORT (
        x: IN std_logic_vector(7 downto 0);
                clk, rst : IN std_logic;
        y: OUT std_logic_vector(23 downto 0)
        );
END my_fir;
------------------------------------------------------

---------------- ARCHITECTURE ------------------------

ARCHITECTURE rtl of my_fir IS

TYPE registers IS ARRAY (10 downto 0)
OF std_logic_vector(7 downto 0);

SIGNAL reg: registers;


----------------------PROCESS-----------------------------
BEGIN



        process(clk, rst, reg)

        VARIABLE coeff0: std_logic_vector(23 downto 0) ;
        VARIABLE coeff1: std_logic_vector(23 downto 0) ;
        VARIABLE coeff2: std_logic_vector(23 downto 0) ;
        VARIABLE coeff3: std_logic_vector(23 downto 0) ;
        VARIABLE coeff4: std_logic_vector(23 downto 0) ;
        VARIABLE coeff5: std_logic_vector(23 downto 0) ;
        VARIABLE coeff6: std_logic_vector(23 downto 0) ;
        VARIABLE coeff7: std_logic_vector(23 downto 0) ;
        VARIABLE coeff8: std_logic_vector(23 downto 0) ;
        VARIABLE coeff9: std_logic_vector(23 downto 0) ;
        VARIABLE coeff10: std_logic_vector(23 downto 0);


        BEGIN
        --- reset registers ---
        if (rst = '1') then
                for i in 10 downto 0 loop
                        for j in 7 downto 0 loop
                                reg(i)(j) <= '0';
                        end loop;
                end loop;
        --- update registers ---
        elsif (clk'EVENT and clk ='1') then
                for k in 10 downto 1 loop
```

```
                reg(k) <= reg(k-1);
            end loop;
            reg(0) <= x;

        end if;

        --- multiply coeff with x and accumulate ---
        coeff0 := coeff_0(reg(0)(7 downto 0));
        coeff1 := coeff_1(reg(1)(7 downto 0));
        coeff2 := coeff_2(reg(2)(7 downto 0));
        coeff3 := coeff_3(reg(3)(7 downto 0));
        coeff4 := coeff_4(reg(4)(7 downto 0));
        coeff5 := coeff_5(reg(5)(7 downto 0));
        coeff6 := coeff_6(reg(6)(7 downto 0));
        coeff7 := coeff_7(reg(7)(7 downto 0));
        coeff8 := coeff_8(reg(8)(7 downto 0));
        coeff9 := coeff_9(reg(9)(7 downto 0));
        coeff10 := coeff_10(reg(10)(7 downto 0));

        y <= coeff0 + coeff1 + coeff2 + coeff3
        + coeff4 + coeff5 + coeff6 + coeff7
        + coeff8 + coeff9  + coeff10;
        END PROCESS;
END rtl;
-------------------------------------------------
```

C.2 my_functions.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE work.my_package.all;

-------------------------------------------------
-- MY FIR filter
-- Computation of the coefficients
-- Version 3.0
-- 11 coefficients
-- Use of multiplication character "*"
-- Author: Vebjørn Bystrøm, NTNU 14.may 2008
-------------------------------------------------

PACKAGE my_functions is

FUNCTION coeff_0 (signal c: in std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_1 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_2 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_3 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_4 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_5 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_6 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_7 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_8 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_9 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_10 (signal c : std_logic_vector)
RETURN std_logic_vector;

END my_functions;

PACKAGE BODY my_functions is


---------------  COEFF_0 -------------------------------
        FUNCTION coeff_0 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE temp : std_logic_vector (23 DOWNTO 0);


        BEGIN
                temp := c * "0000000000011000";
```

```
                return temp;
        END coeff_0;
-------------------------------------------------------------

---------------  COEFF_1 ------------------------------
        FUNCTION coeff_1 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE temp : std_logic_vector (23 DOWNTO 0);


        BEGIN
                temp := c * "0000000010001001";


                return temp;

        END coeff_1;
-------------------------------------------------------------

---------------  COEFF_2 ------------------------------
        FUNCTION coeff_2 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE temp : std_logic_vector (23 DOWNTO 0);


        BEGIN
                temp := c * "1111111001000101";


                return temp;

        END coeff_2;
-------------------------------------------------------------

---------------  COEFF_3 ------------------------------
        FUNCTION coeff_3 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE temp : std_logic_vector (23 DOWNTO 0);


        BEGIN
                temp := c * "1111111001000100";


                return temp;

        END coeff_3;
-------------------------------------------------------------


---------------  COEFF_4 ------------------------------
        FUNCTION coeff_4 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE temp : std_logic_vector (23 DOWNTO 0);


        BEGIN
                temp := c * "0000101101010010";


                return temp;

        END coeff_4;
-------------------------------------------------------------

---------------  COEFF_5 ------------------------------
        FUNCTION coeff_5 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE temp : std_logic_vector (23 DOWNTO 0);


        BEGIN
                temp := c * "0001010111001000";


                return temp;

        END coeff_5;
-------------------------------------------------------------
```

```
-----------------  COEFF_6 ---------------------------------
        FUNCTION coeff_6 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE temp : std_logic_vector (23 DOWNTO 0);


        BEGIN
                temp := c * "0000101101010010";


                return temp;

        END coeff_6;
-----------------------------------------------------------

-----------------  COEFF_7 ---------------------------------
        FUNCTION coeff_7 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE temp : std_logic_vector (23 DOWNTO 0);


        BEGIN
                temp := c * "1111111001000100";


                return temp;

        END coeff_7;
-----------------------------------------------------------


-----------------  COEFF_8 ---------------------------------
        FUNCTION coeff_8 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE temp : std_logic_vector (23 DOWNTO 0);


        BEGIN
                temp := c * "1111111001000101";


                return temp;

        END coeff_8;
-----------------------------------------------------------

-----------------  COEFF_9 ---------------------------------
        FUNCTION coeff_9 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE temp : std_logic_vector (23 DOWNTO 0);


        BEGIN
                temp := c * "0000000010001001";


                return temp;

        END coeff_9;
-----------------------------------------------------------

-----------------  COEFF_10 ---------------------------------
        FUNCTION coeff_10 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE temp : std_logic_vector (23 DOWNTO 0);


        BEGIN
                temp := c * "0000000000011000";


                return temp;

        END coeff_10;
-----------------------------------------------------------


END my_functions;
```

C.3 my_fir_tb.vhd

```
library ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

-------------------------------------------------------
-- MY FIR filter
-- Test bench
-- Version 3.0
-- 11 coefficients
-- Use of multiplication character "*"
-- Author: Vebjørn Bystrøm, NTNU 14.may 2008
-------------------------------------------------------

---------------- ENTITY -------------------------------
entity my_fir_tb is

end my_fir_tb;

-------------------------------------------------------

---------------- ARCHITECTURE -------------------------
architecture my_fir_tb_arch of my_fir_tb is


---------------- COMPONENTS ---------------------------
component my_fir
        PORT (
                x:              IN std_logic_vector (7 DOWNTO 0);
                clk, rst: IN std_logic;
                y:              OUT std_logic_vector (39 DOWNTO 0)
        );
end component;
-------------------------------------------------------

signal x                : std_logic_vector (7 DOWNTO 0)
:= "00000000";
signal y                : std_logic_vector (39 DOWNTO 0)
:= "0000000000000000000000000000000000000000";
signal clk              : std_logic := '1';
signal rst              : std_logic;


begin

-- device under test --
dut: my_fir
port map (
        x => x,
        y => y,
        clk => clk,
        rst => rst
);

-- clk generator --
clk <= not clk after 1 ms;


-- stimulate divece under test --
wstim: process
        begin
                rst <= '1';
                wait until clk'event and clk = '1';
                rst <= '0';
                x <= "00000000"; --0
                wait until clk'event and clk = '1';
                x <= "00000001"; --1
                wait until clk'event and clk = '1';
                x <= "00000010"; --2
                wait until clk'event and clk = '1';
                x <= "00000011"; --3
                wait until clk'event and clk = '1';
                x <= "00000100"; --4
                wait until clk'event and clk = '1';
                x <= "00000101"; --5
                wait until clk'event and clk = '1';
                x <= "00000110"; --6
                wait until clk'event and clk = '1';
                x <= "00000111"; --7
                wait until clk'event and clk = '1';
                x <= "11111111"; -- -1
                wait until clk'event and clk = '1';
```

```
                x <= "11111110"; -- -2
                wait until clk'event and clk = '1';
                x <= "11111101"; -- -3
                wait until clk'event and clk = '1';
                wait;
        end process;
end;
```

D. MY FIR FILTER VERSION 4

D.1 my_fir.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE work.my_package.all;
USE work.my_functions.all;


------------------------------------------------------
-- MY FIR filter
-- Version 4.0
-- 11 coefficients
-- Binary representation of the coefficients
-- Author: Vebjørn Bystrøm, NTNU 22.may 2008
------------------------------------------------------

---------------- ENTITY ------------------------------
ENTITY my_fir IS
    PORT (
        x: IN std_logic_vector(7 downto 0);
                clk, rst : IN std_logic;
        y: OUT std_logic_vector(23 downto 0)
        );
END my_fir;
------------------------------------------------------

---------------- ARCHITECTURE ------------------------

ARCHITECTURE rtl of my_fir IS

TYPE registers IS ARRAY (10 downto 0)
OF std_logic_vector(7 downto 0);

SIGNAL reg: registers;


----------------------PROCESS-----------------------------
BEGIN



        process(clk, rst, reg)

        VARIABLE coeff0: std_logic_vector(23 downto 0) ;
        VARIABLE coeff1: std_logic_vector(23 downto 0) ;
        VARIABLE coeff2: std_logic_vector(23 downto 0) ;
        VARIABLE coeff3: std_logic_vector(23 downto 0) ;
        VARIABLE coeff4: std_logic_vector(23 downto 0) ;
        VARIABLE coeff5: std_logic_vector(23 downto 0) ;
        VARIABLE coeff6: std_logic_vector(23 downto 0) ;
        VARIABLE coeff7: std_logic_vector(23 downto 0) ;
        VARIABLE coeff8: std_logic_vector(23 downto 0) ;
        VARIABLE coeff9: std_logic_vector(23 downto 0) ;
        VARIABLE coeff10: std_logic_vector(23 downto 0);


        BEGIN
        --- reset registers ---
        if (rst = '1') then
                for i in 10 downto 0 loop
                        for j in 7 downto 0 loop
                                reg(i)(j) <= '0';
                        end loop;
                end loop;
        --- update registers ---
        elsif (clk'EVENT and clk ='1') then
                for k in 10 downto 1 loop
```

```
                reg(k) <= reg(k-1);
            end loop;
            reg(0) <= x;

        end if;

        --- multiply coeff with x and accumulate ---
        coeff0 := coeff_0(reg(0)(7 downto 0));
        coeff1 := coeff_1(reg(1)(7 downto 0));
        coeff2 := coeff_2(reg(2)(7 downto 0));
        coeff3 := coeff_3(reg(3)(7 downto 0));
        coeff4 := coeff_4(reg(4)(7 downto 0));
        coeff5 := coeff_5(reg(5)(7 downto 0));
        coeff6 := coeff_6(reg(6)(7 downto 0));
        coeff7 := coeff_7(reg(7)(7 downto 0));
        coeff8 := coeff_8(reg(8)(7 downto 0));
        coeff9 := coeff_9(reg(9)(7 downto 0));
        coeff10 := coeff_10(reg(10)(7 downto 0));

