ABSTRACT

COMPARING BINARY, CSD AND CSD4 APPROACHES ON THE ASPECT OF POWER CONSUMPTION IN FIR FILTERS USING MULTIPLIERLESS ARCHITECTURES ON FPGAS

The aim of this thesis is to compare several different algorithms of FIR-filter design on the aspect of the amount of power they consume. Three different approaches are presented: One based on binary, two's complement representation of the coefficients in the filter. The second approach is based on CSD representation, and the third approach is based on CSD4 representation of the coefficients. The three approaches are compared due to their overall power consumption when implemented on an FPGA.

In theory, representing coefficients in CSD number representation, yields a reduction of non-zero bits in the implementation by 33% compared to binary representation for long wordlengths. Representing them in CSD4 yields a further reduction of 36% over CSD representation. These are the theoretical numbers. This thesis presents a practical example, simulated in distributed arithmetic on Xilinx's FPGAs. 12 different filters have been simulated with number of taps between 4 and 200.

An automatic design generation tool has been developed in C to ease the process of VHDL-code generation. The automation tool generates two basic architectures, each consisting of three designs. The designs are one design based on binary numbers, one design based on CSD and the last design based on CSD4 number representation.

The simulations have been done on Xilinx - Project Navigator 7.1.02i, on device family Spartan II for the smaller filters and on Spartan 3 for the larger filters. The power analysis is done using Xilinx - XPower.

The results from this thesis are not what the theory states: For filters with number of taps between 4 and 32, simulated on Spartan II, the results show an increased difference between the binary approach and the CSD4 approach power consumption, in favour of the binary one. On average for these designs, binary consumes 24,5% less power than CSD4.

The filters with larger number of taps (62-200) simulated on Spartan 3, the results show a power consumption equal for all the three different approaches in a filter. In other words, the percentage difference between binary, CSD and CSD4 numbers are almost zero.

In this thesis it has not been shown that the binary approach in any case consumes less power than the CSD4 approach. This is, however, only a novel start on the big research field exploiting the possibilities of CSD4 number representation. The future will show whether the CSD4 number representation will turn out to be beneficial or not and if the use of it in FIR-filters will exceed the efficiency of RAG-n and other currently optimal algorithms.

ACKNOWLEDGEMENTS

This is my master thesis, in other words the last thesis delivered before receiving the title of Master of Technology in Computer Science. The master studies have been studied at NTNU in Trondheim, Norway, with an exchange period of one and a half year at Boğaziçi University in Istanbul, Turkey. This thesis were written at Boğaziçi University under supervising of Assistant Professor Arda Yurdakul. I would like to thank her for her advices, help and ideas contributing to the thesis. I would also like to thank PhD student Mustafa Akin at the department of Electrical Engineering at Boğaziçi University for all help and advice about EVA and ModelSim.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

For many decades the aim of developing new hardware was to make devices smaller and faster. The amount of power they consumed was hardly an issue, since the power wasn't the bottle neck. Today we face a different situation, as the demand for real time application with a long battery lifetime is increasing.

Today, design of low power signal processing circuits is an important part of the electronics industry. This is because portable computing and communication devices like cellular phones and laptop computers have become very popular. The development seems to take a turn towards wireless communication and flexibility in the sense that a stationary port is unnecessary. This challenges the industry to produce low power devices and it challenges researchers to find new, less power consuming algorithms for the processes carried out by portable devices. An important process is the digital signal processing (DSP). DSP is the transference of data between devices, such as cellular phone communication, wireless Internet usage and digital TV transmission to mention a few. Finite impulse response (FIR) filtering is a central task in DSP. By reducing the power needed to process data through such a filter, power can be significantly reduced.

The aim of this work is to compare different approaches of number representation implemented on a multiplierless FIR-filter structure on an FPGA. The three approaches to compare are three different implementations of the same filter. The first approach represents the tap values in binary number representation, the second approach in CSD (canonic signed digit) number representation and the third approach in CSD4 (radix-4 CSD) number representation. A design automation tool for designing such filters has been developed.

The thesis has this structure: In chapter 2 some important definitions are stated. Chapter 3 is the background chapter and contains a review of different approaches presented in literature and the results found. The implemented simulations are described in chapter 4. Chapter 5 contains a description of the simulations, before the results

and discussion is presented in chapter 6. In the end, conclusion and further works are proposed (chapter 7).

2. DEFINITIONS

Before describing the background work and theory behind the subject, some issues are necessary to explain and define. The issues explained are: The sources of power consumption (section 2.1), the FIR filter (section 2.2), the FPGA architecture (section 2.3) and the binary, signed digits, CSD and CSD4 number representations (section 2.4).

2.1. The sources of power consumption

The power consumed in devices can be roughly divided into to categories: Static power and dynamic power. Static power consumption is mainly leakage power dissipation [1]. To reduce leakage, the industry has taken steps to improve their technology [2], [3]. Dynamic power is another implementation specific issue, and is still the main source of power consumption. The main consumption of dynamic power is due to charging and discharging of capacitances, in other words switching [1]. Routing is another highly power consuming task in a design. A large and badly routed design can consume more power on routing than on switching.

The power equation for a CMOS gate with capacitance load C is given by equation 2.1:

$$P = n * C * V^2, \tag{2.1}$$

where n is the switching frequency and V is the operation voltage [4]. If the switching frequency can be reduced, power can be saved. Maybe more important is that if the switching frequency is reduced, the operation voltage can be reduced too for many designs. Together this can significantly reduce the power.

In the approach presented here an attempt is made to reduce the power consumed

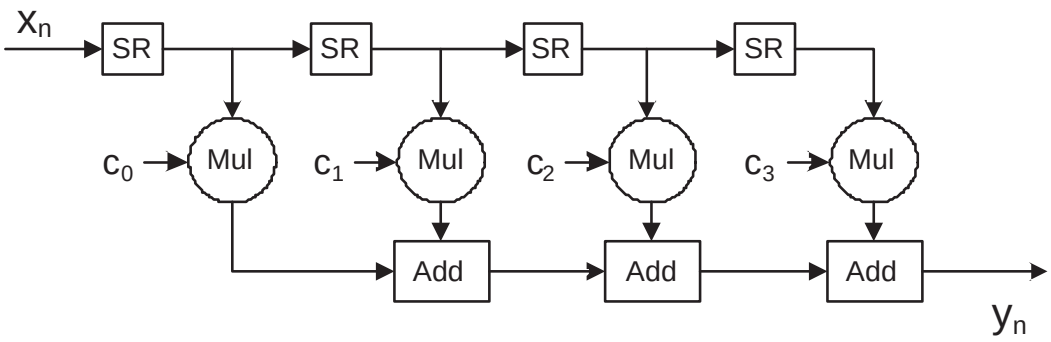

Figure 2.1. Traditional FIR filter

by switching as lower number of non-zero digits in LUTs (Look-Up Tables) results in lower switching frequency. This will be discussed further in the following chapters.

2.2. FIR filter

Processing of data quick enough to achieve real-time performance is a highly power-consuming task. Data is transferred through a network and filtered through filters containing multiplications and/or additions. The power consumed in such a filter is directly related to how many times multiplications or additions must be done. In this thesis the focus is on one-dimensional Finite Impulse Response (FIR) filters.

The FIR-filter's equation is given in equation 2.2.

$$y_n = \sum_{i=0}^{N-1} c_i * x_{n-i}, \tag{2.2}$$

where N is the number of taps, c_i is the coefficient of tap i, x_i is the input data samples. y is the output of the filter.

2.2.1. Direct form FIR filter

A direct form FIR filter design is shown in figure 2.1.

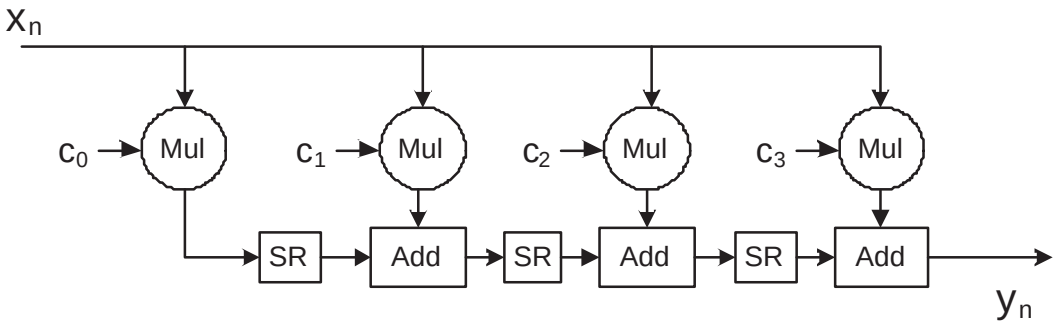

Figure 2.2. Transposed FIR filter

A set of inputs are shifted through n number of registers (SR), also called "taps". They are then multiplied by a number of constant coefficients (c_x) and added up. Basically a FIR filter is a data stream multiplied by a set of constants.

2.2.2. Transposed form FIR filter

The transposed form FIR filter is shown in figure 2.2.

The transposed form of the FIR filter produces the same output as the direct form. The difference is that it performs all the multiplications of a variable at the same time. The shift registers are moved to delay *the output* from a multiplication instead of the input. This way different algorithms can be applied to the FIR filter, as will be discussed in the background chapter (chapter 3).

2.2.3. Multiplierless FIR filter

It is widely accepted that the multiplication is the main source of complexity in a FIR filter. In other words the multiplication consumes most area and is the main source of delaying the throughput, in addition to its high consume of power. An array multiplier of length L consumes an area $O(L^2)$, whereas an adder spreads over only $O(L)$ area [5].

The multiplication by a constant can be represented behaviourally as in figure 2.3.

```
x3 x2 x1 x0    x    1 0 0 1    =              x3 x2 x1 x0
                                  +   x3 x2 x1 x0          .
                                     y6 y5 y4 y3 y2 y1 y0
```

Figure 2.3. Multiplication using reduced additions

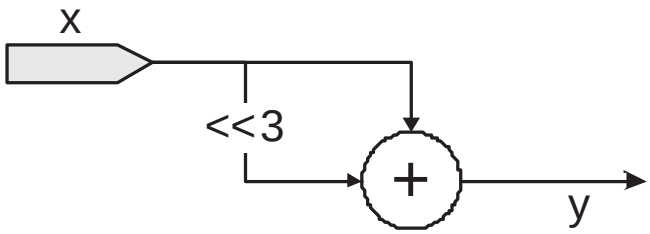

Figure 2.4. Multiplication with coefficient 1001

Here the coefficient value is c=1001. Figure 2.4 shows how the multiplication can be implemented in hardware.

FIR filters using this structure is called multiplierless FIR filter. The algorithm is discussed in chapter 4.

Since additions are the main source of power consumption in a multiplication block, it can be easily understood from the example in figure 2.3 that reducing the number of 1s, i.e. non-zero bits from the coefficients can save power. In section 2.4 about binary, signed digits, CSD and CSD4 number representations this aspect will be discussed further.

2.3. The FPGA architecture

The two Field-Programmable Gate Arrays (FPGAs) used in this thesis are both from Xilinx. They are Spartan II and Spartan 3. The only differences between these are that Spartan 3 includes a built-in multiplier and it has a higher number of configurable logic blocks (CLBs). Spartan 3 is used for larger designs due to its increased number of CLBs, the multiplier is not used in the designs presented here.

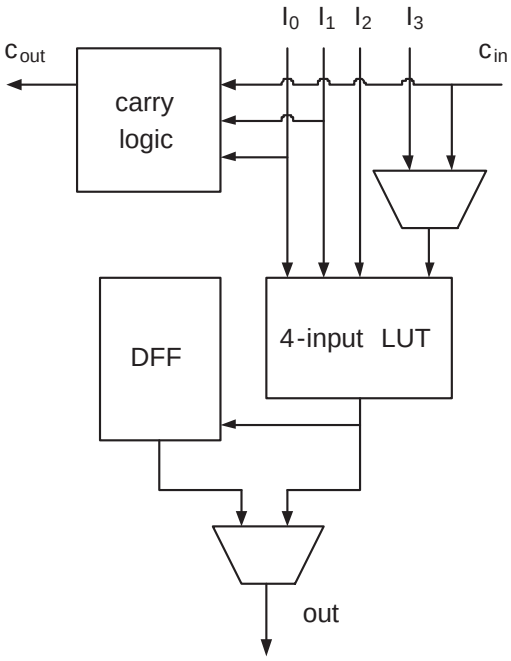

Figure 2.5. One logic cell

The Xilinx FPGAs use look-up tables (LUTs) to construct logic. The LUTs are placed in logic cells (LCs). A LC also contains carry logic and a storage element, as shown in figure 2.5. In each CLB (configurable logic block) there are four LCs. The CLB is in its turn placed in SRAM (static random access memory) [6].

The LCs can be configured to some configuration modes. Those are:

1 LC:

- 4-input LUT
- 4-input function generator
- 16 x 1-bit synchronous RAM

2 LCs:

- 16 x 2-bit synchronous RAM
- 32 x 1-bit synchronous RAM
- 16 x 1-bit dual-port synchronous RAM
- 16-bit shift-register

In this project the 4-input LUT and 16-bit shift-register modes are the most interesting ones, as they are used in Distributed Arithmetic (discussed in section 3.5), which is the architecture the FIR-filters are implemented in here.

2.4. Binary, Signed Digits, CSD and CSD4 number representations

Binary number representation consists of the alphabet $A = \{0, 1\}$. Signed digit (SD) representation consists of the alphabet $A = \{-1, 0, 1\}$. A SD number is shown to be canonic when there is no consecutive non-zero bits [7].

Canonic signed digits (CSD) have two properties:

1. Minimal representation
2. Unique representation

Using CSD representation, the number of non-zero bits in a coefficient can be reduced by 33%, compared to binary representation [8]. A typical example is the representation of the number 7. Binary: $7 = 0111$, i.e. $(2^2 + 2^1 + 2^0 = 7)$, CSD: $7 = 100\text{-}1$, i.e. $(2^3 - 2^0 = 7)$. Here the number of non-zero bits is reduced by $\frac{(3-2)}{3} = 33\%$, without changing the word length. This has been shown to be the average reduction.

Coleman presents in [9] an algorithm for producing any type of Radix-CSD alphabets. He specifically proposes the use of Radix4-CSD representation of numbers for the coefficients in FIR-filter designs. From here on the term "CSD4" will be used for

Radix4-CSD representation. CSD4 numbers have as their base 4 instead of 2 as in CSD number representation (from here on called "CSD2"). The weights are as follows: ... $x_1 * 4^1$ $x_0 * 4^0$. $x_{-1} * 4^{-1}$ $x_{-2} * 4^{-2}$ ... , where "." is the decimal separator. The xs are numbers from the CSD4 alphabet: A = {-15, -13, -11, -9, -7, -5, 0, 5, 7, 9, 11, 13, 15}.

CSD4 number representation has the same property as CSD2 representation, i.e. it is minimal and unique. Using the example above, 7 is represented using CSD2 as: 100-1, and by CSD4 simply: 0007. The reduction of non-zero bits is $\frac{(2-1)}{2} = 50\%$ in this example. Coleman showed the average reduction to be 36% relative to CSD2 representation.

CSD4 number representation requires fewer number of digits than CSD2 for the same precision of a coefficient. This has been exploited in this thesis. It is discussed further in the implementation chapter, in section 4.1.5

The theoretical reductions discussed above are only applicable in theory. In the practical examples, both the CSD2 and CSD4 implemented approaches requires quite a lot of overhead. This is further discussed in the implementation chapter.

3. BACKGROUND

What is in common for FIR-filters and equivalent processes is the multiplication of a data stream by a set of constants. Several lately presented algorithms attempt to reduce the power consumed in these processes.

In this section several algorithms are described. Of these, only the last subsection, 3.5 Distributed Arithmetic algorithms, is directly relevant for this thesis. The rest of the algorithms are included shortly for the reader to have an overview over the field. It would also be interesting to try to combine some of these algorithms with the CSD4 number representation. This is one of the proposed further works.

On a multiplierless structure, the main source of power consumption is the additions. Several algorithms with the aim of reducing the number of adders needed in such a structure have been presented in literature. Among the efficient algorithms are:

- Reducing the number of additions needed by taking advantage of common sub-expressions (CSE).
- Exploiting the reduction of non-zero bits in different number representations.
- Representing the multiplication as a graph-structure.
- Implementing the design in a Distributed Arithmetic manner on a FPGA.

In section 3.1 the Multiple Constant Multiplication (MCM) principle that lead to efficient multiplierless structures is described. Section 3.2 describes the Common Sub-Expression sharing (CSE), before Number Representation algorithms are discussed in section 3.3. Section 3.4 explains shortly the Graph Representation algorithms, and finally, section 3.5 presents the Distributed Arithmetic (DA).

Table 3.1. Filter coefficients

a	1110
b	1011

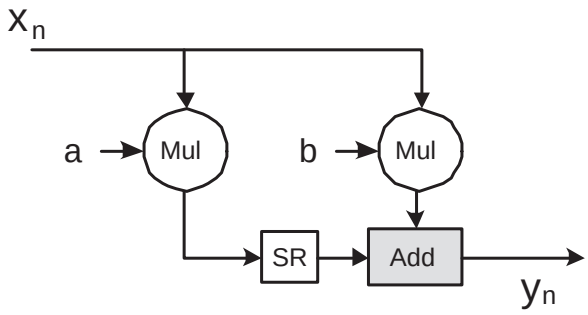

Figure 3.1. Transposed form FIR filter for coefficient a and b

3.1. Multiple Constant Multiplication

Potkonjak, Srivastava and Chandrakasan presented the Multiple Constant Multiplication (MCM) block in [10]. As scarcely discussed in section 2.2.3, the multiplication with a constant can be replaced by a series of shifts and adders. The coefficients in table 3.1 are depict in a transposed form FIR filter in figure 3.1.

The multiplication by coefficient a is shown as a series of adders and shifts in figure 3.2, and the multiplication by coefficient b is shown in figure 3.3.

When comparing coefficient a and b it can easily be observed that two bits are in common for them, that being the 1st and the 3rd bit most significant bit. When constructing a filter where both coefficients are represented, their common digits can be exploited to reduce the number of adders needed. The resulting filter is shown in figure 3.4.

In figure 3.4 the grey adder is the adder in figure 3.1. The number of saved adders is only 1, while the saved shift-registers are 2, compared to a structure were a and b are calculated separately as shown in figure 3.2 and 3.3. For larger filters the reduction can be significant.

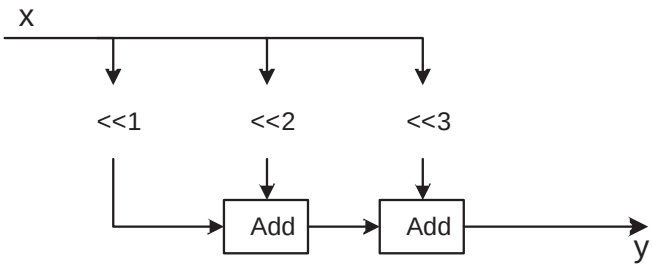

Figure 3.2. Multiplication by coefficient a = 1110 using additions and shifts

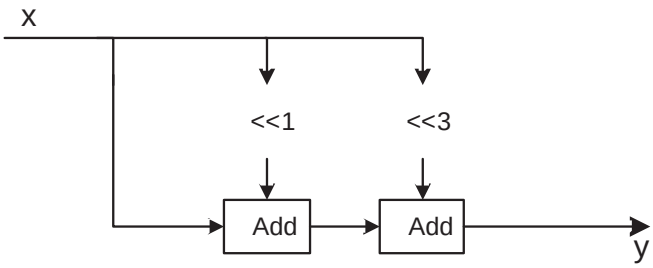

Figure 3.3. Multiplication by coefficient b = 1011 using additions and shifts

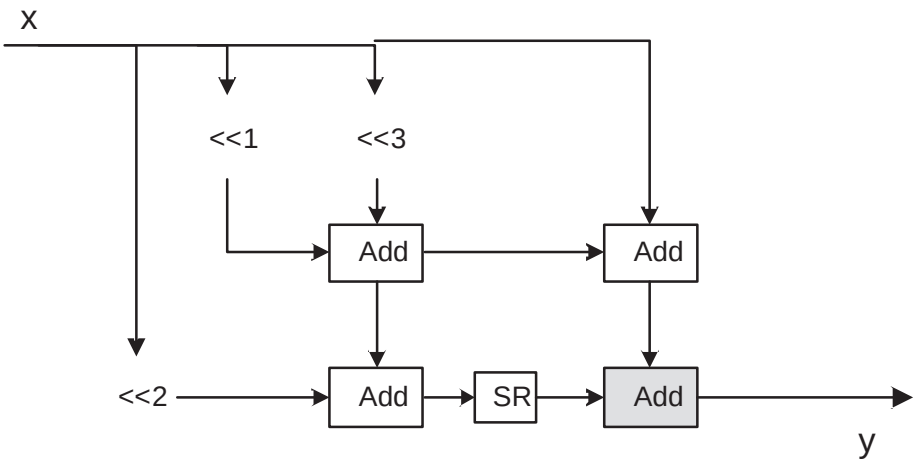

Figure 3.4. Coefficient a and b in a MCM

c_1	1				-1					
c_2		-1		-1		1			1	
c_3					1					-1

Figure 3.5. Table representation of coefficient set S

The MCM structure is the start of the Common Sub-Expression sharing algorithms described next.

3.2. Common Sub-Expression algorithms

Numerous algorithms using Common Sub-Expressions (CSE) have been presented. Among them are [4], [10], [14-20]. The basic example is taken from [14]. The idea is as follows:

The coefficient set $S = \{c_1, c_2, c_3\}$, $c_1 =$ 1000-10000, $c_2 =$ 0-10-1010010, $c_3 =$ 000010000-1, can be represented in a table like figure 3.5.

In figure 3.5 a blank cell represents a 0. The input stream is fed through this filter, starting from c_1, going through c_2 one clock cycle later. Without any algorithm, this filter needs 7 additions, 1 for c_1, 3 for c_2 and 1 for c_3 in addition to 2 adders to add the shifted results as described in section 3.1 about the MCM block.

In the example the common sub-expression $x2 = x - x[-1] >> 1$ and its negative form can be found 4 times as shown in figure 3.6. x is a coefficient's non-zero bit. $x[-1]$ denotes the next coefficient's non-zero bit used in the sub-expression. $x >> 1$ denotes the coefficient shifted once to the right.

The resulting formula becomes: $x2 = x - x[-1] >> 1$, $y = x2 - x2 >> 4 - x2[-1] >> 3 + x2[-1] >> 8$. This requires 4 subtractions and additions, compared to 7 needed in the first case.

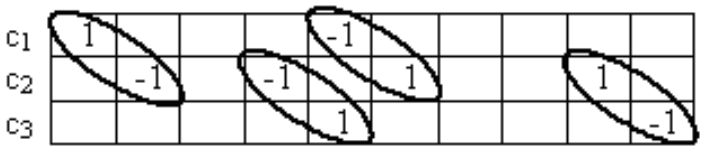

Figure 3.6. Table representation of coefficient set S. CSE highlighted

In other filters finding the common sub-expressions from the found sub-expressions, can further reduce the number of additions and subtractions.

3.3. Number representations

Several number representations have been proposed lately. In addition to the CSD and CSD4 representation discussed earlier, splitting of a number was early suggested, and MSD (Minimal Signed Digit Representation) has shown to be advantageous when applying the common sub-expression algorithms.

Concerning one single coefficient, it has been shown that CSD representation is a SD representation using minimal number of non-zero bits.

The reason why methods like number splitting or MSD was proposed instead of using the optimal CSD, is because a *set* of coefficients is concerned. This means that what is optimal for one number might not be optimal for the set. Two different methods has been proposed, the number splitting, discussed in section 3.3.1 and the Minimal Signed Digit also extended to allow non-minimal signed digits, discussed in section 3.3.2. The last one has shown to be as efficient as the RAG-n graph algorithm [21].

3.3.1. Number splitting

Number splitting was early proposed as a means of improving the common subexpression algorithm. Hartley mentioned in [16] this possibility.

Table 3.2. Number Splitting example

a	10101
b	00101
c	10001
d	-10101

The idea is that in a set of coefficients, if splitting a number leads to more occurrences of a CSE, that would be a benefit. An example is the number 10001, which can be split into 10-100 and 00101. If the coefficient set is the one given in table 3.2, it can be advantageous to split 10001 into 10-100 and 00101 to obtain one more occurrence of the subexpression "101".

Unfortunately the method has shown not to be optimal and a Minimal Signed Digit representation has been presented as a better alternative.

3.3.2. Minimal Signed Digit representation

The MSD (Minimal Signed Digit) approach offers some redundancy in the numbers that CSD does not offer, but still has the minimum number of non-zero bits. An example is the CSD representation of the number 3. 3 is represented in CSD as 10-1, using two non-zero bits. Another minimal representation is 011, also using two non-zero bits, but this representation is not a CSD. Park and Kang presented an algorithm in [22] that exploits the redundancy of MSD to find the best possible *set* of coefficients. By searching the space of MSDs the common sub-expressions can be better utilized as the set of which consists of numbers containing most CSEs can be chosen.

Dempster and Macleod extended Park and Kang's work to find the best possible set of coefficients for Common Sub-Expression Sharing by searching a set of SD number representations, not only the minimal ones in [21]. They show that their algorithm is optimal, i.e. it gives the same result as the graph algorithm RAG-n, but doesn't have the precomputed table problem as RAG-n has. RAG-n is discussed next.

3.4. Graph representation algorithms

Bull and Horrocks presented in [5] an algorithm based on graph representation of the MCM. They observed that the partial products from the multiplications performed in a MCM could be reused in larger number multiplications. Their idea was to decompose each coefficient into a number of primitive operations, i.e. additions, subtractions and shifts, perform these on the smallest coefficients and try to reuse them as much as possible in the larger coefficients. Their idea was refined in [11] by Dempster and Macleod.

In [12] Dempster and Macleod presents the n-dimensional reduces adder graph algorithm, RAG-n. RAG-n has proved to be the best algorithm in most cases [13] and is considered optimal. The problem with it is that it uses a pre-computed table containing all graphs for each coefficient. The search for these graphs is very large and often not feasible for larger wordlength coefficients.

3.5. FIR filters implemented on FPGAs: Distributed Arithmetic algorithms

FPGAs were for a long time not considered fit when implementing FIR filters and other real-time, low power processes. This was because better alternatives could be found in dedicated hardware and ASICs (Application-Specific Integrated Circuit). The low area, low power and high speed these alternatives offered were by far superior to the FPGA. The distributed arithmetic approach, made the FPGA a worthy candidate since this architecture fits perfectly on a FPGA.

In multiplierless architectures the replacement of the multiplier with a series of additions and shifts is on the cost of speed. The Distributed Arithmetic (DA) is a bit-serial computation process that offers speeds approaching the original array multiplier's speed, and at the same time keeping the power consumption down on the same level as the competing dedicated hardware and ASIC's power consumption. [23].

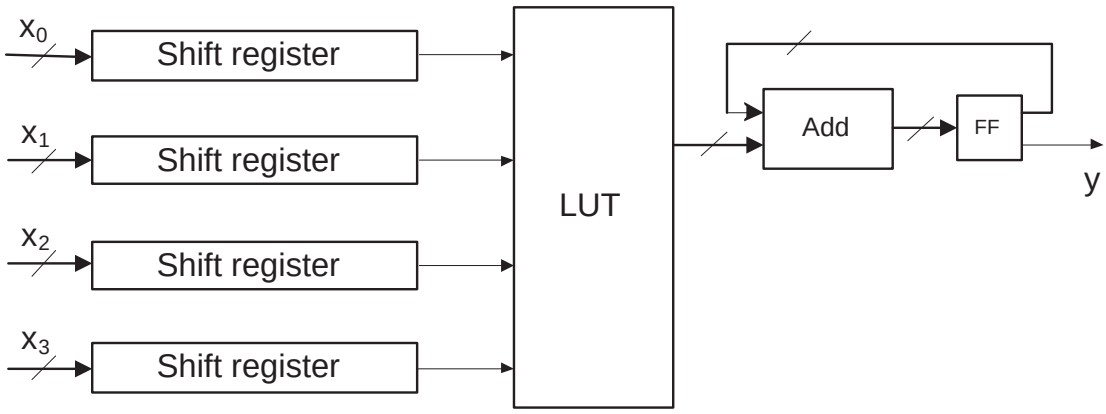

Figure 3.7. Distributed Arithmetic, the principle

Table 3.3. Content of the LUTs

Input $x_0x_1x_2x_3$	Output		Input $x_0x_1x_2x_3$	Output
0 0 0 0	0		1 0 0 0	D
0 0 0 1	A		1 0 0 1	A + D
0 0 1 0	B		1 0 1 0	B + D
0 0 1 1	A + B		1 0 1 1	A + B + D
0 1 0 0	C		1 1 0 0	C + D
0 1 0 1	A + C		1 1 0 1	A + C + D
0 1 1 0	B + C		1 1 1 0	B + C + D
0 1 1 1	A + B + C		1 1 1 1	A + B + C + D

The DA architecture fits very well on an FPGA because the FPGA's Configurable Logic Blocks easily can be used as shift-registers and LUTs as discussed in chapter 2.3. The DA standard architecture is shown in figure 3.7.

The variables are fed into the shift-registers. From there, one by one they are fed into the Look Up Table (LUT). The LUT is the replacement of a multiplication, since it consists of precomputed data. The values are shown in table 3.3. Here A, B, C and D are the coefficients.

A transposed form FIR filter implemented in a Distributed Arithmetic design has a structure as shown in figure 3.8.

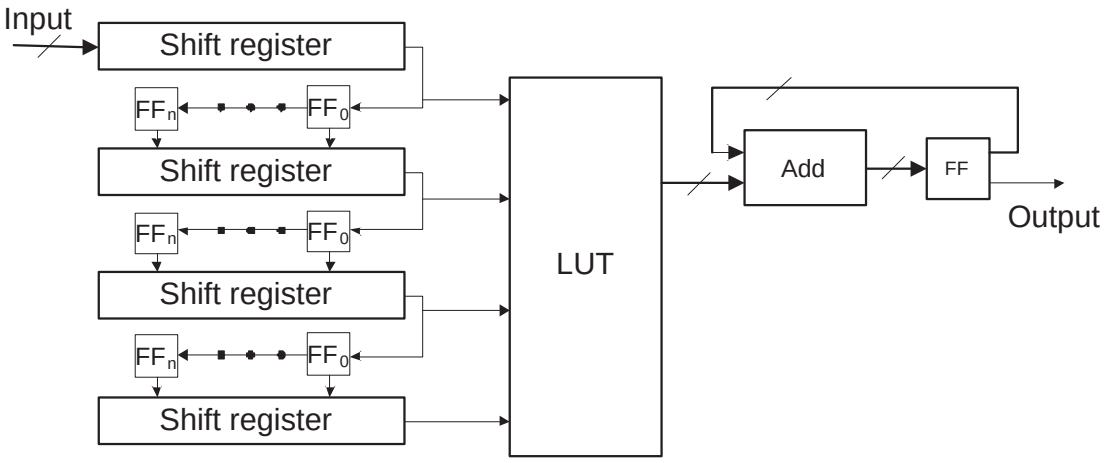

Figure 3.8. Simple FIR filter implemented in DA

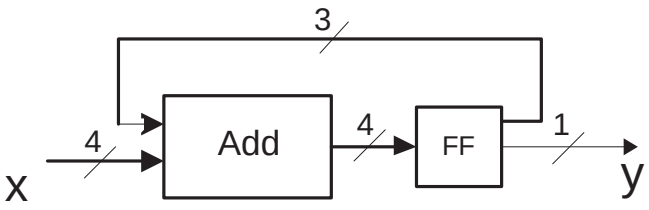

Figure 3.9. Add-and-flip-flop structure

The add and flip-flop structure highlighted in figure 3.9 has the behaviour shown in figure 3.10. x_0 is the first input word from the LUT, x_1 is the second and so on. y is the output.

Initially, in figure 3.9, there is no number stored in the flip-flop. Therefore the output $y0$ is the least significant bit of x_0. The three remaining bits are stored in the flip-flop for one clock-cycle, shifted one to the right and fed to the next addition. The least significant bit of the result is the next output. The answer to the multiplication comes bitwise, the least significant bit first, using 8 clock-cycles in this case. In the general case, the clock-cycles needed to produce the result is given by the equation: $clock\text{-}cycles = length(x) + length(c)$, where $length(x)$ is the wordlength of the input bit stream and $length(c)$ is the wordlength of the coefficient.

In the literature different approaches for optimizing the DA structure have been presented, among them [24-27].

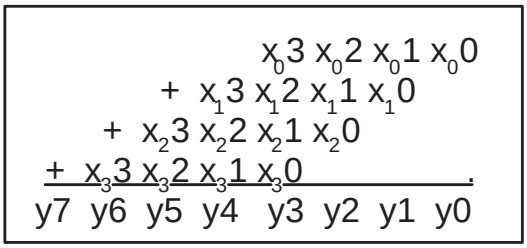

Figure 3.10. Behaviour of the add-and-flip-flop structure

In this thesis the implemented FIR filter is in DA and of a simple form as shown in figure 3.8. The focus is not on an optimised distributed arithmetic, but on the use of different number representation, i.e. binary representation, CSD2 representation and CSD4 representation.

4. IMPLEMENTATION

In this chapter the implementation issues are discussed. The architectures implemented in VHDL are presented in section 4.1, followed by the discussion of the implemented design automation tool in section 4.2. In section 4.3 the tools used are discussed. The VHDL code of an 8 tap FIR filter in all implemented designs can be found in Appendix C. The full source code of the DAT tool can be found in Appendix D.

4.1. Architecture implementation in VHDL

The architectures implemented is of the type Distributed Arithmetic. Two different architectures are implemented, presented in chapter 4.1.4. For each of the architectures 3 designs are made, one binary FIR filter design (section 4.1.1), one CSD2 FIR filter design (section 4.1.2) and one CSD4 FIR filter design (section 4.1.3). Other implementation issues are discussed in section 4.1.5.

4.1.1. Binary FIR filter design

Figure 4.1 shows a 4 tap FIR filter implemented in binary representation.