        y <= coeff0 + coeff1 + coeff2 + coeff3
        + coeff4 + coeff5 + coeff6 + coeff7
        + coeff8 + coeff9  + coeff10;
        END PROCESS;
END rtl;
-------------------------------------------------
```

D.2 my_functions.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE work.my_package.all;

-------------------------------------------------
-- MY FIR filter
-- Version 4.0
-- 11 coefficients
-- Binary representation of the coefficients
-- Author: Vebjørn Bystrøm, NTNU 22.may 2008
-------------------------------------------------

PACKAGE my_functions is

FUNCTION coeff_0 (signal c: in std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_1 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_2 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_3 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_4 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_5 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_6 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_7 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_8 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_9 (signal c : std_logic_vector)
RETURN std_logic_vector;
FUNCTION coeff_10 (signal c : std_logic_vector)
RETURN std_logic_vector;

END my_functions;

PACKAGE BODY my_functions is


---------------  COEFF_0 ------------------------------------------------
        FUNCTION coeff_0 (signal c: in std_logic_vector)
 RETURN std_logic_vector IS

                VARIABLE temp : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift2 : std_logic_vector (23 DOWNTO 0);

        BEGIN
                temp := c & "0000000000000000";
                temp := (temp sra 16);
                shift1 := (temp sll 3);
```

```
            shift2 := (temp sll 4);

            return shift1 + shift2;
        END coeff_0;
-------------------------------------------------------------------------------

---------------  COEFF_1 ------------------------------------------------------
        FUNCTION coeff_1 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

            VARIABLE temp : std_logic_vector (23 DOWNTO 0);
            VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);
            VARIABLE shift2 :std_logic_vector (23 DOWNTO 0);

        BEGIN
            temp := c & "0000000000000000";
            temp := (temp sra 16);
            shift1 := (temp sll 3);
            shift2 := (temp sll 7);

            return (temp + shift1 + shift2);

        END coeff_1;
-------------------------------------------------------------------------------

---------------  COEFF_2 ------------------------------------------------------
        FUNCTION coeff_2 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

            VARIABLE temp : std_logic_vector (23 DOWNTO 0);
            VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);
            VARIABLE shift2 :std_logic_vector (23 DOWNTO 0);
            VARIABLE shift3 :std_logic_vector (23 DOWNTO 0);
            VARIABLE shift4 :std_logic_vector (23 DOWNTO 0);
            VARIABLE shift5 :std_logic_vector (23 DOWNTO 0);
            VARIABLE shift6 :std_logic_vector (23 DOWNTO 0);
            VARIABLE shift7 :std_logic_vector (23 DOWNTO 0);
            VARIABLE shift8 :std_logic_vector (23 DOWNTO 0);
            VARIABLE shift9 :std_logic_vector (23 DOWNTO 0);

        BEGIN
            temp := c & "0000000000000000";
            temp := (temp sra 16);
            shift1 := (temp sll 2);
            shift2 := (temp sll 6);
            shift3 := (temp sll 9);
            shift4 := (temp sll 10);
            shift5 := (temp sll 11);
            shift6 := (temp sll 12);
            shift7 := (temp sll 13);
            shift8 := (temp sll 14);
        shift9 := (temp sll 15);

            return (temp + shift1 + shift2 + shift3 +
            shift4 + shift5 + shift6 + shift7 + shift8
            + shift9);

        END coeff_2;
-------------------------------------------------------------------------------

---------------  COEFF_3 ------------------------------------------------------
        FUNCTION coeff_3 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

            VARIABLE temp : std_logic_vector (23 DOWNTO 0);
            VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);
            VARIABLE shift2 :std_logic_vector (23 DOWNTO 0);
            VARIABLE shift3 :std_logic_vector (23 DOWNTO 0);
            VARIABLE shift4 :std_logic_vector (23 DOWNTO 0);
            VARIABLE shift5 :std_logic_vector (23 DOWNTO 0);
            VARIABLE shift6 :std_logic_vector (23 DOWNTO 0);
            VARIABLE shift7 :std_logic_vector (23 DOWNTO 0);
            VARIABLE shift8 :std_logic_vector (23 DOWNTO 0);
            VARIABLE shift9 :std_logic_vector (23 DOWNTO 0);

        BEGIN
            temp := c & "0000000000000000";
            temp := (temp sra 16);
            shift1 := (temp sll 2);
            shift2 := (temp sll 6);
            shift3 := (temp sll 9);
            shift4 := (temp sll 10);
            shift5 := (temp sll 11);
            shift6 := (temp sll 12);
```

```
                shift7 := (temp sll 13);
                shift8 := (temp sll 14);
        shift9 := (temp sll 15);

                return (shift1 + shift2 + shift3
                + shift4 + shift5 + shift6 + shift7
                + shift8 + shift9);

        END coeff_3;
-------------------------------------------------------------------------------


---------------  COEFF_4 ------------------------------------------------------
        FUNCTION coeff_4 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE temp : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift2 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift3 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift4 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift5 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift6 :std_logic_vector (23 DOWNTO 0);

        BEGIN
                temp := c & "0000000000000000";
                temp := (temp sra 16);
                shift1 := (temp srl 1);
                shift2 := (temp srl 4);
                shift3 := (temp srl 6);
                shift4 := (temp srl 8);
                shift5 := (temp srl 9);
                shift6 := (temp srl 11);

                return (shift1 + shift2 + shift3
                + shift4 + shift5 + shift6);

        END coeff_4;
-------------------------------------------------------------------------------

---------------  COEFF_5 ------------------------------------------------------
        FUNCTION coeff_5 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE temp : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift2 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift3 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift4 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift5 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift6 :std_logic_vector (23 DOWNTO 0);

        BEGIN
                temp := c & "0000000000000000";
                temp := (temp sra 16);
                shift1 := (temp srl 3);
                shift2 := (temp srl 6);
                shift3 := (temp srl 7);
                shift4 := (temp srl 8);
                shift5 := (temp srl 10);
                shift6 := (temp srl 12);

                return (shift1 + shift2 + shift3
                + shift4 + shift5 + shift6);

        END coeff_5;
-------------------------------------------------------------------------------

---------------  COEFF_6 ------------------------------------------------------
        FUNCTION coeff_6 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE temp : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift2 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift3 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift4 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift5 :std_logic_vector (23 DOWNTO 0);
                VARIABLE shift6 :std_logic_vector (23 DOWNTO 0);

        BEGIN
                temp := c & "0000000000000000";
                temp := (temp sra 16);
                shift1 := (temp srl 1);
```

```
            shift2 := (temp srl 4);
            shift3 := (temp srl 6);
            shift4 := (temp srl 8);
            shift5 := (temp srl 9);
            shift6 := (temp srl 11);

            return (shift1 + shift2 + shift3
            + shift4 + shift5 + shift6);

    END coeff_6;
-------------------------------------------------------------------------------

---------------  COEFF_7 ------------------------------------------------------
    FUNCTION coeff_7 (signal c: in std_logic_vector)
    RETURN std_logic_vector IS

            VARIABLE temp : std_logic_vector (23 DOWNTO 0);
            VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);
            VARIABLE shift2 :std_logic_vector (23 DOWNTO 0);
            VARIABLE shift3 :std_logic_vector (23 DOWNTO 0);
            VARIABLE shift4 :std_logic_vector (23 DOWNTO 0);
            VARIABLE shift5 :std_logic_vector (23 DOWNTO 0);
            VARIABLE shift6 :std_logic_vector (23 DOWNTO 0);
            VARIABLE shift7 :std_logic_vector (23 DOWNTO 0);
            VARIABLE shift8 :std_logic_vector (23 DOWNTO 0);
            VARIABLE shift9 :std_logic_vector (23 DOWNTO 0);

    BEGIN
            temp := c & "0000000000000000";
            temp := (temp sra 16);
            shift1 := (temp sll 2);
            shift2 := (temp sll 6);
            shift3 := (temp sll 9);
            shift4 := (temp sll 10);
            shift5 := (temp sll 11);
            shift6 := (temp sll 12);
            shift7 := (temp sll 13);
            shift8 := (temp sll 14);
    shift9 := (temp sll 15);

            return (shift1 + shift2 + shift3
            + shift4 + shift5 + shift6 + shift7
            + shift8 + shift9);

    END coeff_7;
-------------------------------------------------------------------------------


---------------  COEFF_8 ------------------------------------------------------
    FUNCTION coeff_8 (signal c: in std_logic_vector)
    RETURN std_logic_vector IS

            VARIABLE temp : std_logic_vector (23 DOWNTO 0);
            VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);
            VARIABLE shift2 :std_logic_vector (23 DOWNTO 0);
            VARIABLE shift3 :std_logic_vector (23 DOWNTO 0);
            VARIABLE shift4 :std_logic_vector (23 DOWNTO 0);
            VARIABLE shift5 :std_logic_vector (23 DOWNTO 0);
            VARIABLE shift6 :std_logic_vector (23 DOWNTO 0);
            VARIABLE shift7 :std_logic_vector (23 DOWNTO 0);
            VARIABLE shift8 :std_logic_vector (23 DOWNTO 0);
            VARIABLE shift9 :std_logic_vector (23 DOWNTO 0);

    BEGIN
            temp := c & "0000000000000000";
            temp := (temp sra 16);
            shift1 := (temp sll 2);
            shift2 := (temp sll 6);
            shift3 := (temp sll 9);
            shift4 := (temp sll 10);
            shift5 := (temp sll 11);
            shift6 := (temp sll 12);
            shift7 := (temp sll 13);
            shift8 := (temp sll 14);
    shift9 := (temp sll 15);

            return (temp + shift1 + shift2 + shift3
            + shift4 + shift5 + shift6 + shift7
            + shift8 + shift9);

    END coeff_8;
-------------------------------------------------------------------------------

---------------  COEFF_9 ------------------------------------------------------
```

```
        FUNCTION coeff_9 (signal c: in std_logic_vector)
        RETURN std_logic_vector IS

                VARIABLE temp : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift2 :std_logic_vector (23 DOWNTO 0);

        BEGIN
                temp := c & "0000000000000000";
                temp := (temp sra 16);
                shift1 := (temp sll 3);
                shift2 := (temp sll 7);

                return (temp + shift1 + shift2);

        END coeff_9;
-------------------------------------------------------------------------------

---------------  COEFF_10 -----------------------------------------------------
        FUNCTION coeff_10 (signal c: in std_logic_vector)
 RETURN std_logic_vector IS

                VARIABLE temp : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift1 : std_logic_vector (23 DOWNTO 0);
                VARIABLE shift2 : std_logic_vector (23 DOWNTO 0);

        BEGIN
                temp := c & "0000000000000000";
                temp := (temp sra 16);
                shift1 := (temp sll 3);
                shift2 := (temp sll 4);

                return shift1 + shift2;

        END coeff_10;
-------------------------------------------------------------------------------

END my_functions;
```

D.3 my_fir_tb.vhd

```
library ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

-------------------------------------------------
-- MY FIR filter
-- Version 4.0
-- 11 coefficients
-- Binary representation of the coefficients
-- Author: Vebjørn Bystrøm, NTNU 22.may 2008
-------------------------------------------------

---------------- ENTITY -------------------------
entity my_fir_tb is

end my_fir_tb;

-------------------------------------------------

---------------- ARCHITECTURE -------------------
architecture my_fir_tb_arch of my_fir_tb is


---------------- COMPONENTS ---------------------
component my_fir
        PORT (
                x:              IN std_logic_vector (7 DOWNTO 0);
                clk, rst: IN std_logic;
                y:              OUT std_logic_vector (23 DOWNTO 0)
        );
end component;
-------------------------------------------------

signal x                : std_logic_vector (7 DOWNTO 0)
:= "00000000";
signal y                : std_logic_vector (23 DOWNTO 0)
:= "000000000000000000000000";
signal clk              : std_logic := '1';
signal rst              : std_logic;


begin
```

```
-- device under test --
dut: my_fir
port map (
        x => x,
        y => y,
        clk => clk,
        rst => rst
);

-- clk generator --
clk <= not clk after 1 ms;


-- stimulate divece under test --
wstim: process
        begin
                rst <= '1';
                wait until clk'event and clk = '1';
                rst <= '0';
                x <= "00000000"; --0
                wait until clk'event and clk = '1';
                x <= "00000001"; --1
                wait until clk'event and clk = '1';
                x <= "00000010"; --2
                wait until clk'event and clk = '1';
                x <= "00000011"; --3
                wait until clk'event and clk = '1';
                x <= "00000100"; --4
                wait until clk'event and clk = '1';
                x <= "00000101"; --5
                wait until clk'event and clk = '1';
                x <= "00000110"; --6
                wait until clk'event and clk = '1';
                x <= "00000111"; --7
                wait until clk'event and clk = '1';
                x <= "11111111"; -- -1
                wait until clk'event and clk = '1';
                x <= "11111110"; -- -2
                wait until clk'event and clk = '1';
                x <= "11111101"; -- -3
                wait until clk'event and clk = '1';
                wait;
        end process;
end;
```

E. PACKAGE USED IN MY VHDL CODE

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;


-------------------------------------------------
-- SHIFT OPERTATORS
-- Version 1.0
-- Author: Vebjørn Bystrøm, NTNU 8.may 2008
-------------------------------------------------

PACKAGE my_package is
        function "sll" ( l : std_logic_vector;  r : integer )
        RETURN std_logic_vector;
   function "sll" ( l : std_ulogic_vector; r : integer )
   RETURN std_ulogic_vector;

   function "srl" ( l : std_logic_vector;  r : integer )
   RETURN std_logic_vector;
   function "srl" ( l : std_ulogic_vector; r : integer )
   RETURN std_ulogic_vector;

   function "sla" ( l : std_logic_vector;  r : integer )
   RETURN std_logic_vector;
   function "sla" ( l : std_ulogic_vector; r : integer )
   RETURN std_ulogic_vector;

   function "sra" ( l : std_logic_vector;  r : integer )
   RETURN std_logic_vector;
   function "sra" ( l : std_ulogic_vector; r : integer )
   RETURN std_ulogic_vector;

   function "rol" ( l : std_logic_vector;  r : integer )
   RETURN std_logic_vector;
   function "rol" ( l : std_ulogic_vector; r : integer )
   RETURN std_ulogic_vector;

   function "ror" ( l : std_logic_vector;  r : integer )
   RETURN std_logic_vector;
   function "ror" ( l : std_ulogic_vector; r : integer )
   RETURN std_ulogic_vector;
END my_package;

PACKAGE BODY my_package is
    FUNCTION "sll" ( l : std_logic_vector;  r : integer )
        RETURN std_logic_vector IS
        ALIAS lv : std_logic_vector ( 1 TO l'LENGTH ) IS l;
        VARIABLE result : std_logic_vector ( l'RANGE )
                := (OTHERS => '0');
        ALIAS resultv : std_logic_vector ( 1 TO l'LENGTH ) IS result;
    BEGIN
        IF r = 0 OR l'LENGTH = 0 THEN
            RETURN l;
        ELSIF r > 0 THEN
            IF r < l'LENGTH THEN
                resultv(1 TO l'LENGTH - r) := lv(r+1 TO l'LENGTH);
            END IF;
            RETURN result;
        ELSE
            RETURN l SRL (-r);
        END IF;
    END "sll";
    ---------------------------------------------------------------------
    FUNCTION "sll" ( l : std_ulogic_vector; r : integer )
        RETURN std_ulogic_vector IS
        ALIAS lv : std_ulogic_vector ( 1 TO l'LENGTH ) IS l;
        VARIABLE result : std_ulogic_vector ( l'RANGE )
                := (OTHERS => '0');
        ALIAS resultv : std_ulogic_vector ( 1 TO l'LENGTH ) IS result;
    BEGIN
```

```
    IF r = 0 OR l'LENGTH = 0 THEN
        RETURN l;
    ELSIF r > 0 THEN
        IF r < l'LENGTH THEN
            resultv(1 TO l'LENGTH - r) := lv(r+1 TO l'LENGTH);
        END IF;
        RETURN result;
    ELSE
        RETURN l SRL (-r);
    END IF;
END "sll";

-------------------------------------------------------------------------
-- srl
-------------------------------------------------------------------------
FUNCTION "srl" ( l : std_logic_vector;  r : integer )
    RETURN std_logic_vector IS
    ALIAS lv : std_logic_vector ( 1 TO l'LENGTH ) IS l;
    VARIABLE result : std_logic_vector ( l'RANGE )
            := (OTHERS => '0');
    ALIAS resultv : std_logic_vector ( 1 TO l'LENGTH ) IS result;
BEGIN
    IF r = 0 OR l'LENGTH = 0 THEN
        RETURN l;
    ELSIF r > 0 THEN
        IF r < l'LENGTH THEN
            resultv(r+1 TO l'LENGTH) := lv(1 TO l'LENGTH - r);
        END IF;
        RETURN result;
    ELSE
        RETURN l SLL (-r);
    END IF;
END "srl";
-------------------------------------------------------------------------
FUNCTION "srl" ( l : std_ulogic_vector; r : integer )
    RETURN std_ulogic_vector IS
    ALIAS lv : std_ulogic_vector ( 1 TO l'LENGTH ) IS l;
    VARIABLE result : std_ulogic_vector ( l'RANGE )
            := (OTHERS => '0');
    ALIAS resultv : std_ulogic_vector ( 1 TO l'LENGTH ) IS result;
BEGIN
    IF r = 0 OR l'LENGTH = 0 THEN
        RETURN l;
    ELSIF r > 0 THEN
        IF r < l'LENGTH THEN
            resultv(r+1 TO l'LENGTH) := lv(1 TO l'LENGTH - r);
        END IF;
        RETURN result;
    ELSE
        RETURN l SLL (-r);
    END IF;
END "srl";

-------------------------------------------------------------------------
-- sla
-------------------------------------------------------------------------
FUNCTION "sla" ( l : std_logic_vector;  r : integer )
    RETURN std_logic_vector IS
    ALIAS lv : std_logic_vector ( 1 TO l'LENGTH ) IS l;
    VARIABLE result : std_logic_vector ( l'RANGE )
            := (OTHERS => l(l'RIGHT));
    ALIAS resultv : std_logic_vector ( 1 TO l'LENGTH ) IS result;
BEGIN
    IF r = 0 OR l'LENGTH = 0 THEN
        RETURN l;
    ELSIF r > 0 THEN
        IF r < l'LENGTH THEN
            resultv(1 TO l'LENGTH - r) := lv(r+1 TO l'LENGTH);
        END IF;
        RETURN result;
    ELSE
        RETURN l SRA (-r);
    END IF;
END "sla";
-------------------------------------------------------------------------
FUNCTION "sla" ( l : std_ulogic_vector; r : integer )
    RETURN std_ulogic_vector IS
    ALIAS lv : std_ulogic_vector ( 1 TO l'LENGTH ) IS l;
    VARIABLE result : std_ulogic_vector ( l'RANGE )
            := (OTHERS => l(l'RIGHT));
    ALIAS resultv : std_ulogic_vector ( 1 TO l'LENGTH ) IS result;
BEGIN
    IF r = 0 OR l'LENGTH = 0 THEN
        RETURN l;
```

```
    ELSIF r > 0 THEN
        IF r < l'LENGTH THEN
            resultv(1 TO l'LENGTH - r) := lv(r+1 TO l'LENGTH);
        END IF;
        RETURN result;
    ELSE
        RETURN l SRA (-r);
    END IF;
END "sla";

-------------------------------------------------------------------------
-- sra
-------------------------------------------------------------------------
FUNCTION "sra" ( l : std_logic_vector;  r : integer )
    RETURN std_logic_vector IS
    ALIAS lv : std_logic_vector ( 1 TO l'LENGTH ) IS l;
    VARIABLE result : std_logic_vector ( l'RANGE )
            := (OTHERS => l(l'LEFT));
    ALIAS resultv : std_logic_vector ( 1 TO l'LENGTH ) IS result;
BEGIN
    IF r = 0 OR l'LENGTH = 0 THEN
        RETURN l;
    ELSIF r > 0 THEN
        IF r < l'LENGTH THEN
            resultv(r+1 TO l'LENGTH) := lv(1 TO l'LENGTH - r);
        END IF;
        RETURN result;
    ELSE
        RETURN l SLA (-r);
    END IF;
END "sra";
-------------------------------------------------------------------------
FUNCTION "sra" ( l : std_ulogic_vector; r : integer )
    RETURN std_ulogic_vector IS
    ALIAS lv : std_ulogic_vector ( 1 TO l'LENGTH ) IS l;
    VARIABLE result : std_ulogic_vector ( l'RANGE )
            := (OTHERS => l(l'LEFT));
    ALIAS resultv : std_ulogic_vector ( 1 TO l'LENGTH ) IS result;
BEGIN
    IF r = 0 OR l'LENGTH = 0 THEN
        RETURN l;
    ELSIF r > 0 THEN
        IF r < l'LENGTH THEN
            resultv(r+1 TO l'LENGTH) := lv(1 TO l'LENGTH - r);
        END IF;
        RETURN result;
    ELSE
        RETURN l SLA (-r);
    END IF;
END "sra";

-------------------------------------------------------------------------
-- rol
-------------------------------------------------------------------------
FUNCTION "rol" ( l : std_logic_vector;  r : integer )
    RETURN std_logic_vector IS
    ALIAS lv : std_logic_vector ( 1 TO l'LENGTH ) IS l;
    VARIABLE result : std_logic_vector ( l'RANGE );
    ALIAS resultv : std_logic_vector ( 1 TO l'LENGTH ) IS result;
    VARIABLE rv : integer;
BEGIN
    IF r = 0 OR l'LENGTH = 0 THEN
        RETURN l;
    ELSIF r > 0 THEN
        rv := r MOD l'length;
        resultv(1 TO l'LENGTH - rv ) := lv(rv+1 TO l'LENGTH);
        resultv(l'LENGTH - rv + 1 TO l'length) := lv(1 TO rv);
        RETURN result;
    ELSE
        RETURN l ROR (-r);
    END IF;
END "rol";
-------------------------------------------------------------------------
FUNCTION "rol" ( l : std_ulogic_vector; r : integer )
    RETURN std_ulogic_vector IS
    ALIAS lv : std_ulogic_vector ( 1 TO l'LENGTH ) IS l;
    VARIABLE result : std_ulogic_vector ( l'RANGE );
    ALIAS resultv : std_ulogic_vector ( 1 TO l'LENGTH ) IS result;
    VARIABLE rv : integer;
BEGIN
    IF r = 0 OR l'LENGTH = 0 THEN
        RETURN l;
    ELSIF r > 0 THEN
        rv := r MOD l'length;
```

```
            resultv(1 TO l'LENGTH - rv ) := lv(rv+1 TO l'LENGTH);
            resultv(l'LENGTH - rv + 1 TO l'LENGTH) := lv(1 TO rv);
            RETURN result;
        ELSE
            RETURN l ROR (-r);
        END IF;
    END "rol";

    ------------------------------------------------------------------
    -- ror
    ------------------------------------------------------------------
    FUNCTION "ror" ( l : std_logic_vector;  r : integer )
        RETURN std_logic_vector IS
        ALIAS lv : std_logic_vector ( 1 TO l'LENGTH ) IS l;
        VARIABLE result : std_logic_vector ( l'RANGE );
        ALIAS resultv : std_logic_vector ( 1 TO l'LENGTH ) IS result;
        VARIABLE rv : integer;
    BEGIN
        IF r = 0 OR l'LENGTH = 0 THEN
            RETURN l;
        ELSIF r > 0 THEN
            rv := r MOD l'length;
            resultv(rv+1 TO l'LENGTH) := lv(1 TO l'LENGTH - rv);
            resultv(1 TO rv) := lv(l'LENGTH - rv + 1 TO l'LENGTH);
            RETURN result;
        ELSE
            RETURN l ROL (-r);
        END IF;
    END "ror";
    ------------------------------------------------------------------
    FUNCTION "ror" ( l : std_ulogic_vector; r : integer )
        RETURN std_ulogic_vector IS
        ALIAS lv : std_ulogic_vector ( 1 TO l'LENGTH ) IS l;
        VARIABLE result : std_ulogic_vector ( l'RANGE );
        ALIAS resultv : std_ulogic_vector ( 1 TO l'LENGTH ) IS result;
        VARIABLE rv : integer;
    BEGIN
        IF r = 0 OR l'LENGTH = 0 THEN
            RETURN l;
        ELSIF r > 0 THEN
            rv := r MOD l'length;
            resultv(rv+1 TO l'LENGTH) := lv(1 TO l'LENGTH - rv);
            resultv(1 TO rv) := lv(l'LENGTH - rv + 1 TO l'LENGTH);
            RETURN result;
        ELSE
            RETURN l ROL (-r);
        END IF;
    END "ror";
END;
```

F. SYNTHESIS REPORTS

F.1 Synthesis Report for version 1

```
Release 10.1 - xst K.31 (nt)
Copyright (c) 1995-2008 Xilinx, Inc. All rights reserved.
--> Parameter TMPDIR set to D:/Master/Latex/my_FIR/my_FIR/xst/projnav.tmp
Total REAL time to Xst completion: 0.00 secs
Total CPU time to Xst completion: 0.17 secs
--> Parameter xsthdpdir set to D:/Master/Latex/my_FIR/my_FIR/xst
Total REAL time to Xst completion: 0.00 secs
Total CPU time to Xst completion: 0.17 secs
--> Reading design: my_fir.prj
TABLE OF CONTENTS
1) Synthesis Options Summary
2) HDL Compilation
3) Design Hierarchy Analysis
4) HDL Analysis
5) HDL Synthesis
5.1) HDL Synthesis Report
6) Advanced HDL Synthesis
6.1) Advanced HDL Synthesis Report
7) Low Level Synthesis
8) Partition Report
9) Final Report
9.1) Device utilization summary
9.2) Partition Resource Summary
9.3) TIMING REPORT
=========================================================================
* Synthesis Options Summary *
=========================================================================
---- Source Parameters
Input File Name : "my_fir.prj"
Input Format : mixed
Ignore Synthesis Constraint File : NO
---- Target Parameters
Output File Name : "my_fir"
Output Format : NGC
Target Device : xc3s500e-4-fg320
---- Source Options
Top Module Name : my_fir
Automatic FSM Extraction : YES
FSM Encoding Algorithm : Auto
Safe Implementation : No
FSM Style : lut
RAM Extraction : Yes
RAM Style : Auto
ROM Extraction : Yes
Mux Style : Auto
Decoder Extraction : YES
Priority Encoder Extraction : YES
Shift Register Extraction : YES
Logical Shifter Extraction : YES
XOR Collapsing : YES
ROM Style : Auto
Mux Extraction : YES
Resource Sharing : YES
Asynchronous To Synchronous : NO
Multiplier Style : auto
Automatic Register Balancing : No
---- Target Options
Add IO Buffers : YES
Global Maximum Fanout : 500
Add Generic Clock Buffer(BUFG) : 24
Register Duplication : YES
Slice Packing : YES
Optimize Instantiated Primitives : NO
Use Clock Enable : Yes
Use Synchronous Set : Yes
```

```
Use Synchronous Reset : Yes
Pack IO Registers into IOBs : auto
Equivalent register Removal : YES
---- General Options
Optimization Goal : Speed
Optimization Effort : 1
Library Search Order : my_fir.lso
Keep Hierarchy : NO
Netlist Hierarchy : as_optimized
RTL Output : Yes
Global Optimization : AllClockNets
Read Cores : YES
Write Timing Constraints : NO
Cross Clock Analysis : NO
Hierarchy Separator : /
Bus Delimiter : <>
Case Specifier : maintain
Slice Utilization Ratio : 100
BRAM Utilization Ratio : 100
Verilog 2001 : YES
Auto BRAM Packing : NO
Slice Utilization Ratio Delta : 5
=========================================================================
=========================================================================
* HDL Compilation *
=========================================================================
Compiling vhdl file "D:/Master/Latex/my_FIR/my_FIR/my_package.vhd" in
Library work.
Architecture my_package of Entity my_package is up to date.
Compiling vhdl file "D:/Master/Latex/my_FIR/my_FIR/my_functions.vhd" in
Library work.
Architecture my_functions of Entity my_functions is up to date.
Compiling vhdl file "D:/Master/Latex/my_FIR/my_FIR/my_fir.vhd" in
Library work.
Entity <my_fir> compiled.
Entity <my_fir> (Architecture <rtl>) compiled.
=========================================================================
* Design Hierarchy Analysis *
=========================================================================
Analyzing hierarchy for entity <my_fir> in library <work>
(architecture <rtl>).
=========================================================================
* HDL Analysis *
=========================================================================
Analyzing Entity <my_fir> in library <work> (Architecture <rtl>).
Entity <my_fir> analyzed. Unit <my_fir> generated.
=========================================================================
* HDL Synthesis *
=========================================================================
Performing bidirectional port resolution...
Synthesizing Unit <my_fir>.
Related source file is "D:/Master/Latex/my_FIR/my_FIR/my_fir.vhd".
Found 24-bit adder for signal <y>.
Found 24-bit subtractor for signal <coeff0$sub0000> created at line 57.
Found 24-bit adder for signal <coeff1$add0000> created at line 75.
Found 24-bit adder for signal <coeff1$addsub0000> created at line 75.
Found 24-bit subtractor for signal <coeff10$sub0000> created at line 279.
Found 24-bit adder for signal <coeff2$add0000> created at line 96.
Found 24-bit subtractor for signal <coeff2$addsub0000>.
Found 24-bit adder for signal <coeff2$addsub0001> created at line 96.
Found 24-bit adder for signal <coeff3$add0000> created at line 117.
Found 24-bit subtractor for signal <coeff3$addsub0000>.
Found 24-bit adder for signal <coeff4$add0000> created at line 144.
Found 24-bit subtractor for signal <coeff4$addsub0000> created at line 144.
Found 24-bit subtractor for signal <coeff4$addsub0001> created at line 144.
Found 24-bit adder for signal <coeff4$addsub0002> created at line 144.
Found 24-bit adder for signal <coeff4$addsub0003> created at line 144.
Found 24-bit adder for signal <coeff5$add0000> created at line 169.
Found 24-bit subtractor for signal <coeff5$addsub0000> created at line 169.
Found 24-bit subtractor for signal <coeff5$addsub0001> created at line 169.
Found 24-bit subtractor for signal <coeff5$addsub0002> created at line 169.
Found 24-bit adder for signal <coeff6$add0000> created at line 196.
Found 24-bit subtractor for signal <coeff6$addsub0000> created at line 196.
Found 24-bit subtractor for signal <coeff6$addsub0001> created at line 196.
Found 24-bit adder for signal <coeff6$addsub0002> created at line 196.
Found 24-bit adder for signal <coeff6$addsub0003> created at line 196.
Found 24-bit adder for signal <coeff7$add0000> created at line 218.
Found 24-bit subtractor for signal <coeff7$addsub0000>.
Found 24-bit adder for signal <coeff8$add0000> created at line 241.
Found 24-bit subtractor for signal <coeff8$addsub0000>.
Found 24-bit adder for signal <coeff8$addsub0001> created at line 241.
Found 24-bit adder for signal <coeff9$add0000> created at line 261.
Found 24-bit adder for signal <coeff9$addsub0000> created at line 261.
Found 24-bit adder for signal <y$addsub0000> created at line 85.
```