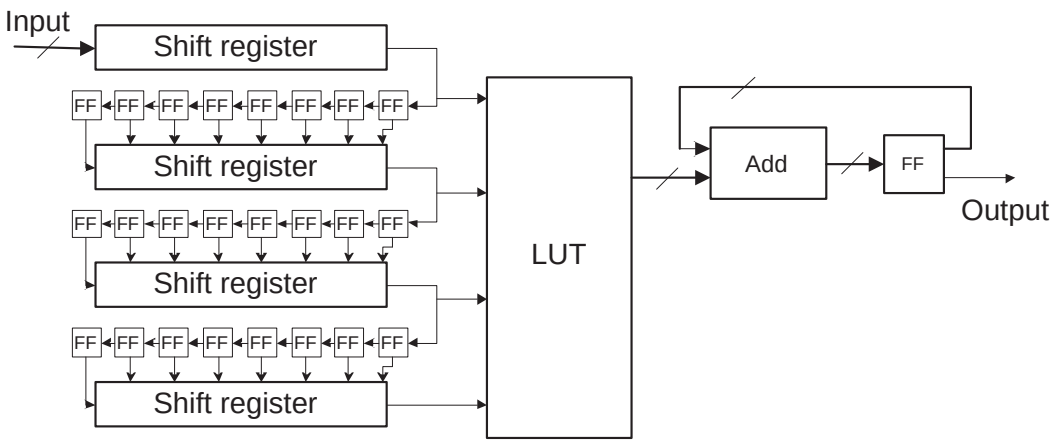

Figure 4.1. 4 tap input binary FIR filter implementation

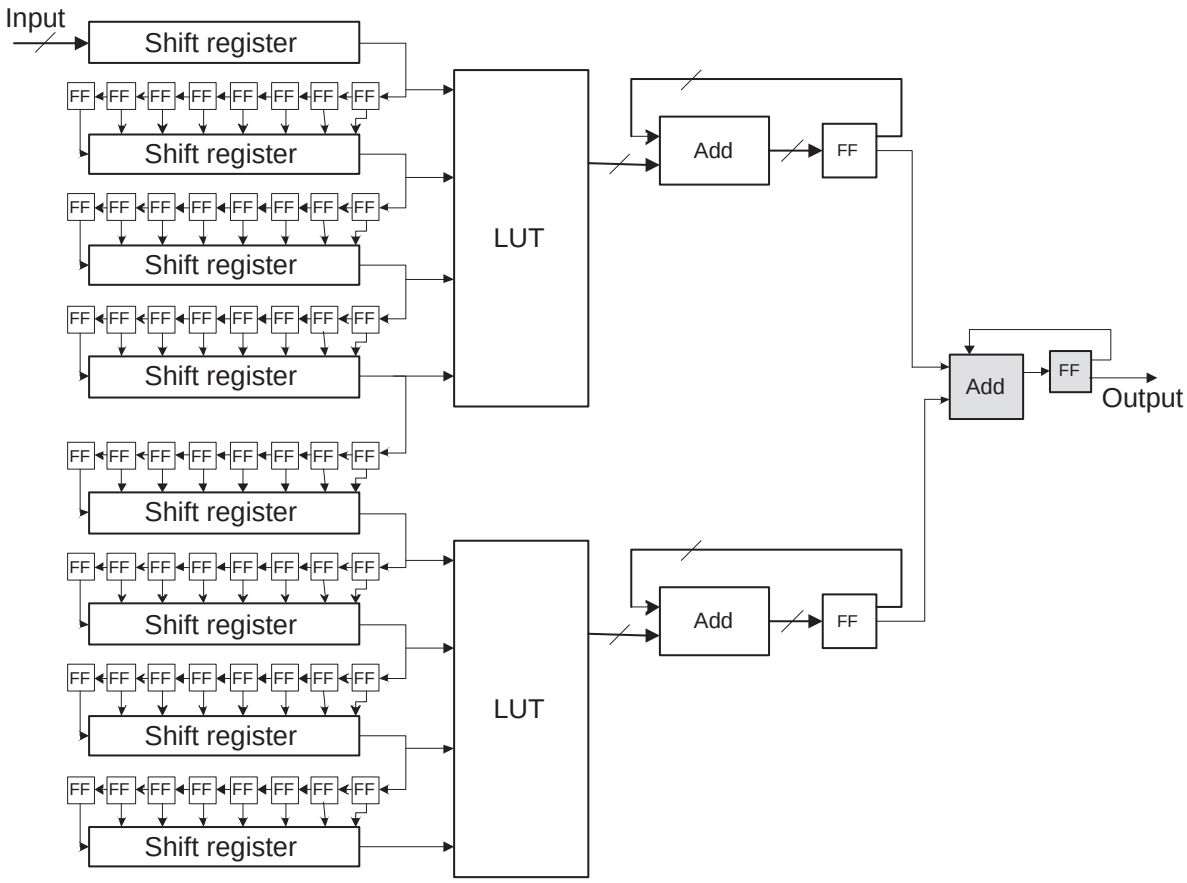

Figure 4.2. 8 tap binary FIR filter implementation

The design grows in size according to the number of taps used. An 8 tap implementation is illustrated in figure 4.2.

4.1.2. CSD2 design

To take advantage of CSD2 numbers, a double set of LUTs must be used (see figure 4.3). One LUT is needed for the set of positive digits and another is needed for the set of negative digits. For instance, to put the coefficient 10-10010-1 into the LUTs, 10000100 is placed in the positive LUT, while 00100001 is placed in the negative LUT. Later the outputs from the two add-and-flip-flop structures are subtracted, i.e. subtracting the negative numbers from the positive numbers.

The CSD2 architecture takes two more clock-cycles before the correct answer is output, compared to the binary architecture. This is due to the subtractor. It produces its

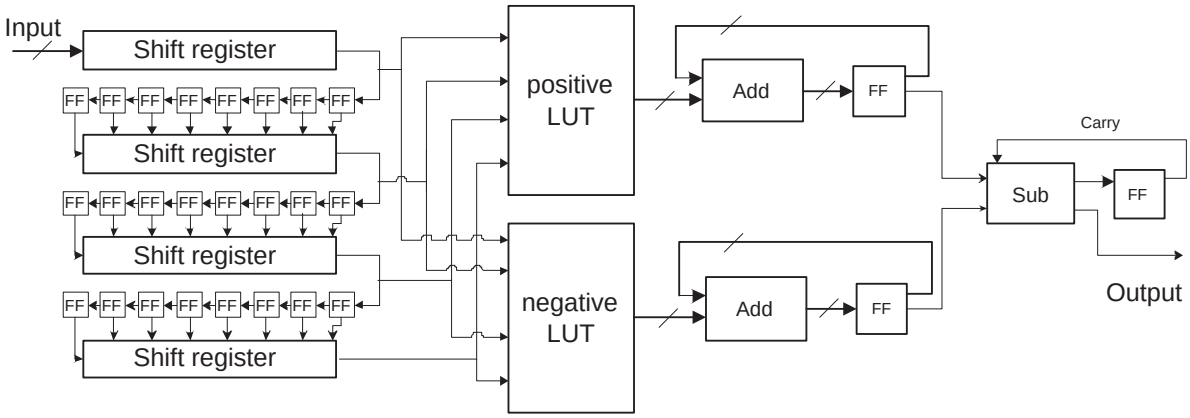

Figure 4.3. The basic unit of CSD2 FIR filter implementation

result in two clock-cycles; therefore two extra clock-cycles are needed for the overall result to be put out.

The design grows according to the number of taps, as illustrated for the binary case in figure 4.2.

4.1.3. CSD4 design

The focus in this thesis lies on weather or not it is advantageous to use CSD4 number representation. The theory, as described in section 2.4 states that using CSD4 representation reduces the number of non-zero bits by 36% relative to CSD2. This is on the cost of scaling that must be done to transform the representation from CSD4 to binary.

In the case of CSD4 numbers, for each set of digits {-X, X}, a double set of LUTs is needed. This set is the same as the set used for CSD2 numbers. According to the CSD4 alphabet, 6 of these sets are needed, resulting in 6 times as many LUTs as is needed for CSD2. The architecture is illustrated in figure 4.4. Here each CSD4_X unit corresponds to one unit of CSD2, illustrated in figure 4.3. Note that the shift-registers are put outside the basic design.

Each CSD4_X component must be multiplied by X in the end to convert the numbers

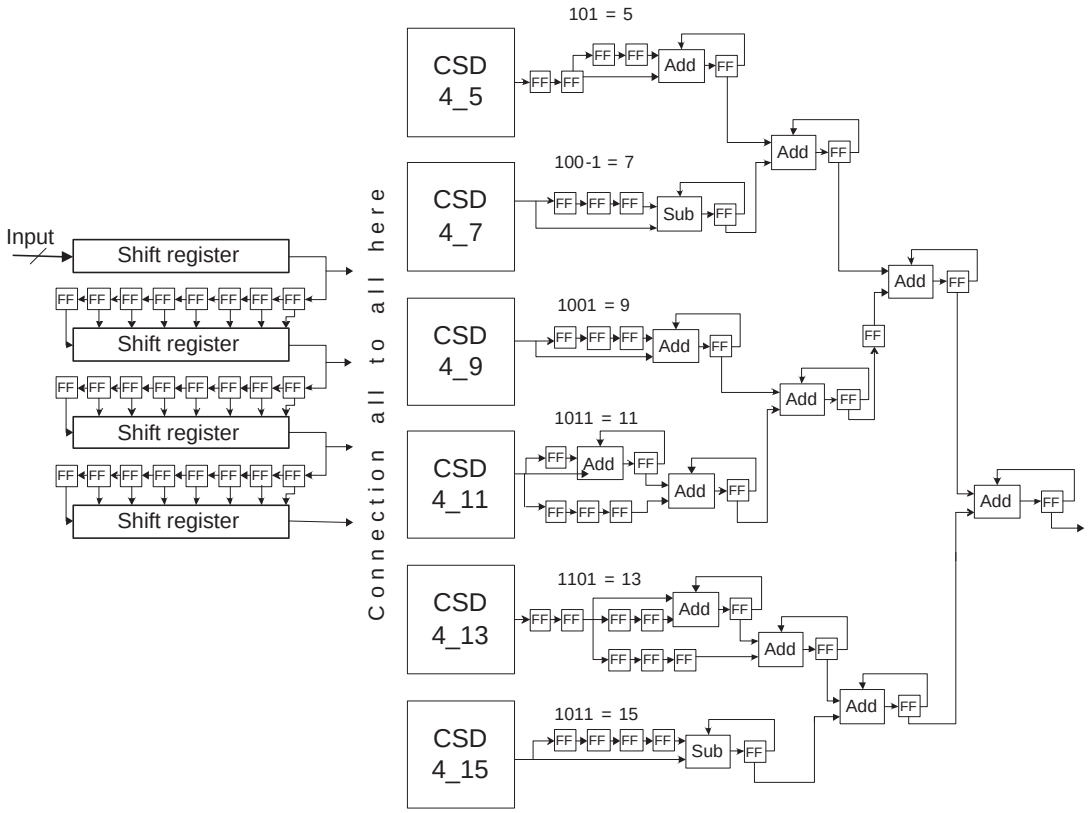

Figure 4.4. The CSD4 FIR filter implementation

from numbers represented in CSD4 format to a binary bit stream. This is not done by multiplication but in a systolic manner. For instance, 7=100-1. Therefore the signal coming from CSD4_7 is split into two signals, where one of them is delayed 3 clock-cycles to achieve 1000, while the other one is fed into the subtraction directly, achieving -1.

After all the outputs from the CSD-units have been scaled, 5 additional adders are required to get the final output of the architecture. Note the use of flip-flops to delay the signal where necessary to avoid glitching[1] .

[1]Glitching occurs when data is switching within a clock-cycle. For instance by putting two consecutive adders, without any flip-flops between will result in glitching in the last adder, since there will be two outputs from the adder in the same clock-cycle. The first output is reasoning from the input before the clock-cycle and the second output is reasoning from the first adder's output in the clock-cycle. I.e. the input to the second adder is not ready at the start of the clock-cycle. This results in glitching and can easily be avoided by the use of flip-flops to store temporary results. Glitching can in some designs be a tremendous cause of power consumption. In [1] it is stated that glitching may contribute to up to 70 % of the power dissipation. However, the use of flip-flops can significantly reduce glitching.

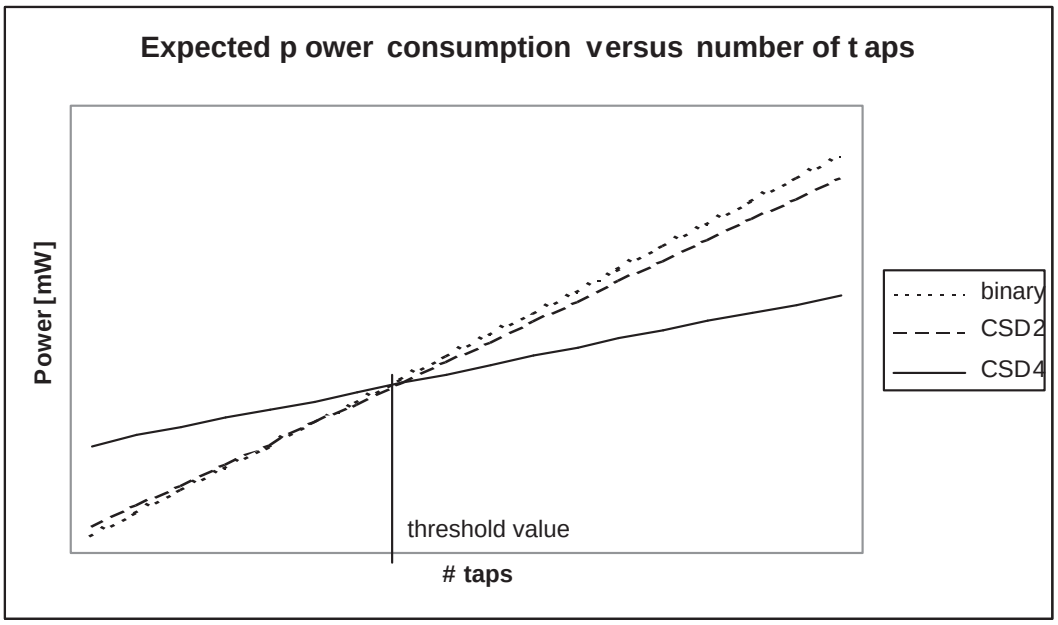

Figure 4.5. Expected power versus taps behavior

The extra adders and flip-flops needed to scale the result are expected to result in a power increase for filters with few taps. But the power consumed here is static for all designs, and therefore the power saved by the number of non-zero bit reduced in CSD4 representation is expected to exceed the extra power needed.

A behaviour like the one pictured in figure 4.5 is expected. A threshold value is expected to be found.

4.1.4. The architectures

Two different architectures are implemented. For filters where number of taps is 4 or less, there is no difference between the designs. For the binary implementation of the filters, there is never any difference between architecture 1 and architecture 2.

For the other designs, the difference between architecture 1 and 2 is the placement of the subtractor, as shown in figure 4.6 and 4.7. The full VHDL code for a 8 tap filter can be found in appendix C.

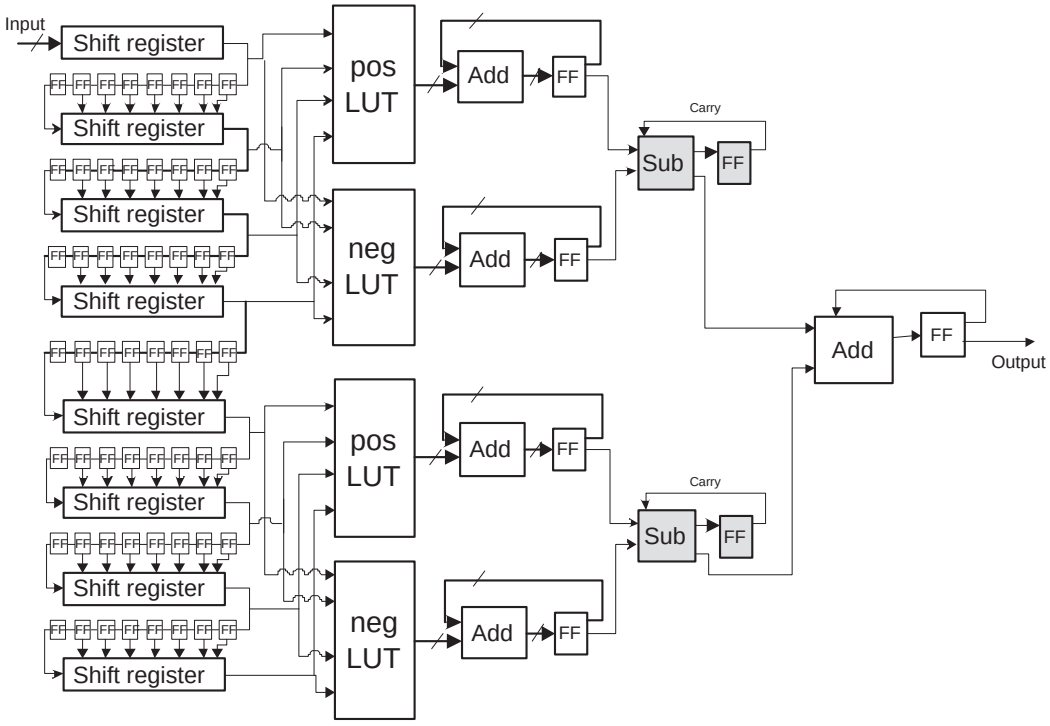

Figure 4.6. An 8 tap CSD2 filter on architecture 1

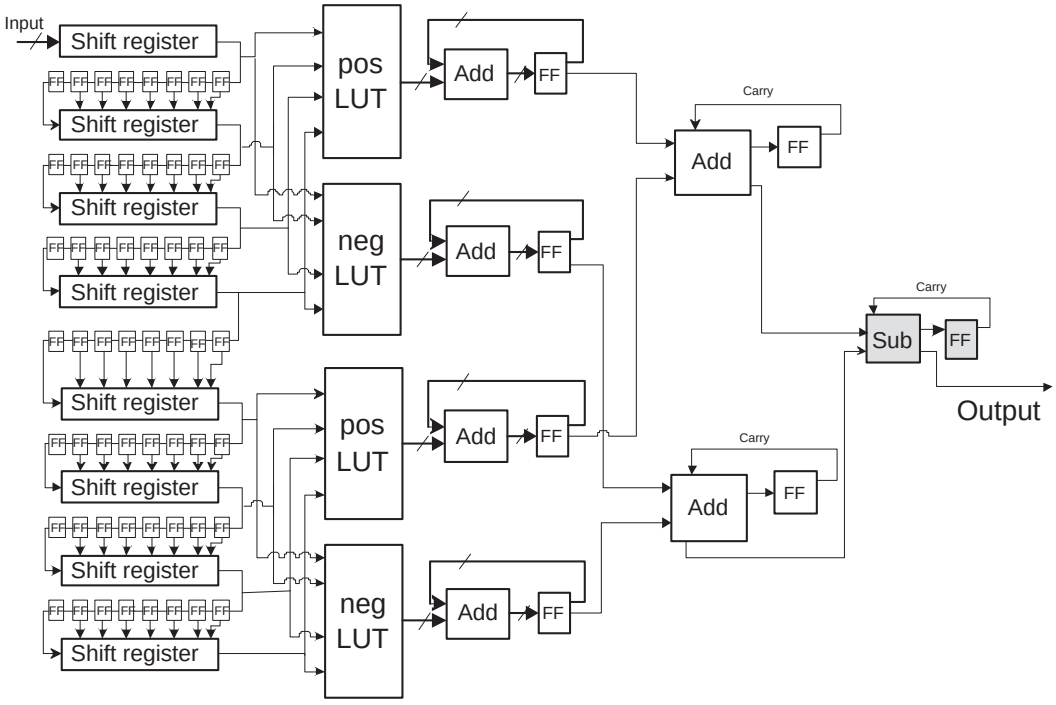

Figure 4.7. An 8 tap CSD2 filter on architecture 2

4.1.5. Other implementation issues

The main implementation issues have been discussed in the text above. While implementing the designs, problems concerning details of the designs were faced. They and their solutions are presented in this subsection.

4.1.5.1. The initial shift-registers and flip-flops. In the implemented design, the input stream is broken into 8 bit words. Between the initial shift-registers they are stored in flip-flops, waiting to be fed into the next shift-register. The number of flip-flops needed between the shift-registers is found by $max(length(x), length(c))$, where $length(x)$ is the wordlength of the input bit stream and $length(c)$ is the wordlength of the coefficient. A counter is needed to count the clock-cycles until the shift-registers need to be fed again. At that time the eight last flip-flop's values are fed into the next shift-register.

4.1.5.2. The coefficient's wordlength and precision issue. The number of bits, or wordlength of the coefficients is determined by the user. The user chooses the wordlength for the coefficients in the binary and CSD2 case. Usual choices are 8, 12, 16 etc. The number of digits for CSD4 is determined from the precision of CSD2. There will always be an error when converting a random decimal number into binary, CSD2 or CSD4 form. When determining the number of digits, the aim is that the error produced by CSD4 shall be closest possible to the error produced by CSD2. CSD4 needs, in the general case, a lower number of digits than CSD2, for the same precision. This is due to that one digit holds a higher precision.

4.1.5.3. The choice of adder. The chosen adder is a Ripple Carry Adder. Zimmermann presented in [28] a comprehensive comparison of a great number of different adders with respect to delay, area, and power. The paper concludes that the power consumption is directly related to the hardware area an adder consumes. Ripple Carry Adder is the one consuming least area and therefore is the one consuming least power.

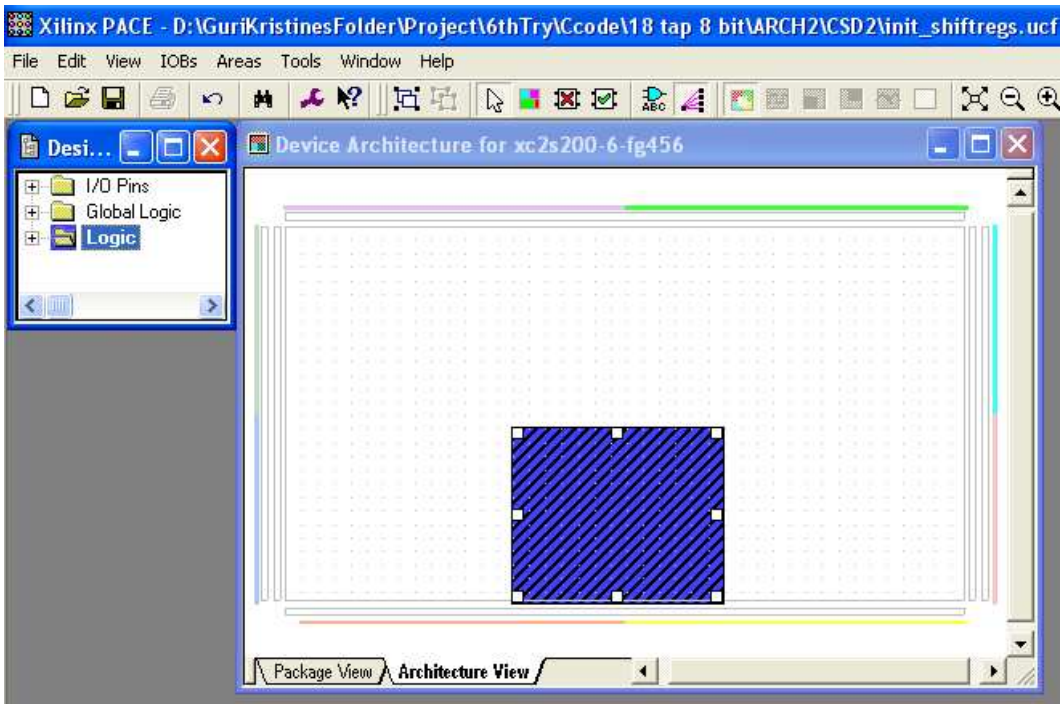

Figure 4.8. Floorplanning a CSD2 design architecture

4.1.5.4. Area Constraints. The floorplanning, as shown in figure 4.8 is the only area constraints done. It is done in Xilinx PACE. The architecture is placed close to a global clock.

In the beginning of the project the inclusion of RLOCs (Relative LOCation) were attempted to achieve a better floorplan. These, however, caused some mapping errors. They were therefore omitted. A more thorough study of architecture placement, would be interesting and is one of the proposed further works.

4.2. Design automation tool implementation in C

A design automation tool has been developed in C for automatic design of FIR filters with any number of taps. The created tool creates, for a given input file, two different architectures, and for each of the architectures three different designs. The first being a binary design, the second a CSD2 design, and the third a CSD4 design, as described in section 4. The designs are different approaches of FIR-filters.

The main task of the tool is to scale the designs according to the number of taps in a filter, and to precalculate the LUT values.

In the rest of this chapter details while creating the architectures are discussed.

4.2.1. Converting numbers from binary form to CSD form

Arda Yurdakul implemented the CSD algorithm used. The theory behind this algorithm can be found in [29]. This version has some problems since there are blind zones where the algorithm does not give a valid result. One of the blind zones covers for instance number 0.5 for CSD4. This has been overcome by adding a small fraction to the precision of the decimal number in such cases. In the example the CSD4 representation for 0.5000001 is found instead of the original number.

Another problem is that the algorithm does not find CSD numbers for coefficients above 1.0. This has been solved by dividing the decimal number by the base (base is 2 for CSD2 and 4 for CSD4), and shifting the result once to the left.

4.2.2. Pipelining

The designs are to some extend pipelined. When one iteration is done with the add-and-flip-flop structure, a new one is ready to start this action. Meanwhile the old one goes through the end additions. The pipelining for an 8-tap CSD2 filter is shown in figure 4.9. The dotted black line shows where the critical point is. The old value must have passed this point not to be mixed with the new value.

4.3. Tools

On the start-up of the project that lead to this thesis, Xilinx's Project Navigator 6.3.3 was used as the tool for implementing the architectures on the FPGA. It turned out to contain serious errors when calculating the power consumed by an architecture and

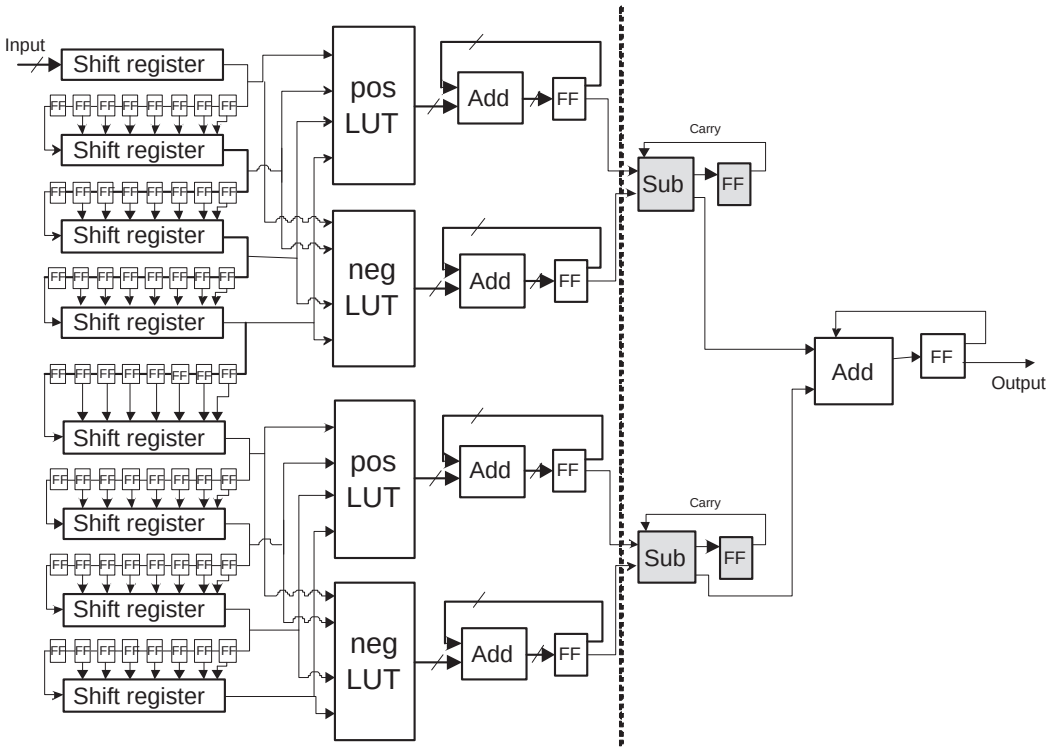

Figure 4.9. Pipelining the architectures

was not reliable. In its place a program developed by Mustafa Aktar, a PhD student at Boğaziçi University's Electrical Department, was used for some time. The program is called EVA and calculates power consumption to an accurate level. EVA's problem was that it couldn't calculate LUTs and therefore MUXs were used instead of LUTs. The disadvantage of MUXs are that they consume more power than LUTs because of their internal negator.

With the release of Xilinx's Project Navigator 7.1.02i, this tool was used. The results presented are found using Project Navigator and XPower.

The design automation tool was developed in C using Microsoft Visual C++ 6.0.

5. SIMULATIONS

The simulation chapter contains the simulations and explains how they are found. A "User guide" for students who want to continue this work is included in Appendix E.

The FIR filters used are a subset of those found from [17, 18] and some new filters. The filters have been chosen to represent different number of taps, from 4 to 62 and one filter with number of taps = 200. The filters used are included in Appendix A.

5.1. Simulation issues

While simulating the designs and making test files some problems were faced and some decisions had to be made. These are discussed in this section.

5.1.1. The input stream

The input stream is broken into words of length 8. Between each word, x number of zeros are added, where x = wordlength of the coefficients. These zeroes are also added before the input stream starts to have a correct output result.

5.1.2. The output stream

The output from the architecture comes bitwise as discussed in section 3.5 about Distributed Arithmetics. The delay, i.e. the number of cycles before the first valid result is output, is:

- Delay for the binary case: min($length(x)$, $length(c)$) + logic depth of the end additions
- Delay for the CSD2 case: min($length(x)$, $length(c)$) + logic depth of the end additions + 2

- Delay for the CSD4 case: min($length(x)$, $length(c)$) + logic depth of the end additions + 2 + 8

where $length(x)$ = input wordlength, $length(c)$ = coefficient's wordlength.

The 2 extra clock-cycles needed for the CSD2 and CSD4 case is due to the subtractor that takes two clock-cycles to produce its output.

The 8 extra clock-cycles needed for CSD4 is due to the end additions and delays to convert the result from CSD4 number representation to a bit stream of binary form.

As discussed in section 3.5 one result needs $clock\text{-}cycles = length(x) + length(c)$ to produce the whole output.

5.2. Simulation platforms

The filters were simulated on Spartan II and Spartan 3. The basic differences between these device families are that the Spartan 3 includes a multiplier. In these simulations this multiplier is not used. Spartan 3 is used for simulations of the filters with 62 and 200 taps since it has more CLBs than Spartan II.

The device used on Spartan II is "xc2s200", and on Spartan 3 "xc3s1500".

The simulations were done on ModelSim XE III/Starter 6.0a for the smaller designs, and on ModelSim SE PLUS 5.7f for the larger designs.

6. RESULTS AND DISCUSSION

Section 6.1 presents the results from the simulations and section 6.2 contains the discussion.

6.1. Results

The results are divided into the power results, presented in section 6.1.1 and the area results presented in section 6.1.2.

The values used in these figures are the values obtained from architecture 1. The two architectures consume approximately the same amount of power and area for all filters. The detailed results from the simulations are included in Appendix B.

6.1.1. Power results

Figure 6.1 shows the power consumption for binary, CSD2 and CSD4 designs for number of taps up until 32 done on Spartan II, using ModelSim XE III/Starter 6.0a. As the figure shows, the power consumption of a filter increases with increasing number of taps, as expected. For all the simulated filters, the results are that the binary approach consumes least power, CSD2 consumes a little more than the binary approach and CSD4 produces significantly more power than the two previous approaches.

Figure 6.2 shows the difference in percent between binary and CSD4 designs. The difference increase with the number of taps increasing.

Figure 6.3 shows the power consumption for binary, CSD2 and CSD4 designs when number of taps is 62 and 200 done on Spartan 3, using ModelSim SE PLUS 5.7f. For the 62-tap filter, the power consumed is the same for all the approaches. For the 200-tap filter there was a small difference between the results. The binary approach

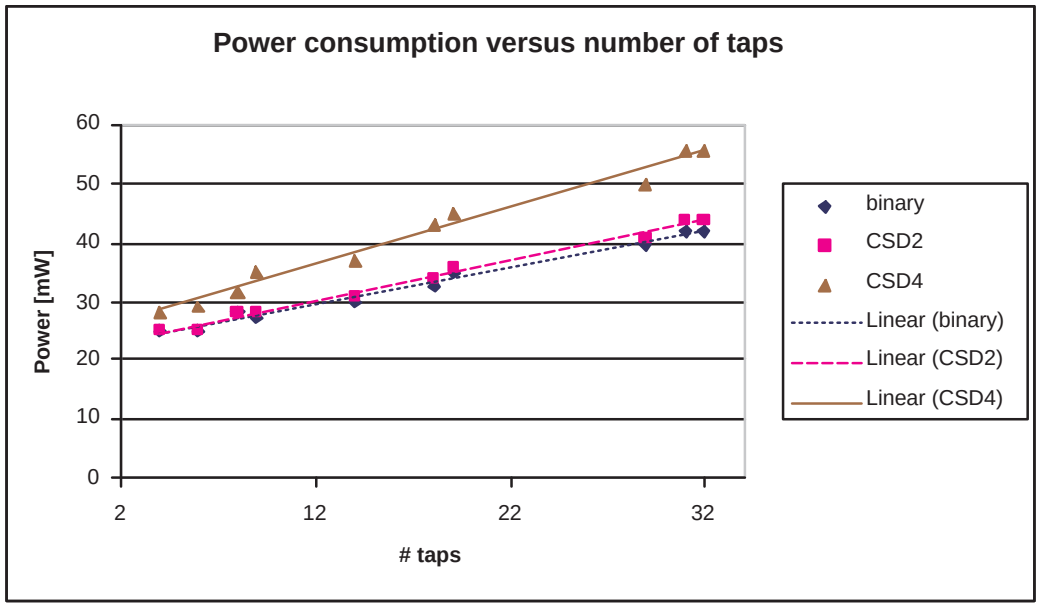

Figure 6.1. Binary, CSD2 and CSD4 power consumption for filters with number of taps up until 32

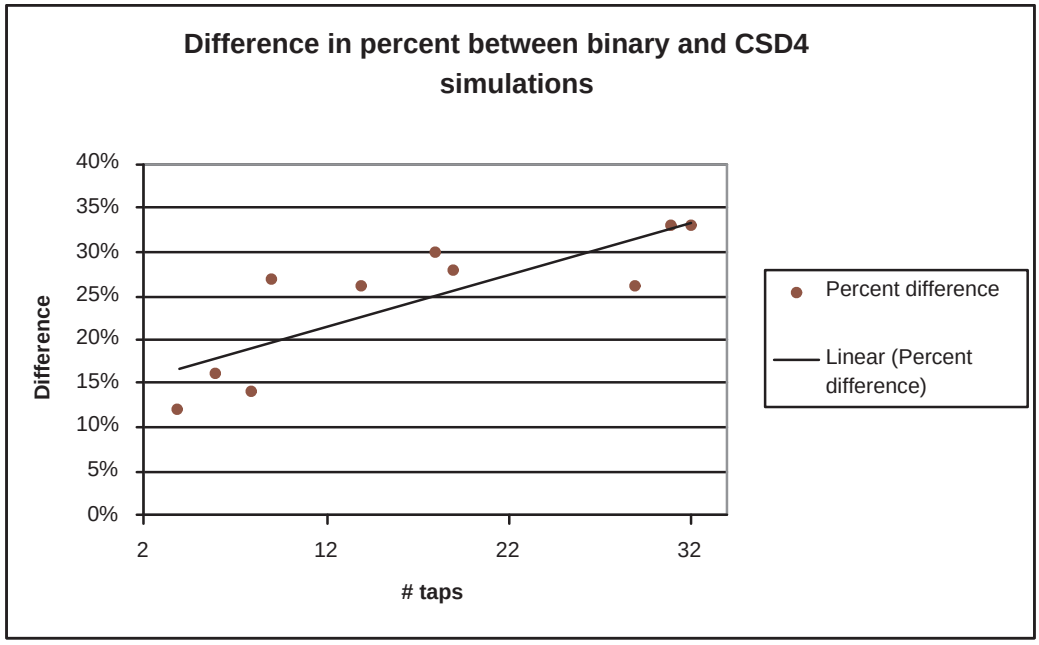

Figure 6.2. Difference in percent between binary and CSD4 design for filters with number of taps up until 32

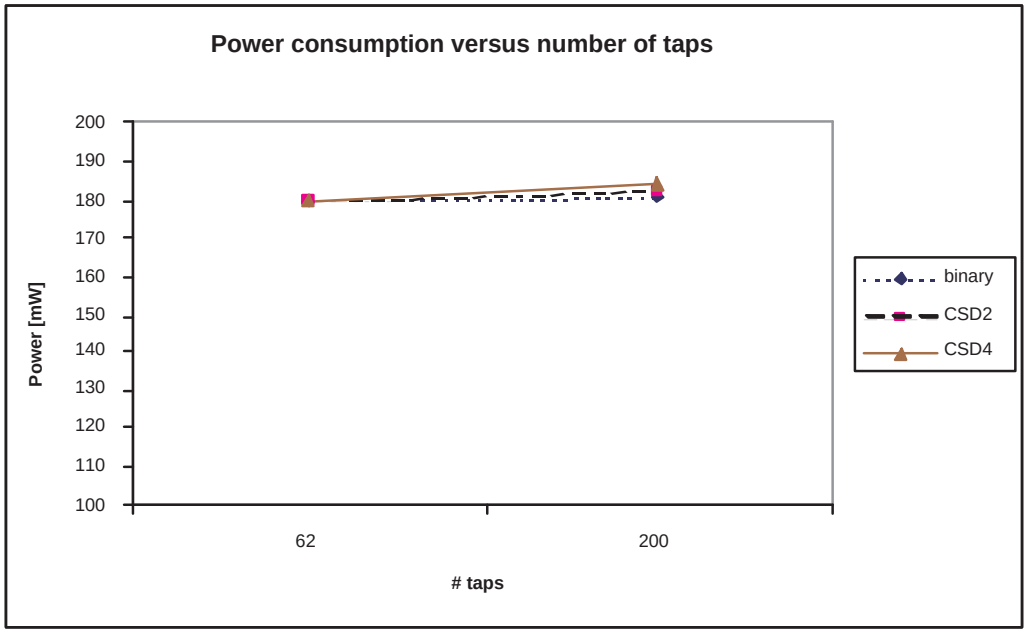

Figure 6.3. Binary, CSD2 and CSD4 power consumption for filters with number of taps: 62 and 200

produced 181 mW, the CSD2 approach produced 182 mW and the CSD4 approach produced 184 mW. The difference in percent between the binary and CSD4 approach is 1,6 %.