```
Found 24-bit adder for signal <y$addsub0001> created at line 85.
Found 24-bit adder for signal <y$addsub0002> created at line 85.
Found 24-bit adder for signal <y$addsub0003> created at line 85.
Found 24-bit adder for signal <y$addsub0004> created at line 85.
Found 24-bit adder for signal <y$addsub0005> created at line 85.
Found 24-bit adder for signal <y$addsub0006> created at line 85.
Found 24-bit adder for signal <y$addsub0007> created at line 85.
Found 24-bit adder for signal <y$addsub0008> created at line 85.
Summary:
inferred 88 D-type flip-flop(s).
inferred 40 Adder/Subtractor(s).
Unit <my_fir> synthesized.
=========================================================================
HDL Synthesis Report
Macro Statistics
# Adders/Subtractors : 40
24-bit adder : 27
24-bit subtractor : 13
# Registers : 88
1-bit register : 88
=========================================================================
=========================================================================
* Advanced HDL Synthesis *
=========================================================================
Loading device for application Rf_Device from file '3s500e.nph' in
environment C:\Xilinx\10.1\ISE.
=========================================================================
Advanced HDL Synthesis Report
Macro Statistics
# Adders/Subtractors : 40
24-bit adder : 27
24-bit subtractor : 13
# Registers : 88
Flip-Flops : 88
=========================================================================
=========================================================================
* Low Level Synthesis *
=========================================================================
Optimizing unit <my_fir> ...
Mapping all equations...
Building and optimizing final netlist ...
Found area constraint ratio of 100 (+ 5) on block my_fir, actual ratio
is 9.
FlipFlop reg<1>_7 has been replicated 1 time(s)
FlipFlop reg<3>_7 has been replicated 1 time(s)
FlipFlop reg<4>_1 has been replicated 1 time(s)
FlipFlop reg<4>_5 has been replicated 1 time(s)
FlipFlop reg<4>_7 has been replicated 1 time(s)
FlipFlop reg<5>_7 has been replicated 1 time(s)
FlipFlop reg<6>_1 has been replicated 1 time(s)
FlipFlop reg<6>_5 has been replicated 1 time(s)
FlipFlop reg<6>_7 has been replicated 1 time(s)
FlipFlop reg<7>_7 has been replicated 1 time(s)
FlipFlop reg<9>_7 has been replicated 1 time(s)
Final Macro Processing ...
=========================================================================
Final Register Report
Macro Statistics
# Registers : 99
Flip-Flops : 99
=========================================================================
=========================================================================
* Partition Report *
=========================================================================
Partition Implementation Status
-------------------------------
No Partitions were found in this design.
-------------------------------
=========================================================================
* Final Report *
=========================================================================
Final Results
RTL Top Level Output File Name : my_fir.ngr
Top Level Output File Name : my_fir
Output Format : NGC
Optimization Goal : Speed
Keep Hierarchy : NO
Design Statistics
# IOs : 34
Cell Usage :
# BELS : 1206
# GND : 1
# INV : 9
# LUT2 : 84
```

```
# LUT3 : 214
# LUT4 : 321
# MUXCY : 280
# MUXF5 : 12
# VCC : 1
# XORCY : 284
# FlipFlops/Latches : 99
# FDC : 99
# Clock Buffers : 1
# BUFGP : 1
# IO Buffers : 33
# IBUF : 9
# OBUF : 24
=========================================================================
Device utilization summary:
---------------------------

Selected Device : 3s500efg320-4
Number of Slices: 343 out of 4656 7%
Number of Slice Flip Flops: 99 out of 9312 1%
Number of 4 input LUTs: 628 out of 9312 6%
Number of IOs: 34
Number of bonded IOBs: 34 out of 232 14%
Number of GCLKs: 1 out of 24 4%
---------------------------
Partition Resource Summary:
---------------------------

No Partitions were found in this design.
---------------------------

=========================================================================
TIMING REPORT
NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
GENERATED AFTER PLACE-and-ROUTE.
Clock Information:
------------------
-----------------------------------+------------------------+-------+
Clock Signal | Clock buffer(FF name) | Load |
-----------------------------------+------------------------+-------+
clk | BUFGP | 99 |
-----------------------------------+------------------------+-------+
Asynchronous Control Signals Information:
-----------------------------------------
-----------------------------------+------------------------+-------+
Control Signal | Buffer(FF name) | Load |
-----------------------------------+------------------------+-------+
rst | IBUF | 99 |
-----------------------------------+------------------------+-------+
Timing Summary:
---------------
Speed Grade: -4
Minimum period: 1.984ns (Maximum Frequency: 504.032MHz)
Minimum input arrival time before clock: 1.946ns
Maximum output required time after clock: 40.720ns
Maximum combinational path delay: No path found
Timing Detail:
--------------
All values displayed in nanoseconds (ns)
=========================================================================
Timing constraint: Default period analysis for Clock 'clk'
Clock period: 1.984ns (frequency: 504.032MHz)
Total number of paths / destination ports: 91 / 91
-------------------------------------------------------------------------
Delay: 1.984ns (Levels of Logic = 0)
Source: reg<2>_7 (FF)
Destination: reg<3>_7 (FF)
Source Clock: clk rising
Destination Clock: clk rising
Data Path: reg<2>_7 to reg<3>_7
Gate Net
Cell:in->out fanout Delay Delay Logical Name (Net Name)
---------------------------------------- ------------
FDC:C->Q 19 0.591 1.085 reg<2>_7 (reg<2>_7)
FDC:D 0.308 reg<3>_7
----------------------------------------
Total 1.984ns (0.899ns logic, 1.085ns route)
(45.3% logic, 54.7% route)
=========================================================================
Timing constraint: Default OFFSET IN BEFORE for Clock 'clk'
Total number of paths / destination ports: 8 / 8
-------------------------------------------------------------------------
Offset: 1.946ns (Levels of Logic = 1)
Source: x<0> (PAD)
Destination: reg<0>_0 (FF)
Destination Clock: clk rising
```

```
Data Path: x<0> to reg<0>_0
Gate Net
Cell:in->out fanout Delay Delay Logical Name (Net Name)
---------------------------------------- ------------
IBUF:I->O 1 1.218 0.420 x_0_IBUF (x_0_IBUF)
FDC:D 0.308 reg<0>_0
----------------------------------------
Total 1.946ns (1.526ns logic, 0.420ns route)
(78.4% logic, 21.6% route)
```

```
Timing constraint: Default OFFSET OUT AFTER for Clock 'clk'
Total number of paths / destination ports: 48395541245 / 23
```

```
Offset: 40.720ns (Levels of Logic = 36)
Source: reg<1>_3 (FF)
Destination: y<23> (PAD)
Source Clock: clk rising
Data Path: reg<1>_3 to y<23>
Gate Net
Cell:in->out fanout Delay Delay Logical Name (Net Name)
---------------------------------------- ------------
FDC:C->Q 12 0.591 1.136 reg<1>_3 (reg<1>_3)
LUT4:I0->O 2 0.704 0.622 Madd_coeff1_add0000_Madd_cy<5>11
(Madd_coeff1_add0000_Madd_cy<5>)
LUT4:I0->O 2 0.704 0.622 Madd_coeff1_add0000_Madd_cy<6>11
(Madd_coeff1_add0000_Madd_cy<6>)
LUT4:I0->O 2 0.704 0.622 Madd_coeff1_add0000_Madd_cy<7>11
(Madd_coeff1_add0000_Madd_cy<7>)
LUT4:I0->O 2 0.704 0.622 Madd_coeff1_add0000_Madd_cy<8>11
(Madd_coeff1_add0000_Madd_cy<8>)
LUT4:I0->O 2 0.704 0.622 Madd_coeff1_add0000_Madd_cy<9>11
(Madd_coeff1_add0000_Madd_cy<9>)
LUT4:I0->O 4 0.704 0.762 Madd_coeff1_add0000_Madd_cy<10>11
(Madd_coeff1_add0000_Madd_cy<10>)
LUT3:I0->O 2 0.704 0.622 Madd_coeff1_add0000_Madd_xor<11>11
(coeff1_add0000<11>)
LUT3:I0->O 1 0.704 0.595 Madd_y_addsub0001C101 (Madd_y_addsub0001C10)
LUT4:I0->O 1 0.704 0.000 Madd_y_addsub0001_Madd_lut<12>
(Madd_y_addsub0001_Madd_lut<12>)
MUXCY:S->O 1 0.464 0.000 Madd_y_addsub0001_Madd_cy<12>
(Madd_y_addsub0001_Madd_cy<12>)
XORCY:CI->O 2 0.804 0.526 Madd_y_addsub0001_Madd_xor<13>
(y_addsub0001<13>)
LUT3:I1->O 1 0.704 0.595 Madd_y_addsub0002C121 (Madd_y_addsub0002C12)
LUT4:I0->O 1 0.704 0.000 Madd_y_addsub0002_Madd_lut<14>
(Madd_y_addsub0002_Madd_lut<14>)
MUXCY:S->O 1 0.464 0.000 Madd_y_addsub0002_Madd_cy<14>
(Madd_y_addsub0002_Madd_cy<14>)
XORCY:CI->O 2 0.804 0.526 Madd_y_addsub0002_Madd_xor<15>
(y_addsub0002<15>)
LUT3:I1->O 1 0.704 0.595 Madd_y_addsub0003C141 (Madd_y_addsub0003C14)
LUT4:I0->O 1 0.704 0.000 Madd_y_addsub0003_Madd_lut<16>
(Madd_y_addsub0003_Madd_lut<16>)
MUXCY:S->O 1 0.464 0.000 Madd_y_addsub0003_Madd_cy<16>
(Madd_y_addsub0003_Madd_cy<16>)
XORCY:CI->O 2 0.804 0.526 Madd_y_addsub0003_Madd_xor<17>
(y_addsub0003<17>)
LUT3:I1->O 1 0.704 0.595 Madd_y_addsub0004C161 (Madd_y_addsub0004C16)
LUT4:I0->O 1 0.704 0.000 Madd_y_addsub0004_Madd_lut<18>
(Madd_y_addsub0004_Madd_lut<18>)
MUXCY:S->O 1 0.464 0.000 Madd_y_addsub0004_Madd_cy<18>
(Madd_y_addsub0004_Madd_cy<18>)
XORCY:CI->O 2 0.804 0.526 Madd_y_addsub0004_Madd_xor<19>
(y_addsub0004<19>)
LUT3:I1->O 1 0.704 0.595 Madd_y_addsub0005C181 (Madd_y_addsub0005C18)
LUT4:I0->O 1 0.704 0.000 Madd_y_addsub0005_Madd_lut<20>
(Madd_y_addsub0005_Madd_lut<20>)
MUXCY:S->O 1 0.464 0.000 Madd_y_addsub0005_Madd_cy<20>
(Madd_y_addsub0005_Madd_cy<20>)
XORCY:CI->O 3 0.804 0.566 Madd_y_addsub0005_Madd_xor<21>
(y_addsub0005<21>)
LUT4:I2->O 1 0.704 0.000 Madd_y_addsub0006_Madd_lut<21>
(Madd_y_addsub0006_Madd_lut<21>)
XORCY:LI->O 2 0.527 0.526 Madd_y_addsub0006_Madd_xor<21>
(y_addsub0006<21>)
LUT3:I1->O 1 0.704 0.595 Madd_y_addsub0008C201 (Madd_y_addsub0008C20)
LUT4:I0->O 1 0.704 0.000 Madd_y_addsub0008_Madd_lut<22>
(Madd_y_addsub0008_Madd_lut<22>)
MUXCY:S->O 0 0.464 0.000 Madd_y_addsub0008_Madd_cy<22>
(Madd_y_addsub0008_Madd_cy<22>)
XORCY:CI->O 1 0.804 0.595 Madd_y_addsub0008_Madd_xor<23>
(y_addsub0008<23>)
LUT2:I0->O 0 0.704 0.000 Madd_y_lut<23> (Madd_y_lut<23>)
XORCY:LI->O 1 0.527 0.420 Madd_y_xor<23> (y_23_OBUF)
```

```
OBUF:I->O  3.272  y_23_OBUF (y<23>)
----------------------------------------
Total  40.720ns (27.309ns logic, 13.411ns route)
(67.1% logic, 32.9% route)
==========================================================================

Total REAL time to Xst completion: 24.00 secs
Total CPU time to Xst completion: 24.20 secs
-->
Total memory usage is 173420 kilobytes
Number of errors   :  0 ( 0 filtered)
Number of warnings :  0 ( 0 filtered)
Number of infos    :  0 ( 0 filtered)
```

F.2 Synthesis Report for version 2

```
Release 10.1 - xst K.31 (nt)
Copyright (c) 1995-2008 Xilinx, Inc. All rights reserved.
--> Parameter TMPDIR set to D:/Master/Latex/my_fir_version_2/my_fir
/xst/projnav.tmp
Total REAL time to Xst completion: 0.00 secs
Total CPU time to Xst completion: 0.17 secs
--> Parameter xsthdpdir set to D:/Master/Latex/my_fir_version_2/my_fir/xst
Total REAL time to Xst completion: 0.00 secs
Total CPU time to Xst completion: 0.17 secs
--> Reading design: my_fir.prj
TABLE OF CONTENTS
1) Synthesis Options Summary
2) HDL Compilation
3) Design Hierarchy Analysis
4) HDL Analysis
5) HDL Synthesis
5.1) HDL Synthesis Report
6) Advanced HDL Synthesis
6.1) Advanced HDL Synthesis Report
7) Low Level Synthesis
8) Partition Report
9) Final Report
9.1) Device utilization summary
9.2) Partition Resource Summary
9.3) TIMING REPORT
=========================================================================
* Synthesis Options Summary *
=========================================================================
---- Source Parameters
Input File Name : "my_fir.prj"
Input Format : mixed
Ignore Synthesis Constraint File : NO
---- Target Parameters
Output File Name : "my_fir"
Output Format : NGC
Target Device : xc3s500e-4-fg320
---- Source Options
Top Module Name : my_fir
Automatic FSM Extraction : YES
FSM Encoding Algorithm : Auto
Safe Implementation : No
FSM Style : lut
RAM Extraction : Yes
RAM Style : Auto
ROM Extraction : Yes
Mux Style : Auto
Decoder Extraction : YES
Priority Encoder Extraction : YES
Shift Register Extraction : YES
Logical Shifter Extraction : YES
XOR Collapsing : YES
ROM Style : Auto
Mux Extraction : YES
Resource Sharing : YES
Asynchronous To Synchronous : NO
Multiplier Style : auto
Automatic Register Balancing : No
---- Target Options
Add IO Buffers : YES
Global Maximum Fanout : 500
Add Generic Clock Buffer(BUFG) : 24
Register Duplication : YES
Slice Packing : YES
Optimize Instantiated Primitives : NO
Use Clock Enable : Yes
Use Synchronous Set : Yes
Use Synchronous Reset : Yes
Pack IO Registers into IOBs : auto
Equivalent register Removal : YES
---- General Options
Optimization Goal : Speed
Optimization Effort : 1
Library Search Order : my_fir.lso
Keep Hierarchy : NO
Netlist Hierarchy : as_optimized
RTL Output : Yes
Global Optimization : AllClockNets
Read Cores : YES
Write Timing Constraints : NO
Cross Clock Analysis : NO
```

```
Hierarchy Separator : /
Bus Delimiter : <>
Case Specifier : maintain
Slice Utilization Ratio : 100
BRAM Utilization Ratio : 100
Verilog 2001 : YES
Auto BRAM Packing : NO
Slice Utilization Ratio Delta : 5
=========================================================================
=========================================================================
* HDL Compilation *
=========================================================================
Compiling vhdl file "D:/Master/Latex/my_fir_version_2/my_fir/my_package.vhd"
in Library work.
Architecture my_package of Entity my_package is up to date.
Compiling vhdl file "D:/Master/Latex/my_fir_version_2/my_fir/my_functions.vhd"
in Library work.
Package <my_functions> compiled.
Package body <my_functions> compiled.
Compiling vhdl file "D:/Master/Latex/my_fir_version_2/my_fir/my_fir.vhd"
in Library work.
Entity <my_fir> compiled.
Entity <my_fir> (Architecture <rtl>) compiled.
=========================================================================
* Design Hierarchy Analysis *
=========================================================================
Analyzing hierarchy for entity <my_fir> in library <work>
(architecture <rtl>).
=========================================================================
* HDL Analysis *
=========================================================================
Analyzing Entity <my_fir> in library <work> (Architecture <rtl>).
Entity <my_fir> analyzed. Unit <my_fir> generated.
=========================================================================
* HDL Synthesis *
=========================================================================
Performing bidirectional port resolution...
Synthesizing Unit <my_fir>.
Related source file is "D:/Master/Latex/my_fir_version_2/my_fir
/my_fir.vhd".
Found 24-bit adder for signal <y>.
Found 24-bit subtractor for signal <coeff1$sub0000> created at line 75.
Found 24-bit subtractor for signal <coeff2$addsub0000>.
Found 24-bit adder for signal <coeff2$addsub0001> created at line 104.
Found 24-bit subtractor for signal <coeff2$sub0000> created at line 104.
Found 24-bit subtractor for signal <coeff3$addsub0000>.
Found 24-bit adder for signal <coeff3$addsub0001> created at line 132.
Found 24-bit subtractor for signal <coeff3$sub0000> created at line 132.
Found 24-bit adder for signal <coeff4$add0000> created at line 157.
Found 24-bit adder for signal <coeff4$addsub0000> created at line 157.
Found 24-bit adder for signal <coeff5$addsub0000> created at line 185.
Found 24-bit subtractor for signal <coeff5$addsub0001> created at line 185.
Found 24-bit subtractor for signal <coeff5$sub0000> created at line 185.
Found 24-bit adder for signal <coeff6$add0000> created at line 209.
Found 24-bit adder for signal <coeff6$addsub0000> created at line 209.
Found 24-bit subtractor for signal <coeff7$addsub0000>.
Found 24-bit adder for signal <coeff7$addsub0001> created at line 236.
Found 24-bit subtractor for signal <coeff7$sub0000> created at line 236.
Found 24-bit subtractor for signal <coeff8$addsub0000>.
Found 24-bit adder for signal <coeff8$addsub0001> created at line 265.
Found 24-bit subtractor for signal <coeff8$sub0000> created at line 265.
Found 24-bit subtractor for signal <coeff9$sub0000> created at line 286.
Found 24-bit adder for signal <y$addsub0000> created at line 85.
Found 24-bit adder for signal <y$addsub0001> created at line 85.
Found 24-bit adder for signal <y$addsub0002> created at line 85.
Found 24-bit adder for signal <y$addsub0003> created at line 85.
Found 24-bit adder for signal <y$addsub0004> created at line 85.
Found 24-bit adder for signal <y$addsub0005> created at line 85.
Found 24-bit adder for signal <y$addsub0006> created at line 85.
Found 24-bit adder for signal <y$addsub0007> created at line 85.
Found 24-bit adder for signal <y$addsub0008> created at line 85.
Summary:
inferred 88 D-type flip-flop(s).
inferred 31 Adder/Subtractor(s).
Unit <my_fir> synthesized.
=========================================================================
HDL Synthesis Report
Macro Statistics
# Adders/Subtractors : 31
24-bit adder : 19
24-bit subtractor : 12
# Registers : 88
1-bit register : 88
=========================================================================
```

```
=========================================================================
* Advanced HDL Synthesis *
=========================================================================
Loading device for application Rf_Device from file '3s500e.nph' in
environment C:\Xilinx\10.1\ISE.
=========================================================================
Advanced HDL Synthesis Report
Macro Statistics
# Adders/Subtractors : 31
24-bit adder : 19
24-bit subtractor : 12
# Registers : 88
Flip-Flops : 88
=========================================================================
=========================================================================
* Low Level Synthesis *
=========================================================================
Optimizing unit <my_fir> ...
Mapping all equations...
Building and optimizing final netlist ...
Found area constraint ratio of 100 (+ 5) on block my_fir, actual
ratio is 7.
FlipFlop reg<0>_7 has been replicated 1 time(s)
INFO:Xst:1843 - HDL ADVISOR - FlipFlop reg<0>_7 connected to a
primary input has been replicated
FlipFlop reg<4>_7 has been replicated 1 time(s)
FlipFlop reg<5>_7 has been replicated 1 time(s)
FlipFlop reg<6>_7 has been replicated 1 time(s)
Final Macro Processing ...
=========================================================================
Final Register Report
Macro Statistics
# Registers : 92
Flip-Flops : 92
=========================================================================
=========================================================================
* Partition Report *
=========================================================================
Partition Implementation Status
-------------------------------
No Partitions were found in this design.
-------------------------------
=========================================================================
* Final Report *
=========================================================================
Final Results
RTL Top Level Output File Name : my_fir.ngr
Top Level Output File Name : my_fir
Output Format : NGC
Optimization Goal : Speed
Keep Hierarchy : NO
Design Statistics
# IOs : 34
Cell Usage :
# BELS : 994
# GND : 1
# INV : 17
# LUT2 : 96
# LUT3 : 181
# LUT4 : 190
# MUXCY : 250
# MUXF5 : 1
# VCC : 1
# XORCY : 257
# FlipFlops/Latches : 92
# FDC : 92
# Clock Buffers : 1
# BUFGP : 1
# IO Buffers : 33
# IBUF : 9
# OBUF : 24
=========================================================================
Device utilization summary:
---------------------------
Selected Device : 3s500efg320-4
Number of Slices: 267 out of 4656 5%
Number of Slice Flip Flops: 92 out of 9312 0%
Number of 4 input LUTs: 484 out of 9312 5%
Number of IOs: 34
Number of bonded IOBs: 34 out of 232 14%
Number of GCLKs: 1 out of 24 4%
---------------------------
Partition Resource Summary:
---------------------------
```

```
No Partitions were found in this design.
--------------------------------

==========================================================================
TIMING REPORT
NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
GENERATED AFTER PLACE-and-ROUTE.
Clock Information:
------------------
-----------------------------------+------------------------+-------+
Clock Signal | Clock buffer(FF name) | Load |
-----------------------------------+------------------------+-------+
clk | BUFGP | 92 |
-----------------------------------+------------------------+-------+
Asynchronous Control Signals Information:
----------------------------------------
-----------------------------------+------------------------+-------+
Control Signal | Buffer(FF name) | Load |
-----------------------------------+------------------------+-------+
rst | IBUF | 92 |
-----------------------------------+------------------------+-------+
Timing Summary:
---------------
Speed Grade: -4
Minimum period: 1.950ns (Maximum Frequency: 512.821MHz)
Minimum input arrival time before clock: 1.973ns
Maximum output required time after clock: 32.072ns
Maximum combinational path delay: No path found
Timing Detail:
--------------
All values displayed in nanoseconds (ns)
==========================================================================
Timing constraint: Default period analysis for Clock 'clk'
Clock period: 1.950ns (frequency: 512.821MHz)
Total number of paths / destination ports: 83 / 83
--------------------------------------------------------------------------
Delay: 1.950ns (Levels of Logic = 0)
Source: reg<3>_7 (FF)
Destination: reg<4>_7 (FF)
Source Clock: clk rising
Destination Clock: clk rising
Data Path: reg<3>_7 to reg<4>_7
Gate Net
Cell:in->out fanout Delay Delay Logical Name (Net Name)
---------------------------------------- ------------
FDC:C->Q 17 0.591 1.051 reg<3>_7 (reg<3>_7)
FDC:D 0.308 reg<4>_7
----------------------------------------
Total 1.950ns (0.899ns logic, 1.051ns route)
(46.1% logic, 53.9% route)
==========================================================================
Timing constraint: Default OFFSET IN BEFORE for Clock 'clk'
Total number of paths / destination ports: 9 / 9
--------------------------------------------------------------------------
Offset: 1.973ns (Levels of Logic = 1)
Source: x<7> (PAD)
Destination: reg<0>_7 (FF)
Destination Clock: clk rising
Data Path: x<7> to reg<0>_7
Gate Net
Cell:in->out fanout Delay Delay Logical Name (Net Name)
---------------------------------------- ------------
IBUF:I->O 2 1.218 0.447 x_7_IBUF (x_7_IBUF)
FDC:D 0.308 reg<0>_7
----------------------------------------
Total 1.973ns (1.526ns logic, 0.447ns route)
(77.3% logic, 22.7% route)
==========================================================================
Timing constraint: Default OFFSET OUT AFTER for Clock 'clk'
Total number of paths / destination ports: 1141597533 / 24
--------------------------------------------------------------------------
Offset: 32.072ns (Levels of Logic = 29)
Source: reg<2>_3 (FF)
Destination: y<23> (PAD)
Source Clock: clk rising
Data Path: reg<2>_3 to y<23>
Gate Net
Cell:in->out fanout Delay Delay Logical Name (Net Name)
---------------------------------------- ------------
FDC:C->Q 7 0.591 0.883 reg<2>_3 (reg<2>_3)
LUT4:I0->O 2 0.704 0.622 Msub_coeff2_addsub0000_cy<5>11
(Msub_coeff2_addsub0000_cy<5>)
LUT3:I0->O 2 0.704 0.622 Msub_coeff2_addsub0000_cy<6>11
(Msub_coeff2_addsub0000_cy<6>)
```

```
LUT3:I0->O 3 0.704 0.706 Msub_coeff2_addsub0000_cy<7>11
(Msub_coeff2_addsub0000_cy<7>)
LUT3:I0->O 4 0.704 0.666 Msub_coeff2_addsub0000_cy<8>11
(Msub_coeff2_addsub0000_cy<8>)
LUT4:I1->O 1 0.704 0.000 Madd_coeff2_addsub0001_lut<11>
(Madd_coeff2_addsub0001_lut<11>)
MUXCY:S->O 1 0.464 0.000 Madd_coeff2_addsub0001_cy<11>
(Madd_coeff2_addsub0001_cy<11>)
XORCY:CI->O 1 0.804 0.595 Madd_coeff2_addsub0001_xor<12>
(coeff2_addsub0001<13>)
LUT2:I0->O 1 0.704 0.000 Msub_coeff2_sub0000_lut<13>
(Msub_coeff2_sub0000_lut<13>)
MUXCY:S->O 1 0.464 0.000 Msub_coeff2_sub0000_cy<13>
(Msub_coeff2_sub0000_cy<13>)
XORCY:CI->O 3 0.804 0.566 Msub_coeff2_sub0000_xor<14>
(coeff2_sub0000<14>)
LUT4:I2->O 1 0.704 0.000 Madd_y_addsub0001_Madd_lut<14>
(Madd_y_addsub0001_Madd_lut<14>)
MUXCY:S->O 1 0.464 0.000 Madd_y_addsub0001_Madd_cy<14>
(Madd_y_addsub0001_Madd_cy<14>)
XORCY:CI->O 2 0.804 0.526 Madd_y_addsub0001_Madd_xor<15>
(y_addsub0001<15>)
LUT3:I1->O 1 0.704 0.595 Madd_y_addsub0003C141
(Madd_y_addsub0003C14)
LUT4:I0->O 1 0.704 0.000 Madd_y_addsub0003_Madd_lut<16>
(Madd_y_addsub0003_Madd_lut<16>)
MUXCY:S->O 1 0.464 0.000 Madd_y_addsub0003_Madd_cy<16>
(Madd_y_addsub0003_Madd_cy<16>)
XORCY:CI->O 2 0.804 0.526 Madd_y_addsub0003_Madd_xor<17>
(y_addsub0003<17>)
LUT3:I1->O 1 0.704 0.595 Madd_y_addsub0005C161
(Madd_y_addsub0005C16)
LUT4:I0->O 1 0.704 0.000 Madd_y_addsub0005_Madd_lut<18>
(Madd_y_addsub0005_Madd_lut<18>)
MUXCY:S->O 1 0.464 0.000 Madd_y_addsub0005_Madd_cy<18>
(Madd_y_addsub0005_Madd_cy<18>)
XORCY:CI->O 3 0.804 0.610 Madd_y_addsub0005_Madd_xor<19>
(y_addsub0005<19>)
LUT3:I1->O 0 0.704 0.000 Madd_y_addsub0007C181 (Madd_y_addsub0007C18)
MUXCY:DI->O 1 0.888 0.000 Madd_y_addsub0007_Madd_cy<20>
(Madd_y_addsub0007_Madd_cy<20>)
XORCY:CI->O 2 0.804 0.526 Madd_y_addsub0007_Madd_xor<21>
(y_addsub0007<21>)
LUT3:I1->O 1 0.704 0.595 Madd_yC201 (Madd_yC20)
LUT3:I0->O 1 0.704 0.000 Madd_y_Madd_lut<22> (Madd_y_Madd_lut<22>)
MUXCY:S->O 0 0.464 0.000 Madd_y_Madd_cy<22> (Madd_y_Madd_cy<22>)
XORCY:CI->O 1 0.804 0.420 Madd_y_Madd_xor<23> (y_23_OBUF)
OBUF:I->O 3.272 y_23_OBUF (y<23>)
----------------------------------------
Total 32.072ns (23.019ns logic, 9.053ns route)
(71.8% logic, 28.2% route)
========================================================================