6.1.2. Area results

Figure 6.4 shows the area consumed by the designs with number of taps up until 32. It is remarkable that the CSD4 approach consumes more than double the area of the binary and CSD2 approaches for all taps.

Figure 6.5 shows the area consumed by the designs where number of taps is 62 and 200. These results show the same as the results for filters with number of taps from 4-32, i.e. that the CSD4 approach for all designs consumes more than double the are of the binary and CSD2 approaches.

The values of the two figures show that the designs have the same linear trendline. They are consuming proportionally the same amount of slices.

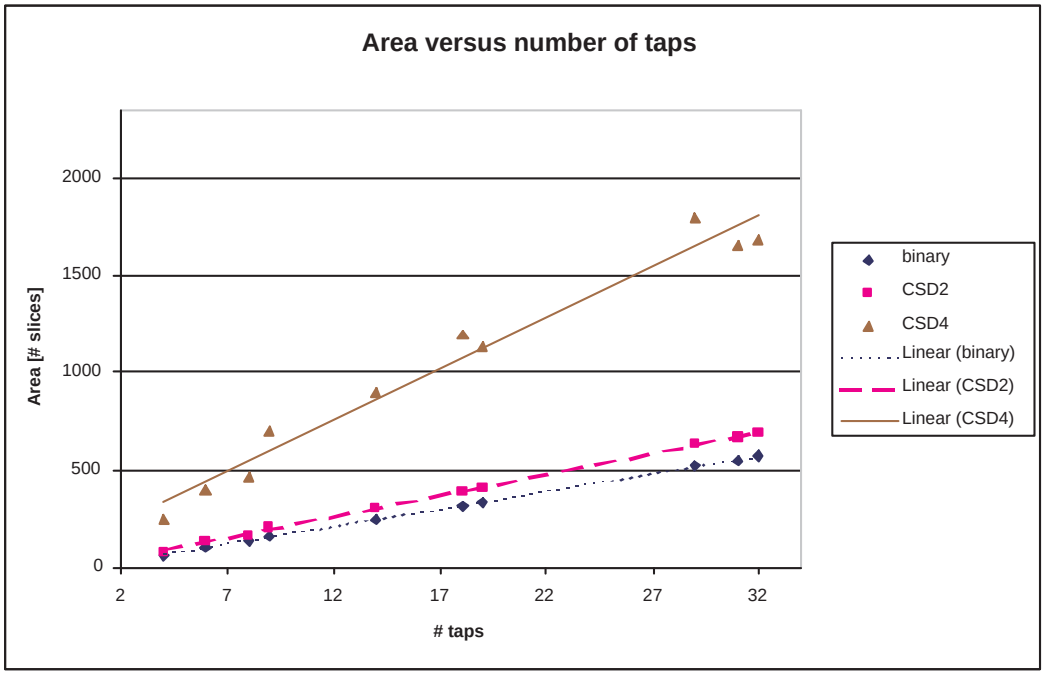

Figure 6.4. Binary, CSD2 and CSD4 area consumption for filters with number of taps up until 32

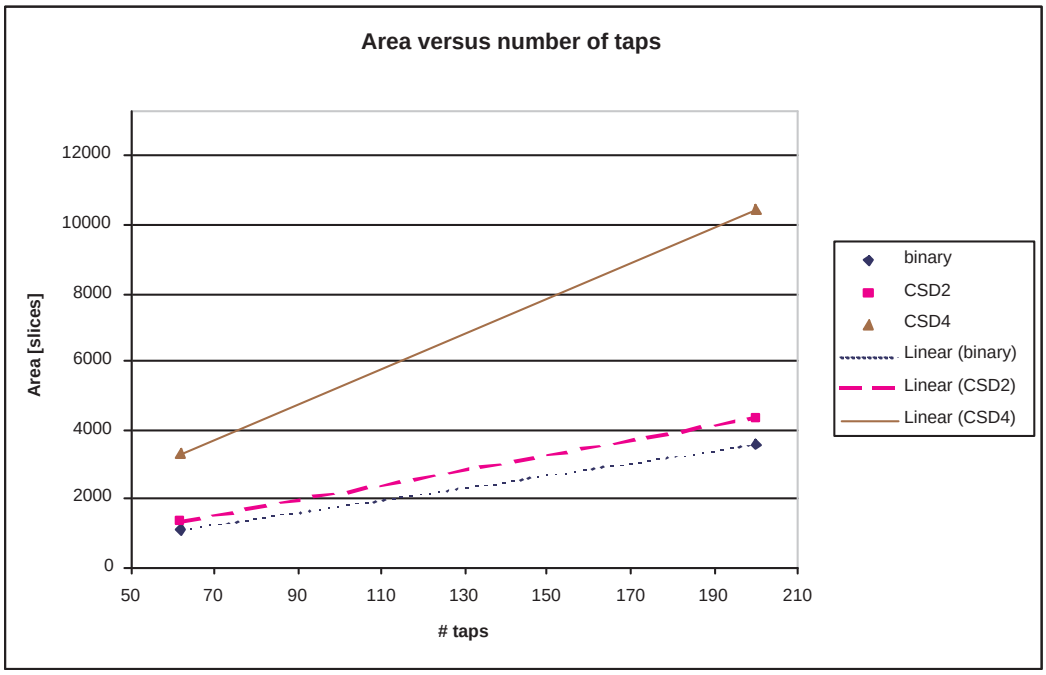

Figure 6.5. Binary, CSD2 and CSD4 area consumption for filters with number of taps: 62 and 200

6.2. Discussion

For the designs containing 4-32 taps, done on Spartan II there are some observations done and some points to discuss.

The difference in percent between the binary and the CSD4 approach as given in figure 6.2 is increasing as the number of taps in the filters increase. This is most probably due to the increase in the area for the CSD4 approach. An increase in area means more power consumed at routing.

It is remarkable that though CSD4 covers a much larger field than the binary approach, and therefore have a lot more complicated routing which is power consuming, the difference in their power consumption is not bigger. This is an indication that power is saved in the CSD4 approach, and that a careful floorplanning will show this.

It is hard to conclude something certain from figure 6.3. After finding the results from the filters with number of taps between 4 and 32, the expected results from larger number of taps filters is that they will have an increased difference between the binary, CSD2 and CSD4 approach. This is not the case for these two simulations. The two figures are somehow contradicting each other, and therefore a clear conclusion is hard to draw. The conclusion is here left for others to draw, after having simulated more filters, since it is outside of the time scope of this thesis.

The expected power consumption results were presented in chapter 4.1.3 . The figure is included here for clarity (figure 6.6). Unfortunately the results obtained do not give clear conclusions. For few tap designs done on Spartan II, the difference between the binary approach and the CSD4 approach increase with the number of tap of a filter. This is opposite of the expected result.

The expected threshold value, as shown in figure 6.6 has not been found.

The reasons why the expected results are not found are only hypothetical and can be

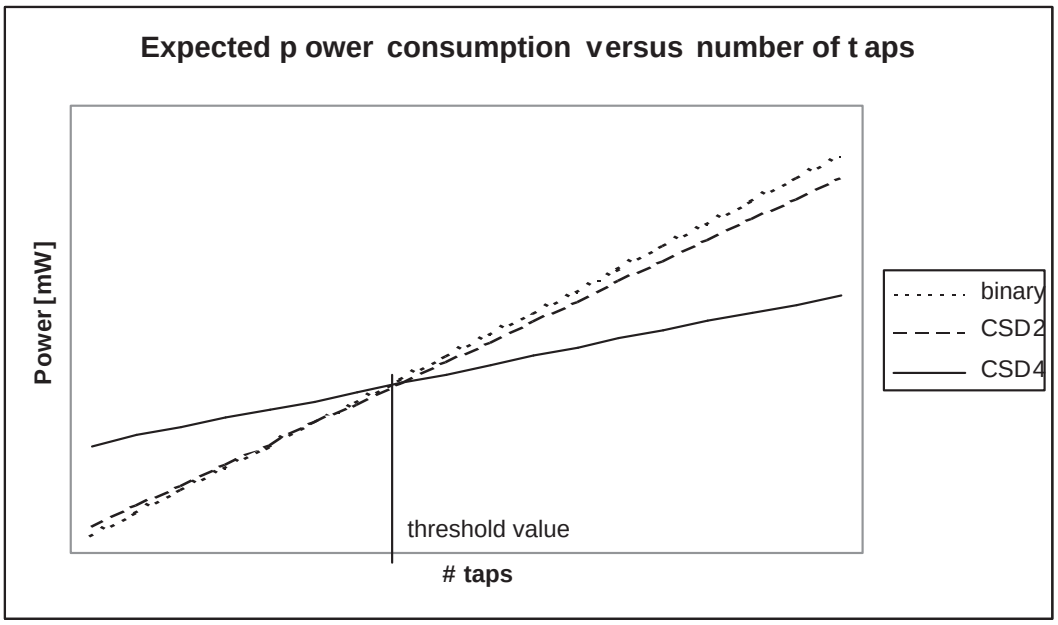

Figure 6.6. Expected power versus taps behavior

many. Some hypotheses are:

- A more detailed floorplanning, using RLOCs and HUSETs [30] could have shown better results
- The threshold value is later than 200 taps
- Xilinx's XPower's accuracy is not certain. It has not been compared to other program's accuracy

The most probable reason why CSD4, in the way we have tried to exploit it, was not successful, is the extra wiring and routing needed for a larger design. A large portion of the power consumed in a design is consumed in routing ([2], [3]). Switching is a big power consuming task, too, but a bad floorplanning can easily ruin the gain from won from switching.

One remarkable point when analysing the results is that for none of the filters is the CSD2 approach better than the binary approach. These designs are close to each other in the area they consume and therefore a gain in power consumed was expected. Despite that this gain is not shown in this thesis, it is believed that there is a good chance that it can be shown after careful floorplanning.

7. CONCLUSIONS AND FURTHER WORK

In section 7.1 the conclusions to the thesis are presented. Section 7.2 presents suggestions for further work.

7.1. Conclusion

FIR filters of various number of taps have been implemented in a Distributed Arithmetic manner on Xilinx's Spartan II and Spartan 3 FPGAs. For each filter six different designs have been built. From each of architecture 1 and 2, one binary, one CSD2 and one CSD4 design have been built and simulated. These have been compared with respect to their consumption of area and power.

The focus of the thesis has been the gain won from representing number in radix-4 CSD (CSD4) representation. The CSD4 number representation represents a number with 36 % less non-zero digits than the well-known CSD representation. In theory a lower number of non-zero bits leads to less charging and discharging of capacitances, in other words, less switching. Since switching has been found to be a highly power consuming task, reducing this activity in a circuit reduces the power this circuit consumes.

The critical point when implementing CSD4 is that the area it consumes is significantly larger than that consumed by the CSD2 and the binary approach. The end additions and subtractions needed to convert the numbers from CSD4 number representation to a binary bit stream, together with the extra area consumed for LUTs in CSD4, complicates the routing significantly. Therefore the power consumed in routing has shown to exceed the gain won from less switching. Our goal was to show a lower power consumption for the CSD4 approach. This has not been show.

A novel start of the research using the CSD4 number representation is done here. The results presented are not what was hoped for, but a refinement of the work done,

using fine floorplanning on the FPGA is expected to result in power consumption gain. Time will tell if representing numbers by CSD4 will produce even better results than the until now optimal graph-representation algorithm RAG-n [18] for FIR filters and similar processes. The theory favours CSD4 representation, but the practical approach in this thesis did not show a benefit from CSD4.

7.2. Further work

The work done in this thesis is just a novel start of a large field of research using CSD4 number representation. The further works are many, and the foreseen "carrot", the significant reduction of power consumed in a FIR-filter, is expected to be achieved.

The first step of further work is to carefully floorplan the designs presented in this thesis. The simulations of more examples of large number of taps filters to se which results are obtained from them are also an obvious further work.

Other architectures than the two presented can be developed. An approach based on LUTs placed in the FPGA's block RAM could be interesting. Another idea is to optimise the distributed arithmetics as described in some of the papers mentioned in chapter 3.5.

Truncation of the result can be done to reduce power further.

Apart from what can be done further from this thesis, it would be interesting to look at totally new approaches, for instance a match between CSD4 representation and the Common Sub-Expression Sharing algorithm presented in the Background section.

REFERENCES

1. L. Shang, A. S. Kaviani and K. Bathala: "Dynamic Power Consumption in VirtexTH-II FPGA Family", *FPGA '02, 24-26 February, 2002.*

2. A. Tehlikepalli: "Power vs. Performance", *Xilinx, April 2005, http://www.xilinx.com/bvdocs/whitepapers/WP223.pdf, last visited July 2005.*

3. ——, "Stratix II 90-nm Silicon Power Optimization", *Altera, http://www.altera.com/products/devices/stratix2/features/st2-power.html, last visited July 2003.*

4. H. T. Nguyen and A. Chatterjee: "Number-Splitting with Shift-and-Add Decomposition for Power and Hardware Optimization in Linear DSP Synthesis", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 8, no. 4, pp. 419-424, August 2000.*

5. D. R. Bull and D. H. Horrocks: "Primitive operator digital filters", *IEE Proceedings G. Circuits, Devices and Systems, vol. 138, no. 3, pp. 401-412, June 1991.*

6. ——, "Spartan-II 2.5V FPGA Family: Functional Description", *Xilinx, DS001-2 (v2.2), September 3, 2003, http://direct.xilinx.com/bvdocs/publications/ds001_2.pdf, last visited July 2005.*

7. G. W. Reitweisner: "Binary arithmetic", *Advanced Computing vol. 1, pp. 232-308, 1960.*

8. H. L. Garner: "Number systems and arithmetic", *Advances in Computers, vol. 6, pp. 131-194, 1964.*

9. J. O. Coleman: "Express Coefficients in 13-ary, Radix-4 CSD to Create Computationally Efficient Multiplierless FIR Filters", *The 15th European Conference On*

Circuit Theory and Design (ECCTD '01), Espoo, Finland, 28-31 August, 2001.

10. M. Potkonjak, M. B. Srivastava and A. P. Chandrakasan: "Multiple Constant Multiplication: Efficient and Versatile Framework and Algorithms for Exploring Common Subexpression Elimination", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 15, no. 2, pp. 151-165, February 1996.*

11. A. G. Dempster and M. D. Macleod: "Constant integer multiplication using minimum adders", *IEE Proceedings. Circuits, Devices and Systems, vol. 141, no. 5, pp. 407-413, October 1994.*

12. A. G. Dempster and M. D. Macleod: "Use of minimum adder multiplier blocks in FIR digital filters", *IEEE Transactions on Circuits and Systems II, Analog and Digital Signal Processing, vol. 42, no. 9, pp. 569-577, September 1995.*

13. M. D. Macleod and A. G. Dempster: "Multiplierless FIR Filter Design Algorithms", *IEEE Signal Processing Letters, vol. 12, no. 3, pp. 186-189, March 2005.*

14. R. I. Hartley: "Optimization of canonic signed digit multipliers for filter design", *Proceedings on IEEE International Symposium on Circuit and Systems, Singapore, pp. 1992-1995, June 1991.*

15. M. Mehendale, S. D. Sherlekar and G. Vekantesh: "Synthesis of multiplierless FIR filters with minimum number of additions", *Proceedings 1995 IEEE/ACM International Conference on Computer-Aided Design, Los Alamitos, CA, pp. 668-671, 5-9 November 1995.*

16. R. I. Hartley: "Subexpression Sharing in Filters Using Canonic Signed Digit Multipliers", *IEEE Transactions on Circuits and Systems II, Analog and Digital Signal Processing, vol. 43, no. 10, pp. 677-688, October 1995.*

17. A. Yurdakul and G. Dundar: "Multiplierless Realization of Linear DSP Transforms by Using Common Two-Term Expressions", *Journal of VLSI Signal Processing 22,*

pp. 163-172, 1999.

18. A. Yurdakul and G. Dundar: "Fast and efficient algorithm for the multiplierless realization of linear DSP transforms", *IEE Proceedings. Circuits, Devices and Systems, vol. 149, no. 4, pp. 2005-211, August 2002.*

19. M. Martnez-Peir, E. I. Boemo and L. Wanhammer: "Design of High-Speed Multiplierless Filters Using a Nonrecursive Signed Common Subexpression Algorithm", *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, vol. 49, no.3, pp. 196-203, March 2002.*

20. R. Pasko, P. Schaumont, V. Derudder, S. Vernalde, D. Durackova: "A new algorithm for elimination of common subexpressions", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 18, no. 1, pp. 58-68, January 1999.*

21. A. G. Dempster and M. D. Macleod: "Generation of Signed-Digit Representations for Integer Multiplication", *IEEE Signal Processing Letters, vol. 11, no. 8, pp. 663-665, August 2004.*

22. I.-C. Park and H.-J. Kang: "Digital Filter Synthesis Based on an Algorithm to Generate All Minimal Signed Digit Representations", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 21, no. 12, pp. 1525-1529, December 2002.*

23. L. Mintzer: "Digital Filtering in FPGAs", *IEEE Momentum Data Systems, Costa Mesa, California, 1995.*

24. S. Hwang, G. Han, S. Kang and J. Kim: "New Distributed Arithmetic Algorithm for Low-Power FIR Filter Implementation", *IEEE Signal Processing Letters, vol. 11, no. 5, pp. 463-466, May 2004.*

25. D. Akopian and J. Astola: "An Optimal Nonlinear Extension of Linear Filters

Based on Distributed Arithmetic", *IEEE Transactions on Image Processing, vol. 14, no. 5, pp. 616-623, May 2005.*

26. Sinha and M. Mehendale: "Improving Area Efficiency of FIR Filters Implemented Using Distributed Arithmetic", *VLSI Design, 1998. Proceedings., 1998 Eleventh International Conference on, pp. 104-109, 4-7 January 1998.*

27. Mehendale, A. Sinha and S. D. Sherlekar: "Low Power Realization of FIR Filters Implemented Using Distributed Arithmetic", *Design Automation Conference 1998. Proceedings of the ASP-DAC '98. Asia and South Pacific, pp. 151-156, 11-13 February 1998.*

28. R. Zimmermann: "Binary Adder Architectures for Cell-Based VLSI and their Synthesis Acknowledgments", *Diss. ETH No. 12480, 1997.*

29. J. O. Coleman and A. Yurdakul: "Fractions in the Canonical-Signed-Digit Number System", *2001 Conference on Information Sciences and Systems, The Johns Hopkins University, March 21-23, 2001.*

30. ——, "RLOC", *Xilinx, http://toolbox.xilinx.com/docsan/xilinx6/books/data/docs/cgd/cgd0161_121.html#wp240391, last visited July 2005.*

APPENDIX A: FILTERS

This appendix lists all the filters used for the simulations. They are sorted by their number of taps, the filter with the least number of taps first.

POTDAT.DAT, 4 taps:

0.795898	0.606445	0.811523	0.102539

D6H.DAT, 6 taps:

-0.051429728471	0.602859456942	-0.051429972847
-0.238929728471	-0.272140543058	0.011070271529

CAGLAR_1.DAT, 8 taps:

-0.067371764	0.40580489	0.56737176	0.094195111
0.094195111	0.56737176	0.40580489	-0.067371764

S1DAT.DAT, 9 taps:

0.0117188	0.0390625	-0.15625	0.21875	0.480469
-0.0273438	0.078125	-0.132812	0.71875	

ICSPAT.DAT, 14 taps:

0.00387498186446	0.05021063927542	-0.09754914332358	-0.00790962562117
0.00790962562117	0.09754914332358	-0.05021063927542	-0.00387498186446
0.01556811687396	0.31537477112986	-0.02820740953154	
0.02820740953154	-0.31537477112986	-0.01556811687396	

EX3_1.DAT, 18 taps:

0.234375	0.03515625	-0.0390625	0.0234375	-0.002929688	-0.0078125
0.1875	-0.01953125	-0.017578125	0.01953125	-0.013671875	0.000976563
0.1171875	-0.046875	0.008789063	0.009765625	-0.013671875	0.009765625

D.DAT, 19 taps:

0.094195111	0.56737176	-0.067371764	0.56737176	-0.067371764
0.067371764	0.40580489	0.094195111	0.40580489	0.094195111
0.40580489	0.094195111	0.40580489	0.094195111	0.40580489
0.56737176	-0.067371764	0.56737176	-0.067371764	

S2DAT.DAT, 29 taps:

0.00378418	-0.00720215	-0.0527344	-0.0410156	0.179688	0.501709
0.00341797	-0.0141602	-0.0610352	-0.0157471	0.245117	0.546631
0.00354004	-0.0229492	-0.0649414	0.0192871	0.314453	0.578247
0.00268555	-0.0327148	-0.0645752	0.064209	0.382812	0.594238
-0.0020752	-0.0429688	-0.0566406	0.117676	0.445312	

F_I.DAT, 31 taps:

-2.25642473421915E-02	1.75795945894805E-01	4.43102449133471E-03
-5.92642269619601E-02	2.57115613202005E-01	-4.44879597993632E-02
-1.77162471069205E-01	-1.77448194734674E-01	1.63328069012013E-01
-9.10609172708082E-02	-9.89160579756001E-02	-3.98649937149391E-02
-3.58527292535449E-01	5.86635173405536E-02	2.49564514832325E-01
3.76432007825370E-02	-2.90434180520017E-02	-1.50336801567996E-02
-1.69397020474223E-01	3.65904720400722E-02	1.79928590185281E-01
6.84038156849593E-02	-1.44419871273342E-02	1.04729975491613E-02
5.01834007246984E-02	2.47173571586727E-02	9.98115316448967E-02
-8.11229200371396E-02	-4.19109749997295E-02	
-1.60825548414954E-01	-2.54015713650747E-02	

EX2.DAT, 32 taps:

0.23046875	-0.040039063	-0.010253906	0.003662109	0.001708984
0.210449219	-0.017333984	-0.014404297	-0.001220703	0.001953125
0.156738281	0.007080078	-0.011230469	-0.004882813	0.001464844
0.085693359	0.022460938	-0.003662109	-0.005859375	0.000732422
0.018066406	0.024169922	0.00390625	-0.004638672	
-0.029052734	0.014892578	0.008056641	-0.002197266	
-0.047607422	0.000976563	0.007568359	0.000244141	

ARDA.DAT, 62 taps:

2.19394141740085E-02	-7.79587861897311E-02	1.75795945894805E-01
-9.58818201492958E-02	-2.33934407340529E-02	2.57115613202005E-01
1.59488285531023E-01	3.65788147048449E-02	-1.77448194734674E-01
-1.63448210936294E-01	-2.91554712130640E-02	-9.89160579756001E-02
3.36378355603590E-01	1.11606851656514E-01	5.86635173405536E-02
-2.20861062740320E-02	-3.00545134543690E-02	-2.90434180520017E-02
2.25636763706701E-01	1.07406404497879E-01	3.65904720400722E-02
-4.39717063821545E-02	-2.24432727511082E-02	-1.44419871273342E-02
-6.48466631070307E-03	8.28070481202828E-02	2.47173571586727E-02
-2.09409039355906E-01	-2.09636238440438E-02	-4.19109749997295E-02
8.91086704774373E-02	-2.25642473421915E-02	-2.54015713650747E-02
2.14723118885948E-01	-5.92642269619601E-02	4.43102449133471E-03
-2.42872252792656E-01	-1.77162471069205E-01	-4.44879597993632E-02
-1.39416733317628E-01	-9.10609172708082E-02	1.63328069012013E-01
1.23046028181698E-01	-3.58527292535449E-01	-3.98649937149391E-02
8.02303012254496E-02	3.76432007825370E-02	2.49564514832325E-01
3.69102160920208E-02	-1.69397020474223E-01	-1.50336801567996E-02
2.72943419583064E-02	6.84038156849593E-02	1.79928590185281E-01
2.08796910304073E-02	5.01834007246984E-02	1.04729975491613E-02
-7.69512593593722E-02	-8.11229200371396E-02	9.98115316448967E-02
-2.04676874968026E-02	-1.60825548414954E-01	

CP2S.DAT, 200 taps:

1.43609221233881E-05	-4.81165211295745E-03	-4.42120481078983E-02
-4.50490922576851E-05	1.18879678989135E-02	7.66924805388164E-03
4.55811265309450E-05	-5.90826659971121E-03	7.45211914802909E-02
3.01237112315495E-05	-6.74021569815144E-03	-1.02325961144905E-01
-9.89734520191030E-05	1.05979147620579E-02	7.69451534933483E-03

1.46083613857205E-11	-3.48702541339324E-03	1.28392001212250E-01
1.77638791895636E-04	-3.14203498089210E-03	-1.44622339266935E-01
-7.98895386082297E-06	2.55111300533457E-03	4.17818472564574E-10
-6.41246872173804E-04	1.79668147380937E-10	1.53605053431718E-01
1.01253054244009E-03	1.25635647754205E-03	-1.44697378628604E-01
-2.15418424329076E-04	-3.50019629442508E-03	-9.62543582902926E-03
-1.22541530518120E-03	1.62055560022173E-03	1.30498822274146E-01
1.56932780505052E-03	2.80962620579770E-03	-1.00255013403714E-01
-3.24209901652086E-04	-4.17473603442819E-03	-1.20485833270755E-02
-6.49189260038137E-04	1.30008726858909E-03	6.79750903063995E-02
4.34635608029441E-11	1.63931890666012E-03	-3.62979083173576E-02
3.28402054443668E-04	-1.66034032599121E-03	-4.04009843609086E-03
1.99332619363609E-03	2.89624207868867E-04	2.37795057997910E-11
-4.43928011199537E-03	4.48231076673493E-11	1.32423503915457E-02
1.69787560392129E-03	5.70117182559212E-04	7.68569432506482E-03
5.41240025896900E-03	-3.87520247159353E-04	-3.84289915285227E-02
-8.27415542548219E-03	-6.18707335502734E-04	2.95743468516687E-02
2.19009222420915E-03	1.09191802785951E-03	1.26848928518730E-02
5.07411657773013E-03	-4.68132971116870E-04	-3.59627440553104E-02
-4.18133880505073E-03	-3.73685623043933E-04	1.85220713542255E-02
2.23291491766925E-10	5.40167292297070E-04	7.01122390548820E-03
-4.26835327039399E-03	-1.76238519280672E-04	-9.35498218510437E-03
1.22898530714509E-02	-7.72929687332286E-05	-3.72611156433441E-10
-4.16905175945655E-03	3.91105846219205E-11	-3.35919854695790E-03
-2.14238244068196E-02	1.56754627008990E-04	1.41746408073206E-02
3.28391656379695E-02	-1.74604290226326E-04	-9.92264715757673E-03
-6.26273660053395E-03	9.00605781840030E-05	-9.12080045433690E-03
-3.11862200684734E-02	-2.06087894924414E-05	1.89794128679496E-02
3.21220458259990E-02	-6.69621455744112E-06	-8.36955766242994E-03
-2.49189603101114E-03	1.06697307238947E-12	-6.62266558494226E-03
1.25261664053327E-10	1.46373373239906E-05	8.63711717854915E-03

-3.32478095520890E-02	-3.31166206449692E-05	-1.91970790071074E-03
5.59039778407169E-03	-4.14617601871704E-05	-1.75302693523147E-10
1.76716830223407E-01	3.04609551063911E-04	-3.44347485545506E-03
-4.45101259417769E-01	-6.04749667630307E-04	2.53347095679455E-03
5.88487095577307E-01	5.46714606578425E-04	4.32465878680511E-03
-4.72747978510001E-01	-1.09959353208188E-05	-7.85150878521306E-03
1.99159318297400E-01	-4.65893921341457E-04	2.95107453603702E-03
6.99328724513562E-03	3.28991278068776E-04	3.86667605913607E-03
-4.24015718692188E-02	-5.26152435072793E-11	-4.81761366963502E-03
-3.66498827828269E-10	3.98160932624551E-04	1.17740224478815E-03
-3.91481909532878E-03	-1.14018410780922E-03	9.12904958508843E-04
4.93868179448823E-02	4.46236381401434E-04	4.13369834195003E-11
-4.99597512697674E-02	1.99800532242371E-03	-4.22423081893732E-04
-1.24341435391760E-02	-3.33800382656061E-03	-1.19103583966677E-03
6.11742365651817E-02	1.01071608027689E-03	2.28526916977082E-03
-4.07557316434372E-02	2.74358116946566E-03	-8.91372301032444E-04
-1.05784530523227E-02	-3.22532347331188E-03	-1.30498822274146E-03
2.79203614757754E-02	5.51672750082648E-04	1.75463863514720E-03
-9.60926683955201E-03	-7.90091691918908E-11	-5.33373890501699E-04
-3.79135941576188E-10	2.68843677019814E-03	-4.49516970584760E-04
-1.16849803600636E-02	-1.59119666659536E-03	3.54785408763241E-04
1.34570704125780E-02	-6.98391165745375E-03	3.68328887087430E-11
9.65011390085734E-03	1.28688346035467E-02	1.21417499802692E-04
-2.87917192366714E-02	-3.84492677532691E-03	-3.92454128514307E-04
1.68380009132384E-02	-1.31366367411586E-02	2.42571235382614E-04
1.03385697164770E-02	1.69155358102601E-02	2.37883441541676E-04
-1.95097141305474E-02	-3.02900802551096E-03	-5.49260399241027E-04
7.20922037363134E-03	-6.96100125650099E-03	4.82440707143039E-04
2.96353488269741E-03	1.75051406977144E-10	-2.40321787543229E-04
7.66004576726597E-11	2.03488378931287E-03	6.54329369661879E-05
-2.71944033094283E-03	2.26580746003562E-02	

APPENDIX B: POWER AND AREA RESULTS

This appendix contains the power and area data for all the simulations done. They are sorted by their number of taps, the filter with the least number of taps first. The "Difference between bin and csd4"-percentage value is the difference between the binary power consumption and the CSD4 power consumption for architecture 1.

The area values are the number of slices consumed by a design. The values are shown only for architecture 1. They are the same for both architectures. The percent value is how much area the design consumes compared to how large the FPGA is. The Spartan II FPGA used consists of 2352 slices. The Spartan 3 FPGA consists of 13312 slices.

Filter: POTDAT
Taps: 4
Wordlength: 12
Device Family: Spartan II

Power:

	ARCH1:	ARCH2:
BIN:	25 mW	25 mW
CSD2:	25 mW	25 mW
CSD4:	28 mW	28 mW

Difference between bin and csd4: 12%

Area:

	ARCH1:	Percent:
BIN:	67	2%
CSD2:	82	3%
CSD4:	250	10%

Filter: D6H

Taps: 6

Wordlength: 12

Device Family: Spartan II

Power:

	ARCH1:	ARCH2:
BIN:	25 mW	25 mW
CSD2:	25 mW	25 mW
CSD4:	29 mW	29 mW

Difference between bin and csd4: 16%

Area:

	ARCH1:	Percent:
BIN:	104	4%
CSD2:	135	5%
CSD4:	401	17%

Filter: CAGLAR_1

Taps: 8

Wordlength: 12

Device Family: Spartan II

Power:

	ARCH1:	ARCH2:
BIN:	28 mW	28 mW
CSD2:	28 mW	28 mW
CSD4:	32 mW	31 mW

Difference between bin and csd4: 14%

Area:

	ARCH1:	Percent:
BIN:	138	5%
CSD2:	169	7%
CSD4:	461	19%

Filter: S1DAT

Taps: 9

Wordlength: 12

Device Family: Spartan II

Power:

	ARCH1:	ARCH2:
BIN:	27 mW	27 mW
CSD2:	28 mW	28 mW
CSD4:	35 mW	35 mW

Difference between bin and csd4: 27%

Area:

	ARCH1:	Percent:
BIN:	162	6%
CSD2:	205	8%
CSD4:	699	29%

Filter: ICSPAT

Taps: 14

Wordlength: 12

Device Family: Spartan II

Power:

	ARCH1:	ARCH2:
BIN:	30 mW	30 mW
CSD2:	31 mW	31 mW
CSD4:	38 mW	37 mW

Difference between bin and csd4: 26%

Area:

	ARCH1:	Percent:
BIN:	248	10%
CSD2:	302	12%
CSD4:	894	39%

Filter: EX3_1

Taps: 18

Wordlength: 12

Device Family: Spartan II

Power:

	ARCH1:	ARCH2:
BIN:	33 mW	33 mW
CSD2:	34 mW	34 mW
CSD4:	43 mW	43 mW

Difference between bin and csd4: 30%

Area:

	ARCH1:	Percent:
BIN:	321	13%
CSD2:	390	16%
CSD4:	1201	51%

Filter: D

Taps: 19

Wordlength: 12

Device Family: Spartan II

Power:

	ARCH1:	ARCH2:
BIN:	35 mW	35 mW
CSD2:	36 mW	36 mW
CSD4:	45 mW	45 mW

Difference between bin and csd4: 28%

Area:

	ARCH1:	Percent:
BIN:	335	14%
CSD2:	405	17%
CSD4:	1132	48%

Filter: S2DAT

Taps: 29

Wordlength: 12

Device Family: Spartan II

Power:

	ARCH1:	ARCH2:
BIN:	40 mW	40 mW
CSD2:	41 mW	41 mW
CSD4:	50 mW	50 mW

Difference between bin and csd4: 26%

Area:

	ARCH1:	Percent:
BIN:	521	22%
CSD2:	637	27%
CSD4:	1792	76%

Filter: F_I

Taps: 31

Wordlength: 12

Device Family: Spartan II

Power:

	ARCH1:	ARCH2:
BIN:	42 mW	42 mW
CSD2:	44 mW	44 mW
CSD4:	56 mW	56 mW

Difference between bin and csd4: 33%

Area:

	ARCH1:	Percent:
BIN:	550	23%
CSD2:	666	28%
CSD4:	1664	70%

Filter: EX3_1

Taps: 32

Wordlength: 12

Device Family: Spartan II

Power:

	ARCH1:	ARCH2:
BIN:	42 mW	42 mW
CSD2:	44 mW	44 mW
CSD4:	56 mW	56 mW

Difference between bin and csd4: 33%

Area:

	ARCH1:	Percent:
BIN:	570	24%
CSD2:	692	29%
CSD4:	1685	71%

Filter: ARDA

Taps: 62

Wordlength: 12

Device Family: Spartan 3

Power:

	ARCH1:	ARCH2:
BIN:	180 mW	180 mW
CSD2:	180 mW	180 mW
CSD4:	180 mW	180 mW

Difference between bin and csd4: 0%

Area:

	ARCH1:	Percent:
BIN:	1109	8%
CSD2:	1346	10%
CSD4:	3326	24%

Filter: CP2S

Taps: 200

Wordlength: 12

Device Family: Spartan 3

Power:

	ARCH1:	ARCH2:
BIN:	181 mW	181 mW
CSD2:	182 mW	182 mW
CSD4:	184 mW	183 mW

Difference between bin and csd4: 2%

Area:

	ARCH1:	Percent:
BIN:	3585	26%
CSD2:	4342	32%
CSD4:	10452	78%

APPENDIX C: VHDL SOURCE CODE

This appendix contains the VHDL-code for tap POTDAT.DAT a 4 tap filter, using wordlength 8. The code is constructed using the implemented DAT tool. Section C.1 presents the three files common for all designs. Those are Add.vhdl, Add16.vhd and Sub.vhdl. Section C.2.1 lists the files produced for the binary approach, section C.2.2 lists the file produced for the CSD2 approach and section C.2.3 lists those produced for the CSD4 approach of architecture 1. Equivalent for architecture 2 are the sections C.3.1, C.3.2 and C.3.3.