Total REAL time to Xst completion: 19.00 secs
Total CPU time to Xst completion: 18.44 secs
-->
Total memory usage is 169772 kilobytes
Number of errors : 0 ( 0 filtered)
Number of warnings : 0 ( 0 filtered)
Number of infos : 1 ( 0 filtered)
```

F.3 Synthesis Report for version 3

```
Release 10.1 - xst K.31 (nt)
Copyright (c) 1995-2008 Xilinx, Inc. All rights reserved.
--> Parameter TMPDIR set to D:/Master/Latex/my_fir_version_3/
my_fir/xst/projnav.tmp
Total REAL time to Xst completion: 0.00 secs
Total CPU time to Xst completion: 0.17 secs
--> Parameter xsthdpdir set to D:/Master/Latex/my_fir_version_3/
my_fir/xst
Total REAL time to Xst completion: 0.00 secs
Total CPU time to Xst completion: 0.17 secs
--> Reading design: my_fir.prj
TABLE OF CONTENTS
1) Synthesis Options Summary
2) HDL Compilation
3) Design Hierarchy Analysis
4) HDL Analysis
5) HDL Synthesis
5.1) HDL Synthesis Report
6) Advanced HDL Synthesis
6.1) Advanced HDL Synthesis Report
7) Low Level Synthesis
8) Partition Report
9) Final Report
9.1) Device utilization summary
9.2) Partition Resource Summary
9.3) TIMING REPORT
=========================================================================
* Synthesis Options Summary *
=========================================================================
---- Source Parameters
Input File Name : "my_fir.prj"
Input Format : mixed
Ignore Synthesis Constraint File : NO
---- Target Parameters
Output File Name : "my_fir"
Output Format : NGC
Target Device : xc3s500e-4-fg320
---- Source Options
Top Module Name : my_fir
Automatic FSM Extraction : YES
FSM Encoding Algorithm : Auto
Safe Implementation : No
FSM Style : lut
RAM Extraction : Yes
RAM Style : Auto
ROM Extraction : Yes
Mux Style : Auto
Decoder Extraction : YES
Priority Encoder Extraction : YES
Shift Register Extraction : YES
Logical Shifter Extraction : YES
XOR Collapsing : YES
ROM Style : Auto
Mux Extraction : YES
Resource Sharing : YES
Asynchronous To Synchronous : NO
Multiplier Style : auto
Automatic Register Balancing : No
---- Target Options
Add IO Buffers : YES
Global Maximum Fanout : 500
Add Generic Clock Buffer(BUFG) : 24
Register Duplication : YES
Slice Packing : YES
Optimize Instantiated Primitives : NO
Use Clock Enable : Yes
Use Synchronous Set : Yes
Use Synchronous Reset : Yes
Pack IO Registers into IOBs : auto
Equivalent register Removal : YES
---- General Options
Optimization Goal : Speed
Optimization Effort : 1
Library Search Order : my_fir.lso
Keep Hierarchy : NO
Netlist Hierarchy : as_optimized
RTL Output : Yes
Global Optimization : AllClockNets
Read Cores : YES
Write Timing Constraints : NO
```

```
Cross Clock Analysis : NO
Hierarchy Separator : /
Bus Delimiter : <>
Case Specifier : maintain
Slice Utilization Ratio : 100
BRAM Utilization Ratio : 100
Verilog 2001 : YES
Auto BRAM Packing : NO
Slice Utilization Ratio Delta : 5
=========================================================================
=========================================================================
* HDL Compilation *
=========================================================================
Compiling vhdl file "D:/Master/Latex/my_fir_version_3/my_fir/
my_package.vhd" in Library work.
Architecture my_package of Entity my_package is up to date.
Compiling vhdl file "D:/Master/Latex/my_fir_version_3/my_fir/
my_functions.vhd" in Library work.
Architecture my_functions of Entity my_functions is up to date.
Compiling vhdl file "D:/Master/Latex/my_fir_version_3/my_fir/
my_fir.vhd" in Library work.
Entity <my_fir> compiled.
Entity <my_fir> (Architecture <rtl>) compiled.
=========================================================================
* Design Hierarchy Analysis *
=========================================================================
Analyzing hierarchy for entity <my_fir> in library <work>
(architecture <rtl>).
=========================================================================
* HDL Analysis *
=========================================================================
Analyzing Entity <my_fir> in library <work> (Architecture <rtl>).
Entity <my_fir> analyzed. Unit <my_fir> generated.
=========================================================================
* HDL Synthesis *
=========================================================================
Performing bidirectional port resolution...
Synthesizing Unit <my_fir>.
Related source file is "D:/Master/Latex/my_fir_version_3/my_fir/
my_fir.vhd".
Found 24-bit adder for signal <y>.
Found 8x16-bit multiplier for signal <coeff0$mult0000>
created at line 53.
Found 8x16-bit multiplier for signal <coeff1$mult0000>
created at line 68.
Found 8x16-bit multiplier for signal <coeff10$mult0000>
created at line 214.
Found 8x16-bit multiplier for signal <coeff2$mult0000>
created at line 84.
Found 8x16-bit multiplier for signal <coeff3$mult0000>
created at line 100.
Found 8x16-bit multiplier for signal <coeff4$mult0000>
created at line 117.
Found 8x16-bit multiplier for signal <coeff5$mult0000>
created at line 133.
Found 8x16-bit multiplier for signal <coeff6$mult0000>
created at line 149.
Found 8x16-bit multiplier for signal <coeff7$mult0000>
created at line 165.
Found 8x16-bit multiplier for signal <coeff8$mult0000>
created at line 182.
Found 8x16-bit multiplier for signal <coeff9$mult0000>
created at line 198.
Found 24-bit adder for signal <y$addsub0000> created at line 86.
Found 24-bit adder for signal <y$addsub0001> created at line 86.
Found 24-bit adder for signal <y$addsub0002> created at line 86.
Found 24-bit adder for signal <y$addsub0003> created at line 86.
Found 24-bit adder for signal <y$addsub0004> created at line 86.
Found 24-bit adder for signal <y$addsub0005> created at line 86.
Found 24-bit adder for signal <y$addsub0006> created at line 86.
Found 24-bit adder for signal <y$addsub0007> created at line 86.
Found 24-bit adder for signal <y$addsub0008> created at line 86.
Summary:
inferred 88 D-type flip-flop(s).
inferred 10 Adder/Subtractor(s).
inferred 11 Multiplier(s).
Unit <my_fir> synthesized.
=========================================================================
HDL Synthesis Report
Macro Statistics
# Multipliers : 11
8x16-bit multiplier : 11
# Adders/Subtractors : 10
24-bit adder : 10
```

```
# Registers : 88
1-bit register : 88
=========================================================================
=========================================================================
* Advanced HDL Synthesis *
=========================================================================
Loading device for application Rf_Device from file '3s500e.nph'
in environment C:\Xilinx\10.1\ISE.
Synthesizing (advanced) Unit <my_fir>.
INFO:Xst:2385 - HDL ADVISOR - You can improve the performance
of the multiplier
Mmult_coeff10_mult0000 by adding 1 register level(s).
INFO:Xst:2385 - HDL ADVISOR - You can improve the performance
of the multiplier
Mmult_coeff9_mult0000 by adding 1 register level(s).
INFO:Xst:2385 - HDL ADVISOR - You can improve the performance
of the multiplier
Mmult_coeff8_mult0000 by adding 1 register level(s).
INFO:Xst:2385 - HDL ADVISOR - You can improve the performance
of the multiplier
Mmult_coeff7_mult0000 by adding 1 register level(s).
INFO:Xst:2385 - HDL ADVISOR - You can improve the performance
of the multiplier
Mmult_coeff6_mult0000 by adding 1 register level(s).
INFO:Xst:2385 - HDL ADVISOR - You can improve the performance
of the multiplier
Mmult_coeff5_mult0000 by adding 1 register level(s).
INFO:Xst:2385 - HDL ADVISOR - You can improve the performance
of the multiplier
Mmult_coeff4_mult0000 by adding 1 register level(s).
INFO:Xst:2385 - HDL ADVISOR - You can improve the performance
of the multiplier
Mmult_coeff3_mult0000 by adding 1 register level(s).
INFO:Xst:2385 - HDL ADVISOR - You can improve the performance
of the multiplier
Mmult_coeff2_mult0000 by adding 1 register level(s).
INFO:Xst:2385 - HDL ADVISOR - You can improve the performance
of the multiplier
Mmult_coeff1_mult0000 by adding 1 register level(s).
INFO:Xst:2385 - HDL ADVISOR - You can improve the performance
of the multiplier
Mmult_coeff0_mult0000 by adding 1 register level(s).
Unit <my_fir> synthesized (advanced).
=========================================================================
Advanced HDL Synthesis Report
Macro Statistics
# Multipliers : 11
8x16-bit multiplier : 11
# Adders/Subtractors : 10
24-bit adder : 10
# Registers : 88
Flip-Flops : 88
=========================================================================
=========================================================================
* Low Level Synthesis *
=========================================================================
Optimizing unit <my_fir> ...
Mapping all equations...
Building and optimizing final netlist ...
Found area constraint ratio of 100 (+ 5) on block my_fir,
actual ratio is 4.
Final Macro Processing ...
=========================================================================
Final Register Report
Macro Statistics
# Registers : 88
Flip-Flops : 88
=========================================================================
=========================================================================
* Partition Report *
=========================================================================
Partition Implementation Status
-------------------------------
No Partitions were found in this design.
-------------------------------
=========================================================================
* Final Report *
=========================================================================
Final Results
RTL Top Level Output File Name : my_fir.ngr
Top Level Output File Name : my_fir
Output Format : NGC
Optimization Goal : Speed
Keep Hierarchy : NO
```

```
Design Statistics
# IOs : 34
Cell Usage :
# BELS : 465
# GND : 1
# LUT2 : 5
# LUT3 : 111
# LUT4 : 111
# MUXCY : 115
# MUXF5 : 1
# VCC : 1
# XORCY : 120
# FlipFlops/Latches : 88
# FDC : 88
# Clock Buffers : 1
# BUFGP : 1
# IO Buffers : 33
# IBUF : 9
# OBUF : 24
# MULTs : 11
# MULT18X18SIO : 11
=========================================================================
Device utilization summary:
---------------------------

Selected Device : 3s500efg320-4
Number of Slices: 130 out of 4656 2%
Number of Slice Flip Flops: 88 out of 9312 0%
Number of 4 input LUTs: 227 out of 9312 2%
Number of IOs: 34
Number of bonded IOBs: 34 out of 232 14%
Number of MULT18X18SIOs: 11 out of 20 55%
Number of GCLKs: 1 out of 24 4%
---------------------------
Partition Resource Summary:
---------------------------

No Partitions were found in this design.
---------------------------

=========================================================================
TIMING REPORT
NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
GENERATED AFTER PLACE-and-ROUTE.
Clock Information:
------------------
-----------------------------------+------------------------+-------+
Clock Signal | Clock buffer(FF name) | Load |
-----------------------------------+------------------------+-------+
clk | BUFGP | 88 |
-----------------------------------+------------------------+-------+
Asynchronous Control Signals Information:
-----------------------------------------
-----------------------------------+------------------------+-------+
Control Signal | Buffer(FF name) | Load |
-----------------------------------+------------------------+-------+
rst | IBUF | 88 |
-----------------------------------+------------------------+-------+
Timing Summary:
---------------

Speed Grade: -4
Minimum period: 1.346ns (Maximum Frequency: 742.942MHz)
Minimum input arrival time before clock: 1.946ns
Maximum output required time after clock: 28.509ns
Maximum combinational path delay: No path found
Timing Detail:
--------------

All values displayed in nanoseconds (ns)
=========================================================================
Timing constraint: Default period analysis for Clock 'clk'
Clock period: 1.346ns (frequency: 742.942MHz)
Total number of paths / destination ports: 80 / 80
-------------------------------------------------------------------------
Delay: 1.346ns (Levels of Logic = 0)
Source: reg<0>_0 (FF)
Destination: reg<1>_0 (FF)
Source Clock: clk rising
Destination Clock: clk rising
Data Path: reg<0>_0 to reg<1>_0
Gate Net
Cell:in->out fanout Delay Delay Logical Name (Net Name)
---------------------------------------- ------------
FDC:C->Q 2 0.591 0.447 reg<0>_0 (reg<0>_0)
FDC:D 0.308 reg<1>_0
----------------------------------------
Total 1.346ns (0.899ns logic, 0.447ns route)
```

```
(66.8% logic, 33.2% route)
=========================================================================
Timing constraint: Default OFFSET IN BEFORE for Clock 'clk'
Total number of paths / destination ports: 8 / 8
-------------------------------------------------------------------------
Offset: 1.946ns (Levels of Logic = 1)
Source: x<0> (PAD)
Destination: reg<0>_0 (FF)
Destination Clock: clk rising
Data Path: x<0> to reg<0>_0
Gate Net
Cell:in->out fanout Delay Delay Logical Name (Net Name)
---------------------------------------- ------------
IBUF:I->O 1 1.218 0.420 x_0_IBUF (x_0_IBUF)
FDC:D 0.308 reg<0>_0
----------------------------------------
Total 1.946ns (1.526ns logic, 0.420ns route)
(78.4% logic, 21.6% route)
=========================================================================
Timing constraint: Default OFFSET OUT AFTER for Clock 'clk'
Total number of paths / destination ports: 400137001 / 24
-------------------------------------------------------------------------
Offset: 28.509ns (Levels of Logic = 28)
Source: reg<1>_7 (FF)
Destination: y<23> (PAD)
Source Clock: clk rising
Data Path: reg<1>_7 to y<23>
Gate Net
Cell:in->out fanout Delay Delay Logical Name (Net Name)
---------------------------------------- ------------
FDC:C->Q 2 0.591 0.447 reg<1>_7 (reg<1>_7)
MULT18X18SIO:A7->P7 2 4.344 0.622 Mmult_coeff1_mult0000
(coeff1_mult0000<7>)
LUT3:I0->O 1 0.704 0.595 Madd_y_addsub0001C61
(Madd_y_addsub0001C6)
LUT4:I0->O 1 0.704 0.000 Madd_y_addsub0001_Madd_lut<8>
(Madd_y_addsub0001_Madd_lut<8>)
MUXCY:S->O 1 0.464 0.000 Madd_y_addsub0001_Madd_cy<8>
(Madd_y_addsub0001_Madd_cy<8>)
XORCY:CI->O 2 0.804 0.526 Madd_y_addsub0001_Madd_xor<9>
(y_addsub0001<9>)
LUT3:I1->O 1 0.704 0.595 Madd_y_addsub0003C81
(Madd_y_addsub0003C8)
LUT4:I0->O 1 0.704 0.000 Madd_y_addsub0003_Madd_lut<10>
(Madd_y_addsub0003_Madd_lut<10>)
MUXCY:S->O 1 0.464 0.000 Madd_y_addsub0003_Madd_cy<10>
(Madd_y_addsub0003_Madd_cy<10>)
XORCY:CI->O 2 0.804 0.526 Madd_y_addsub0003_Madd_xor<11>
(y_addsub0003<11>)
LUT3:I1->O 1 0.704 0.595 Madd_y_addsub0005C101
(Madd_y_addsub0005C10)
LUT4:I0->O 1 0.704 0.000 Madd_y_addsub0005_Madd_lut<12>
(Madd_y_addsub0005_Madd_lut<12>)
MUXCY:S->O 1 0.464 0.000 Madd_y_addsub0005_Madd_cy<12>
(Madd_y_addsub0005_Madd_cy<12>)
XORCY:CI->O 2 0.804 0.526 Madd_y_addsub0005_Madd_xor<13>
(y_addsub0005<13>)
LUT3:I1->O 1 0.704 0.595 Madd_y_addsub0007C121
(Madd_y_addsub0007C12)
LUT4:I0->O 1 0.704 0.000 Madd_y_addsub0007_Madd_lut<14>
(Madd_y_addsub0007_Madd_lut<14>)
MUXCY:S->O 1 0.464 0.000 Madd_y_addsub0007_Madd_cy<14>
(Madd_y_addsub0007_Madd_cy<14>)
XORCY:CI->O 2 0.804 0.526 Madd_y_addsub0007_Madd_xor<15>
(y_addsub0007<15>)
LUT3:I1->O 1 0.704 0.595 Madd_yC141 (Madd_yC14)
LUT4:I0->O 1 0.704 0.000 Madd_y_Madd_lut<16>
(Madd_y_Madd_lut<16>)
MUXCY:S->O 1 0.464 0.000 Madd_y_Madd_cy<16>
(Madd_y_Madd_cy<16>)
MUXCY:CI->O 1 0.059 0.000 Madd_y_Madd_cy<17> (Madd_y_Madd_cy<17>)
MUXCY:CI->O 1 0.059 0.000 Madd_y_Madd_cy<18> (Madd_y_Madd_cy<18>)
MUXCY:CI->O 1 0.059 0.000 Madd_y_Madd_cy<19> (Madd_y_Madd_cy<19>)
MUXCY:CI->O 1 0.059 0.000 Madd_y_Madd_cy<20> (Madd_y_Madd_cy<20>)
MUXCY:CI->O 1 0.059 0.000 Madd_y_Madd_cy<21> (Madd_y_Madd_cy<21>)
MUXCY:CI->O 0 0.059 0.000 Madd_y_Madd_cy<22> (Madd_y_Madd_cy<22>)
XORCY:CI->O 1 0.804 0.420 Madd_y_Madd_xor<23> (y_23_OBUF)
OBUF:I->O 3.272 y_23_OBUF (y<23>)
----------------------------------------
Total 28.509ns (21.941ns logic, 6.568ns route)
(77.0% logic, 23.0% route)
=========================================================================
Total REAL time to Xst completion: 10.00 secs
Total CPU time to Xst completion: 10.03 secs
```

```
-->
Total memory usage is 163628 kilobytes
Number of errors : 0 ( 0 filtered)
Number of warnings : 0 ( 0 filtered)
Number of infos : 11 ( 0 filtered)
```

F.4 Synthesis Report for version 4

```
Release 10.1 - xst K.31 (nt)
Copyright (c) 1995-2008 Xilinx, Inc. All rights reserved.
--> Parameter TMPDIR set to D:/Master/Latex/my_fir_version_4/my_fir/
xst/projnav.tmp
Total REAL time to Xst completion: 0.00 secs
Total CPU time to Xst completion: 0.16 secs
--> Parameter xsthdpdir set to D:/Master/Latex/my_fir_version_4/
my_fir/xst
Total REAL time to Xst completion: 0.00 secs
Total CPU time to Xst completion: 0.16 secs
--> Reading design: my_fir.prj
TABLE OF CONTENTS
1) Synthesis Options Summary
2) HDL Compilation
3) Design Hierarchy Analysis
4) HDL Analysis
5) HDL Synthesis
5.1) HDL Synthesis Report
6) Advanced HDL Synthesis
6.1) Advanced HDL Synthesis Report
7) Low Level Synthesis
8) Partition Report
9) Final Report
9.1) Device utilization summary
9.2) Partition Resource Summary
9.3) TIMING REPORT
=========================================================================
* Synthesis Options Summary *
=========================================================================
---- Source Parameters
Input File Name : "my_fir.prj"
Input Format : mixed
Ignore Synthesis Constraint File : NO
---- Target Parameters
Output File Name : "my_fir"
Output Format : NGC
Target Device : xc3s500e-4-fg320
---- Source Options
Top Module Name : my_fir
Automatic FSM Extraction : YES
FSM Encoding Algorithm : Auto
Safe Implementation : No
FSM Style : lut
RAM Extraction : Yes
RAM Style : Auto
ROM Extraction : Yes
Mux Style : Auto
Decoder Extraction : YES
Priority Encoder Extraction : YES
Shift Register Extraction : YES
Logical Shifter Extraction : YES
XOR Collapsing : YES
ROM Style : Auto
Mux Extraction : YES
Resource Sharing : YES
Asynchronous To Synchronous : NO
Multiplier Style : auto
Automatic Register Balancing : No
---- Target Options
Add IO Buffers : YES
Global Maximum Fanout : 500
Add Generic Clock Buffer(BUFG) : 24
Register Duplication : YES
Slice Packing : YES
Optimize Instantiated Primitives : NO
Use Clock Enable : Yes
Use Synchronous Set : Yes
Use Synchronous Reset : Yes
Pack IO Registers into IOBs : auto
Equivalent register Removal : YES
---- General Options
Optimization Goal : Speed
Optimization Effort : 1
Library Search Order : my_fir.lso
Keep Hierarchy : NO
Netlist Hierarchy : as_optimized
RTL Output : Yes
Global Optimization : AllClockNets
Read Cores : YES
Write Timing Constraints : NO
```

```
Cross Clock Analysis : NO
Hierarchy Separator : /
Bus Delimiter : <>
Case Specifier : maintain
Slice Utilization Ratio : 100
BRAM Utilization Ratio : 100
Verilog 2001 : YES
Auto BRAM Packing : NO
Slice Utilization Ratio Delta : 5
=========================================================================
=========================================================================
* HDL Compilation *
=========================================================================
Compiling vhdl file "D:/Master/Latex/my_fir_version_4/my_fir/
my_package.vhd" in Library work.
Architecture my_package of Entity my_package is up to date.
Compiling vhdl file "D:/Master/Latex/my_fir_version_4/my_fir/
my_functions.vhd" in Library work.
Architecture my_functions of Entity my_functions is up to date.
Compiling vhdl file "D:/Master/Latex/my_fir_version_4/my_fir/
my_fir.vhd" in Library work.
Entity <my_fir> compiled.
Entity <my_fir> (Architecture <rtl>) compiled.
=========================================================================
* Design Hierarchy Analysis *
=========================================================================
Analyzing hierarchy for entity <my_fir> in library <work>
(architecture <rtl>).
=========================================================================
* HDL Analysis *
=========================================================================
Analyzing Entity <my_fir> in library <work> (Architecture <rtl>).
Entity <my_fir> analyzed. Unit <my_fir> generated.
=========================================================================
* HDL Synthesis *
=========================================================================
Performing bidirectional port resolution...
Synthesizing Unit <my_fir>.
Related source file is "D:/Master/Latex/my_fir_version_4/my_fir/
my_fir.vhd".
Found 24-bit adder for signal <y>.
Found 24-bit adder for signal <coeff0$add0000> created at line 58.
Found 24-bit adder for signal <coeff1$add0000> created at line 76.
Found 24-bit adder for signal <coeff1$addsub0000> created at line 76.
Found 24-bit adder for signal <coeff10$add0000> created at line 340.
Found 24-bit adder for signal <coeff2$add0000> created at line 109.
Found 24-bit adder for signal <coeff2$addsub0000> created at line 109.
Found 24-bit adder for signal <coeff2$addsub0001> created at line 109.
Found 24-bit adder for signal <coeff2$addsub0002> created at line 109.
Found 24-bit adder for signal <coeff2$addsub0003> created at line 109.
Found 24-bit adder for signal <coeff2$addsub0004> created at line 109.
Found 24-bit adder for signal <coeff2$addsub0005> created at line 109.
Found 24-bit adder for signal <coeff2$addsub0006> created at line 109.
Found 24-bit adder for signal <coeff2$addsub0007> created at line 109.
Found 24-bit adder for signal <coeff3$add0000> created at line 144.
Found 24-bit adder for signal <coeff3$addsub0000> created at line 144.
Found 24-bit adder for signal <coeff3$addsub0001> created at line 144.
Found 24-bit adder for signal <coeff3$addsub0002> created at line 144.
Found 24-bit adder for signal <coeff3$addsub0003> created at line 144.
Found 24-bit adder for signal <coeff3$addsub0004> created at line 144.
Found 24-bit adder for signal <coeff3$addsub0005> created at line 144.
Found 24-bit adder for signal <coeff3$addsub0006> created at line 144.
Found 24-bit adder for signal <coeff4$add0000> created at line 174.
Found 24-bit adder for signal <coeff4$addsub0000> created at line 174.
Found 24-bit adder for signal <coeff4$addsub0001> created at line 174.
Found 24-bit adder for signal <coeff4$addsub0002> created at line 174.
Found 24-bit adder for signal <coeff4$addsub0003> created at line 174.
Found 24-bit adder for signal <coeff5$add0000> created at line 202.
Found 24-bit adder for signal <coeff5$addsub0000> created at line 202.
Found 24-bit adder for signal <coeff5$addsub0001> created at line 202.
Found 24-bit adder for signal <coeff5$addsub0002> created at line 202.
Found 24-bit adder for signal <coeff5$addsub0003> created at line 202.
Found 24-bit adder for signal <coeff6$add0000> created at line 230.
Found 24-bit adder for signal <coeff6$addsub0000> created at line 230.
Found 24-bit adder for signal <coeff6$addsub0001> created at line 230.
Found 24-bit adder for signal <coeff6$addsub0002> created at line 230.
Found 24-bit adder for signal <coeff6$addsub0003> created at line 230.
Found 24-bit adder for signal <coeff7$add0000> created at line 264.
Found 24-bit adder for signal <coeff7$addsub0000> created at line 264.
Found 24-bit adder for signal <coeff7$addsub0001> created at line 264.
Found 24-bit adder for signal <coeff7$addsub0002> created at line 264.
Found 24-bit adder for signal <coeff7$addsub0003> created at line 264.
Found 24-bit adder for signal <coeff7$addsub0004> created at line 264.
Found 24-bit adder for signal <coeff7$addsub0005> created at line 264.
```

```
Found 24-bit adder for signal <coeff7$addsub0006> created at line 264.
Found 24-bit adder for signal <coeff8$add0000> created at line 300.
Found 24-bit adder for signal <coeff8$addsub0000> created at line 300.
Found 24-bit adder for signal <coeff8$addsub0001> created at line 300.
Found 24-bit adder for signal <coeff8$addsub0002> created at line 300.
Found 24-bit adder for signal <coeff8$addsub0003> created at line 300.
Found 24-bit adder for signal <coeff8$addsub0004> created at line 300.
Found 24-bit adder for signal <coeff8$addsub0005> created at line 300.
Found 24-bit adder for signal <coeff8$addsub0006> created at line 300.
Found 24-bit adder for signal <coeff8$addsub0007> created at line 300.
Found 24-bit adder for signal <coeff9$add0000> created at line 321.
Found 24-bit adder for signal <coeff9$addsub0000> created at line 321.
Found 24-bit adder for signal <y$addsub0000> created at line 86.
Found 24-bit adder for signal <y$addsub0001> created at line 86.
Found 24-bit adder for signal <y$addsub0002> created at line 86.
Found 24-bit adder for signal <y$addsub0003> created at line 86.
Found 24-bit adder for signal <y$addsub0004> created at line 86.
Found 24-bit adder for signal <y$addsub0005> created at line 86.
Found 24-bit adder for signal <y$addsub0006> created at line 86.
Found 24-bit adder for signal <y$addsub0007> created at line 86.
Found 24-bit adder for signal <y$addsub0008> created at line 86.
Summary:
inferred 88 D-type flip-flop(s).
inferred 65 Adder/Subtractor(s).
Unit <my_fir> synthesized.
=========================================================================
HDL Synthesis Report
Macro Statistics
# Adders/Subtractors : 65
24-bit adder : 65
# Registers : 88
1-bit register : 88
=========================================================================
=========================================================================
* Advanced HDL Synthesis *
=========================================================================
Loading device for application Rf_Device from file '3s500e.nph'
in environment C:\Xilinx\10.1\ISE.
=========================================================================
Advanced HDL Synthesis Report
Macro Statistics
# Adders/Subtractors : 65
24-bit adder : 65
# Registers : 88
Flip-Flops : 88
=========================================================================
=========================================================================
* Low Level Synthesis *
=========================================================================
Optimizing unit <my_fir> ...
Mapping all equations...
Building and optimizing final netlist ...
Found area constraint ratio of 100 (+ 5) on block my_fir, actual
ratio is 11.
FlipFlop reg<1>_7 has been replicated 1 time(s)
FlipFlop reg<2>_1 has been replicated 1 time(s)
FlipFlop reg<2>_2 has been replicated 1 time(s)
FlipFlop reg<2>_3 has been replicated 1 time(s)
FlipFlop reg<2>_4 has been replicated 1 time(s)
FlipFlop reg<2>_5 has been replicated 1 time(s)
FlipFlop reg<2>_6 has been replicated 1 time(s)
FlipFlop reg<2>_7 has been replicated 1 time(s)
FlipFlop reg<3>_1 has been replicated 1 time(s)
FlipFlop reg<3>_2 has been replicated 1 time(s)
FlipFlop reg<3>_4 has been replicated 1 time(s)
FlipFlop reg<3>_5 has been replicated 1 time(s)
FlipFlop reg<3>_6 has been replicated 1 time(s)
FlipFlop reg<3>_7 has been replicated 1 time(s)
FlipFlop reg<4>_7 has been replicated 1 time(s)
FlipFlop reg<5>_7 has been replicated 1 time(s)
FlipFlop reg<6>_7 has been replicated 1 time(s)
FlipFlop reg<7>_1 has been replicated 1 time(s)
FlipFlop reg<7>_2 has been replicated 1 time(s)
FlipFlop reg<7>_3 has been replicated 1 time(s)
FlipFlop reg<7>_4 has been replicated 1 time(s)
FlipFlop reg<7>_5 has been replicated 1 time(s)
FlipFlop reg<7>_6 has been replicated 1 time(s)
FlipFlop reg<7>_7 has been replicated 1 time(s)
FlipFlop reg<8>_1 has been replicated 1 time(s)
FlipFlop reg<8>_2 has been replicated 1 time(s)
FlipFlop reg<8>_3 has been replicated 1 time(s)
FlipFlop reg<8>_4 has been replicated 1 time(s)
FlipFlop reg<8>_5 has been replicated 1 time(s)
FlipFlop reg<8>_6 has been replicated 1 time(s)
```