C.1. Common files for all the designs

The files listed in this section are common for all the designs.

Add.vhdl

```
   -- Created by Guri Kristine Birkeland 2005

 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
 use UNISIM.VComponents.all;

 entity ADD2 is
   port
   (
      a:  in std_logic;
      b:  in std_logic;
      cii : in std_logic;
      s: out std_logic;
      coo : out std_logic
   );
end ADD2;
```

```
architecture FULLADDER of ADD2 is

signal q : std_logic;

begin

   q <= (A xor B);
   s <= (q xor cii);
   coo <= ( (A and B) or (q and cii) );

end FULLADDER;
```

Add16.vhdl

```
   -- Created by Guri Kristine Birkeland 2005

 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
 use UNISIM.VComponents.all;

 entity add16bit is
    port ( A   : in    std_logic_vector (15 downto 0);
           B   : in    std_logic_vector (15 downto 0);
           CI  : in    std_logic;
           CO  : out   std_logic;
           OFL : out   std_logic;
           S   : out   std_logic_vector (15 downto 0));
 end add16bit;

 architecture BEHAVIORAL of add16bit is
    attribute BOX_TYPE   : STRING ;
--    attribute RLOC       : STRING ;
    signal C0       : std_logic;
    signal C1       : std_logic;
    signal C2       : std_logic;
    signal C3       : std_logic;
    signal C4       : std_logic;
    signal C5       : std_logic;
    signal C6       : std_logic;
    signal C7       : std_logic;
```

```
signal C8       : std_logic;
signal C9       : std_logic;
signal C10       : std_logic;
signal C11       : std_logic;
signal C12       : std_logic;
signal C13       : std_logic;
signal C14       : std_logic;
signal C14O     : std_logic;
signal dummy    : std_logic;
signal I0       : std_logic;
signal I1       : std_logic;
signal I2       : std_logic;
signal I3       : std_logic;
signal I4       : std_logic;
signal I5       : std_logic;
signal I6       : std_logic;
signal I7       : std_logic;
signal I8       : std_logic;
signal I9       : std_logic;
signal I10       : std_logic;
signal I11       : std_logic;
signal I12       : std_logic;
signal I13       : std_logic;
signal I14       : std_logic;
signal I15       : std_logic;
signal CO_DUMMY : std_logic;
component FMAP
   port ( I1 : in    std_logic;
          I2 : in    std_logic;
          I3 : in    std_logic;
          I4 : in    std_logic;
          O  : in    std_logic);
end component;
attribute BOX_TYPE of FMAP : COMPONENT is "BLACK_BOX";

component MUXCY_L
   port ( CI : in    std_logic;
          DI : in    std_logic;
          S  : in    std_logic;
          LO : out   std_logic);
end component;
attribute BOX_TYPE of MUXCY_L : COMPONENT is "BLACK_BOX";

component MUXCY
   port ( CI : in    std_logic;
          DI : in    std_logic;
          S  : in    std_logic;
          O  : out   std_logic);
```

```
   end component;
   attribute BOX_TYPE of MUXCY : COMPONENT is "BLACK_BOX";

   component XORCY
      port ( CI : in    std_logic;
             LI : in    std_logic;
             O  : out   std_logic);
   end component;
   attribute BOX_TYPE of XORCY : COMPONENT is "BLACK_BOX";

   component MUXCY_D
      port ( CI : in    std_logic;
             DI : in    std_logic;
             S  : in    std_logic;
             LO : out   std_logic;
             O  : out   std_logic);
   end component;
   attribute BOX_TYPE of MUXCY_D : COMPONENT is "BLACK_BOX";

   component XOR2
      port ( I0 : in    std_logic;
             I1 : in    std_logic;
             O  : out   std_logic);
   end component;
   attribute BOX_TYPE of XOR2 : COMPONENT is "BLACK_BOX";

begin
   CO <= CO_DUMMY;
   I_36_16 : FMAP
      port map (I1=>A(8), I2=>B(8), I3=>dummy, I4=>dummy, O=>I8);

   I_36_17 : FMAP
      port map (I1=>A(9), I2=>B(9), I3=>dummy, I4=>dummy, O=>I9);

   I_36_18 : FMAP
      port map (I1=>A(10), I2=>B(10), I3=>dummy, I4=>dummy, O=>I10);

   I_36_19 : FMAP
      port map (I1=>A(11), I2=>B(11), I3=>dummy, I4=>dummy, O=>I11);

   I_36_20 : FMAP
      port map (I1=>A(12), I2=>B(12), I3=>dummy, I4=>dummy, O=>I12);

   I_36_21 : FMAP
      port map (I1=>A(13), I2=>B(13), I3=>dummy, I4=>dummy, O=>I13);

   I_36_22 : FMAP
      port map (I1=>A(14), I2=>B(14), I3=>dummy, I4=>dummy, O=>I14);
```

```
I_36_23 : FMAP
   port map (I1=>A(15), I2=>B(15), I3=>dummy, I4=>dummy, O=>I15);

I_36_55 : MUXCY_L
   port map (CI=>C8, DI=>A(9), S=>I9, LO=>C9);

I_36_58 : MUXCY_L
   port map (CI=>C10, DI=>A(11), S=>I11, LO=>C11);

I_36_62 : MUXCY_L
   port map (CI=>C9, DI=>A(10), S=>I10, LO=>C10);

I_36_63 : MUXCY_L
   port map (CI=>C11, DI=>A(12), S=>I12, LO=>C12);

I_36_64 : MUXCY
   port map (CI=>C14, DI=>A(15), S=>I15, O=>CO_DUMMY);

I_36_73 : XORCY
   port map (CI=>C7, LI=>I8, O=>S(8));

I_36_74 : XORCY
   port map (CI=>C8, LI=>I9, O=>S(9));

I_36_75 : XORCY
   port map (CI=>C10, LI=>I11, O=>S(11));

I_36_76 : XORCY
   port map (CI=>C9, LI=>I10, O=>S(10));

I_36_77 : XORCY
   port map (CI=>C12, LI=>I13, O=>S(13));

I_36_78 : XORCY
   port map (CI=>C11, LI=>I12, O=>S(12));

I_36_80 : XORCY
   port map (CI=>C14, LI=>I15, O=>S(15));

I_36_81 : XORCY
   port map (CI=>C13, LI=>I14, O=>S(14));

I_36_107 : MUXCY_D
   port map (CI=>C13, DI=>A(14), S=>I14, LO=>C14, O=>C14O);

I_36_110 : MUXCY_L
   port map (CI=>C12, DI=>A(13), S=>I13, LO=>C13);
```

```
    I_36_111 : MUXCY_L
       port map (CI=>C7, DI=>A(8), S=>I8, LO=>C8);

    I_36_226 : XORCY
       port map (CI=>CI, LI=>I0, O=>S(0));

    I_36_227 : XORCY
       port map (CI=>C0, LI=>I1, O=>S(1));

    I_36_228 : XORCY
       port map (CI=>C2, LI=>I3, O=>S(3));

    I_36_229 : XORCY
       port map (CI=>C1, LI=>I2, O=>S(2));

    I_36_230 : XORCY
       port map (CI=>C4, LI=>I5, O=>S(5));

    I_36_231 : XORCY
       port map (CI=>C3, LI=>I4, O=>S(4));

    I_36_233 : XORCY
       port map (CI=>C6, LI=>I7, O=>S(7));

    I_36_234 : XORCY
       port map (CI=>C5, LI=>I6, O=>S(6));

    I_36_248 : MUXCY_L
       port map (CI=>C6, DI=>A(7), S=>I7, LO=>C7);

    I_36_249 : MUXCY_L
       port map (CI=>C5, DI=>A(6), S=>I6, LO=>C6);

    I_36_250 : MUXCY_L
       port map (CI=>C4, DI=>A(5), S=>I5, LO=>C5);

    I_36_251 : MUXCY_L
       port map (CI=>C3, DI=>A(4), S=>I4, LO=>C4);

    I_36_252 : MUXCY_L
port map (CI=>C2, DI=>A(3), S=>I3, LO=>C3);

    I_36_253 : MUXCY_L
       port map (CI=>C1, DI=>A(2), S=>I2, LO=>C2);

    I_36_254 : MUXCY_L
       port map (CI=>C0, DI=>A(1), S=>I1, LO=>C1);
```

```
I_36_255 : MUXCY_L
   port map (CI=>CI, DI=>A(0), S=>I0, LO=>C0);

I_36_272 : FMAP
   port map (I1=>A(1), I2=>B(1), I3=>dummy, I4=>dummy, O=>I1);

I_36_275 : FMAP
   port map (I1=>A(0), I2=>B(0), I3=>dummy, I4=>dummy, O=>I0);

I_36_279 : FMAP
   port map (I1=>A(2), I2=>B(2), I3=>dummy, I4=>dummy, O=>I2);

I_36_283 : FMAP
   port map (I1=>A(3), I2=>B(3), I3=>dummy, I4=>dummy, O=>I3);

I_36_287 : FMAP
   port map (I1=>A(4), I2=>B(4), I3=>dummy, I4=>dummy, O=>I4);

I_36_291 : FMAP
   port map (I1=>A(5), I2=>B(5), I3=>dummy, I4=>dummy, O=>I5);

I_36_295 : FMAP
   port map (I1=>A(6), I2=>B(6), I3=>dummy, I4=>dummy, O=>I6);

I_36_299 : FMAP
   port map (I1=>A(7), I2=>B(7), I3=>dummy, I4=>dummy, O=>I7);

I_36_354 : XOR2
   port map (I0=>A(0), I1=>B(0), O=>I0);

I_36_355 : XOR2
   port map (I0=>A(1), I1=>B(1), O=>I1);

I_36_356 : XOR2
   port map (I0=>A(2), I1=>B(2), O=>I2);

I_36_357 : XOR2
   port map (I0=>A(3), I1=>B(3), O=>I3);

I_36_358 : XOR2
   port map (I0=>A(4), I1=>B(4), O=>I4);

I_36_359 : XOR2
   port map (I0=>A(5), I1=>B(5), O=>I5);

I_36_360 : XOR2
   port map (I0=>A(6), I1=>B(6), O=>I6);
```

```
    I_36_361 : XOR2
       port map (I0=>A(7), I1=>B(7), O=>I7);

    I_36_362 : XOR2
       port map (I0=>A(8), I1=>B(8), O=>I8);

    I_36_363 : XOR2
       port map (I0=>A(9), I1=>B(9), O=>I9);

    I_36_364 : XOR2
       port map (I0=>A(10), I1=>B(10), O=>I10);

    I_36_365 : XOR2
       port map (I0=>A(11), I1=>B(11), O=>I11);

    I_36_366 : XOR2
       port map (I0=>A(12), I1=>B(12), O=>I12);

    I_36_367 : XOR2
port map (I0=>A(13), I1=>B(13), O=>I13);

    I_36_368 : XOR2
       port map (I0=>A(14), I1=>B(14), O=>I14);

    I_36_369 : XOR2
       port map (I0=>A(15), I1=>B(15), O=>I15);

    I_36_375 : XOR2
       port map (I0=>C14O, I1=>CO_DUMMY, O=>OFL);

 end BEHAVIORAL;
```

Sub.vhdl

```
   -- Created by Guri Kristine Birkeland 2005

 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
 use UNISIM.VComponents.all;
```

```
ENTITY SUB2 IS
    PORT(
           A:  IN STD_LOGIC;
           B:  IN STD_LOGIC;
           bin : in std_logic;
           Q:  OUT STD_LOGIC;
           bout :   OUT STD_LOGIC
        );
END SUB2;

ARCHITECTURE logic OF SUB2 IS

BEGIN
   Q <= (A xor B xor bin);

   bout <= (not A and B) or (not A and bin) or (B and bin);

END logic;
```

C.2. Architecture 1 VHDL files

This section lists the files needed to produce a 4-tap filter of architecture 1.

C.2.1. Binary design

The files needed to produce the binary design of architecture 1 is listed below.

AddAndFlipFlop.vhdl

```
   -- Created by Guri Kristine Birkeland 2005

 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
 use UNISIM.VComponents.all;
```

```
entity add_ff is
   port ( clock   : in    std_logic;
          CE  : in    std_logic;
          CLR : in    std_logic;
          afi0 : in std_logic_vector(7 downto 0);
          afso : out std_logic);
end add_ff;

architecture BEHAVIORAL of add_ff is

component add16bit
    port ( A   : in    std_logic_vector (15 downto 0);
           B   : in    std_logic_vector (15 downto 0);
           CI  : in    std_logic;
           CO  : out   std_logic;
           OFL : out   std_logic;
           S   : out   std_logic_vector (15 downto 0));
end component;

component flipflop
   port ( C   : in    std_logic;
          CE  : in    std_logic;
          CLR : in    std_logic;
          D   : in    std_logic_vector (8 downto 0);
          Q   : out   std_logic_vector (7 downto 0);
        so : out std_logic);
end component;

signal outadder : std_logic_vector(8 downto 0); signal inadder :
std_logic_vector(7 downto 0);

signal dummy0 : std_logic; signal dummy1 : std_logic_vector(6
downto 0); signal dummy2 : std_logic;

begin
  adder : add16bit
    port map ( A(7 downto 0) => afi0, A(15 downto 8)=>"00000000",
             B(7 downto 0) => inadder, B(15 downto 8)=>"00000000",
             CI => '0', CO=> dummy0, OFL => dummy2,
             S(8 downto 0)=> outadder, S(15 downto 9)=> dummy1);

  ff : flipflop
    port map (C => clock, CE => CE, CLR => CLR, D => outadder,
              Q => inadder, so => afso);

 end BEHAVIORAL;
```

Arch1_BIN.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity ARCH1_BIN is
    Port ( clock : in std_logic;
         enable :  in std_logic;
         tin : in std_logic_vector(3 downto 0);
       tout : out std_logic
         );
end ARCH1_BIN;

architecture Behavioral of ARCH1_BIN is

component lut_0
    Port ( clock : in std_logic;
         enable :  in std_logic;
         ti : in std_logic_vector(3 downto 0);
       tout : out std_logic);
end component;

component add2
  port
  (
     a:  in std_logic;
     b:  in std_logic;
     cii : in std_logic;
     s: out std_logic;
     coo : out std_logic
  );
end component;

signal partsum0 : std_logic;

begin

LUT_00 : lut_0
```

```
    port map (clock=>clock, enable=>enable, ti=>tin(3 downto 0), tout=>partsum0);

 tout<=partsum0;

 end Behavioral;
```

FlipFlop.vhdl

```
   -- Created by Guri Kristine Birkeland 2005

 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
 use UNISIM.VComponents.all;

entity flipflop is
   port ( C   : in    std_logic;
          CE  : in    std_logic;
          CLR : in    std_logic;
          D   : in    std_logic_vector (8 downto 0);
          Q   : out   std_logic_vector (7 downto 0);
        so : out std_logic);
end flipflop;

architecture BEHAVIORAL of flipflop is

   attribute INIT       : STRING ;
   attribute BOX_TYPE   : STRING ;
   component FDCE
      -- synopsys translate_off
      generic( INIT : bit :=  '0');
      -- synopsys translate_on
      port ( C   : in    std_logic;
             CE  : in    std_logic;
             CLR : in    std_logic;
             D   : in    std_logic;
             Q   : out   std_logic);
   end component;
   attribute INIT of FDCE : COMPONENT is "0";
   attribute BOX_TYPE of FDCE : COMPONENT is "BLACK_BOX";
```

```
begin

   I_Q0 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(0), Q=>so);

   I_Q1 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(1), Q=>Q(0));

   I_Q2 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(2), Q=>Q(1));

   I_Q3 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(3), Q=>Q(2));

   I_Q4 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(4), Q=>Q(3));

   I_Q5 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(5), Q=>Q(4));

   I_Q6 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(6), Q=>Q(5));

   I_Q7 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(7), Q=>Q(6));

   I_Q8 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(8), Q=>Q(7));

end BEHAVIORAL;
```

Init_ShiftRegisters.vhdl

```
   -- Created by Guri Kristine Birkeland 2005

 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
 use UNISIM.VComponents.all;

 entity INIT_SHIFTREGS is
```

```
    Port ( clk, enable, load : in std_logic;
         tin : in std_logic_vector(7 downto 0);
         tout : out std_logic
         );
end INIT_SHIFTREGS;

architecture Behavioral of INIT_SHIFTREGS is

component shiftreg16bit
   port ( C    : in    std_logic;        -- clock
          CE   : in    std_logic;        -- enable
          CLR  : in    std_logic;        -- clear
          D    : in    std_logic_vector (15 downto 0);  -- input
          L    : in    std_logic;        -- load
          LEFT : in    std_logic;        -- shift left
          SLI  : in    std_logic;        -- add to left 0/1
          SRI  : in    std_logic;        -- add to right 0/1
          Q    : out   std_logic_vector (15 downto 0));     -- output
end component;

component FDCE
     -- synopsys translate_off
     generic( INIT : bit :=  '0');
     -- synopsys translate_on
     port ( C   : in    std_logic;
            CE  : in    std_logic;
            CLR : in    std_logic;
            D   : in    std_logic;
            Q   : out   std_logic);
end component;

component ARCH1_BIN
    Port ( clock : in std_logic;
         enable :  in std_logic;
         tin : in std_logic_vector(3 downto 0);
       tout : out std_logic
         );
end component;

signal shiftOut : std_logic_vector(3 downto 0);

signal out0 : std_logic_vector(14 downto 0);
signal out1 : std_logic_vector(14 downto 0);
signal out2 : std_logic_vector(14 downto 0);
signal out3 : std_logic_vector(14 downto 0);

signal Q0 : std_logic_vector(7 downto 0);
signal Q1 : std_logic_vector(7 downto 0);
```

```
signal Q2 : std_logic_vector(7 downto 0);

signal GND : std_logic := '0';

begin

shiftreg_0 : shiftreg16bit
   port map (C => clk, CE => enable, CLR => GND,
           D(7 downto 0) => "00000000",
           D(15 downto 8) => tin,
           L => load, LEFT => GND, SLI => GND, SRI => GND,
           Q(0) => shiftOut(0), Q(15 downto 1) => out0
           );

-- flipflops to hold the value for next shiftregister.
flipflop_00 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>shiftOut(0), Q=>Q0(7));

flipflop_01 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(7), Q=>Q0(6));

flipflop_02 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(6), Q=>Q0(5));

flipflop_03 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(5), Q=>Q0(4));

flipflop_04 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(4), Q=>Q0(3));

flipflop_05 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(3), Q=>Q0(2));

flipflop_06 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(2), Q=>Q0(1));

flipflop_07 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(1), Q=>Q0(0));


shiftreg_1 : shiftreg16bit
   port map (C => clk, CE => enable, CLR => GND,
           D(7 downto 0) => "00000000",
           D(15 downto 8) => Q0,
           L => load, LEFT => GND, SLI => GND, SRI => GND,
           Q(0) => shiftOut(1), Q(15 downto 1) => out1
           );
flipflop_10 :  FDCE
```

```
     port map (C=>clk, CE=>enable, CLR=>GND, D=>shiftOut(1), Q=>Q1(7));

flipflop_11 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(7), Q=>Q1(6));

flipflop_12 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(6), Q=>Q1(5));

flipflop_13 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(5), Q=>Q1(4));

flipflop_14 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(4), Q=>Q1(3));

flipflop_15 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(3), Q=>Q1(2));

flipflop_16 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(2), Q=>Q1(1));

flipflop_17 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(1), Q=>Q1(0));


shiftreg_2 : shiftreg16bit
   port map (C => clk, CE => enable, CLR => GND,
           D(7 downto 0) => "00000000",
           D(15 downto 8) => Q1,
           L => load, LEFT => GND, SLI => GND, SRI => GND,
           Q(0) => shiftOut(2), Q(15 downto 1) => out2
           );
flipflop_20 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>shiftOut(2), Q=>Q2(7));

flipflop_21 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(7), Q=>Q2(6));

flipflop_22 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(6), Q=>Q2(5));

flipflop_23 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(5), Q=>Q2(4));

flipflop_24 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(4), Q=>Q2(3));

flipflop_25 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(3), Q=>Q2(2));
```

```
flipflop_26 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(2), Q=>Q2(1));

flipflop_27 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(1), Q=>Q2(0));


shiftreg_3 : shiftreg16bit
   port map (C => clk, CE => enable, CLR => GND,
           D(7 downto 0) => "00000000",
           D(15 downto 8) => Q2,
           L => load, LEFT => GND, SLI => GND, SRI => GND,
           Q(0) => shiftOut(3), Q(15 downto 1) => out3
           );

arch0 : ARCH1_BIN
    port map (clock=>clk, enable=>enable,
                   tin=>shiftOut, tout=>tout
         );

end Behavioral;
```

Lut8Bits.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity lut is
    generic (  INIT0 : bit_vector := x"16";
               INIT1 : bit_vector := x"16";
               INIT2 : bit_vector := x"16";
               INIT3 : bit_vector := x"16";
               INIT4 : bit_vector := x"16";
               INIT5 : bit_vector := x"16";
               INIT6 : bit_vector := x"16";
               INIT7 : bit_vector := x"16");
```

```
   Port ( ui0 : in std_logic;
          ui1 : in std_logic;
          ui2 : in std_logic;
          ui3 : in std_logic;
          uo : out std_logic_vector(7 downto 0));
end lut;

architecture Behavioral of lut is
       -- Component Declaration for LUT4 should be placed
-- after architecture statement but before begin keyword
component LUT4
   generic (INIT : bit_vector := x"16");
   port (O : out STD_LOGIC;
           I0 : in STD_LOGIC;
           I1 : in STD_LOGIC;
           I2 : in STD_LOGIC;
           I3 : in STD_LOGIC);
end component;

-- Component Attribute specification for LUT4
-- should be placed after architecture declaration but
-- before the begin keyword
attribute INIT : string;
begin
LUT4_p0 : LUT4 generic map (INIT => INIT0)
   port map (O => uO(0),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p1 : LUT4 generic map (INIT => INIT1)
   port map (O => uO(1),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p2 : LUT4 generic map (INIT => INIT2)
   port map (O => uO(2),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p3 : LUT4 generic map (INIT => INIT3)
   port map (O => uO(3),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p4 : LUT4 generic map (INIT => INIT4)
   port map (O => uO(4),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p5 : LUT4 generic map (INIT => INIT5)
   port map (O => uO(5),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p6 : LUT4 generic map (INIT => INIT6)
   port map (O => uO(6),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p7 : LUT4 generic map (INIT => INIT7)
   port map (O => uO(7),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);

end Behavioral;
```

Lut_0.vhdl

```
  -- Created by Guri Kristine Birkeland 2005
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity lut_0 is
    Port ( clock : in std_logic;
                enable :  in std_logic;
                ti : in std_logic_vector(3 downto 0);
              tout : out std_logic
                );
end lut_0;

architecture Behavioral of lut_0 is


component lut
    generic (
             INIT0 : bit_vector := x"16";
             INIT1 : bit_vector := x"16";
             INIT2 : bit_vector := x"16";
             INIT3 : bit_vector := x"16";
             INIT4 : bit_vector := x"16";
             INIT5 : bit_vector := x"16";
             INIT6 : bit_vector := x"16";
             INIT7 : bit_vector := x"16");
   port (
             UI0   : in    std_logic;
             UI1   : in    std_logic;
             UI2   : in    std_logic;
             UI3   : in    std_logic;
             UO    : out   std_logic_vector(7 downto 0)
       );
end component;

component add_ff
  port ( clock   : in    std_logic;
         CE  : in    std_logic;
         CLR : in    std_logic;
            afi0 : in std_logic_vector(7 downto 0);
            afso : out std_logic
            );
```

```
end component;

signal poslutout : std_logic_vector(7 downto 0);

begin

-- lut
poslut_0 : lut
   generic map (
           INIT0=>"0000111111110010",
           INIT1=>"0001010101011010",
           INIT2=>"0000110000110110",
           INIT3=>"0001101001101000",
           INIT4=>"0001001101001100",
           INIT5=>"0010111110111100",
           INIT6=>"0011110011000000",
           INIT7=>"0000000000000000")
   port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),
             UO=>poslutout);

 add_ff_0: add_ff
   port map (clock, enable, '0', poslutout, tout);

end Behavioral;
```

ShiftReg16.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity M2_1_MXILINX_test is
   port ( D0 : in    std_logic;
          D1 : in    std_logic;
          S0 : in    std_logic;
          O  : out   std_logic);
end M2_1_MXILINX_test;
```

```
architecture BEHAVIORAL of M2_1_MXILINX_test is
   attribute BOX_TYPE   : STRING ;
   signal M0 : std_logic;
   signal M1 : std_logic;
   component AND2B1
      port ( I0 : in    std_logic;
             I1 : in    std_logic;
             O  : out   std_logic);
   end component;
   attribute BOX_TYPE of AND2B1 : COMPONENT is "BLACK_BOX";

   component OR2
      port ( I0 : in    std_logic;
             I1 : in    std_logic;
             O  : out   std_logic);
   end component;
   attribute BOX_TYPE of OR2 : COMPONENT is "BLACK_BOX";

   component AND2
      port ( I0 : in    std_logic;
             I1 : in    std_logic;
             O  : out   std_logic);
   end component;
   attribute BOX_TYPE of AND2 : COMPONENT is "BLACK_BOX";

begin
   I_36_7 : AND2B1
      port map (I0=>S0, I1=>D0, O=>M0);

   I_36_8 : OR2
      port map (I0=>M1, I1=>M0, O=>O);

   I_36_9 : AND2
      port map (I0=>D1, I1=>S0, O=>M1);

end BEHAVIORAL;


  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
```

```
use UNISIM.VComponents.all;

entity shiftreg16bit is
   port ( C    : in    std_logic;   -- clock
          CE   : in    std_logic;   -- enable
          CLR  : in    std_logic;   -- clear
          D    : in    std_logic_vector (15 downto 0);   -- input
          L    : in    std_logic;   -- load
          LEFT : in    std_logic;   -- shift left
          SLI  : in    std_logic;   -- add to left 0/1
          SRI  : in    std_logic;   -- add to right 0/1
          Q    : out   std_logic_vector (15 downto 0));  -- output
end shiftreg16bit;

architecture BEHAVIORAL of shiftreg16bit is
   attribute HU_SET     : STRING ;
   attribute INIT       : STRING ;
   attribute BOX_TYPE   : STRING ;
   signal L_LEFT  : std_logic;
   signal L_OR_CE : std_logic;
   signal MDL0    : std_logic;
   signal MDL1    : std_logic;
   signal MDL2    : std_logic;
   signal MDL3    : std_logic;
   signal MDL4    : std_logic;
   signal MDL5    : std_logic;
   signal MDL6    : std_logic;
   signal MDL7    : std_logic;
   signal MDL8    : std_logic;
   signal MDL9    : std_logic;
   signal MDL10    : std_logic;
   signal MDL11    : std_logic;
   signal MDL12    : std_logic;
   signal MDL13    : std_logic;
   signal MDL14    : std_logic;
   signal MDL15    : std_logic;
   signal MDR0    : std_logic;
   signal MDR1    : std_logic;
   signal MDR2    : std_logic;
   signal MDR3    : std_logic;
   signal MDR4    : std_logic;
   signal MDR5    : std_logic;
   signal MDR6    : std_logic;
   signal MDR7    : std_logic;
   signal MDR8    : std_logic;
   signal MDR9    : std_logic;
   signal MDR10    : std_logic;
   signal MDR11    : std_logic;
```

```
   signal MDR12    : std_logic;
   signal MDR13    : std_logic;
   signal MDR14    : std_logic;
   signal MDR15    : std_logic;
   signal Q_DUMMY : std_logic_vector (15 downto 0);
   component M2_1_MXILINX_test
      port ( D0 : in    std_logic;
             D1 : in    std_logic;
             S0 : in    std_logic;
             O  : out   std_logic);
   end component;

   component FDCE
      -- synopsys translate_off
      generic( INIT : bit :=  '0');
      -- synopsys translate_on
      port ( C   : in    std_logic;
             CE  : in    std_logic;
             CLR : in    std_logic;
             D   : in    std_logic;
             Q   : out   std_logic);
   end component;
   attribute INIT of FDCE : COMPONENT is "0";
   attribute BOX_TYPE of FDCE : COMPONENT is "BLACK_BOX";

   component OR2
      port ( I0 : in    std_logic;
             I1 : in    std_logic;
             O  : out   std_logic);
   end component;
   attribute BOX_TYPE of OR2 : COMPONENT is "BLACK_BOX";

begin
   Q(15 downto 0) <= Q_DUMMY(15 downto 0);
   I_MDL0 : M2_1_MXILINX_test
      port map (D0=>SLI, D1=>D(0), S0=>L, O=>MDL0);

   I_MDL1 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(0), D1=>D(1), S0=>L, O=>MDL1);

   I_MDL2 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(1), D1=>D(2), S0=>L, O=>MDL2);

   I_MDL3 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(2), D1=>D(3), S0=>L, O=>MDL3);

   I_MDL4 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(3), D1=>D(4), S0=>L, O=>MDL4);
```

```
I_MDL5 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(4), D1=>D(5), S0=>L, O=>MDL5);

I_MDL6 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(5), D1=>D(6), S0=>L, O=>MDL6);

I_MDL7 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(6), D1=>D(7), S0=>L, O=>MDL7);

I_MDL8 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(7), D1=>D(8), S0=>L, O=>MDL8);

I_MDL9 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(8), D1=>D(9), S0=>L, O=>MDL9);

I_MDL10 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(9), D1=>D(10), S0=>L, O=>MDL10);

I_MDL11 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(10), D1=>D(11), S0=>L, O=>MDL11);

I_MDL12 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(11), D1=>D(12), S0=>L, O=>MDL12);

I_MDL13 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(12), D1=>D(13), S0=>L, O=>MDL13);

I_MDL14 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(13), D1=>D(14), S0=>L, O=>MDL14);

I_MDL15 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(14), D1=>D(15), S0=>L, O=>MDL15);

I_MDR0 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(1), D1=>MDL0, S0=>L_LEFT, O=>MDR0);

I_MDR1 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(2), D1=>MDL1, S0=>L_LEFT, O=>MDR1);

I_MDR2 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(3), D1=>MDL2, S0=>L_LEFT, O=>MDR2);

I_MDR3 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(4), D1=>MDL3, S0=>L_LEFT, O=>MDR3);

I_MDR4 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(5), D1=>MDL4, S0=>L_LEFT, O=>MDR4);
```

```
I_MDR5 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(6), D1=>MDL5, S0=>L_LEFT, O=>MDR5);

I_MDR6 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(7), D1=>MDL6, S0=>L_LEFT, O=>MDR6);

I_MDR7 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(8), D1=>MDL7, S0=>L_LEFT, O=>MDR7);

I_MDR8 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(9), D1=>MDL8, S0=>L_LEFT, O=>MDR8);

I_MDR9 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(10), D1=>MDL9, S0=>L_LEFT, O=>MDR9);

I_MDR10 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(11), D1=>MDL10, S0=>L_LEFT, O=>MDR10);

I_MDR11 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(12), D1=>MDL11, S0=>L_LEFT, O=>MDR11);

I_MDR12 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(13), D1=>MDL12, S0=>L_LEFT, O=>MDR12);

I_MDR13 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(14), D1=>MDL13, S0=>L_LEFT, O=>MDR13);

I_MDR14 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(15), D1=>MDL14, S0=>L_LEFT, O=>MDR14);

I_MDR15 : M2_1_MXILINX_test
   port map (D0=>SRI, D1=>MDL15, S0=>L_LEFT, O=>MDR15);

I_Q0 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR0, Q=>Q_DUMMY(0));

I_Q1 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR1, Q=>Q_DUMMY(1));

I_Q2 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR2, Q=>Q_DUMMY(2));

I_Q3 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR3, Q=>Q_DUMMY(3));

I_Q4 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR4, Q=>Q_DUMMY(4));
```

```
   I_Q5 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR5, Q=>Q_DUMMY(5));

   I_Q6 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR6, Q=>Q_DUMMY(6));

   I_Q7 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR7, Q=>Q_DUMMY(7));

   I_Q8 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR8, Q=>Q_DUMMY(8));

   I_Q9 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR9, Q=>Q_DUMMY(9));

   I_Q10 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR10, Q=>Q_DUMMY(10));

   I_Q11 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR11, Q=>Q_DUMMY(11));

   I_Q12 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR12, Q=>Q_DUMMY(12));

   I_Q13 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR13, Q=>Q_DUMMY(13));

   I_Q14 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR14, Q=>Q_DUMMY(14));

   I_Q15 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR15, Q=>Q_DUMMY(15));

   I_36_145 : OR2
      port map (I0=>L, I1=>CE, O=>L_OR_CE);

   I_36_161 : OR2
      port map (I0=>LEFT, I1=>L, O=>L_LEFT);

end BEHAVIORAL;
```

InitShift_Test.vhd

```
  -- Created by Guri Kristine Birkeland 2005
```

```
 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
 use UNISIM.VComponents.all;


LIBRARY ieee; USE ieee.std_logic_1164.ALL; USE
ieee.std_logic_unsigned.all; USE ieee.numeric_std.ALL;

ENTITY init_shift_test_vhd IS END init_shift_test_vhd;

ARCHITECTURE behavior OF init_shift_test_vhd IS

    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT init_shiftregs
    PORT(
        clk : IN std_logic;
        enable : IN std_logic;
        load : IN std_logic;
        tin : IN std_logic_vector(7 downto 0);
        tout : OUT std_logic
        );
    END COMPONENT;