```
FlipFlop reg<8>_7 has been replicated 1 time(s)
FlipFlop reg<9>_7 has been replicated 1 time(s)
Final Macro Processing ...
Processing Unit <my_fir> :
Found 4-bit shift register for signal <reg<7>_0>.
Unit <my_fir> processed.
=========================================================================
Final Register Report
Macro Statistics
# Registers : 116
Flip-Flops : 116
# Shift Registers : 1
4-bit shift register : 1
=========================================================================
=========================================================================
* Partition Report *
=========================================================================
Partition Implementation Status
-------------------------------
No Partitions were found in this design.
-------------------------------
=========================================================================
* Final Report *
=========================================================================
Final Results
RTL Top Level Output File Name : my_fir.ngr
Top Level Output File Name : my_fir
Output Format : NGC
Optimization Goal : Speed
Keep Hierarchy : NO
Design Statistics
# IOs : 34
Cell Usage :
# BELS : 1564
# GND : 1
# LUT1 : 11
# LUT2 : 104
# LUT3 : 255
# LUT4 : 357
# MULT_AND : 68
# MUXCY : 397
# MUXF5 : 22
# VCC : 1
# XORCY : 348
# FlipFlops/Latches : 121
# FD : 1
# FDC : 120
# Shift Registers : 1
# SRL16 : 1
# Clock Buffers : 1
# BUFGP : 1
# IO Buffers : 33
# IBUF : 9
# OBUF : 24
=========================================================================
Device utilization summary:
---------------------------
Selected Device : 3s500efg320-4
Number of Slices: 418 out of 4656 8%
Number of Slice Flip Flops: 121 out of 9312 1%
Number of 4 input LUTs: 728 out of 9312 7%
Number used as logic: 727
Number used as Shift registers: 1
Number of IOs: 34
Number of bonded IOBs: 34 out of 232 14%
Number of GCLKs: 1 out of 24 4%
---------------------------
Partition Resource Summary:
---------------------------
No Partitions were found in this design.
---------------------------
=========================================================================
TIMING REPORT
NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
GENERATED AFTER PLACE-and-ROUTE.
Clock Information:
------------------
+-----------------------------------+------------------------+-------+
| Clock Signal | Clock buffer(FF name) | Load |
+-----------------------------------+------------------------+-------+
| clk | BUFGP | 122 |
+-----------------------------------+------------------------+-------+
Asynchronous Control Signals Information:
```

```
-----------------------------------
-----------------------------------+-----------------------+-------+
Control Signal | Buffer(FF name) | Load |
-----------------------------------+-----------------------+-------+
rst | IBUF | 120 |
-----------------------------------+-----------------------+-------+
Timing Summary:
---------------
Speed Grade: -4
Minimum period: 4.014ns (Maximum Frequency: 249.128MHz)
Minimum input arrival time before clock: 1.946ns
Maximum output required time after clock: 49.355ns
Maximum combinational path delay: No path found
Timing Detail:
--------------
All values displayed in nanoseconds (ns)
=========================================================================
Timing constraint: Default period analysis for Clock 'clk'
Clock period: 4.014ns (frequency: 249.128MHz)
Total number of paths / destination ports: 114 / 113
-------------------------------------------------------------------------
Delay: 4.014ns (Levels of Logic = 0)
Source: Mshreg_reg_7 (FF)
Destination: reg_7 (FF)
Source Clock: clk rising
Destination Clock: clk rising
Data Path: Mshreg_reg_7 to reg_7
Gate Net
Cell:in->out fanout Delay Delay Logical Name (Net Name)
---------------------------------------- ------------
SRL16:CLK->Q 1 3.706 0.000 Mshreg_reg_7 (Mshreg_reg_7)
FD:D 0.308 reg_7
----------------------------------------
Total 4.014ns (4.014ns logic, 0.000ns route)
(100.0% logic, 0.0% route)
=========================================================================
Timing constraint: Default OFFSET IN BEFORE for Clock 'clk'
Total number of paths / destination ports: 8 / 8
-------------------------------------------------------------------------
Offset: 1.946ns (Levels of Logic = 1)
Source: x<0> (PAD)
Destination: reg<0>_0 (FF)
Destination Clock: clk rising
Data Path: x<0> to reg<0>_0
Gate Net
Cell:in->out fanout Delay Delay Logical Name (Net Name)
---------------------------------------- ------------
IBUF:I->O 1 1.218 0.420 x_0_IBUF (x_0_IBUF)
FDC:D 0.308 reg<0>_0
----------------------------------------
Total 1.946ns (1.526ns logic, 0.420ns route)
(78.4% logic, 21.6% route)
=========================================================================
Timing constraint: Default OFFSET OUT AFTER for Clock 'clk'
Total number of paths / destination ports: 297493869922 / 24
-------------------------------------------------------------------------
Offset: 49.355ns (Levels of Logic = 46)
Source: reg<2>_3 (FF)
Destination: y<23> (PAD)
Source Clock: clk rising
Data Path: reg<2>_3 to y<23>
Gate Net
Cell:in->out fanout Delay Delay Logical Name (Net Name)
---------------------------------------- ------------
FDC:C->Q 27 0.591 1.296 reg<2>_3 (reg<2>_3)
LUT4:I2->O 2 0.704 0.622 Madd_coeff2_addsub0001_Madd_cy<3>11
(Madd_coeff2_addsub0001_Madd_cy<3>)
LUT4:I0->O 2 0.704 0.622 Madd_coeff2_addsub0001_Madd_cy<4>11
(Madd_coeff2_addsub0001_Madd_cy<4>)
LUT4:I0->O 2 0.704 0.622 Madd_coeff2_addsub0001_Madd_cy<5>11
(Madd_coeff2_addsub0001_Madd_cy<5>)
LUT4:I0->O 2 0.704 0.622 Madd_coeff2_addsub0001_Madd_cy<6>11
(Madd_coeff2_addsub0001_Madd_cy<6>)
LUT4:I0->O 2 0.704 0.622 Madd_coeff2_addsub0001_Madd_cy<7>11
(Madd_coeff2_addsub0001_Madd_cy<7>)
LUT4:I0->O 4 0.704 0.622 Madd_coeff2_addsub0001_Madd_cy<8>11
(Madd_coeff2_addsub0001_Madd_cy<8>)
LUT3:I2->O 4 0.704 0.762 Madd_coeff2_addsub0001_Madd_cy<9>11
(Madd_coeff2_addsub0001_Madd_cy<9>)
LUT4:I0->O 6 0.704 0.748 Madd_coeff2_addsub0001_Madd_cy<11>11
(Madd_coeff2_addsub0001_Madd_cy<11>)
LUT4:I1->O 1 0.704 0.424 Madd_coeff2_addsub0003_Madd_lut<12>_SW1
 (N368)
LUT4:I3->O 1 0.704 0.000 Madd_coeff2_addsub0003_Madd_lut<12>
```

```
(Madd_coeff2_addsub0003_Madd_lut<12>)
MUXCY:S->O 1 0.464 0.000 Madd_coeff2_addsub0003_Madd_cy<12>
(Madd_coeff2_addsub0003_Madd_cy<12>)
XORCY:CI->O 2 0.804 0.482 Madd_coeff2_addsub0003_Madd_xor<13>
(coeff2_addsub0003<13>)
LUT3:I2->O 1 0.704 0.595 Madd_coeff2_addsub0005C21
(Madd_coeff2_addsub0005C2)
LUT4:I0->O 1 0.704 0.000 Madd_coeff2_addsub0005_Madd_lut<14>
(Madd_coeff2_addsub0005_Madd_lut<14>)
XORCY:LI->O 2 0.527 0.482 Madd_coeff2_addsub0005_Madd_xor<14>
(coeff2_addsub0005<14>)
LUT3:I2->O 1 0.704 0.595 Madd_coeff2_addsub0007C11
(Madd_coeff2_addsub0007C1)
LUT4:I0->O 1 0.704 0.000 Madd_coeff2_addsub0007_Madd_lut<15>
(Madd_coeff2_addsub0007_Madd_lut<15>)
XORCY:LI->O 2 0.527 0.622 Madd_coeff2_addsub0007_Madd_xor<15>
(coeff2_addsub0007<15>)
LUT3:I0->O 1 0.704 0.499 Madd_y_addsub0001R141
(Madd_y_addsub0001R14)
LUT3:I1->O 1 0.704 0.000 Madd_y_addsub0001_Madd_lut<15>
(Madd_y_addsub0001_Madd_lut<15>)
MUXCY:S->O 1 0.464 0.000 Madd_y_addsub0001_Madd_cy<15>
(Madd_y_addsub0001_Madd_cy<15>)
XORCY:CI->O 1 0.804 0.595 Madd_y_addsub0001_Madd_xor<16>
(y_addsub0001<16>)
LUT2:I0->O 1 0.704 0.000 Madd_y_addsub0002_lut<16>
(Madd_y_addsub0002_lut<16>)
MUXCY:S->O 1 0.464 0.000 Madd_y_addsub0002_cy<16>
(Madd_y_addsub0002_cy<16>)
XORCY:CI->O 2 0.804 0.526 Madd_y_addsub0002_xor<17>
(Madd_y_addsub0003C16_mand)
LUT4:I1->O 1 0.704 0.000 Madd_y_addsub0003_Madd_lut<17>
(Madd_y_addsub0003_Madd_lut<17>)
MUXCY:S->O 1 0.464 0.000 Madd_y_addsub0003_Madd_cy<17>
(Madd_y_addsub0003_Madd_cy<17>)
XORCY:CI->O 2 0.804 0.526 Madd_y_addsub0003_Madd_xor<18>
(Madd_y_addsub0004C17_mand)
LUT4:I1->O 1 0.704 0.000 Madd_y_addsub0004_Madd_lut<18>
(Madd_y_addsub0004_Madd_lut<18>)
MUXCY:S->O 1 0.464 0.000 Madd_y_addsub0004_Madd_cy<18>
(Madd_y_addsub0004_Madd_cy<18>)
XORCY:CI->O 2 0.804 0.526 Madd_y_addsub0004_Madd_xor<19>
(Madd_y_addsub0005C18_mand)
LUT4:I1->O 1 0.704 0.000 Madd_y_addsub0005_Madd_lut<19>
(Madd_y_addsub0005_Madd_lut<19>)
MUXCY:S->O 1 0.464 0.000 Madd_y_addsub0005_Madd_cy<19>
(Madd_y_addsub0005_Madd_cy<19>)
XORCY:CI->O 1 0.804 0.595 Madd_y_addsub0005_Madd_xor<20>
(y_addsub0005<20>)
LUT2:I0->O 1 0.704 0.000 Madd_y_addsub0006_lut<20>
(Madd_y_addsub0006_lut<20>)
MUXCY:S->O 1 0.464 0.000 Madd_y_addsub0006_cy<20>
(Madd_y_addsub0006_cy<20>)
XORCY:CI->O 2 0.804 0.526 Madd_y_addsub0006_xor<21>
(y_addsub0006<21>)
LUT3:I1->O 1 0.704 0.595 Madd_y_addsub0007C201
(Madd_y_addsub0007C20)
LUT4:I0->O 1 0.704 0.000 Madd_y_addsub0007_Madd_lut<22>
(Madd_y_addsub0007_Madd_lut<22>)
MUXCY:S->O 0 0.464 0.000 Madd_y_addsub0007_Madd_cy<22>
(Madd_y_addsub0007_Madd_cy<22>)
XORCY:CI->O 1 0.804 0.595 Madd_y_addsub0007_Madd_xor<23>
(y_addsub0007<23>)
LUT2:I0->O 0 0.704 0.000 Madd_y_addsub0008_lut<23>
(Madd_y_addsub0008_lut<23>)
XORCY:LI->O 1 0.527 0.499 Madd_y_addsub0008_xor<23>
(y_addsub0008<23>)
LUT3:I1->O 0 0.704 0.000 Madd_y_Madd_lut<23>
(Madd_y_Madd_lut<23>)
XORCY:LI->O 1 0.527 0.420 Madd_y_Madd_xor<23> (y_23_OBUF)
OBUF:I->O 3.272 y_23_OBUF (y<23>)
----------------------------------------
Total 49.355ns (33.715ns logic, 15.640ns route)
(68.3% logic, 31.7% route)
========================================

Total REAL time to Xst completion: 27.00 secs
Total CPU time to Xst completion: 27.44 secs
-->
Total memory usage is 178540 kilobytes
Number of errors : 0 ( 0 filtered)
Number of warnings : 0 ( 0 filtered)
Number of infos : 0 ( 0 filtered)
```