    --Inputs
    SIGNAL clk :  std_logic := '0';
    SIGNAL enable :  std_logic := '0';
    SIGNAL load :  std_logic := '0';
    SIGNAL tin :  std_logic_vector(7 downto 0) := (others=>'0');

    --Outputs
    SIGNAL tout :  std_logic;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: init_shiftregs PORT MAP(
        clk => clk,
        enable => enable,
        load => load,
        tin => tin,
        tout => tout
    );
```

```
PROCESS -- Process for clock
 BEGIN
     CLOCK_LOOP : LOOP
     clk <= transport '0';
     WAIT FOR 10 ns;
     clk <= transport '1';
     WAIT FOR 10 ns;
     WAIT FOR 40 ns;
     clk <= transport '0';
     WAIT FOR 40 ns;
     END LOOP CLOCK_LOOP;
 END PROCESS;

 tb : PROCESS
 BEGIN

     -- Wait 100 ns for global reset to finish
     wait for 100 ns;

     -- 1st
     -- --------------------
     enable <= transport '1';
     load <= transport '0';
     tin <= transport std_logic_vector'("00000001");
     -- --------------------
     WAIT FOR 100 ns; -- Time=100 ns
     load <= transport '1';
     -- --------------------
     WAIT FOR 100 ns; -- Time=200 ns
     load <= transport '0';
     -- --------------------
     --
     WAIT FOR 1400 ns; -- Time=1500 ns
     load <= transport '1';
     tin <= transport std_logic_vector'("00111111");
     -- --------------------
     WAIT FOR 100 ns;
     load <= transport '0';
     -- --------------------
     --
     WAIT FOR 1500 ns;
     load <= transport '1';
     tin <= transport std_logic_vector'("11111100");
     -- --------------------
     WAIT FOR 100 ns;
     load <= transport '0';
     -- --------------------
```

```
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("01111111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11111110");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11111100");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11111001");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("10011111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00111111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
```

```
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("01111111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11100111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00000011");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00000110");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("01100000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11000000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
```

```
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("10000000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00011000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00100100");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("01000010");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("10000001");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00000000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
```

```
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11100000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00000111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11110000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00001111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00011111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11111111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
```

```
        --
        WAIT FOR 1500 ns;
        load <= transport '1';
        tin <= transport std_logic_vector'("11111000");
        -- --------------------
        WAIT FOR 100 ns;
        load <= transport '0';
        -- --------------------
        --
        WAIT FOR 1500 ns;
        -- --------------------
        -- end waiting
        WAIT FOR 800 ns;
        -- --------------------

-- accumulated runtime is: 45700 ns

    END PROCESS;

END;
```

C.2.2. CSD2 design

The files needed to produce the CSD2 design of architecture 1 is listed below.

AddAndFlipFlop.vhdl

```
   -- Created by Guri Kristine Birkeland 2005

 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
 use UNISIM.VComponents.all;

entity add_ff is
   port ( clock   : in    std_logic;
          CE  : in    std_logic;
          CLR : in    std_logic;
          afi0 : in std_logic_vector(7 downto 0);
```

```
          afso : out std_logic);
end add_ff;

architecture BEHAVIORAL of add_ff is

component add16bit
    port ( A   : in    std_logic_vector (15 downto 0);
           B   : in    std_logic_vector (15 downto 0);
           CI  : in    std_logic;
           CO  : out   std_logic;
           OFL : out   std_logic;
           S   : out   std_logic_vector (15 downto 0));
end component;

component flipflop
   port ( C   : in    std_logic;
          CE  : in    std_logic;
          CLR : in    std_logic;
          D   : in    std_logic_vector (8 downto 0);
          Q   : out   std_logic_vector (7 downto 0);
        so : out std_logic);
end component;

signal outadder : std_logic_vector(8 downto 0); signal inadder :
std_logic_vector(7 downto 0);

signal dummy0 : std_logic; signal dummy1 : std_logic_vector(6
downto 0); signal dummy2 : std_logic;

begin

  adder : add16bit
    port map ( A(7 downto 0) => afi0, A(15 downto 8)=>"00000000",
               B(7 downto 0) => inadder, B(15 downto 8)=>"00000000",
               CI => '0', CO=> dummy0, OFL => dummy2, S(8 downto 0)=> outadder,
               S(15 downto 9)=> dummy1);

  ff : flipflop
    port map (C => clock, CE => CE, CLR => CLR, D => outadder, Q => inadder,
             so => afso);

 end BEHAVIORAL;
```

Arch1_CSD2.vhdl

```
   -- Created by Guri Kristine Birkeland 2005
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity ARCH1_CSD2 is
    Port ( clock : in std_logic;
         enable :  in std_logic;
         tin : in std_logic_vector(3 downto 0);
       tout : out std_logic
         );
end ARCH1_CSD2;

architecture Behavioral of ARCH1_CSD2 is

component lutsub0
    Port ( clock : in std_logic;
         enable :  in std_logic;
         ti : in std_logic_vector(3 downto 0);
       tout : out std_logic);
end component;

component add2
  port
  (
     a:  in std_logic;
     b:  in std_logic;
     cii : in std_logic;
     s: out std_logic;
     coo : out std_logic
  );
end component;

signal partsum0 : std_logic;

begin

LUTsub00 : lutsub0
   port map (clock=>clock, enable=>enable, ti=>tin(3 downto 0), tout=>partsum0);

tout<=partsum0;
```

```
 end Behavioral;
```

FlipFlop.vhdl

```
   -- Created by Guri Kristine Birkeland 2005

 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
 use UNISIM.VComponents.all;

entity flipflop is
   port ( C   : in    std_logic;
          CE  : in    std_logic;
          CLR : in    std_logic;
          D   : in    std_logic_vector (8 downto 0);
          Q   : out   std_logic_vector (7 downto 0);
        so : out std_logic);
end flipflop;

architecture BEHAVIORAL of flipflop is

   attribute INIT       : STRING ;
   attribute BOX_TYPE   : STRING ;
   component FDCE
      -- synopsys translate_off
      generic( INIT : bit :=  '0');
      -- synopsys translate_on
      port ( C   : in    std_logic;
             CE  : in    std_logic;
             CLR : in    std_logic;
             D   : in    std_logic;
             Q   : out   std_logic);
   end component;
   attribute INIT of FDCE : COMPONENT is "0";
   attribute BOX_TYPE of FDCE : COMPONENT is "BLACK_BOX";

begin

   I_Q0 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(0), Q=>so);
```

```
   I_Q1 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(1), Q=>Q(0));

   I_Q2 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(2), Q=>Q(1));

   I_Q3 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(3), Q=>Q(2));

   I_Q4 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(4), Q=>Q(3));

   I_Q5 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(5), Q=>Q(4));

   I_Q6 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(6), Q=>Q(5));

   I_Q7 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(7), Q=>Q(6));

   I_Q8 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(8), Q=>Q(7));

end BEHAVIORAL;
```

Init_ShiftRegisters.vhdl

```
   -- Created by Guri Kristine Birkeland 2005

 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
 use UNISIM.VComponents.all;

 entity INIT_SHIFTREGS is
     Port ( clk, enable, load : in std_logic;
          tin : in std_logic_vector(7 downto 0);
          tout : out std_logic
          );
```

```
end INIT_SHIFTREGS;

architecture Behavioral of INIT_SHIFTREGS is

component shiftreg16bit
   port ( C    : in    std_logic;        -- clock
          CE   : in    std_logic;        -- enable
          CLR  : in    std_logic;        -- clear
          D    : in    std_logic_vector (15 downto 0);  -- input
          L    : in    std_logic;        -- load
          LEFT : in    std_logic;        -- shift left
          SLI  : in    std_logic;        -- add to left 0/1
          SRI  : in    std_logic;        -- add to right 0/1
          Q    : out   std_logic_vector (15 downto 0));     -- output
end component;

component FDCE
     -- synopsys translate_off
     generic( INIT : bit :=  '0');
     -- synopsys translate_on
     port ( C   : in    std_logic;
            CE  : in    std_logic;
            CLR : in    std_logic;
            D   : in    std_logic;
            Q   : out   std_logic);
end component;

component ARCH1_CSD2
    Port ( clock : in std_logic;
         enable :  in std_logic;
         tin : in std_logic_vector(3 downto 0);
       tout : out std_logic
         );
end component;

signal shiftOut : std_logic_vector(3 downto 0);

signal out0 : std_logic_vector(14 downto 0);
signal out1 : std_logic_vector(14 downto 0);
signal out2 : std_logic_vector(14 downto 0);
signal out3 : std_logic_vector(14 downto 0);

signal Q0 : std_logic_vector(7 downto 0);
signal Q1 : std_logic_vector(7 downto 0);
signal Q2 : std_logic_vector(7 downto 0);

signal GND : std_logic := '0';
```

```
begin

shiftreg_0 : shiftreg16bit
   port map (C => clk, CE => enable, CLR => GND,
           D(7 downto 0) => "00000000",
           D(15 downto 8) => tin,
           L => load, LEFT => GND, SLI => GND, SRI => GND,
           Q(0) => shiftOut(0), Q(15 downto 1) => out0
           );

-- flipflops to hold the value for next shiftregister.
flipflop_00 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>shiftOut(0), Q=>Q0(7));

flipflop_01 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(7), Q=>Q0(6));

flipflop_02 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(6), Q=>Q0(5));

flipflop_03 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(5), Q=>Q0(4));

flipflop_04 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(4), Q=>Q0(3));

flipflop_05 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(3), Q=>Q0(2));

flipflop_06 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(2), Q=>Q0(1));

flipflop_07 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(1), Q=>Q0(0));


shiftreg_1 : shiftreg16bit
   port map (C => clk, CE => enable, CLR => GND,
           D(7 downto 0) => "00000000",
           D(15 downto 8) => Q0,
           L => load, LEFT => GND, SLI => GND, SRI => GND,
           Q(0) => shiftOut(1), Q(15 downto 1) => out1
           );
flipflop_10 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>shiftOut(1), Q=>Q1(7));

flipflop_11 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(7), Q=>Q1(6));
```

```
flipflop_12 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(6), Q=>Q1(5));

flipflop_13 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(5), Q=>Q1(4));

flipflop_14 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(4), Q=>Q1(3));

flipflop_15 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(3), Q=>Q1(2));

flipflop_16 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(2), Q=>Q1(1));

flipflop_17 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(1), Q=>Q1(0));


shiftreg_2 : shiftreg16bit
   port map (C => clk, CE => enable, CLR => GND,
           D(7 downto 0) => "00000000",
           D(15 downto 8) => Q1,
           L => load, LEFT => GND, SLI => GND, SRI => GND,
           Q(0) => shiftOut(2), Q(15 downto 1) => out2
           );
flipflop_20 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>shiftOut(2), Q=>Q2(7));

flipflop_21 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(7), Q=>Q2(6));

flipflop_22 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(6), Q=>Q2(5));

flipflop_23 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(5), Q=>Q2(4));

flipflop_24 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(4), Q=>Q2(3));

flipflop_25 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(3), Q=>Q2(2));

flipflop_26 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(2), Q=>Q2(1));
```

```
flipflop_27 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(1), Q=>Q2(0));


shiftreg_3 : shiftreg16bit
   port map (C => clk, CE => enable, CLR => GND,
           D(7 downto 0) => "00000000",
           D(15 downto 8) => Q2,
           L => load, LEFT => GND, SLI => GND, SRI => GND,
           Q(0) => shiftOut(3), Q(15 downto 1) => out3
           );

arch0 : ARCH1_CSD2
    port map (clock=>clk, enable=>enable,
                   tin=>shiftOut, tout=>tout
         );

end Behavioral;
```

Lut8Bits.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity lut is
    generic (              INIT0 : bit_vector := x"16";
               INIT1 : bit_vector := x"16";
               INIT2 : bit_vector := x"16";
               INIT3 : bit_vector := x"16";
               INIT4 : bit_vector := x"16";
               INIT5 : bit_vector := x"16";
               INIT6 : bit_vector := x"16";
               INIT7 : bit_vector := x"16");
   Port ( ui0 : in std_logic;
           ui1 : in std_logic;
           ui2 : in std_logic;
           ui3 : in std_logic;
```

```
           uo : out std_logic_vector(7 downto 0));
end lut;

architecture Behavioral of lut is
       -- Component Declaration for LUT4 should be placed
-- after architecture statement but before begin keyword
component LUT4
   generic (INIT : bit_vector := x"16");
   port (O : out STD_LOGIC;
           I0 : in STD_LOGIC;
           I1 : in STD_LOGIC;
           I2 : in STD_LOGIC;
           I3 : in STD_LOGIC);
end component;

-- Component Attribute specification for LUT4
-- should be placed after architecture declaration but
-- before the begin keyword
attribute INIT : string;
begin
LUT4_p0 : LUT4 generic map (INIT => INIT0)
   port map (O => uO(0),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p1 : LUT4 generic map (INIT => INIT1)
   port map (O => uO(1),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p2 : LUT4 generic map (INIT => INIT2)
   port map (O => uO(2),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p3 : LUT4 generic map (INIT => INIT3)
   port map (O => uO(3),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p4 : LUT4 generic map (INIT => INIT4)
   port map (O => uO(4),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p5 : LUT4 generic map (INIT => INIT5)
   port map (O => uO(5),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p6 : LUT4 generic map (INIT => INIT6)
   port map (O => uO(6),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p7 : LUT4 generic map (INIT => INIT7)
   port map (O => uO(7),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);

end Behavioral;
```

LutSub0.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
```

```
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity LUTsub0 is
    Port ( clock : in std_logic;
               enable :  in std_logic;
               ti : in std_logic_vector(3 downto 0);
           tout : out std_logic
               );
end LUTsub0;

architecture Behavioral of LUTsub0 is

component lut
    generic (
             INIT0 : bit_vector := x"16";
             INIT1 : bit_vector := x"16";
             INIT2 : bit_vector := x"16";
             INIT3 : bit_vector := x"16";
             INIT4 : bit_vector := x"16";
             INIT5 : bit_vector := x"16";
             INIT6 : bit_vector := x"16";
             INIT7 : bit_vector := x"16");
   port (
            UI0   : in    std_logic;
            UI1   : in    std_logic;
            UI2   : in    std_logic;
            UI3   : in    std_logic;
            UO    : out   std_logic_vector(7 downto 0)
       );
end component;

component SUB2
   PORT(  A:  IN STD_LOGIC;
          B:  IN STD_LOGIC;
          bin : in std_logic;
             Q:  OUT STD_LOGIC;
          bout :   OUT STD_LOGIC
       );
END component;

component add_ff
  port ( clock   : in    std_logic;
         CE  : in    std_logic;
```

```
         CLR : in    std_logic;
            afi0 : in std_logic_vector(7 downto 0);
            afso : out std_logic
            );
end component;

signal poslutout : std_logic_vector(7 downto 0);
signal neglutout : std_logic_vector(7 downto 0);

signal posffout : std_logic;
signal negffout : std_logic;

signal sign0 : std_logic;
signal bin0 : std_logic;
signal diff : std_logic;

begin

-- lut
poslut_0 : lut
   generic map (
            INIT0=>"0000000000001010",
            INIT1=>"0000000010000000",
            INIT2=>"0001001000001000",
            INIT3=>"1000000100000100",
            INIT4=>"0000010000010010",
            INIT5=>"1100000000000000",
            INIT6=>"0000000000010000",
            INIT7=>"0000001100101100")
   port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),
             UO=>poslutout);

neglut_0 : lut
   generic map (
            INIT0=>"0000101000100000",
            INIT1=>"0000010101010000",
            INIT2=>"0100100000100010",
            INIT3=>"0000000001000000",
            INIT4=>"0001001000001000",
            INIT5=>"0000100100100100",
            INIT6=>"0011010011000000",
            INIT7=>"0000000000000000")
   port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),
             UO=>neglutout);

 add_ff_0: add_ff
   port map (clock, enable, '0', poslutout, posffout);
```

```
 add_ff_1: add_ff
   port map (clock, enable, '0', neglutout, negffout);

 -- sub neg from pos
 sub1bit_0 : SUB2
       port map (A=>posffout, B=>negffout, bin=>bin0, Q=>diff, bout=>sign0);

 -- store carry signal
 flip_flop_0 : FDCE
       port map (C=>clock, CE=>enable, CLR=>'0', D=>sign0, Q=>bin0);

 -- store diff
 flip_flop_1 : FDCE
       port map (C=>clock, CE=>enable, CLR=>'0', D=>diff, Q=>tout);

end Behavioral;
```

ShiftReg16.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity M2_1_MXILINX_test is
   port ( D0 : in    std_logic;
          D1 : in    std_logic;
          S0 : in    std_logic;
          O  : out   std_logic);
end M2_1_MXILINX_test;

architecture BEHAVIORAL of M2_1_MXILINX_test is
   attribute BOX_TYPE   : STRING ;
   signal M0 : std_logic;
   signal M1 : std_logic;
   component AND2B1
      port ( I0 : in    std_logic;
             I1 : in    std_logic;
             O  : out   std_logic);
```

```
   end component;
   attribute BOX_TYPE of AND2B1 : COMPONENT is "BLACK_BOX";

   component OR2
      port ( I0 : in    std_logic;
             I1 : in    std_logic;
             O  : out   std_logic);
   end component;
   attribute BOX_TYPE of OR2 : COMPONENT is "BLACK_BOX";

   component AND2
      port ( I0 : in    std_logic;
             I1 : in    std_logic;
             O  : out   std_logic);
   end component;
   attribute BOX_TYPE of AND2 : COMPONENT is "BLACK_BOX";

begin
   I_36_7 : AND2B1
      port map (I0=>S0, I1=>D0, O=>M0);

   I_36_8 : OR2
      port map (I0=>M1, I1=>M0, O=>O);

   I_36_9 : AND2
      port map (I0=>D1, I1=>S0, O=>M1);

end BEHAVIORAL;

  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity shiftreg16bit is
   port ( C    : in    std_logic;   -- clock
          CE   : in    std_logic;   -- enable
          CLR  : in    std_logic;   -- clear
          D    : in    std_logic_vector (15 downto 0);   -- input
          L    : in    std_logic;   -- load
          LEFT : in    std_logic;   -- shift left
```

```
          SLI  : in    std_logic;   -- add to left 0/1
          SRI  : in    std_logic;   -- add to right 0/1
          Q    : out   std_logic_vector (15 downto 0));  -- output
end shiftreg16bit;

architecture BEHAVIORAL of shiftreg16bit is
   attribute HU_SET     : STRING ;
   attribute INIT       : STRING ;
   attribute BOX_TYPE   : STRING ;
   signal L_LEFT  : std_logic;
   signal L_OR_CE : std_logic;
   signal MDL0    : std_logic;
   signal MDL1    : std_logic;
   signal MDL2    : std_logic;
   signal MDL3    : std_logic;
   signal MDL4    : std_logic;
   signal MDL5    : std_logic;
   signal MDL6    : std_logic;
   signal MDL7    : std_logic;
   signal MDL8    : std_logic;
   signal MDL9    : std_logic;
   signal MDL10    : std_logic;
   signal MDL11    : std_logic;
   signal MDL12    : std_logic;
   signal MDL13    : std_logic;
   signal MDL14    : std_logic;
   signal MDL15    : std_logic;
   signal MDR0    : std_logic;
   signal MDR1    : std_logic;
   signal MDR2    : std_logic;
   signal MDR3    : std_logic;
   signal MDR4    : std_logic;
   signal MDR5    : std_logic;
   signal MDR6    : std_logic;
   signal MDR7    : std_logic;
   signal MDR8    : std_logic;
   signal MDR9    : std_logic;
   signal MDR10    : std_logic;
   signal MDR11    : std_logic;
   signal MDR12    : std_logic;
   signal MDR13    : std_logic;
   signal MDR14    : std_logic;
   signal MDR15    : std_logic;
   signal Q_DUMMY : std_logic_vector (15 downto 0);
   component M2_1_MXILINX_test
      port ( D0 : in    std_logic;
             D1 : in    std_logic;
             S0 : in    std_logic;
```

```
            O  : out   std_logic);
    end component;

    component FDCE
       -- synopsys translate_off
       generic( INIT : bit :=  '0');
       -- synopsys translate_on
       port ( C   : in    std_logic;
              CE  : in    std_logic;
              CLR : in    std_logic;
              D   : in    std_logic;
              Q   : out   std_logic);
    end component;
    attribute INIT of FDCE : COMPONENT is "0";
    attribute BOX_TYPE of FDCE : COMPONENT is "BLACK_BOX";

    component OR2
       port ( I0 : in    std_logic;
              I1 : in    std_logic;
              O  : out   std_logic);
    end component;
    attribute BOX_TYPE of OR2 : COMPONENT is "BLACK_BOX";

begin
    Q(15 downto 0) <= Q_DUMMY(15 downto 0);
    I_MDL0 : M2_1_MXILINX_test
       port map (D0=>SLI, D1=>D(0), S0=>L, O=>MDL0);

    I_MDL1 : M2_1_MXILINX_test
       port map (D0=>Q_DUMMY(0), D1=>D(1), S0=>L, O=>MDL1);

    I_MDL2 : M2_1_MXILINX_test
       port map (D0=>Q_DUMMY(1), D1=>D(2), S0=>L, O=>MDL2);

    I_MDL3 : M2_1_MXILINX_test
       port map (D0=>Q_DUMMY(2), D1=>D(3), S0=>L, O=>MDL3);

    I_MDL4 : M2_1_MXILINX_test
       port map (D0=>Q_DUMMY(3), D1=>D(4), S0=>L, O=>MDL4);

    I_MDL5 : M2_1_MXILINX_test
       port map (D0=>Q_DUMMY(4), D1=>D(5), S0=>L, O=>MDL5);

    I_MDL6 : M2_1_MXILINX_test
       port map (D0=>Q_DUMMY(5), D1=>D(6), S0=>L, O=>MDL6);

    I_MDL7 : M2_1_MXILINX_test
       port map (D0=>Q_DUMMY(6), D1=>D(7), S0=>L, O=>MDL7);
```

```
I_MDL8 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(7), D1=>D(8), S0=>L, O=>MDL8);

I_MDL9 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(8), D1=>D(9), S0=>L, O=>MDL9);

I_MDL10 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(9), D1=>D(10), S0=>L, O=>MDL10);

I_MDL11 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(10), D1=>D(11), S0=>L, O=>MDL11);

I_MDL12 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(11), D1=>D(12), S0=>L, O=>MDL12);

I_MDL13 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(12), D1=>D(13), S0=>L, O=>MDL13);

I_MDL14 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(13), D1=>D(14), S0=>L, O=>MDL14);

I_MDL15 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(14), D1=>D(15), S0=>L, O=>MDL15);

I_MDR0 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(1), D1=>MDL0, S0=>L_LEFT, O=>MDR0);

I_MDR1 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(2), D1=>MDL1, S0=>L_LEFT, O=>MDR1);

I_MDR2 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(3), D1=>MDL2, S0=>L_LEFT, O=>MDR2);

I_MDR3 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(4), D1=>MDL3, S0=>L_LEFT, O=>MDR3);

I_MDR4 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(5), D1=>MDL4, S0=>L_LEFT, O=>MDR4);

I_MDR5 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(6), D1=>MDL5, S0=>L_LEFT, O=>MDR5);

I_MDR6 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(7), D1=>MDL6, S0=>L_LEFT, O=>MDR6);

I_MDR7 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(8), D1=>MDL7, S0=>L_LEFT, O=>MDR7);
```

```
I_MDR8 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(9), D1=>MDL8, S0=>L_LEFT, O=>MDR8);

I_MDR9 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(10), D1=>MDL9, S0=>L_LEFT, O=>MDR9);

I_MDR10 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(11), D1=>MDL10, S0=>L_LEFT, O=>MDR10);

I_MDR11 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(12), D1=>MDL11, S0=>L_LEFT, O=>MDR11);

I_MDR12 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(13), D1=>MDL12, S0=>L_LEFT, O=>MDR12);

I_MDR13 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(14), D1=>MDL13, S0=>L_LEFT, O=>MDR13);

I_MDR14 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(15), D1=>MDL14, S0=>L_LEFT, O=>MDR14);

I_MDR15 : M2_1_MXILINX_test
   port map (D0=>SRI, D1=>MDL15, S0=>L_LEFT, O=>MDR15);

I_Q0 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR0, Q=>Q_DUMMY(0));

I_Q1 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR1, Q=>Q_DUMMY(1));

I_Q2 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR2, Q=>Q_DUMMY(2));

I_Q3 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR3, Q=>Q_DUMMY(3));

I_Q4 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR4, Q=>Q_DUMMY(4));

I_Q5 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR5, Q=>Q_DUMMY(5));

I_Q6 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR6, Q=>Q_DUMMY(6));

I_Q7 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR7, Q=>Q_DUMMY(7));
```

```
   I_Q8 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR8, Q=>Q_DUMMY(8));

   I_Q9 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR9, Q=>Q_DUMMY(9));

   I_Q10 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR10, Q=>Q_DUMMY(10));

   I_Q11 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR11, Q=>Q_DUMMY(11));

   I_Q12 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR12, Q=>Q_DUMMY(12));

   I_Q13 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR13, Q=>Q_DUMMY(13));

   I_Q14 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR14, Q=>Q_DUMMY(14));

   I_Q15 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR15, Q=>Q_DUMMY(15));

   I_36_145 : OR2
      port map (I0=>L, I1=>CE, O=>L_OR_CE);

   I_36_161 : OR2
      port map (I0=>LEFT, I1=>L, O=>L_LEFT);

end BEHAVIORAL;
```

InitShift_Test.vhd

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;
```

```
LIBRARY ieee; USE ieee.std_logic_1164.ALL; USE
ieee.std_logic_unsigned.all; USE ieee.numeric_std.ALL;

ENTITY init_shift_test_vhd IS END init_shift_test_vhd;

ARCHITECTURE behavior OF init_shift_test_vhd IS

    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT init_shiftregs
    PORT(
        clk : IN std_logic;
        enable : IN std_logic;
        load : IN std_logic;
        tin : IN std_logic_vector(7 downto 0);
        tout : OUT std_logic
        );
    END COMPONENT;

    --Inputs
    SIGNAL clk :  std_logic := '0';
    SIGNAL enable :  std_logic := '0';
    SIGNAL load :  std_logic := '0';
    SIGNAL tin :  std_logic_vector(7 downto 0) := (others=>'0');

    --Outputs
    SIGNAL tout :  std_logic;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: init_shiftregs PORT MAP(
        clk => clk,
        enable => enable,
        load => load,
        tin => tin,
        tout => tout
    );

   PROCESS -- Process for clock
    BEGIN
        CLOCK_LOOP : LOOP
        clk <= transport '0';
        WAIT FOR 10 ns;
        clk <= transport '1';
        WAIT FOR 10 ns;
        WAIT FOR 40 ns;
```

```
    clk <= transport '0';
    WAIT FOR 40 ns;
    END LOOP CLOCK_LOOP;
END PROCESS;

tb : PROCESS
BEGIN

    -- Wait 100 ns for global reset to finish
    wait for 100 ns;

    -- 1st
    -- --------------------
    enable <= transport '1';
    load <= transport '0';
    tin <= transport std_logic_vector'("00000001");
    -- --------------------
    WAIT FOR 100 ns; -- Time=100 ns
    load <= transport '1';
    -- --------------------
    WAIT FOR 100 ns; -- Time=200 ns
    load <= transport '0';
    -- --------------------
    --
    WAIT FOR 1400 ns; -- Time=1500 ns
    load <= transport '1';
    tin <= transport std_logic_vector'("00111111");
    -- --------------------
    WAIT FOR 100 ns;
    load <= transport '0';
    -- --------------------
    --
    WAIT FOR 1500 ns;
    load <= transport '1';
    tin <= transport std_logic_vector'("11111100");
    -- --------------------
    WAIT FOR 100 ns;
    load <= transport '0';
    -- --------------------
    --
    WAIT FOR 1500 ns;
    load <= transport '1';
    tin <= transport std_logic_vector'("01111111");
    -- --------------------
    WAIT FOR 100 ns;
    load <= transport '0';
    -- --------------------
    --
```

```
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11111110");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11111100");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11111001");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("10011111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00111111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("01111111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
```

```
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11100111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00000011");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00000110");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("01100000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11000000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("10000000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
```

```
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00011000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00100100");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("01000010");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("10000001");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00000000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11100000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
```

```
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00000111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11110000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00001111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00011111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11111111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11111000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
```

```
        WAIT FOR 1500 ns;
        -- --------------------
        -- end waiting
        WAIT FOR 1000 ns;
        -- --------------------

-- accumulated runtime is: 45900 ns

    END PROCESS;

END;
```

C.2.3. CSD4 design

The files needed to produce the CSD4 design of architecture 1 is listed below.

Add5And7.vhdl

```
   -- Created by Guri Kristine Birkeland 2005

 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
 use UNISIM.VComponents.all;

 entity add5and7 is
     Port ( clock : in std_logic;
                enable :  in std_logic;
                tin : in std_logic_vector(3 downto 0);
            tout : out std_logic
                );
 end add5and7;

 architecture Behavioral of add5and7 is

 component ARCH1_CSD4_5
     Port ( clock : in std_logic;
                enable :  in std_logic;
                tin : in std_logic_vector(3 downto 0);
```

```
           tout : out std_logic
               );
end component;

component ARCH1_CSD4_7
    Port ( clock : in std_logic;
               enable :  in std_logic;
               tin : in std_logic_vector(3 downto 0);
           tout : out std_logic
               );
end component;

component SUB2
   port (
     a:  in std_logic;
     b:  in std_logic;
     bin : in std_logic;
     q: out std_logic;
     bout : out std_logic
   );
end component;

component ADD2
  port
  (
     a:  in std_logic;
     b:  in std_logic;
     cii : in std_logic;
     s: out std_logic;
     coo : out std_logic
  );
end component;

signal f_out0: std_logic;
signal f_out1: std_logic;
signal f_out2: std_logic;
signal f_out3: std_logic;
signal out_f0: std_logic;
signal out_f1: std_logic;

signal s_out0: std_logic;
signal s_out1: std_logic;
signal s_out2: std_logic;
signal s_out3: std_logic;
signal out_s0: std_logic;
signal out_s1: std_logic;

signal cin0 : std_logic;
```

```
 signal sign0 : std_logic;
 signal cin_f : std_logic;
 signal sign_f : std_logic;
 signal cin_s : std_logic;
 signal sign_s : std_logic;

 signal total : std_logic;


begin

-- about number 5
-------------------------------------------------------------------------------------------------
addf_0 : ARCH1_CSD4_5
    port map(clock, enable, tin, f_out0);

flip_flop_f0 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>f_out0, Q=>f_out1);

flip_flop_f1 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>f_out1, Q=>f_out2);

flip_flop_f2 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>f_out2, Q=>f_out3);

add_f : ADD2
    port map(f_out1, f_out3, cin_f, out_f0, sign_f);

flip_flop_f3 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign_f, Q=>cin_f);

flip_flop_f4 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>out_f0, Q=>out_f1);

-- about number 7
-------------------------------------------------------------------------------------------------
adds_0 : ARCH1_CSD4_7
    port map(clock, enable, tin, s_out0);

flip_flop_s0 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>s_out0, Q=>s_out1);

flip_flop_s1 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>s_out1, Q=>s_out2);

flip_flop_s2 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>s_out2, Q=>s_out3);
```

```
sub_s : SUB2
    port map(s_out3, s_out0, cin_s, out_s0, sign_s);

flip_flop_s3 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign_s, Q=>cin_s);

flip_flop_s4 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>out_s0, Q=>out_s1);

-- end adding

add_0 : ADD2
    port map(out_f1, out_s1, cin0, total, sign0);

flip_flop_0 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign0, Q=>cin0);

flip_flop_1 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>total, Q=>tout);

end Behavioral;
```

Add9And11.vhdl

```
   -- Created by Guri Kristine Birkeland 2005

 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
 use UNISIM.VComponents.all;

 entity add9and11 is
     Port ( clock : in std_logic;
                enable :  in std_logic;
                tin : in std_logic_vector(3 downto 0);
            tout : out std_logic
                );
 end add9and11;

 architecture Behavioral of add9and11 is
```

```
component ARCH1_CSD4_9
    Port ( clock : in std_logic;
                enable :  in std_logic;
                tin : in std_logic_vector(3 downto 0);
           tout : out std_logic
                );
end component;

component ARCH1_CSD4_11
    Port ( clock : in std_logic;
                enable :  in std_logic;
                tin : in std_logic_vector(3 downto 0);
           tout : out std_logic
                );
end component;

component SUB2
   port (
     a:  in std_logic;
     b:  in std_logic;
     bin : in std_logic;
     q: out std_logic;
     bout : out std_logic
   );
end component;

component ADD2
  port
  (
     a:  in std_logic;
     b:  in std_logic;
     cii : in std_logic;
     s: out std_logic;
     coo : out std_logic
  );
end component;

signal f_out0: std_logic;
signal f_out1: std_logic;
signal f_out2: std_logic;
signal f_out3: std_logic;
signal out_f0: std_logic;
signal out_f1: std_logic;
signal out_f2: std_logic;

signal s_out0: std_logic;
signal s_out1: std_logic;
signal s_out2: std_logic;
```

```
 signal s_out3: std_logic;
 signal s_out4: std_logic;
 signal out_s_00: std_logic;
 signal out_s_01: std_logic;
 signal out_s0: std_logic;
 signal out_s1: std_logic;

 signal cin0 : std_logic;
 signal sign0 : std_logic;
 signal cin_f : std_logic;
 signal sign_f : std_logic;
 signal cin_s_0: std_logic;
 signal sign_s_0 : std_logic;
 signal cin_s : std_logic;
 signal sign_s : std_logic;

 signal total : std_logic;


begin

-- about number 9
--------------------------------
addf_0 : ARCH1_CSD4_9
    port map(clock, enable, tin, f_out0);

flip_flop_f0 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>f_out0, Q=>f_out1);

flip_flop_f1 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>f_out1, Q=>f_out2);

flip_flop_f2 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>f_out2, Q=>f_out3);

add_f : ADD2
    port map(f_out0, f_out3, cin_f, out_f0, sign_f);

flip_flop_f3 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign_f, Q=>cin_f);

flip_flop_f4 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>out_f0, Q=>out_f1);

flip_flop_f5 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>out_f1, Q=>out_f2);
```

```
-- about number 11
----------------------------------
adds_0 : ARCH1_CSD4_11
    port map(clock, enable, tin, s_out0);

flip_flop_s0 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>s_out0, Q=>s_out1);

add_s_0 : ADD2
    port map(s_out0, s_out1, cin_s_0, out_s_00, sign_s_0);

flip_flop_s1 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign_s_0, Q=>cin_s_0);

flip_flop_s2 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>out_s_00, Q=>out_s_01);



flip_flop_s3 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>s_out1, Q=>s_out2);

flip_flop_s4 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>s_out2, Q=>s_out3);

flip_flop_s5 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>s_out3, Q=>s_out4);

add_s : ADD2
    port map(out_s_01, s_out4, cin_s, out_s0, sign_s);

flip_flop_s6 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign_s, Q=>cin_s);

flip_flop_s7 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>out_s0, Q=>out_s1);

-- end adding

add_0 : ADD2
    port map(out_f2, out_s1, cin0, total, sign0);

flip_flop_0 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign0, Q=>cin0);

flip_flop_1 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>total, Q=>tout);

end Behavioral;
```

Add13And15.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity add13and15 is
    Port ( clock : in std_logic;
                enable :  in std_logic;
                tin : in std_logic_vector(3 downto 0);
           tout : out std_logic
                );
end add13and15;

architecture Behavioral of add13and15 is

component ARCH1_CSD4_13
    Port ( clock : in std_logic;
                enable :  in std_logic;
                tin : in std_logic_vector(3 downto 0);
           tout : out std_logic
                );
end component;

component ARCH1_CSD4_15
    Port ( clock : in std_logic;
                enable :  in std_logic;
                tin : in std_logic_vector(3 downto 0);
           tout : out std_logic
                );
end component;

component SUB2
   port (
     a:  in std_logic;
     b:  in std_logic;
     bin : in std_logic;
     q: out std_logic;
     bout : out std_logic
```

```
    );
 end component;

 component ADD2
   port
   (
      a:  in std_logic;
      b:  in std_logic;
      cii : in std_logic;
      s: out std_logic;
      coo : out std_logic
   );
 end component;

 signal f_out0: std_logic;
 signal f_out1: std_logic;
 signal f_out2: std_logic;
 signal f_out3: std_logic;
 signal f_out4: std_logic;
 signal out_f_00: std_logic;
 signal out_f_01: std_logic;
 signal out_f0: std_logic;
 signal out_f1: std_logic;

 signal s_out0: std_logic;
 signal s_out1: std_logic;
 signal s_out2: std_logic;
 signal s_out3: std_logic;
 signal s_out4: std_logic;
 signal out_s0: std_logic;
 signal out_s1: std_logic;

 signal cin0 : std_logic;
 signal sign0 : std_logic;
 signal cin_f_0: std_logic;
 signal sign_f_0 : std_logic;
 signal cin_f : std_logic;
 signal sign_f : std_logic;
 signal cin_s : std_logic;
 signal sign_s : std_logic;

 signal total : std_logic;


begin

-- about number 13
----------------------------------------
```

```
addf_0 : ARCH1_CSD4_13
    port map(clock, enable, tin, f_out0);

flip_flop_f0 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>f_out0, Q=>f_out1);

flip_flop_f1 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>f_out1, Q=>f_out2);

add_f_0 : ADD2
    port map(f_out0, f_out2, cin_f_0, out_f_00, sign_f_0);

flip_flop_f2 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign_f_0, Q=>cin_f_0);

flip_flop_f3 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>out_f_00, Q=>out_f_01);

flip_flop_f4 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>f_out2, Q=>f_out3);

flip_flop_f5 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>f_out3, Q=>f_out4);

add_f_1 : ADD2
    port map(out_f_01, f_out4, cin_f, out_f0, sign_f);

flip_flop_f6 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign_f, Q=>cin_f);

flip_flop_f7 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>out_f0, Q=>out_f1);

-- about number 15
----------------------------------------
adds_0 : ARCH1_CSD4_15
    port map(clock, enable, tin, s_out0);

flip_flop_s0 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>s_out0, Q=>s_out1);

flip_flop_s1 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>s_out1, Q=>s_out2);

flip_flop_s2 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>s_out2, Q=>s_out3);

flip_flop_s3 : FDCE
```

```
    port map (C=>clock, CE=>enable, CLR=>'0', D=>s_out3, Q=>s_out4);

sub_s : SUB2
    port map(s_out4, s_out0, cin_s, out_s0, sign_s);

flip_flop_s4 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign_s, Q=>cin_s);

flip_flop_s5 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>out_s0, Q=>out_s1);


-- end adding add_0 : ADD2
    port map(out_f1, out_s1, cin0, total, sign0);

flip_flop_0 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign0, Q=>cin0);

flip_flop_1 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>total, Q=>tout);

end Behavioral;
```

AddAndFlipFlop.vhdl

```
   -- Created by Guri Kristine Birkeland 2005

 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
 use UNISIM.VComponents.all;

entity add_ff is
   port ( clock   : in    std_logic;
          CE  : in    std_logic;
          CLR : in    std_logic;
            afi0 : in std_logic_vector(5 downto 0);
        afso : out std_logic);
end add_ff;

architecture BEHAVIORAL of add_ff is
```

```
component add16bit
    port ( A   : in    std_logic_vector (15 downto 0);
           B   : in    std_logic_vector (15 downto 0);
           CI  : in    std_logic;
           CO  : out   std_logic;
           OFL : out   std_logic;
           S   : out   std_logic_vector (15 downto 0));
end component;

component flipflop
   port ( C   : in    std_logic;
          CE  : in    std_logic;
          CLR : in    std_logic;
          D   : in    std_logic_vector (6 downto 0);
          Q   : out   std_logic_vector (5 downto 0);
        so : out std_logic);
end component;

signal outadder : std_logic_vector(6 downto 0); signal inadder :
std_logic_vector(5 downto 0);

signal dummy0 : std_logic; signal dummy1 : std_logic_vector(8
downto 0); signal dummy2 : std_logic;

begin

  adder : add16bit
    port map ( A(5 downto 0) => afi0, A(15 downto 6)=>"0000000000",
              B(5 downto 0) => inadder, B(15 downto 6)=>"0000000000",
              CI => '0', CO=> dummy0, OFL => dummy2, S(6 downto 0)=> outadder,
              S(15 downto 7)=> dummy1);

  ff : flipflop
    port map (C => clock, CE => CE, CLR => CLR, D => outadder, Q => inadder,
             so => afso);

 end BEHAVIORAL;
```

ARCH1_CSD4.vhdl

```
   -- Created by Guri Kristine Birkeland 2005

 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
```

```
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
 use UNISIM.VComponents.all;

entity ARCH1_CSD4 is
     Port ( clock : in std_logic;
          enable :  in std_logic;
          tin : in std_logic_vector(3 downto 0);
        tout : out std_logic
          );
end ARCH1_CSD4;

architecture Behavioral of ARCH1_CSD4 is

 component add5and7
     Port ( clock : in std_logic;
                 enable :  in std_logic;
                 tin : in std_logic_vector(3 downto 0);
            tout : out std_logic
                 );
 end component;

 component add9and11
     Port ( clock : in std_logic;
                 enable :  in std_logic;
                 tin : in std_logic_vector(3 downto 0);
            tout : out std_logic
                 );
 end component;

 component add13and15
     Port ( clock : in std_logic;
                 enable :  in std_logic;
                 tin : in std_logic_vector(3 downto 0);
            tout : out std_logic
                 );
 end component;

 component ADD2
    port (
      a:  in std_logic;
      b:  in std_logic;
      cii : in std_logic;
      s: out std_logic;
      coo : out std_logic
```

```
    );
 end component;

signal partsum0: std_logic; signal partsum01: std_logic; signal
partsum1: std_logic; signal partsum2: std_logic; signal partsum21:
std_logic;

signal sum0: std_logic; signal sum01: std_logic;

signal cin0, cin1 : std_logic; signal sign0, sign1 : std_logic;
signal total : std_logic;

begin

addand_0 : add5and7
    port map(clock, enable, tin, partsum0);

addand_1 : add9and11
    port map(clock, enable, tin, partsum1);

addand_2 : add13and15
    port map(clock, enable, tin, partsum2);

flip_flop_00 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>partsum0, Q=>partsum01);

add_0 : ADD2
    port map( partsum01, partsum1, cin0, sum0, sign0);

flip_flop_0 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign0, Q=>cin0);

flip_flop_01 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sum0, Q=>sum01);

flip_flop_02 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>partsum2, Q=>partsum21);

add_1 : ADD2
    port map( partsum21, sum01, cin1, total, sign1);

flip_flop_1 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign1, Q=>cin1);

flip_flop_2 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>total, Q=>tout);

end Behavioral;
```

Arch1_CSD4_5.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity ARCH1_CSD4_5 is
    Port ( clock : in std_logic;
         enable :  in std_logic;
         tin : in std_logic_vector(3 downto 0);
       tout : out std_logic
         );
end ARCH1_CSD4_5;

architecture Behavioral of ARCH1_CSD4_5 is

component lutsub_50
    Port ( clock : in std_logic;
         enable :  in std_logic;
         ti : in std_logic_vector(3 downto 0);
       tout : out std_logic);
end component;

component add2
  port
  (
     a:  in std_logic;
     b:  in std_logic;
     cii : in std_logic;
     s: out std_logic;
     coo : out std_logic
  );
end component;

signal partsum0 : std_logic;

begin

LUTsub00 : lutsub_50
```

```
   port map (clock=>clock, enable=>enable, ti=>tin(3 downto 0), tout=>partsum0);

tout<=partsum0;

end Behavioral;
```

Arch1_CSD4_7.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity ARCH1_CSD4_7 is
    Port ( clock : in std_logic;
         enable :  in std_logic;
         tin : in std_logic_vector(3 downto 0);
       tout : out std_logic
         );
end ARCH1_CSD4_7;

architecture Behavioral of ARCH1_CSD4_7 is

component lutsub_70
    Port ( clock : in std_logic;
         enable :  in std_logic;
         ti : in std_logic_vector(3 downto 0);
       tout : out std_logic);
end component;

component add2
  port
  (
     a:  in std_logic;
     b:  in std_logic;
     cii : in std_logic;
     s: out std_logic;
     coo : out std_logic
  );
```

```
end component;

signal partsum0 : std_logic;

begin

LUTsub00 : lutsub_70
   port map (clock=>clock, enable=>enable, ti=>tin(3 downto 0), tout=>partsum0);

tout<=partsum0;

end Behavioral;
```

Arch1_CSD4_9.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity ARCH1_CSD4_9 is
    Port ( clock : in std_logic;
         enable :  in std_logic;
         tin : in std_logic_vector(3 downto 0);
       tout : out std_logic
         );
end ARCH1_CSD4_9;

architecture Behavioral of ARCH1_CSD4_9 is

component lutsub_90
    Port ( clock : in std_logic;
         enable :  in std_logic;
         ti : in std_logic_vector(3 downto 0);
       tout : out std_logic);
end component;

component add2
  port
```

```
  (
     a:  in std_logic;
     b:  in std_logic;
     cii : in std_logic;
     s: out std_logic;
     coo : out std_logic
  );
end component;

signal partsum0 : std_logic;

begin

LUTsub00 : lutsub_90
   port map (clock=>clock, enable=>enable, ti=>tin(3 downto 0), tout=>partsum0);

tout<=partsum0;

end Behavioral;
```

Arch1_CSD4_11.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity ARCH1_CSD4_11 is
    Port ( clock : in std_logic;
         enable :  in std_logic;
         tin : in std_logic_vector(3 downto 0);
       tout : out std_logic
         );
end ARCH1_CSD4_11;

architecture Behavioral of ARCH1_CSD4_11 is

component lutsub_110
    Port ( clock : in std_logic;
```

```
         enable :  in std_logic;
         ti : in std_logic_vector(3 downto 0);
       tout : out std_logic);
end component;

component add2
  port
  (
     a:  in std_logic;
     b:  in std_logic;
     cii : in std_logic;
     s: out std_logic;
     coo : out std_logic
  );
end component;

signal partsum0 : std_logic;

begin

LUTsub00 : lutsub_110
   port map (clock=>clock, enable=>enable, ti=>tin(3 downto 0), tout=>partsum0);

tout<=partsum0;

end Behavioral;
```

Arch1_CSD4_13.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity ARCH1_CSD4_13 is
    Port ( clock : in std_logic;
         enable :  in std_logic;
         tin : in std_logic_vector(3 downto 0);
       tout : out std_logic
```

```
        );
end ARCH1_CSD4_13;

architecture Behavioral of ARCH1_CSD4_13 is

component lutsub_130
    Port ( clock : in std_logic;
         enable :  in std_logic;
         ti : in std_logic_vector(3 downto 0);
       tout : out std_logic);
end component;

component add2
  port
  (
     a:  in std_logic;
     b:  in std_logic;
     cii : in std_logic;
     s: out std_logic;
     coo : out std_logic
  );
end component;

signal partsum0 : std_logic;

begin

LUTsub00 : lutsub_130
   port map (clock=>clock, enable=>enable, ti=>tin(3 downto 0), tout=>partsum0);

tout<=partsum0;

end Behavioral;
```

Arch1_CSD4_15.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
```

```
use UNISIM.VComponents.all;

entity ARCH1_CSD4_15 is
    Port ( clock : in std_logic;
         enable :  in std_logic;
         tin : in std_logic_vector(3 downto 0);
       tout : out std_logic
         );
end ARCH1_CSD4_15;

architecture Behavioral of ARCH1_CSD4_15 is

component lutsub_150
    Port ( clock : in std_logic;
         enable :  in std_logic;
         ti : in std_logic_vector(3 downto 0);
       tout : out std_logic);
end component;

component add2
  port
  (
     a:  in std_logic;
     b:  in std_logic;
     cii : in std_logic;
     s: out std_logic;
     coo : out std_logic
  );
end component;

signal partsum0 : std_logic;

begin

LUTsub00 : lutsub_150
   port map (clock=>clock, enable=>enable, ti=>tin(3 downto 0), tout=>partsum0);

tout<=partsum0;

end Behavioral;
```

FlipFlop.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
```

```
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
 use UNISIM.VComponents.all;

entity flipflop is
   port ( C   : in    std_logic;
          CE  : in    std_logic;
          CLR : in    std_logic;
          D   : in    std_logic_vector (6 downto 0);
          Q   : out   std_logic_vector (5 downto 0);
        so : out std_logic);
end flipflop;

architecture BEHAVIORAL of flipflop is

   attribute INIT       : STRING ;
   attribute BOX_TYPE   : STRING ;
   component FDCE
      -- synopsys translate_off
      generic( INIT : bit :=  '0');
      -- synopsys translate_on
      port ( C   : in    std_logic;
             CE  : in    std_logic;
             CLR : in    std_logic;
             D   : in    std_logic;
             Q   : out   std_logic);
   end component;
   attribute INIT of FDCE : COMPONENT is "0";
   attribute BOX_TYPE of FDCE : COMPONENT is "BLACK_BOX";

begin

   I_Q0 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(0), Q=>so);

   I_Q1 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(1), Q=>Q(0));

   I_Q2 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(2), Q=>Q(1));

   I_Q3 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(3), Q=>Q(2));
```

```
   I_Q4 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(4), Q=>Q(3));

   I_Q5 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(5), Q=>Q(4));

   I_Q6 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(6), Q=>Q(5));

end BEHAVIORAL;
```

Init_ShiftRegisters.vhdl

```
   -- Created by Guri Kristine Birkeland 2005

 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
 use UNISIM.VComponents.all;

 entity INIT_SHIFTREGS is
     Port ( clk, enable, load : in std_logic;
          tin : in std_logic_vector(7 downto 0);
          tout : out std_logic
          );
 end INIT_SHIFTREGS;

 architecture Behavioral of INIT_SHIFTREGS is

 component shiftreg14bit
    port ( C    : in    std_logic;        -- clock
           CE   : in    std_logic;        -- enable
           CLR  : in    std_logic;        -- clear
           D    : in    std_logic_vector (13 downto 0);  -- input
           L    : in    std_logic;        -- load
           LEFT : in    std_logic;        -- shift left
           SLI  : in    std_logic;        -- add to left 0/1
           SRI  : in    std_logic;        -- add to right 0/1
           Q    : out   std_logic_vector (13 downto 0));     -- output
 end component;
```

```
component FDCE
     -- synopsys translate_off
     generic( INIT : bit :=  '0');
     -- synopsys translate_on
     port ( C   : in    std_logic;
            CE  : in    std_logic;
            CLR : in    std_logic;
            D   : in    std_logic;
            Q   : out   std_logic);
end component;

component ARCH1_CSD4
    Port ( clock : in std_logic;
         enable :  in std_logic;
         tin : in std_logic_vector(3 downto 0);
       tout : out std_logic
         );
end component;

signal shiftOut : std_logic_vector(3 downto 0);

signal out0 : std_logic_vector(12 downto 0);
signal out1 : std_logic_vector(12 downto 0);
signal out2 : std_logic_vector(12 downto 0);
signal out3 : std_logic_vector(12 downto 0);

signal Q0 : std_logic_vector(7 downto 0);
signal Q1 : std_logic_vector(7 downto 0);
signal Q2 : std_logic_vector(7 downto 0);

signal GND : std_logic := '0';

begin

shiftreg_0 : shiftreg14bit
   port map (C => clk, CE => enable, CLR => GND,
           D(5 downto 0) => "000000",
           D(13 downto 6) => tin,
           L => load, LEFT => GND, SLI => GND, SRI => GND,
           Q(0) => shiftOut(0), Q(13 downto 1) => out0
           );

-- flipflops to hold the value for next shiftregister.
flipflop_00 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>shiftOut(0), Q=>Q0(7));

flipflop_01 :  FDCE
```

```
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(7), Q=>Q0(6));

flipflop_02 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(6), Q=>Q0(5));

flipflop_03 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(5), Q=>Q0(4));

flipflop_04 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(4), Q=>Q0(3));

flipflop_05 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(3), Q=>Q0(2));

flipflop_06 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(2), Q=>Q0(1));

flipflop_07 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(1), Q=>Q0(0));


shiftreg_1 : shiftreg14bit
   port map (C => clk, CE => enable, CLR => GND,
           D(5 downto 0) => "000000",
           D(13 downto 6) => Q0,
           L => load, LEFT => GND, SLI => GND, SRI => GND,
           Q(0) => shiftOut(1), Q(13 downto 1) => out1
           );
flipflop_10 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>shiftOut(1), Q=>Q1(7));

flipflop_11 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(7), Q=>Q1(6));

flipflop_12 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(6), Q=>Q1(5));

flipflop_13 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(5), Q=>Q1(4));

flipflop_14 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(4), Q=>Q1(3));

flipflop_15 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(3), Q=>Q1(2));

flipflop_16 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(2), Q=>Q1(1));
```

```
flipflop_17 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(1), Q=>Q1(0));


shiftreg_2 : shiftreg14bit
   port map (C => clk, CE => enable, CLR => GND,
           D(5 downto 0) => "000000",
           D(13 downto 6) => Q1,
           L => load, LEFT => GND, SLI => GND, SRI => GND,
           Q(0) => shiftOut(2), Q(13 downto 1) => out2
           );
flipflop_20 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>shiftOut(2), Q=>Q2(7));

flipflop_21 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(7), Q=>Q2(6));

flipflop_22 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(6), Q=>Q2(5));

flipflop_23 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(5), Q=>Q2(4));

flipflop_24 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(4), Q=>Q2(3));

flipflop_25 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(3), Q=>Q2(2));

flipflop_26 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(2), Q=>Q2(1));

flipflop_27 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(1), Q=>Q2(0));


shiftreg_3 : shiftreg14bit
   port map (C => clk, CE => enable, CLR => GND,
           D(5 downto 0) => "000000",
           D(13 downto 6) => Q2,
           L => load, LEFT => GND, SLI => GND, SRI => GND,
           Q(0) => shiftOut(3), Q(13 downto 1) => out3
           );

arch0 : ARCH1_CSD4
    port map (clock=>clk, enable=>enable,
                  tin=>shiftOut, tout=>tout
```

```
        );

end Behavioral;
```

Lut6Bits.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity lut is
    generic (              INIT0 : bit_vector := x"16";
               INIT1 : bit_vector := x"16";
               INIT2 : bit_vector := x"16";
               INIT3 : bit_vector := x"16";
               INIT4 : bit_vector := x"16";
               INIT5 : bit_vector := x"16");
   Port ( ui0 : in std_logic;
          ui1 : in std_logic;
          ui2 : in std_logic;
          ui3 : in std_logic;
          uo : out std_logic_vector(5 downto 0));
end lut;

architecture Behavioral of lut is
       -- Component Declaration for LUT4 should be placed
-- after architecture statement but before begin keyword
component LUT4
   generic (INIT : bit_vector := x"16");
   port (O : out STD_LOGIC;
          I0 : in STD_LOGIC;
          I1 : in STD_LOGIC;
          I2 : in STD_LOGIC;
          I3 : in STD_LOGIC);
end component;

-- Component Attribute specification for LUT4
-- should be placed after architecture declaration but
```

```
-- before the begin keyword
attribute INIT : string;
begin
LUT4_p0 : LUT4 generic map (INIT => INIT0)
   port map (O => uO(0),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p1 : LUT4 generic map (INIT => INIT1)
   port map (O => uO(1),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p2 : LUT4 generic map (INIT => INIT2)
   port map (O => uO(2),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p3 : LUT4 generic map (INIT => INIT3)
   port map (O => uO(3),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p4 : LUT4 generic map (INIT => INIT4)
   port map (O => uO(4),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p5 : LUT4 generic map (INIT => INIT5)
   port map (O => uO(5),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);

end Behavioral;
```

LutSub_50.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity LUTsub_50 is
    Port ( clock : in std_logic;
               enable :  in std_logic;
               ti : in std_logic_vector(3 downto 0);
           tout : out std_logic
               );
end LUTsub_50;

architecture Behavioral of LUTsub_50 is


component lut
    generic (
            INIT0 : bit_vector := x"16";
```

```
            INIT1 : bit_vector := x"16";
            INIT2 : bit_vector := x"16";
            INIT3 : bit_vector := x"16";
            INIT4 : bit_vector := x"16";
            INIT5 : bit_vector := x"16");
   port (
           UI0   : in    std_logic;
           UI1   : in    std_logic;
           UI2   : in    std_logic;
           UI3   : in    std_logic;
           UO    : out   std_logic_vector(5 downto 0)
       );
end component;

component SUB2
   PORT(  A:  IN STD_LOGIC;
          B:  IN STD_LOGIC;
          bin : in std_logic;
             Q:  OUT STD_LOGIC;
          bout :   OUT STD_LOGIC
       );
END component;

component add_ff
  port ( clock   : in    std_logic;
         CE  : in    std_logic;
         CLR : in    std_logic;
            afi0 : in std_logic_vector(5 downto 0);
            afso : out std_logic
            );
end component;

signal poslutout : std_logic_vector(5 downto 0);
signal neglutout : std_logic_vector(5 downto 0);

signal posffout : std_logic;
signal negffout : std_logic;

signal sign0 : std_logic;
signal bin0 : std_logic;
signal diff : std_logic;

begin

-- lut
poslut_0 : lut
   generic map (
           INIT0=>"0000000010000000",
```

```
            INIT1=>"0000000000100000",
            INIT2=>"1000000000000000",
            INIT3=>"0000000000000000",
            INIT4=>"0001000001000000",
            INIT5=>"0000000000000000")
   port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),
             UO=>poslutout);

neglut_0 : lut
   generic map (
            INIT0=>"0000000001000000",
            INIT1=>"0000000100000000",
            INIT2=>"0000000000000000",
            INIT3=>"0000000000000000",
            INIT4=>"0000000000000000",
            INIT5=>"0000000000000000")
   port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),
             UO=>neglutout);

 add_ff_0: add_ff
   port map (clock, enable, '0', poslutout, posffout);

 add_ff_1: add_ff
   port map (clock, enable, '0', neglutout, negffout);

 -- sub neg from pos
 sub1bit_0 : SUB2
       port map (A=>posffout, B=>negffout, bin=>bin0, Q=>diff, bout=>sign0);

 -- store carry signal
 flip_flop_0 : FDCE
       port map (C=>clock, CE=>enable, CLR=>'0', D=>sign0, Q=>bin0);

 -- store diff
 flip_flop_1 : FDCE
       port map (C=>clock, CE=>enable, CLR=>'0', D=>diff, Q=>tout);

end Behavioral;
```

LutSub_70.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
```

```
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity LUTsub_70 is
    Port ( clock : in std_logic;
               enable :  in std_logic;
               ti : in std_logic_vector(3 downto 0);
           tout : out std_logic
               );
end LUTsub_70;

architecture Behavioral of LUTsub_70 is


component lut
    generic (
            INIT0 : bit_vector := x"16";
            INIT1 : bit_vector := x"16";
            INIT2 : bit_vector := x"16";
            INIT3 : bit_vector := x"16";
            INIT4 : bit_vector := x"16";
            INIT5 : bit_vector := x"16");
   port (
           UI0   : in    std_logic;
           UI1   : in    std_logic;
           UI2   : in    std_logic;
           UI3   : in    std_logic;
           UO    : out   std_logic_vector(5 downto 0)
       );
end component;

component SUB2
   PORT(  A:  IN STD_LOGIC;
          B:  IN STD_LOGIC;
          bin : in std_logic;
             Q:  OUT STD_LOGIC;
          bout :   OUT STD_LOGIC
       );
END component;

component add_ff
  port ( clock   : in    std_logic;
         CE  : in    std_logic;
         CLR : in    std_logic;
```

```
            afi0 : in std_logic_vector(5 downto 0);
            afso : out std_logic
            );
end component;

signal poslutout : std_logic_vector(5 downto 0);
signal neglutout : std_logic_vector(5 downto 0);

signal posffout : std_logic;
signal negffout : std_logic;

signal sign0 : std_logic;
signal bin0 : std_logic;
signal diff : std_logic;

begin

-- lut
poslut_0 : lut
   generic map (
           INIT0=>"0000000000000000",
           INIT1=>"0000000000000000",
           INIT2=>"0000000000000010",
           INIT3=>"0000000000000000",
           INIT4=>"0010110010000000",
           INIT5=>"0000000000000000")
   port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),
             UO=>poslutout);

neglut_0 : lut
   generic map (
           INIT0=>"0000000000000010",
           INIT1=>"0000000000001000",
           INIT2=>"0000000000000000",
           INIT3=>"0000000000000000",
           INIT4=>"0000000000000000",
           INIT5=>"0000000000000000")
   port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),
             UO=>neglutout);

 add_ff_0: add_ff
   port map (clock, enable, '0', poslutout, posffout);

 add_ff_1: add_ff
   port map (clock, enable, '0', neglutout, negffout);

 -- sub neg from pos
 sub1bit_0 : SUB2
```

```
      port map (A=>posffout, B=>negffout, bin=>bin0, Q=>diff, bout=>sign0);

 -- store carry signal
 flip_flop_0 : FDCE
      port map (C=>clock, CE=>enable, CLR=>'0', D=>sign0, Q=>bin0);

 -- store diff
 flip_flop_1 : FDCE
      port map (C=>clock, CE=>enable, CLR=>'0', D=>diff, Q=>tout);

end Behavioral;
```

LutSub_90.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity LUTsub_90 is
    Port ( clock : in std_logic;
               enable :  in std_logic;
               ti : in std_logic_vector(3 downto 0);
          tout : out std_logic
               );
end LUTsub_90;

architecture Behavioral of LUTsub_90 is


component lut
    generic (
            INIT0 : bit_vector := x"16";
            INIT1 : bit_vector := x"16";
            INIT2 : bit_vector := x"16";
            INIT3 : bit_vector := x"16";
            INIT4 : bit_vector := x"16";
            INIT5 : bit_vector := x"16");
   port (
```

```
           UI0   : in    std_logic;
           UI1   : in    std_logic;
           UI2   : in    std_logic;
           UI3   : in    std_logic;
           UO    : out   std_logic_vector(5 downto 0)
       );
end component;

component SUB2
   PORT(  A:  IN STD_LOGIC;
          B:  IN STD_LOGIC;
          bin : in std_logic;
             Q:  OUT STD_LOGIC;
          bout :   OUT STD_LOGIC
       );
END component;

component add_ff
  port ( clock   : in    std_logic;
         CE  : in    std_logic;
         CLR : in    std_logic;
            afi0 : in std_logic_vector(5 downto 0);
            afso : out std_logic
            );
end component;

signal poslutout : std_logic_vector(5 downto 0);
signal neglutout : std_logic_vector(5 downto 0);

signal posffout : std_logic;
signal negffout : std_logic;

signal sign0 : std_logic;
signal bin0 : std_logic;
signal diff : std_logic;

begin

-- lut
poslut_0 : lut
   generic map (
           INIT0=>"0000000000000000",
           INIT1=>"0000000000000000",
           INIT2=>"0001000000000000",
           INIT3=>"0000000000010000",
           INIT4=>"1100000000000000",
           INIT5=>"0000000000000000")
   port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),
```

```
            UO=>poslutout);

neglut_0 : lut
   generic map (
           INIT0=>"0000000000000000",
           INIT1=>"0100000000000000",
           INIT2=>"0000010000000000",
           INIT3=>"0000000000000000",
           INIT4=>"0000000000000000",
           INIT5=>"0000000000000000")
   port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),
             UO=>neglutout);

 add_ff_0: add_ff
   port map (clock, enable, '0', poslutout, posffout);

 add_ff_1: add_ff
   port map (clock, enable, '0', neglutout, negffout);

 -- sub neg from pos
 sub1bit_0 : SUB2
       port map (A=>posffout, B=>negffout, bin=>bin0, Q=>diff, bout=>sign0);

 -- store carry signal
 flip_flop_0 : FDCE
       port map (C=>clock, CE=>enable, CLR=>'0', D=>sign0, Q=>bin0);

 -- store diff
 flip_flop_1 : FDCE
       port map (C=>clock, CE=>enable, CLR=>'0', D=>diff, Q=>tout);

end Behavioral;
```

LutSub_110.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;
```

```
entity LUTsub_110 is
    Port ( clock : in std_logic;
               enable :  in std_logic;
               ti : in std_logic_vector(3 downto 0);
           tout : out std_logic
               );
end LUTsub_110;

architecture Behavioral of LUTsub_110 is


component lut
    generic (
             INIT0 : bit_vector := x"16";
             INIT1 : bit_vector := x"16";
             INIT2 : bit_vector := x"16";
             INIT3 : bit_vector := x"16";
             INIT4 : bit_vector := x"16";
             INIT5 : bit_vector := x"16");
   port (
            UI0   : in    std_logic;
            UI1   : in    std_logic;
            UI2   : in    std_logic;
            UI3   : in    std_logic;
            UO    : out   std_logic_vector(5 downto 0)
       );
end component;

component SUB2
   PORT(  A:  IN STD_LOGIC;
          B:  IN STD_LOGIC;
          bin : in std_logic;
             Q:  OUT STD_LOGIC;
          bout :   OUT STD_LOGIC
       );
END component;

component add_ff
  port ( clock   : in    std_logic;
         CE  : in    std_logic;
         CLR : in    std_logic;
            afi0 : in std_logic_vector(5 downto 0);
            afso : out std_logic
            );
end component;

signal poslutout : std_logic_vector(5 downto 0);
```

```
signal neglutout : std_logic_vector(5 downto 0);

signal posffout : std_logic;
signal negffout : std_logic;

signal sign0 : std_logic;
signal bin0 : std_logic;
signal diff : std_logic;

begin

-- lut
poslut_0 : lut
   generic map (
           INIT0=>"0001000000000000",
           INIT1=>"0000000000010000",
           INIT2=>"0000000001000000",
           INIT3=>"0000000000100000",
           INIT4=>"0000000000000000",
           INIT5=>"0000000000000000")
   port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),
             UO=>poslutout);

neglut_0 : lut
   generic map (
           INIT0=>"0010000000000000",
           INIT1=>"0000101000000000",
           INIT2=>"0000000000000000",
           INIT3=>"0000000000000000",
           INIT4=>"0000000000000000",
           INIT5=>"0000000000000000")
   port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),
             UO=>neglutout);

 add_ff_0: add_ff
   port map (clock, enable, '0', poslutout, posffout);

 add_ff_1: add_ff
   port map (clock, enable, '0', neglutout, negffout);

 -- sub neg from pos
 sub1bit_0 : SUB2
       port map (A=>posffout, B=>negffout, bin=>bin0, Q=>diff, bout=>sign0);

 -- store carry signal
 flip_flop_0 : FDCE
       port map (C=>clock, CE=>enable, CLR=>'0', D=>sign0, Q=>bin0);
```

```
 -- store diff
 flip_flop_1 : FDCE
       port map (C=>clock, CE=>enable, CLR=>'0', D=>diff, Q=>tout);

end Behavioral;
```

LutSub_130.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity LUTsub_130 is
    Port ( clock : in std_logic;
               enable :  in std_logic;
               ti : in std_logic_vector(3 downto 0);
           tout : out std_logic
               );
end LUTsub_130;

architecture Behavioral of LUTsub_130 is


component lut
    generic (
            INIT0 : bit_vector := x"16";
            INIT1 : bit_vector := x"16";
            INIT2 : bit_vector := x"16";
            INIT3 : bit_vector := x"16";
            INIT4 : bit_vector := x"16";
            INIT5 : bit_vector := x"16");
   port (
           UI0   : in    std_logic;
           UI1   : in    std_logic;
           UI2   : in    std_logic;
           UI3   : in    std_logic;
           UO    : out   std_logic_vector(5 downto 0)
       );
```

```
end component;

component SUB2
   PORT(  A:  IN STD_LOGIC;
          B:  IN STD_LOGIC;
          bin : in std_logic;
             Q:  OUT STD_LOGIC;
          bout :   OUT STD_LOGIC
       );
END component;

component add_ff
  port ( clock   : in    std_logic;
         CE  : in    std_logic;
         CLR : in    std_logic;
            afi0 : in std_logic_vector(5 downto 0);
            afso : out std_logic
            );
end component;

signal poslutout : std_logic_vector(5 downto 0);
signal neglutout : std_logic_vector(5 downto 0);

signal posffout : std_logic;
signal negffout : std_logic;

signal sign0 : std_logic;
signal bin0 : std_logic;
signal diff : std_logic;

begin

-- lut
poslut_0 : lut
   generic map (
            INIT0=>"0000000000000000",
            INIT1=>"0000000000000000",
            INIT2=>"0000000000000000",
            INIT3=>"0000000100000100",
            INIT4=>"0000000000000000",
            INIT5=>"0000000000000000")
   port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),
             UO=>poslutout);

neglut_0 : lut
   generic map (
            INIT0=>"1000000000000000",
            INIT1=>"0000000000000000",
```

```
          INIT2=>"0000000000000000",
          INIT3=>"0000000000000000",
          INIT4=>"0000000000000000",
          INIT5=>"0000000000000000")
  port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),
            UO=>neglutout);

add_ff_0: add_ff
  port map (clock, enable, '0', poslutout, posffout);

add_ff_1: add_ff
  port map (clock, enable, '0', neglutout, negffout);

-- sub neg from pos
sub1bit_0 : SUB2
      port map (A=>posffout, B=>negffout, bin=>bin0, Q=>diff, bout=>sign0);

-- store carry signal
flip_flop_0 : FDCE
      port map (C=>clock, CE=>enable, CLR=>'0', D=>sign0, Q=>bin0);

-- store diff
flip_flop_1 : FDCE
      port map (C=>clock, CE=>enable, CLR=>'0', D=>diff, Q=>tout);

end Behavioral;
```

LutSub_150.vhdl

```
 -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity LUTsub_150 is
    Port ( clock : in std_logic;
           enable :  in std_logic;
           ti : in std_logic_vector(3 downto 0);
          tout : out std_logic
```

```
              );
end LUTsub_150;

architecture Behavioral of LUTsub_150 is


component lut
    generic (
             INIT0 : bit_vector := x"16";
             INIT1 : bit_vector := x"16";
             INIT2 : bit_vector := x"16";
             INIT3 : bit_vector := x"16";
             INIT4 : bit_vector := x"16";
             INIT5 : bit_vector := x"16");
   port (
            UI0   : in    std_logic;
            UI1   : in    std_logic;
            UI2   : in    std_logic;
            UI3   : in    std_logic;
            UO    : out   std_logic_vector(5 downto 0)
       );
end component;

component SUB2
   PORT(  A:  IN STD_LOGIC;
          B:  IN STD_LOGIC;
          bin : in std_logic;
             Q:  OUT STD_LOGIC;
          bout :   OUT STD_LOGIC
       );
END component;

component add_ff
  port ( clock   : in    std_logic;
         CE  : in    std_logic;
         CLR : in    std_logic;
            afi0 : in std_logic_vector(5 downto 0);
            afso : out std_logic
            );
end component;

signal poslutout : std_logic_vector(5 downto 0);
signal neglutout : std_logic_vector(5 downto 0);

signal posffout : std_logic;
signal negffout : std_logic;

signal sign0 : std_logic;
```

```
signal bin0 : std_logic;
signal diff : std_logic;

begin

-- lut
poslut_0 : lut
   generic map (
            INIT0=>"0000000000000000",
            INIT1=>"0000000000000000",
            INIT2=>"0000000000000000",
            INIT3=>"0000001000001000",
            INIT4=>"0000000000000000",
            INIT5=>"0000000000000000")
   port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),
             UO=>poslutout);

neglut_0 : lut
   generic map (
            INIT0=>"0000000000000000",
            INIT1=>"0000000000000000",
            INIT2=>"0010000010000000",
            INIT3=>"0000000000000000",
            INIT4=>"0000000000000000",
            INIT5=>"0000000000000000")
   port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),
             UO=>neglutout);

 add_ff_0: add_ff
   port map (clock, enable, '0', poslutout, posffout);

 add_ff_1: add_ff
   port map (clock, enable, '0', neglutout, negffout);

 -- sub neg from pos
 sub1bit_0 : SUB2
       port map (A=>posffout, B=>negffout, bin=>bin0, Q=>diff, bout=>sign0);

 -- store carry signal
 flip_flop_0 : FDCE
       port map (C=>clock, CE=>enable, CLR=>'0', D=>sign0, Q=>bin0);

 -- store diff
 flip_flop_1 : FDCE
       port map (C=>clock, CE=>enable, CLR=>'0', D=>diff, Q=>tout);

end Behavioral;
```

ShiftReg14.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity M2_1_MXILINX_test is
   port ( D0 : in    std_logic;
          D1 : in    std_logic;
          S0 : in    std_logic;
          O  : out   std_logic);
end M2_1_MXILINX_test;

architecture BEHAVIORAL of M2_1_MXILINX_test is
   attribute BOX_TYPE   : STRING ;
   signal M0 : std_logic;
   signal M1 : std_logic;
   component AND2B1
      port ( I0 : in    std_logic;
             I1 : in    std_logic;
             O  : out   std_logic);
   end component;
   attribute BOX_TYPE of AND2B1 : COMPONENT is "BLACK_BOX";

   component OR2
      port ( I0 : in    std_logic;
             I1 : in    std_logic;
             O  : out   std_logic);
   end component;
   attribute BOX_TYPE of OR2 : COMPONENT is "BLACK_BOX";

   component AND2
      port ( I0 : in    std_logic;
             I1 : in    std_logic;
             O  : out   std_logic);
   end component;
   attribute BOX_TYPE of AND2 : COMPONENT is "BLACK_BOX";

begin
```

```
   I_36_7 : AND2B1
      port map (I0=>S0, I1=>D0, O=>M0);

   I_36_8 : OR2
      port map (I0=>M1, I1=>M0, O=>O);

   I_36_9 : AND2
      port map (I0=>D1, I1=>S0, O=>M1);

end BEHAVIORAL;


  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity shiftreg14bit is
   port ( C    : in    std_logic;   -- clock
          CE   : in    std_logic;   -- enable
          CLR  : in    std_logic;   -- clear
          D    : in    std_logic_vector (13 downto 0);   -- input
          L    : in    std_logic;   -- load
          LEFT : in    std_logic;   -- shift left
          SLI  : in    std_logic;   -- add to left 0/1
          SRI  : in    std_logic;   -- add to right 0/1
          Q    : out   std_logic_vector (13 downto 0));  -- output
end shiftreg14bit;

architecture BEHAVIORAL of shiftreg14bit is
   attribute HU_SET     : STRING ;
   attribute INIT       : STRING ;
   attribute BOX_TYPE   : STRING ;
   signal L_LEFT  : std_logic;
   signal L_OR_CE : std_logic;
   signal MDL0    : std_logic;
   signal MDL1    : std_logic;
   signal MDL2    : std_logic;
   signal MDL3    : std_logic;
   signal MDL4    : std_logic;
   signal MDL5    : std_logic;
```

```
signal MDL6    : std_logic;
signal MDL7    : std_logic;
signal MDL8    : std_logic;
signal MDL9    : std_logic;
signal MDL10    : std_logic;
signal MDL11    : std_logic;
signal MDL12    : std_logic;
signal MDL13    : std_logic;
signal MDR0    : std_logic;
signal MDR1    : std_logic;
signal MDR2    : std_logic;
signal MDR3    : std_logic;
signal MDR4    : std_logic;
signal MDR5    : std_logic;
signal MDR6    : std_logic;
signal MDR7    : std_logic;
signal MDR8    : std_logic;
signal MDR9    : std_logic;
signal MDR10    : std_logic;
signal MDR11    : std_logic;
signal MDR12    : std_logic;
signal MDR13    : std_logic;
signal Q_DUMMY : std_logic_vector (13 downto 0);
component M2_1_MXILINX_test
   port ( D0 : in    std_logic;
          D1 : in    std_logic;
          S0 : in    std_logic;
          O  : out   std_logic);
end component;

component FDCE
   -- synopsys translate_off
   generic( INIT : bit :=  '0');
   -- synopsys translate_on
   port ( C   : in    std_logic;
          CE  : in    std_logic;
          CLR : in    std_logic;
          D   : in    std_logic;
          Q   : out   std_logic);
end component;
attribute INIT of FDCE : COMPONENT is "0";
attribute BOX_TYPE of FDCE : COMPONENT is "BLACK_BOX";

component OR2
   port ( I0 : in    std_logic;
          I1 : in    std_logic;
          O  : out   std_logic);
end component;
```

```
   attribute BOX_TYPE of OR2 : COMPONENT is "BLACK_BOX";


begin
   Q(13 downto 0) <= Q_DUMMY(13 downto 0);
   I_MDL0 : M2_1_MXILINX_test
      port map (D0=>SLI, D1=>D(0), S0=>L, O=>MDL0);

   I_MDL1 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(0), D1=>D(1), S0=>L, O=>MDL1);

   I_MDL2 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(1), D1=>D(2), S0=>L, O=>MDL2);

   I_MDL3 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(2), D1=>D(3), S0=>L, O=>MDL3);

   I_MDL4 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(3), D1=>D(4), S0=>L, O=>MDL4);

   I_MDL5 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(4), D1=>D(5), S0=>L, O=>MDL5);

   I_MDL6 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(5), D1=>D(6), S0=>L, O=>MDL6);

   I_MDL7 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(6), D1=>D(7), S0=>L, O=>MDL7);

   I_MDL8 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(7), D1=>D(8), S0=>L, O=>MDL8);

   I_MDL9 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(8), D1=>D(9), S0=>L, O=>MDL9);

   I_MDL10 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(9), D1=>D(10), S0=>L, O=>MDL10);

   I_MDL11 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(10), D1=>D(11), S0=>L, O=>MDL11);

   I_MDL12 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(11), D1=>D(12), S0=>L, O=>MDL12);

   I_MDL13 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(12), D1=>D(13), S0=>L, O=>MDL13);

   I_MDR0 : M2_1_MXILINX_test
```

```
   port map (D0=>Q_DUMMY(1), D1=>MDL0, S0=>L_LEFT, O=>MDR0);

I_MDR1 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(2), D1=>MDL1, S0=>L_LEFT, O=>MDR1);

I_MDR2 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(3), D1=>MDL2, S0=>L_LEFT, O=>MDR2);

I_MDR3 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(4), D1=>MDL3, S0=>L_LEFT, O=>MDR3);

I_MDR4 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(5), D1=>MDL4, S0=>L_LEFT, O=>MDR4);

I_MDR5 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(6), D1=>MDL5, S0=>L_LEFT, O=>MDR5);

I_MDR6 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(7), D1=>MDL6, S0=>L_LEFT, O=>MDR6);

I_MDR7 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(8), D1=>MDL7, S0=>L_LEFT, O=>MDR7);

I_MDR8 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(9), D1=>MDL8, S0=>L_LEFT, O=>MDR8);

I_MDR9 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(10), D1=>MDL9, S0=>L_LEFT, O=>MDR9);

I_MDR10 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(11), D1=>MDL10, S0=>L_LEFT, O=>MDR10);

I_MDR11 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(12), D1=>MDL11, S0=>L_LEFT, O=>MDR11);

I_MDR12 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(13), D1=>MDL12, S0=>L_LEFT, O=>MDR12);

I_MDR13 : M2_1_MXILINX_test
   port map (D0=>SRI, D1=>MDL13, S0=>L_LEFT, O=>MDR13);

I_Q0 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR0, Q=>Q_DUMMY(0));

I_Q1 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR1, Q=>Q_DUMMY(1));

I_Q2 : FDCE
```

```
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR2, Q=>Q_DUMMY(2));

   I_Q3 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR3, Q=>Q_DUMMY(3));

   I_Q4 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR4, Q=>Q_DUMMY(4));

   I_Q5 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR5, Q=>Q_DUMMY(5));

   I_Q6 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR6, Q=>Q_DUMMY(6));

   I_Q7 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR7, Q=>Q_DUMMY(7));

   I_Q8 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR8, Q=>Q_DUMMY(8));

   I_Q9 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR9, Q=>Q_DUMMY(9));

   I_Q10 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR10, Q=>Q_DUMMY(10));

   I_Q11 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR11, Q=>Q_DUMMY(11));

   I_Q12 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR12, Q=>Q_DUMMY(12));

   I_Q13 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR13, Q=>Q_DUMMY(13));

   I_36_145 : OR2
      port map (I0=>L, I1=>CE, O=>L_OR_CE);

   I_36_161 : OR2
      port map (I0=>LEFT, I1=>L, O=>L_LEFT);

end BEHAVIORAL;
```

InitShift_Test.vhd

```
  -- Created by Guri Kristine Birkeland 2005
```

```
 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
 use UNISIM.VComponents.all;


LIBRARY ieee; USE ieee.std_logic_1164.ALL; USE
ieee.std_logic_unsigned.all; USE ieee.numeric_std.ALL;

ENTITY init_shift_test_vhd IS END init_shift_test_vhd;

ARCHITECTURE behavior OF init_shift_test_vhd IS

    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT init_shiftregs
    PORT(
        clk : IN std_logic;
        enable : IN std_logic;
        load : IN std_logic;
        tin : IN std_logic_vector(7 downto 0);
        tout : OUT std_logic
        );
    END COMPONENT;

    --Inputs
    SIGNAL clk :  std_logic := '0';
    SIGNAL enable :  std_logic := '0';
    SIGNAL load :  std_logic := '0';
    SIGNAL tin :  std_logic_vector(7 downto 0) := (others=>'0');

    --Outputs
    SIGNAL tout :  std_logic;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: init_shiftregs PORT MAP(
        clk => clk,
        enable => enable,
        load => load,
        tin => tin,
        tout => tout
```

```
 );

PROCESS -- Process for clock
 BEGIN
     CLOCK_LOOP : LOOP
     clk <= transport '0';
     WAIT FOR 10 ns;
     clk <= transport '1';
     WAIT FOR 10 ns;
     WAIT FOR 40 ns;
     clk <= transport '0';
     WAIT FOR 40 ns;
     END LOOP CLOCK_LOOP;
 END PROCESS;

 tb : PROCESS
 BEGIN

     -- Wait 100 ns for global reset to finish
     wait for 100 ns;

     -- 1st
     -- --------------------
     enable <= transport '1';
     load <= transport '0';
     tin <= transport std_logic_vector'("00000001");
     -- --------------------
     WAIT FOR 100 ns; -- Time=100 ns
     load <= transport '1';
     -- --------------------
     WAIT FOR 100 ns; -- Time=200 ns
     load <= transport '0';
     -- --------------------
     --
     WAIT FOR 1600 ns; -- Time=1700 ns
     load <= transport '1';
     tin <= transport std_logic_vector'("00111111");
     -- --------------------
     WAIT FOR 100 ns;
     load <= transport '0';
     -- --------------------
     --
     WAIT FOR 1700 ns;
     load <= transport '1';
     tin <= transport std_logic_vector'("11111100");
     -- --------------------
     WAIT FOR 100 ns;
     load <= transport '0';
```

```
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("01111111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11111110");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11111100");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11111001");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("10011111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00111111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
```

```
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("01111111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11100111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00000011");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00000110");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("01100000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11000000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
```

```
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("10000000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00011000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00100100");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("01000010");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("10000001");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00000000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
```

```
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11100000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00000111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11110000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00001111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00011111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11111111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
```

```
        -- --------------------
        --
        WAIT FOR 1700 ns;
        load <= transport '1';
        tin <= transport std_logic_vector'("11111000");
        -- --------------------
        WAIT FOR 100 ns;
        load <= transport '0';
        -- --------------------
        --
        WAIT FOR 1700 ns;
        -- --------------------
        -- end waiting
        WAIT FOR 1300 ns;
        -- --------------------

-- accumulated runtime is: 51800 ns

    END PROCESS;

END;
```

C.3. Architecture 2 VHDL files

This section lists the files needed to produce a 4-tap filter of architecture 2.

C.3.1. Binary design

The files needed to produce the binary design of architecture 2 is listed below.

AddAll.vhdl

```
   -- Created by Guri Kristine Birkeland 2005

 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
```

```
use UNISIM.VComponents.all;

entity addAll is
   port (
        clock : in std_logic;
        enable : in std_logic;
        tin0 : in  std_logic;
        O : out    std_logic
        );
end addAll;

architecture Behavioral of addAll is

component add2
  port
  (
     a:  in std_logic;
     b:  in std_logic;
     cii : in std_logic;
     s: out std_logic;
     coo : out std_logic
  );
end component;

begin

O<=tin0;

end Behavioral;
```

AddAndFlipFlop.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity add_ff is
  port ( clock   : in    std_logic;
```

```
          CE  : in    std_logic;
          CLR : in    std_logic;
            afi0 : in std_logic_vector(7 downto 0);
        afso : out std_logic);
end add_ff;

architecture BEHAVIORAL of add_ff is

component add16bit
    port ( A   : in    std_logic_vector (15 downto 0);
           B   : in    std_logic_vector (15 downto 0);
           CI  : in    std_logic;
           CO  : out   std_logic;
           OFL : out   std_logic;
           S   : out   std_logic_vector (15 downto 0));
end component;

component flipflop
   port ( C   : in    std_logic;
          CE  : in    std_logic;
          CLR : in    std_logic;
          D   : in    std_logic_vector (8 downto 0);
          Q   : out   std_logic_vector (7 downto 0);
        so : out std_logic);
end component;

signal outadder : std_logic_vector(8 downto 0); signal inadder :
std_logic_vector(7 downto 0);

signal dummy0 : std_logic; signal dummy1 : std_logic_vector(6
downto 0); signal dummy2 : std_logic;

begin

  adder : add16bit
    port map ( A(7 downto 0) => afi0, A(15 downto 8)=>"00000000",
              B(7 downto 0) => inadder, B(15 downto 8)=>"00000000",
              CI => '0', CO=> dummy0, OFL => dummy2, S(8 downto 0)=> outadder,
              S(15 downto 9)=> dummy1);

  ff : flipflop
    port map (C => clock, CE => CE, CLR => CLR, D => outadder, Q => inadder,
             so => afso);

 end BEHAVIORAL;
```

Arch2_BIN.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity ARCH2_BIN is
    Port ( clock : in std_logic;
         enable :  in std_logic;
         tin : in std_logic_vector(3 downto 0);
       tout : out std_logic
         );
end ARCH2_BIN;

architecture Behavioral of ARCH2_BIN is

component biglut
    Port ( clock : in std_logic;
              enable :  in std_logic;
              tin : in std_logic_vector(3 downto 0);
           tout0 : out std_logic);
end component;

component addAll
   port (
        clock : in std_logic;
        enable : in std_logic;
        tin0 : in  std_logic;
        O : out    std_logic
        );
end component;

signal partsum0 : std_logic;

begin

biglut_0 : biglut
   port map (clock, enable, tin, partsum0);

addAll_0 : addAll
   port map (clock, enable, partsum0, tout);
```

```
end Behavioral;
```

BigLut.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity biglut is
    Port ( clock : in std_logic;
              enable :  in std_logic;
              tin : in std_logic_vector(3 downto 0);
           tout0 : out std_logic);
end biglut;

architecture Behavioral of biglut is

component lut_0
    Port ( clock : in std_logic;
              enable :  in std_logic;
              ti : in std_logic_vector(3 downto 0);
              tout : out std_logic
       );
 end component;

begin

lut_00 : lut_0
   port map (clock, enable, tin(3 downto 0), tout0);

end Behavioral;
```

FlipFlop.vhdl

```
  -- Created by Guri Kristine Birkeland 2005
```

```
 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
 use UNISIM.VComponents.all;

entity flipflop is
   port ( C   : in    std_logic;
          CE  : in    std_logic;
          CLR : in    std_logic;
          D   : in    std_logic_vector (8 downto 0);
          Q   : out   std_logic_vector (7 downto 0);
         so : out std_logic);
end flipflop;

architecture BEHAVIORAL of flipflop is

   attribute INIT       : STRING ;
   attribute BOX_TYPE   : STRING ;
   component FDCE
      -- synopsys translate_off
      generic( INIT : bit :=  '0');
      -- synopsys translate_on
      port ( C   : in    std_logic;
             CE  : in    std_logic;
             CLR : in    std_logic;
             D   : in    std_logic;
             Q   : out   std_logic);
   end component;
   attribute INIT of FDCE : COMPONENT is "0";
   attribute BOX_TYPE of FDCE : COMPONENT is "BLACK_BOX";

begin

   I_Q0 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(0), Q=>so);

   I_Q1 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(1), Q=>Q(0));

   I_Q2 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(2), Q=>Q(1));
```

```
   I_Q3 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(3), Q=>Q(2));

   I_Q4 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(4), Q=>Q(3));

   I_Q5 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(5), Q=>Q(4));

   I_Q6 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(6), Q=>Q(5));

   I_Q7 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(7), Q=>Q(6));

   I_Q8 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(8), Q=>Q(7));

end BEHAVIORAL;
```

Init_ShiftRegisters.vhdl

```
   -- Created by Guri Kristine Birkeland 2005

 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
 use UNISIM.VComponents.all;

 entity INIT_SHIFTREGS is
     Port ( clk, enable, load : in std_logic;
          tin : in std_logic_vector(7 downto 0);
          tout : out std_logic
          );
 end INIT_SHIFTREGS;

 architecture Behavioral of INIT_SHIFTREGS is

 component shiftreg16bit
    port ( C    : in    std_logic;        -- clock
           CE   : in    std_logic;        -- enable
```

```
          CLR  : in    std_logic;        -- clear
          D    : in    std_logic_vector (15 downto 0);  -- input
          L    : in    std_logic;        -- load
          LEFT : in    std_logic;        -- shift left
          SLI  : in    std_logic;        -- add to left 0/1
          SRI  : in    std_logic;        -- add to right 0/1
          Q    : out   std_logic_vector (15 downto 0));     -- output
end component;

component FDCE
     -- synopsys translate_off
     generic( INIT : bit :=  '0');
     -- synopsys translate_on
     port ( C   : in    std_logic;
            CE  : in    std_logic;
            CLR : in    std_logic;
            D   : in    std_logic;
            Q   : out   std_logic);
end component;

component ARCH2_BIN
    Port ( clock : in std_logic;
         enable :  in std_logic;
         tin : in std_logic_vector(3 downto 0);
       tout : out std_logic
         );
end component;

signal shiftOut : std_logic_vector(3 downto 0);

signal out0 : std_logic_vector(14 downto 0);
signal out1 : std_logic_vector(14 downto 0);
signal out2 : std_logic_vector(14 downto 0);
signal out3 : std_logic_vector(14 downto 0);

signal Q0 : std_logic_vector(7 downto 0);
signal Q1 : std_logic_vector(7 downto 0);
signal Q2 : std_logic_vector(7 downto 0);

signal GND : std_logic := '0';

begin

shiftreg_0 : shiftreg16bit
   port map (C => clk, CE => enable, CLR => GND,
           D(7 downto 0) => "00000000",
           D(15 downto 8) => tin,
           L => load, LEFT => GND, SLI => GND, SRI => GND,
```

```
           Q(0) => shiftOut(0), Q(15 downto 1) => out0
           );

-- flipflops to hold the value for next shiftregister.
flipflop_00 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>shiftOut(0), Q=>Q0(7));

flipflop_01 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(7), Q=>Q0(6));

flipflop_02 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(6), Q=>Q0(5));

flipflop_03 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(5), Q=>Q0(4));

flipflop_04 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(4), Q=>Q0(3));

flipflop_05 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(3), Q=>Q0(2));

flipflop_06 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(2), Q=>Q0(1));

flipflop_07 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(1), Q=>Q0(0));


shiftreg_1 : shiftreg16bit
   port map (C => clk, CE => enable, CLR => GND,
           D(7 downto 0) => "00000000",
           D(15 downto 8) => Q0,
           L => load, LEFT => GND, SLI => GND, SRI => GND,
           Q(0) => shiftOut(1), Q(15 downto 1) => out1
           );
flipflop_10 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>shiftOut(1), Q=>Q1(7));

flipflop_11 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(7), Q=>Q1(6));

flipflop_12 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(6), Q=>Q1(5));

flipflop_13 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(5), Q=>Q1(4));
```

```
flipflop_14 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(4), Q=>Q1(3));

flipflop_15 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(3), Q=>Q1(2));

flipflop_16 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(2), Q=>Q1(1));

flipflop_17 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(1), Q=>Q1(0));


shiftreg_2 : shiftreg16bit
   port map (C => clk, CE => enable, CLR => GND,
           D(7 downto 0) => "00000000",
           D(15 downto 8) => Q1,
           L => load, LEFT => GND, SLI => GND, SRI => GND,
           Q(0) => shiftOut(2), Q(15 downto 1) => out2
           );
flipflop_20 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>shiftOut(2), Q=>Q2(7));

flipflop_21 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(7), Q=>Q2(6));

flipflop_22 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(6), Q=>Q2(5));

flipflop_23 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(5), Q=>Q2(4));

flipflop_24 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(4), Q=>Q2(3));

flipflop_25 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(3), Q=>Q2(2));

flipflop_26 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(2), Q=>Q2(1));

flipflop_27 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(1), Q=>Q2(0));


shiftreg_3 : shiftreg16bit
   port map (C => clk, CE => enable, CLR => GND,
           D(7 downto 0) => "00000000",
```

```
           D(15 downto 8) => Q2,
           L => load, LEFT => GND, SLI => GND, SRI => GND,
           Q(0) => shiftOut(3), Q(15 downto 1) => out3
           );

arch0 : ARCH2_BIN
    port map (clock=>clk, enable=>enable,
                  tin=>shiftOut, tout=>tout
         );

end Behavioral;
```

Lut8Bits.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity lut is
    generic (              INIT0 : bit_vector := x"16";
               INIT1 : bit_vector := x"16";
               INIT2 : bit_vector := x"16";
               INIT3 : bit_vector := x"16";
               INIT4 : bit_vector := x"16";
               INIT5 : bit_vector := x"16";
               INIT6 : bit_vector := x"16";
               INIT7 : bit_vector := x"16");
   Port ( ui0 : in std_logic;
           ui1 : in std_logic;
           ui2 : in std_logic;
           ui3 : in std_logic;
           uo : out std_logic_vector(7 downto 0));
end lut;

architecture Behavioral of lut is
       -- Component Declaration for LUT4 should be placed
-- after architecture statement but before begin keyword
component LUT4
```

```
   generic (INIT : bit_vector := x"16");
   port (O : out STD_LOGIC;
           I0 : in STD_LOGIC;
           I1 : in STD_LOGIC;
           I2 : in STD_LOGIC;
           I3 : in STD_LOGIC);
end component;

-- Component Attribute specification for LUT4
-- should be placed after architecture declaration but
-- before the begin keyword
attribute INIT : string;
begin
LUT4_p0 : LUT4 generic map (INIT => INIT0)
   port map (O => uO(0),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p1 : LUT4 generic map (INIT => INIT1)
   port map (O => uO(1),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p2 : LUT4 generic map (INIT => INIT2)
   port map (O => uO(2),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p3 : LUT4 generic map (INIT => INIT3)
   port map (O => uO(3),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p4 : LUT4 generic map (INIT => INIT4)
   port map (O => uO(4),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p5 : LUT4 generic map (INIT => INIT5)
   port map (O => uO(5),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p6 : LUT4 generic map (INIT => INIT6)
   port map (O => uO(6),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p7 : LUT4 generic map (INIT => INIT7)
   port map (O => uO(7),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);

end Behavioral;
```

Lut_0.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;
```

```
entity lut_0 is
    Port ( clock : in std_logic;
               enable :  in std_logic;
               ti : in std_logic_vector(3 downto 0);
           tout : out std_logic
               );
end lut_0;

architecture Behavioral of lut_0 is


component lut
    generic (
             INIT0 : bit_vector := x"16";
             INIT1 : bit_vector := x"16";
             INIT2 : bit_vector := x"16";
             INIT3 : bit_vector := x"16";
             INIT4 : bit_vector := x"16";
             INIT5 : bit_vector := x"16";
             INIT6 : bit_vector := x"16";
             INIT7 : bit_vector := x"16");
   port (
            UI0   : in    std_logic;
            UI1   : in    std_logic;
            UI2   : in    std_logic;
            UI3   : in    std_logic;
            UO    : out   std_logic_vector(7 downto 0)
       );
end component;

component add_ff
  port ( clock   : in    std_logic;
         CE  : in    std_logic;
         CLR : in    std_logic;
            afi0 : in std_logic_vector(7 downto 0);
             afso : out std_logic
             );
end component;

signal poslutout : std_logic_vector(7 downto 0);

begin

-- lut
poslut_0 : lut
   generic map (
            INIT0=>"0000111111110010",
            INIT1=>"0001010101011010",
```

```
          INIT2=>"0000110000110110",
          INIT3=>"0001101001101000",
          INIT4=>"0001001101001100",
          INIT5=>"0010111110111100",
          INIT6=>"0011110011000000",
          INIT7=>"0000000000000000")
   port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),
             UO=>poslutout);

 add_ff_0: add_ff
   port map (clock, enable, '0', poslutout, tout);

end Behavioral;
```

ShiftReg16.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity M2_1_MXILINX_test is
   port ( D0 : in    std_logic;
          D1 : in    std_logic;
          S0 : in    std_logic;
          O  : out   std_logic);
end M2_1_MXILINX_test;

architecture BEHAVIORAL of M2_1_MXILINX_test is
   attribute BOX_TYPE   : STRING ;
   signal M0 : std_logic;
   signal M1 : std_logic;
   component AND2B1
      port ( I0 : in    std_logic;
             I1 : in    std_logic;
             O  : out   std_logic);
   end component;
   attribute BOX_TYPE of AND2B1 : COMPONENT is "BLACK_BOX";
```

```
   component OR2
      port ( I0 : in    std_logic;
             I1 : in    std_logic;
             O  : out   std_logic);
   end component;
   attribute BOX_TYPE of OR2 : COMPONENT is "BLACK_BOX";

   component AND2
      port ( I0 : in    std_logic;
             I1 : in    std_logic;
             O  : out   std_logic);
   end component;
   attribute BOX_TYPE of AND2 : COMPONENT is "BLACK_BOX";

begin
   I_36_7 : AND2B1
      port map (I0=>S0, I1=>D0, O=>M0);

   I_36_8 : OR2
      port map (I0=>M1, I1=>M0, O=>O);

   I_36_9 : AND2
      port map (I0=>D1, I1=>S0, O=>M1);

end BEHAVIORAL;



  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity shiftreg16bit is
   port ( C    : in    std_logic;   -- clock
          CE   : in    std_logic;   -- enable
          CLR  : in    std_logic;   -- clear
          D    : in    std_logic_vector (15 downto 0);   -- input
          L    : in    std_logic;   -- load
          LEFT : in    std_logic;   -- shift left
          SLI  : in    std_logic;   -- add to left 0/1
          SRI  : in    std_logic;   -- add to right 0/1
```

```
          Q    : out   std_logic_vector (15 downto 0));  -- output
end shiftreg16bit;

architecture BEHAVIORAL of shiftreg16bit is
   attribute HU_SET     : STRING ;
   attribute INIT       : STRING ;
   attribute BOX_TYPE   : STRING ;
   signal L_LEFT  : std_logic;
   signal L_OR_CE : std_logic;
   signal MDL0    : std_logic;
   signal MDL1    : std_logic;
   signal MDL2    : std_logic;
   signal MDL3    : std_logic;
   signal MDL4    : std_logic;
   signal MDL5    : std_logic;
   signal MDL6    : std_logic;
   signal MDL7    : std_logic;
   signal MDL8    : std_logic;
   signal MDL9    : std_logic;
   signal MDL10    : std_logic;
   signal MDL11    : std_logic;
   signal MDL12    : std_logic;
   signal MDL13    : std_logic;
   signal MDL14    : std_logic;
   signal MDL15    : std_logic;
   signal MDR0    : std_logic;
   signal MDR1    : std_logic;
   signal MDR2    : std_logic;
   signal MDR3    : std_logic;
   signal MDR4    : std_logic;
   signal MDR5    : std_logic;
   signal MDR6    : std_logic;
   signal MDR7    : std_logic;
   signal MDR8    : std_logic;
   signal MDR9    : std_logic;
   signal MDR10    : std_logic;
   signal MDR11    : std_logic;
   signal MDR12    : std_logic;
   signal MDR13    : std_logic;
   signal MDR14    : std_logic;
   signal MDR15    : std_logic;
   signal Q_DUMMY : std_logic_vector (15 downto 0);
   component M2_1_MXILINX_test
      port ( D0 : in    std_logic;
             D1 : in    std_logic;
             S0 : in    std_logic;
             O  : out   std_logic);
   end component;
```

```
   component FDCE
      -- synopsys translate_off
      generic( INIT : bit :=  '0');
      -- synopsys translate_on
      port ( C   : in    std_logic;
             CE  : in    std_logic;
             CLR : in    std_logic;
             D   : in    std_logic;
             Q   : out   std_logic);
   end component;
   attribute INIT of FDCE : COMPONENT is "0";
   attribute BOX_TYPE of FDCE : COMPONENT is "BLACK_BOX";

   component OR2
      port ( I0 : in    std_logic;
             I1 : in    std_logic;
             O  : out   std_logic);
   end component;
   attribute BOX_TYPE of OR2 : COMPONENT is "BLACK_BOX";

begin
   Q(15 downto 0) <= Q_DUMMY(15 downto 0);
   I_MDL0 : M2_1_MXILINX_test
      port map (D0=>SLI, D1=>D(0), S0=>L, O=>MDL0);

   I_MDL1 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(0), D1=>D(1), S0=>L, O=>MDL1);

   I_MDL2 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(1), D1=>D(2), S0=>L, O=>MDL2);

   I_MDL3 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(2), D1=>D(3), S0=>L, O=>MDL3);

   I_MDL4 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(3), D1=>D(4), S0=>L, O=>MDL4);

   I_MDL5 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(4), D1=>D(5), S0=>L, O=>MDL5);

   I_MDL6 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(5), D1=>D(6), S0=>L, O=>MDL6);

   I_MDL7 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(6), D1=>D(7), S0=>L, O=>MDL7);

   I_MDL8 : M2_1_MXILINX_test
```

```
   port map (D0=>Q_DUMMY(7), D1=>D(8), S0=>L, O=>MDL8);

I_MDL9 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(8), D1=>D(9), S0=>L, O=>MDL9);

I_MDL10 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(9), D1=>D(10), S0=>L, O=>MDL10);

I_MDL11 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(10), D1=>D(11), S0=>L, O=>MDL11);

I_MDL12 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(11), D1=>D(12), S0=>L, O=>MDL12);

I_MDL13 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(12), D1=>D(13), S0=>L, O=>MDL13);

I_MDL14 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(13), D1=>D(14), S0=>L, O=>MDL14);

I_MDL15 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(14), D1=>D(15), S0=>L, O=>MDL15);

I_MDR0 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(1), D1=>MDL0, S0=>L_LEFT, O=>MDR0);

I_MDR1 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(2), D1=>MDL1, S0=>L_LEFT, O=>MDR1);

I_MDR2 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(3), D1=>MDL2, S0=>L_LEFT, O=>MDR2);

I_MDR3 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(4), D1=>MDL3, S0=>L_LEFT, O=>MDR3);

I_MDR4 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(5), D1=>MDL4, S0=>L_LEFT, O=>MDR4);

I_MDR5 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(6), D1=>MDL5, S0=>L_LEFT, O=>MDR5);

I_MDR6 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(7), D1=>MDL6, S0=>L_LEFT, O=>MDR6);

I_MDR7 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(8), D1=>MDL7, S0=>L_LEFT, O=>MDR7);

I_MDR8 : M2_1_MXILINX_test
```

```
   port map (D0=>Q_DUMMY(9), D1=>MDL8, S0=>L_LEFT, O=>MDR8);

I_MDR9 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(10), D1=>MDL9, S0=>L_LEFT, O=>MDR9);

I_MDR10 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(11), D1=>MDL10, S0=>L_LEFT, O=>MDR10);

I_MDR11 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(12), D1=>MDL11, S0=>L_LEFT, O=>MDR11);

I_MDR12 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(13), D1=>MDL12, S0=>L_LEFT, O=>MDR12);

I_MDR13 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(14), D1=>MDL13, S0=>L_LEFT, O=>MDR13);

I_MDR14 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(15), D1=>MDL14, S0=>L_LEFT, O=>MDR14);

I_MDR15 : M2_1_MXILINX_test
   port map (D0=>SRI, D1=>MDL15, S0=>L_LEFT, O=>MDR15);

I_Q0 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR0, Q=>Q_DUMMY(0));

I_Q1 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR1, Q=>Q_DUMMY(1));

I_Q2 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR2, Q=>Q_DUMMY(2));

I_Q3 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR3, Q=>Q_DUMMY(3));

I_Q4 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR4, Q=>Q_DUMMY(4));

I_Q5 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR5, Q=>Q_DUMMY(5));

I_Q6 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR6, Q=>Q_DUMMY(6));

I_Q7 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR7, Q=>Q_DUMMY(7));

I_Q8 : FDCE
```

```
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR8, Q=>Q_DUMMY(8));

   I_Q9 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR9, Q=>Q_DUMMY(9));

   I_Q10 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR10, Q=>Q_DUMMY(10));

   I_Q11 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR11, Q=>Q_DUMMY(11));

   I_Q12 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR12, Q=>Q_DUMMY(12));

   I_Q13 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR13, Q=>Q_DUMMY(13));

   I_Q14 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR14, Q=>Q_DUMMY(14));

   I_Q15 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR15, Q=>Q_DUMMY(15));

   I_36_145 : OR2
      port map (I0=>L, I1=>CE, O=>L_OR_CE);

   I_36_161 : OR2
      port map (I0=>LEFT, I1=>L, O=>L_LEFT);

end BEHAVIORAL;
```

InitShift_Test.vhd

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;
```

```
LIBRARY ieee; USE ieee.std_logic_1164.ALL; USE
ieee.std_logic_unsigned.all; USE ieee.numeric_std.ALL;

ENTITY init_shift_test_vhd IS END init_shift_test_vhd;

ARCHITECTURE behavior OF init_shift_test_vhd IS

    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT init_shiftregs
    PORT(
        clk : IN std_logic;
        enable : IN std_logic;
        load : IN std_logic;
        tin : IN std_logic_vector(7 downto 0);
        tout : OUT std_logic
        );
    END COMPONENT;

    --Inputs
    SIGNAL clk :  std_logic := '0';
    SIGNAL enable :  std_logic := '0';
    SIGNAL load :  std_logic := '0';
    SIGNAL tin :  std_logic_vector(7 downto 0) := (others=>'0');

    --Outputs
    SIGNAL tout :  std_logic;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: init_shiftregs PORT MAP(
        clk => clk,
        enable => enable,
        load => load,
        tin => tin,
        tout => tout
    );

   PROCESS -- Process for clock
    BEGIN
        CLOCK_LOOP : LOOP
        clk <= transport '0';
        WAIT FOR 10 ns;
        clk <= transport '1';
        WAIT FOR 10 ns;
        WAIT FOR 40 ns;
        clk <= transport '0';
        WAIT FOR 40 ns;
```

```
    END LOOP CLOCK_LOOP;
END PROCESS;

tb : PROCESS
BEGIN

    -- Wait 100 ns for global reset to finish
    wait for 100 ns;

    -- 1st
    -- --------------------
    enable <= transport '1';
    load <= transport '0';
    tin <= transport std_logic_vector'("00000001");
    -- --------------------
    WAIT FOR 100 ns; -- Time=100 ns
    load <= transport '1';
    -- --------------------
    WAIT FOR 100 ns; -- Time=200 ns
    load <= transport '0';
    -- --------------------
    --
    WAIT FOR 1400 ns; -- Time=1500 ns
    load <= transport '1';
    tin <= transport std_logic_vector'("00111111");
    -- --------------------
    WAIT FOR 100 ns;
    load <= transport '0';
    -- --------------------
    --
    WAIT FOR 1500 ns;
    load <= transport '1';
    tin <= transport std_logic_vector'("11111100");
    -- --------------------
    WAIT FOR 100 ns;
    load <= transport '0';
    -- --------------------
    --
    WAIT FOR 1500 ns;
    load <= transport '1';
    tin <= transport std_logic_vector'("01111111");
    -- --------------------
    WAIT FOR 100 ns;
    load <= transport '0';
    -- --------------------
    --
    WAIT FOR 1500 ns;
    load <= transport '1';
```

```
tin <= transport std_logic_vector'("11111110");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11111100");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11111001");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("10011111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00111111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("01111111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
```

```
tin <= transport std_logic_vector'("11100111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00000011");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00000110");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("01100000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11000000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("10000000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
```

```
tin <= transport std_logic_vector'("00011000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00100100");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("01000010");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("10000001");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00000000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11100000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
```

```
tin <= transport std_logic_vector'("00000111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11110000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00001111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00011111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11111111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11111000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
-- --------------------
```

```
        -- end waiting
        WAIT FOR 800 ns;
        -- --------------------

-- accumulated runtime is: 45700 ns

    END PROCESS;

END;
```

C.3.2. CSD2 design

The files needed to produce the CSD2 design of architecture 2 is listed below.

AddAll.vhdl

```
   -- Created by Guri Kristine Birkeland 2005

 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
 use UNISIM.VComponents.all;

 entity addAll is
    port (
         clock : in std_logic;
         enable : in std_logic;
         tin0 : in  std_logic;
         O : out    std_logic
         );
 end addAll;

 architecture Behavioral of addAll is

 component add2
   port
   (
      a:  in std_logic;
      b:  in std_logic;
```

```
      cii : in std_logic;
      s: out std_logic;
      coo : out std_logic
   );
 end component;

 begin

 O<=tin0;

 end Behavioral;
```

AddAndFlipFlop.vhdl

```
   -- Created by Guri Kristine Birkeland 2005

 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
 use UNISIM.VComponents.all;

entity add_ff is
   port ( clock   : in    std_logic;
          CE  : in    std_logic;
          CLR : in    std_logic;
            afi0 : in std_logic_vector(7 downto 0);
        afso : out std_logic);
end add_ff;

architecture BEHAVIORAL of add_ff is

component add16bit
    port ( A   : in    std_logic_vector (15 downto 0);
           B   : in    std_logic_vector (15 downto 0);
           CI  : in    std_logic;
           CO  : out   std_logic;
           OFL : out   std_logic;
           S   : out   std_logic_vector (15 downto 0));
end component;

component flipflop
```

```
   port ( C   : in    std_logic;
          CE  : in    std_logic;
          CLR : in    std_logic;
          D   : in    std_logic_vector (8 downto 0);
          Q   : out   std_logic_vector (7 downto 0);
         so : out std_logic);
end component;

signal outadder : std_logic_vector(8 downto 0); signal inadder :
std_logic_vector(7 downto 0);

signal dummy0 : std_logic; signal dummy1 : std_logic_vector(6
downto 0); signal dummy2 : std_logic;

begin

  adder : add16bit
    port map ( A(7 downto 0) => afi0, A(15 downto 8)=>"00000000",
              B(7 downto 0) => inadder, B(15 downto 8)=>"00000000",
              CI => '0', CO=> dummy0, OFL => dummy2, S(8 downto 0)=> outadder,
              S(15 downto 9)=> dummy1);

  ff : flipflop
    port map (C => clock, CE => CE, CLR => CLR, D => outadder, Q => inadder,
             so => afso);

 end BEHAVIORAL;
```

Arch2_CSD2.vhdl

```
   -- Created by Guri Kristine Birkeland 2005

 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
 use UNISIM.VComponents.all;

entity add_ff is
   port ( clock   : in    std_logic;
          CE  : in    std_logic;
          CLR : in    std_logic;
```

```
            afi0 : in std_logic_vector(7 downto 0);
        afso : out std_logic);
end add_ff;

architecture BEHAVIORAL of add_ff is

component add16bit
    port ( A   : in    std_logic_vector (15 downto 0);
           B   : in    std_logic_vector (15 downto 0);
           CI  : in    std_logic;
           CO  : out   std_logic;
           OFL : out   std_logic;
           S   : out   std_logic_vector (15 downto 0));
end component;

component flipflop
   port ( C   : in    std_logic;
          CE  : in    std_logic;
          CLR : in    std_logic;
          D   : in    std_logic_vector (8 downto 0);
          Q   : out   std_logic_vector (7 downto 0);
        so : out std_logic);
end component;

signal outadder : std_logic_vector(8 downto 0); signal inadder :
std_logic_vector(7 downto 0);

signal dummy0 : std_logic; signal dummy1 : std_logic_vector(6
downto 0); signal dummy2 : std_logic;

begin

  adder : add16bit
    port map ( A(7 downto 0) => afi0, A(15 downto 8)=>"00000000",
             B(7 downto 0) => inadder, B(15 downto 8)=>"00000000",
             CI => '0', CO=> dummy0, OFL => dummy2, S(8 downto 0)=> outadder,
             S(15 downto 9)=> dummy1);

  ff : flipflop
    port map (C => clock, CE => CE, CLR => CLR, D => outadder, Q => inadder,
            so => afso);

 end BEHAVIORAL;
```

BigLut.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity biglut is
    Port ( clock : in std_logic;
           enable :  in std_logic;
           tin : in std_logic_vector(3 downto 0);
           posPartsum0 : out std_logic;
           negPartsum0 : out std_logic);
end biglut;

architecture Behavioral of biglut is

component lut_0
    Port ( clock : in std_logic;
              enable :  in std_logic;
              ti : in std_logic_vector(3 downto 0);
           toutpos : out std_logic;
              toutneg : out std_logic
       );
 end component;

begin

lut_00 : lut_0
   port map (clock, enable, tin(3 downto 0), posPartsum0, negPartsum0);

end Behavioral;
```

EndAddSub.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
 use UNISIM.VComponents.all;

 entity endAddSub is
    port (
              clock : in std_logic;
              enable : in std_logic;
              pos0 : in std_logic;
              neg0 : in std_logic;
              sum : out std_logic
              );
 end endAddSub;

 architecture Behavioral of endAddSub is

component SUB2
    PORT(
               A:  IN STD_LOGIC;
           B:  IN STD_LOGIC;
           bin : in std_logic;
              Q:  OUT STD_LOGIC;
           bout :   OUT STD_LOGIC
        );
END component;

 component addAll
    port (
         clock : in std_logic;
         enable : in std_logic;
         tin0 : in  std_logic;
         O : out    std_logic
         );
 end component;

 signal sign0, bin0, sum0 : std_logic;

 signal posSum : std_logic;
 signal negSum : std_logic;

 begin

 posAdd_0 : addAll
    port map (clock, enable, pos0, posSum);

 negAdd_0 : addAll
```

```
    port map (clock, enable, neg0, negSum);

 sub_0 : sub2
    port map (posSum, negSum, bin0, sum0, sign0);

 flip_flop_0 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign0, Q=>bin0);

 flip_flop_1 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sum0, Q=>sum);

 end Behavioral;
```

FlipFlop.vhdl

```
   -- Created by Guri Kristine Birkeland 2005

 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
 use UNISIM.VComponents.all;

entity flipflop is
   port ( C   : in    std_logic;
          CE  : in    std_logic;
          CLR : in    std_logic;
          D   : in    std_logic_vector (8 downto 0);
          Q   : out   std_logic_vector (7 downto 0);
        so : out std_logic);
end flipflop;

architecture BEHAVIORAL of flipflop is

   attribute INIT       : STRING ;
   attribute BOX_TYPE   : STRING ;
   component FDCE
      -- synopsys translate_off
      generic( INIT : bit :=  '0');
      -- synopsys translate_on
      port ( C   : in    std_logic;
             CE  : in    std_logic;
```

```
            CLR : in    std_logic;
            D   : in    std_logic;
            Q   : out   std_logic);
   end component;
   attribute INIT of FDCE : COMPONENT is "0";
   attribute BOX_TYPE of FDCE : COMPONENT is "BLACK_BOX";

begin

   I_Q0 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(0), Q=>so);

   I_Q1 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(1), Q=>Q(0));

   I_Q2 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(2), Q=>Q(1));

   I_Q3 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(3), Q=>Q(2));

   I_Q4 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(4), Q=>Q(3));

   I_Q5 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(5), Q=>Q(4));

   I_Q6 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(6), Q=>Q(5));

   I_Q7 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(7), Q=>Q(6));

   I_Q8 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(8), Q=>Q(7));

end BEHAVIORAL;
```

Init_ShiftRegisters.vhdl

```
   -- Created by Guri Kristine Birkeland 2005

 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity INIT_SHIFTREGS is
    Port ( clk, enable, load : in std_logic;
         tin : in std_logic_vector(7 downto 0);
         tout : out std_logic
         );
end INIT_SHIFTREGS;

architecture Behavioral of INIT_SHIFTREGS is

component shiftreg16bit
   port ( C    : in    std_logic;        -- clock
          CE   : in    std_logic;        -- enable
          CLR  : in    std_logic;        -- clear
          D    : in    std_logic_vector (15 downto 0);  -- input
          L    : in    std_logic;        -- load
          LEFT : in    std_logic;        -- shift left
          SLI  : in    std_logic;        -- add to left 0/1
          SRI  : in    std_logic;        -- add to right 0/1
          Q    : out   std_logic_vector (15 downto 0));     -- output
end component;

component FDCE
     -- synopsys translate_off
     generic( INIT : bit :=  '0');
     -- synopsys translate_on
     port ( C   : in    std_logic;
            CE  : in    std_logic;
            CLR : in    std_logic;
            D   : in    std_logic;
            Q   : out   std_logic);
end component;

component ARCH2_CSD2
    Port ( clock : in std_logic;
         enable :  in std_logic;
         tin : in std_logic_vector(3 downto 0);
       tout : out std_logic
         );
end component;

signal shiftOut : std_logic_vector(3 downto 0);
```

```
signal out0 : std_logic_vector(14 downto 0);
signal out1 : std_logic_vector(14 downto 0);
signal out2 : std_logic_vector(14 downto 0);
signal out3 : std_logic_vector(14 downto 0);

signal Q0 : std_logic_vector(7 downto 0);
signal Q1 : std_logic_vector(7 downto 0);
signal Q2 : std_logic_vector(7 downto 0);

signal GND : std_logic := '0';

begin

shiftreg_0 : shiftreg16bit
   port map (C => clk, CE => enable, CLR => GND,
           D(7 downto 0) => "00000000",
           D(15 downto 8) => tin,
           L => load, LEFT => GND, SLI => GND, SRI => GND,
           Q(0) => shiftOut(0), Q(15 downto 1) => out0
           );

-- flipflops to hold the value for next shiftregister.
flipflop_00 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>shiftOut(0), Q=>Q0(7));

flipflop_01 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(7), Q=>Q0(6));

flipflop_02 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(6), Q=>Q0(5));

flipflop_03 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(5), Q=>Q0(4));

flipflop_04 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(4), Q=>Q0(3));

flipflop_05 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(3), Q=>Q0(2));

flipflop_06 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(2), Q=>Q0(1));

flipflop_07 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(1), Q=>Q0(0));


shiftreg_1 : shiftreg16bit
```

```
   port map (C => clk, CE => enable, CLR => GND,
           D(7 downto 0) => "00000000",
           D(15 downto 8) => Q0,
           L => load, LEFT => GND, SLI => GND, SRI => GND,
           Q(0) => shiftOut(1), Q(15 downto 1) => out1
           );
flipflop_10 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>shiftOut(1), Q=>Q1(7));

flipflop_11 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(7), Q=>Q1(6));

flipflop_12 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(6), Q=>Q1(5));

flipflop_13 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(5), Q=>Q1(4));

flipflop_14 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(4), Q=>Q1(3));

flipflop_15 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(3), Q=>Q1(2));

flipflop_16 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(2), Q=>Q1(1));

flipflop_17 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(1), Q=>Q1(0));


shiftreg_2 : shiftreg16bit
   port map (C => clk, CE => enable, CLR => GND,
           D(7 downto 0) => "00000000",
           D(15 downto 8) => Q1,
           L => load, LEFT => GND, SLI => GND, SRI => GND,
           Q(0) => shiftOut(2), Q(15 downto 1) => out2
           );
flipflop_20 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>shiftOut(2), Q=>Q2(7));

flipflop_21 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(7), Q=>Q2(6));

flipflop_22 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(6), Q=>Q2(5));

flipflop_23 :  FDCE
```

```
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(5), Q=>Q2(4));

flipflop_24 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(4), Q=>Q2(3));

flipflop_25 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(3), Q=>Q2(2));

flipflop_26 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(2), Q=>Q2(1));

flipflop_27 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(1), Q=>Q2(0));


shiftreg_3 : shiftreg16bit
   port map (C => clk, CE => enable, CLR => GND,
           D(7 downto 0) => "00000000",
           D(15 downto 8) => Q2,
           L => load, LEFT => GND, SLI => GND, SRI => GND,
           Q(0) => shiftOut(3), Q(15 downto 1) => out3
           );

arch0 : ARCH2_CSD2
    port map (clock=>clk, enable=>enable,
                   tin=>shiftOut, tout=>tout
         );

end Behavioral;
```

Lut0.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity lut_0 is
    Port ( clock : in std_logic;
```

```
               enable :  in std_logic;
               ti : in std_logic_vector(3 downto 0);
           toutpos : out std_logic;
               toutneg : out std_logic
               );
end lut_0;

architecture Behavioral of lut_0 is


component lut
    generic (
             INIT0 : bit_vector := x"16";
             INIT1 : bit_vector := x"16";
             INIT2 : bit_vector := x"16";
             INIT3 : bit_vector := x"16";
             INIT4 : bit_vector := x"16";
             INIT5 : bit_vector := x"16";
             INIT6 : bit_vector := x"16";
             INIT7 : bit_vector := x"16");
   port (
            UI0   : in    std_logic;
            UI1   : in    std_logic;
            UI2   : in    std_logic;
            UI3   : in    std_logic;
            UO    : out   std_logic_vector(7 downto 0)
       );
end component;

component add_ff
  port ( clock   : in    std_logic;
         CE  : in    std_logic;
         CLR : in    std_logic;
            afi0 : in std_logic_vector(7 downto 0);
            afso : out std_logic
            );
end component;

signal outpos : std_logic_vector(7 downto 0);
signal outneg : std_logic_vector(7 downto 0);

begin

-- lut
poslut_0 : lut
   generic map (
            INIT0=>"0000000000001010",
            INIT1=>"0000000010000000",
```

```
            INIT2=>"0001001000001000",
            INIT3=>"1000000100000100",
            INIT4=>"0000010000010010",
            INIT5=>"1100000000000000",
            INIT6=>"0000000000010000",
            INIT7=>"0000001100101100")
   port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),
             UO=>outpos);

neglut_0 : lut
   generic map (
            INIT0=>"0000101000100000",
            INIT1=>"0000010101010000",
            INIT2=>"0100100000100010",
            INIT3=>"0000000001000000",
            INIT4=>"0001001000001000",
            INIT5=>"0000100100100100",
            INIT6=>"0011010011000000",
            INIT7=>"0000000000000000")
   port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),
             UO=>outneg);

 add_ff_0: add_ff
   port map (clock, enable, '0', outpos, toutpos);

 add_ff_1: add_ff
   port map (clock, enable, '0', outneg, toutneg);

end Behavioral;
```

Lut8Bits.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity lut is
    generic (              INIT0 : bit_vector := x"16";
```

```
               INIT1 : bit_vector := x"16";
               INIT2 : bit_vector := x"16";
               INIT3 : bit_vector := x"16";
               INIT4 : bit_vector := x"16";
               INIT5 : bit_vector := x"16";
               INIT6 : bit_vector := x"16";
               INIT7 : bit_vector := x"16");
   Port ( ui0 : in std_logic;
          ui1 : in std_logic;
          ui2 : in std_logic;
          ui3 : in std_logic;
          uo : out std_logic_vector(7 downto 0));
end lut;

architecture Behavioral of lut is
       -- Component Declaration for LUT4 should be placed
-- after architecture statement but before begin keyword
component LUT4
   generic (INIT : bit_vector := x"16");
   port (O : out STD_LOGIC;
          I0 : in STD_LOGIC;
          I1 : in STD_LOGIC;
          I2 : in STD_LOGIC;
          I3 : in STD_LOGIC);
end component;


-- Component Attribute specification for LUT4
-- should be placed after architecture declaration but
-- before the begin keyword
attribute INIT : string;
begin
LUT4_p0 : LUT4 generic map (INIT => INIT0)
   port map (O => uO(0),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p1 : LUT4 generic map (INIT => INIT1)
   port map (O => uO(1),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p2 : LUT4 generic map (INIT => INIT2)
   port map (O => uO(2),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p3 : LUT4 generic map (INIT => INIT3)
   port map (O => uO(3),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p4 : LUT4 generic map (INIT => INIT4)
   port map (O => uO(4),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p5 : LUT4 generic map (INIT => INIT5)
   port map (O => uO(5),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p6 : LUT4 generic map (INIT => INIT6)
   port map (O => uO(6),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p7 : LUT4 generic map (INIT => INIT7)
   port map (O => uO(7),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
```

```
end Behavioral;
```

ShiftReg16.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity M2_1_MXILINX_test is
   port ( D0 : in    std_logic;
          D1 : in    std_logic;
          S0 : in    std_logic;
          O  : out   std_logic);
end M2_1_MXILINX_test;

architecture BEHAVIORAL of M2_1_MXILINX_test is
   attribute BOX_TYPE   : STRING ;
   signal M0 : std_logic;
   signal M1 : std_logic;
   component AND2B1
      port ( I0 : in    std_logic;
             I1 : in    std_logic;
             O  : out   std_logic);
   end component;
   attribute BOX_TYPE of AND2B1 : COMPONENT is "BLACK_BOX";

   component OR2
      port ( I0 : in    std_logic;
             I1 : in    std_logic;
             O  : out   std_logic);
   end component;
   attribute BOX_TYPE of OR2 : COMPONENT is "BLACK_BOX";

   component AND2
      port ( I0 : in    std_logic;
             I1 : in    std_logic;
             O  : out   std_logic);
```

```
   end component;
   attribute BOX_TYPE of AND2 : COMPONENT is "BLACK_BOX";

begin
   I_36_7 : AND2B1
      port map (I0=>S0, I1=>D0, O=>M0);

   I_36_8 : OR2
      port map (I0=>M1, I1=>M0, O=>O);

   I_36_9 : AND2
      port map (I0=>D1, I1=>S0, O=>M1);

end BEHAVIORAL;



  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity shiftreg16bit is
   port ( C    : in    std_logic;   -- clock
          CE   : in    std_logic;   -- enable
          CLR  : in    std_logic;   -- clear
          D    : in    std_logic_vector (15 downto 0);   -- input
          L    : in    std_logic;   -- load
          LEFT : in    std_logic;   -- shift left
          SLI  : in    std_logic;   -- add to left 0/1
          SRI  : in    std_logic;   -- add to right 0/1
          Q    : out   std_logic_vector (15 downto 0));  -- output
end shiftreg16bit;

architecture BEHAVIORAL of shiftreg16bit is
   attribute HU_SET     : STRING ;
   attribute INIT       : STRING ;
   attribute BOX_TYPE   : STRING ;
   signal L_LEFT  : std_logic;
   signal L_OR_CE : std_logic;
   signal MDL0    : std_logic;
   signal MDL1    : std_logic;
```

```
signal MDL2    : std_logic;
signal MDL3    : std_logic;
signal MDL4    : std_logic;
signal MDL5    : std_logic;
signal MDL6    : std_logic;
signal MDL7    : std_logic;
signal MDL8    : std_logic;
signal MDL9    : std_logic;
signal MDL10    : std_logic;
signal MDL11    : std_logic;
signal MDL12    : std_logic;
signal MDL13    : std_logic;
signal MDL14    : std_logic;
signal MDL15    : std_logic;
signal MDR0    : std_logic;
signal MDR1    : std_logic;
signal MDR2    : std_logic;
signal MDR3    : std_logic;
signal MDR4    : std_logic;
signal MDR5    : std_logic;
signal MDR6    : std_logic;
signal MDR7    : std_logic;
signal MDR8    : std_logic;
signal MDR9    : std_logic;
signal MDR10    : std_logic;
signal MDR11    : std_logic;
signal MDR12    : std_logic;
signal MDR13    : std_logic;
signal MDR14    : std_logic;
signal MDR15    : std_logic;
signal Q_DUMMY : std_logic_vector (15 downto 0);
component M2_1_MXILINX_test
   port ( D0 : in    std_logic;
          D1 : in    std_logic;
          S0 : in    std_logic;
          O  : out   std_logic);
end component;

component FDCE
   -- synopsys translate_off
   generic( INIT : bit :=  '0');
   -- synopsys translate_on
   port ( C   : in    std_logic;
          CE  : in    std_logic;
          CLR : in    std_logic;
          D   : in    std_logic;
          Q   : out   std_logic);
end component;
```

```
   attribute INIT of FDCE : COMPONENT is "0";
   attribute BOX_TYPE of FDCE : COMPONENT is "BLACK_BOX";

   component OR2
      port ( I0 : in    std_logic;
             I1 : in    std_logic;
             O  : out   std_logic);
   end component;
   attribute BOX_TYPE of OR2 : COMPONENT is "BLACK_BOX";


begin
   Q(15 downto 0) <= Q_DUMMY(15 downto 0);
   I_MDL0 : M2_1_MXILINX_test
      port map (D0=>SLI, D1=>D(0), S0=>L, O=>MDL0);

   I_MDL1 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(0), D1=>D(1), S0=>L, O=>MDL1);

   I_MDL2 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(1), D1=>D(2), S0=>L, O=>MDL2);

   I_MDL3 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(2), D1=>D(3), S0=>L, O=>MDL3);

   I_MDL4 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(3), D1=>D(4), S0=>L, O=>MDL4);

   I_MDL5 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(4), D1=>D(5), S0=>L, O=>MDL5);

   I_MDL6 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(5), D1=>D(6), S0=>L, O=>MDL6);

   I_MDL7 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(6), D1=>D(7), S0=>L, O=>MDL7);

   I_MDL8 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(7), D1=>D(8), S0=>L, O=>MDL8);

   I_MDL9 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(8), D1=>D(9), S0=>L, O=>MDL9);

   I_MDL10 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(9), D1=>D(10), S0=>L, O=>MDL10);

   I_MDL11 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(10), D1=>D(11), S0=>L, O=>MDL11);
```

```
I_MDL12 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(11), D1=>D(12), S0=>L, O=>MDL12);

I_MDL13 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(12), D1=>D(13), S0=>L, O=>MDL13);

I_MDL14 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(13), D1=>D(14), S0=>L, O=>MDL14);

I_MDL15 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(14), D1=>D(15), S0=>L, O=>MDL15);

I_MDR0 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(1), D1=>MDL0, S0=>L_LEFT, O=>MDR0);

I_MDR1 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(2), D1=>MDL1, S0=>L_LEFT, O=>MDR1);

I_MDR2 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(3), D1=>MDL2, S0=>L_LEFT, O=>MDR2);

I_MDR3 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(4), D1=>MDL3, S0=>L_LEFT, O=>MDR3);

I_MDR4 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(5), D1=>MDL4, S0=>L_LEFT, O=>MDR4);

I_MDR5 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(6), D1=>MDL5, S0=>L_LEFT, O=>MDR5);

I_MDR6 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(7), D1=>MDL6, S0=>L_LEFT, O=>MDR6);

I_MDR7 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(8), D1=>MDL7, S0=>L_LEFT, O=>MDR7);

I_MDR8 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(9), D1=>MDL8, S0=>L_LEFT, O=>MDR8);

I_MDR9 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(10), D1=>MDL9, S0=>L_LEFT, O=>MDR9);

I_MDR10 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(11), D1=>MDL10, S0=>L_LEFT, O=>MDR10);

I_MDR11 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(12), D1=>MDL11, S0=>L_LEFT, O=>MDR11);
```

```
I_MDR12 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(13), D1=>MDL12, S0=>L_LEFT, O=>MDR12);

I_MDR13 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(14), D1=>MDL13, S0=>L_LEFT, O=>MDR13);

I_MDR14 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(15), D1=>MDL14, S0=>L_LEFT, O=>MDR14);

I_MDR15 : M2_1_MXILINX_test
   port map (D0=>SRI, D1=>MDL15, S0=>L_LEFT, O=>MDR15);

I_Q0 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR0, Q=>Q_DUMMY(0));

I_Q1 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR1, Q=>Q_DUMMY(1));

I_Q2 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR2, Q=>Q_DUMMY(2));

I_Q3 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR3, Q=>Q_DUMMY(3));

I_Q4 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR4, Q=>Q_DUMMY(4));

I_Q5 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR5, Q=>Q_DUMMY(5));

I_Q6 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR6, Q=>Q_DUMMY(6));

I_Q7 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR7, Q=>Q_DUMMY(7));

I_Q8 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR8, Q=>Q_DUMMY(8));

I_Q9 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR9, Q=>Q_DUMMY(9));

I_Q10 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR10, Q=>Q_DUMMY(10));

I_Q11 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR11, Q=>Q_DUMMY(11));
```

```
    I_Q12 : FDCE
       port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR12, Q=>Q_DUMMY(12));

    I_Q13 : FDCE
       port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR13, Q=>Q_DUMMY(13));

    I_Q14 : FDCE
       port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR14, Q=>Q_DUMMY(14));

    I_Q15 : FDCE
       port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR15, Q=>Q_DUMMY(15));

    I_36_145 : OR2
       port map (I0=>L, I1=>CE, O=>L_OR_CE);

    I_36_161 : OR2
       port map (I0=>LEFT, I1=>L, O=>L_LEFT);

 end BEHAVIORAL;
```

InitShift_Test.vhd

```
   -- Created by Guri Kristine Birkeland 2005

 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
 use UNISIM.VComponents.all;


LIBRARY ieee; USE ieee.std_logic_1164.ALL; USE
ieee.std_logic_unsigned.all; USE ieee.numeric_std.ALL;

ENTITY init_shift_test_vhd IS END init_shift_test_vhd;

ARCHITECTURE behavior OF init_shift_test_vhd IS

    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT init_shiftregs
    PORT(
```

```
        clk : IN std_logic;
        enable : IN std_logic;
        load : IN std_logic;
        tin : IN std_logic_vector(7 downto 0);
        tout : OUT std_logic
        );
    END COMPONENT;

    --Inputs
    SIGNAL clk :  std_logic := '0';
    SIGNAL enable :  std_logic := '0';
    SIGNAL load :  std_logic := '0';
    SIGNAL tin :  std_logic_vector(7 downto 0) := (others=>'0');

    --Outputs
    SIGNAL tout :  std_logic;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: init_shiftregs PORT MAP(
        clk => clk,
        enable => enable,
        load => load,
        tin => tin,
        tout => tout
    );

   PROCESS -- Process for clock
    BEGIN
        CLOCK_LOOP : LOOP
        clk <= transport '0';
        WAIT FOR 10 ns;
        clk <= transport '1';
        WAIT FOR 10 ns;
        WAIT FOR 40 ns;
        clk <= transport '0';
        WAIT FOR 40 ns;
        END LOOP CLOCK_LOOP;
    END PROCESS;

    tb : PROCESS
    BEGIN

        -- Wait 100 ns for global reset to finish
        wait for 100 ns;

        -- 1st
```

```
-- --------------------
enable <= transport '1';
load <= transport '0';
tin <= transport std_logic_vector'("00000001");
-- --------------------
WAIT FOR 100 ns; -- Time=100 ns
load <= transport '1';
-- --------------------
WAIT FOR 100 ns; -- Time=200 ns
load <= transport '0';
-- --------------------
--
WAIT FOR 1400 ns; -- Time=1500 ns
load <= transport '1';
tin <= transport std_logic_vector'("00111111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11111100");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("01111111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11111110");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11111100");
-- --------------------
```

```
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11111001");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("10011111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00111111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("01111111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11100111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00000011");
-- --------------------
```

```
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00000110");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("01100000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11000000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("10000000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00011000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00100100");
-- --------------------
```

```
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("01000010");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("10000001");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00000000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11100000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00000111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1500 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11110000");
-- --------------------
```

```
        WAIT FOR 100 ns;
        load <= transport '0';
        -- --------------------
        --
        WAIT FOR 1500 ns;
        load <= transport '1';
        tin <= transport std_logic_vector'("00001111");
        -- --------------------
        WAIT FOR 100 ns;
        load <= transport '0';
        -- --------------------
        --
        WAIT FOR 1500 ns;
        load <= transport '1';
        tin <= transport std_logic_vector'("00011111");
        -- --------------------
        WAIT FOR 100 ns;
        load <= transport '0';
        -- --------------------
        --
        WAIT FOR 1500 ns;
        load <= transport '1';
        tin <= transport std_logic_vector'("11111111");
        -- --------------------
        WAIT FOR 100 ns;
        load <= transport '0';
        -- --------------------
        --
        WAIT FOR 1500 ns;
        load <= transport '1';
        tin <= transport std_logic_vector'("11111000");
        -- --------------------
        WAIT FOR 100 ns;
        load <= transport '0';
        -- --------------------
        --
        WAIT FOR 1500 ns;
        -- --------------------
        -- end waiting
        WAIT FOR 1000 ns;
        -- --------------------

-- accumulated runtime is: 45900 ns

    END PROCESS;

END;
```

C.3.3. CSD4 design

The files needed to produce the CSD4 design of architecture 2 is listed below.

Add5And7.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity add5and7 is
    Port ( clock : in std_logic;
                enable :  in std_logic;
                tin : in std_logic_vector(3 downto 0);
           tout : out std_logic
                );
end add5and7;

architecture Behavioral of add5and7 is

component ARCH2_CSD4_5
    Port ( clock : in std_logic;
                enable :  in std_logic;
                tin : in std_logic_vector(3 downto 0);
           tout : out std_logic
                );
end component;

component ARCH2_CSD4_7
    Port ( clock : in std_logic;
                enable :  in std_logic;
                tin : in std_logic_vector(3 downto 0);
           tout : out std_logic
                );
end component;

component SUB2
   port (
```

```
      a:  in std_logic;
      b:  in std_logic;
      bin : in std_logic;
      q: out std_logic;
      bout : out std_logic
   );
end component;

component ADD2
  port
  (
      a:  in std_logic;
      b:  in std_logic;
      cii : in std_logic;
      s: out std_logic;
      coo : out std_logic
  );
end component;

signal f_out0: std_logic;
signal f_out1: std_logic;
signal f_out2: std_logic;
signal f_out3: std_logic;
signal out_f0: std_logic;
signal out_f1: std_logic;

signal s_out0: std_logic;
signal s_out1: std_logic;
signal s_out2: std_logic;
signal s_out3: std_logic;
signal out_s0: std_logic;
signal out_s1: std_logic;

signal cin0 : std_logic;
signal sign0 : std_logic;
signal cin_f : std_logic;
signal sign_f : std_logic;
signal cin_s : std_logic;
signal sign_s : std_logic;

signal total : std_logic;


begin

-- about number 5
-----------------------------------------------------------------------------------------------
addf_0 : ARCH2_CSD4_5
```

```
    port map(clock, enable, tin, f_out0);

flip_flop_f0 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>f_out0, Q=>f_out1);

flip_flop_f1 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>f_out1, Q=>f_out2);

flip_flop_f2 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>f_out2, Q=>f_out3);

add_f : ADD2
    port map(f_out1, f_out3, cin_f, out_f0, sign_f);

flip_flop_f3 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign_f, Q=>cin_f);

flip_flop_f4 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>out_f0, Q=>out_f1);

-- about number 7
-----------------------------------------------------------------------------------------
adds_0 : ARCH2_CSD4_7
    port map(clock, enable, tin, s_out0);

flip_flop_s0 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>s_out0, Q=>s_out1);

flip_flop_s1 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>s_out1, Q=>s_out2);

flip_flop_s2 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>s_out2, Q=>s_out3);

sub_s : SUB2
    port map(s_out3, s_out0, cin_s, out_s0, sign_s);

flip_flop_s3 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign_s, Q=>cin_s);

flip_flop_s4 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>out_s0, Q=>out_s1);

-- end adding

add_0 : ADD2
    port map(out_f1, out_s1, cin0, total, sign0);
```

```
flip_flop_0 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign0, Q=>cin0);

flip_flop_1 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>total, Q=>tout);

end Behavioral;
```

Add9And11.vhdl

```
   -- Created by Guri Kristine Birkeland 2005

 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
 use UNISIM.VComponents.all;

 entity add9and11 is
     Port ( clock : in std_logic;
                enable :  in std_logic;
                tin : in std_logic_vector(3 downto 0);
            tout : out std_logic
                );
 end add9and11;

 architecture Behavioral of add9and11 is

 component ARCH2_CSD4_9
     Port ( clock : in std_logic;
                enable :  in std_logic;
                tin : in std_logic_vector(3 downto 0);
            tout : out std_logic
                );
 end component;

 component ARCH2_CSD4_11
     Port ( clock : in std_logic;
                enable :  in std_logic;
                tin : in std_logic_vector(3 downto 0);
            tout : out std_logic
                );
```

```
end component;

component SUB2
   port (
     a:  in std_logic;
     b:  in std_logic;
     bin : in std_logic;
     q: out std_logic;
     bout : out std_logic
   );
end component;

component ADD2
  port
  (
     a:  in std_logic;
     b:  in std_logic;
     cii : in std_logic;
     s: out std_logic;
     coo : out std_logic
  );
end component;

signal f_out0: std_logic;
signal f_out1: std_logic;
signal f_out2: std_logic;
signal f_out3: std_logic;
signal out_f0: std_logic;
signal out_f1: std_logic;
signal out_f2: std_logic;

signal s_out0: std_logic;
signal s_out1: std_logic;
signal s_out2: std_logic;
signal s_out3: std_logic;
signal s_out4: std_logic;
signal out_s_00: std_logic;
signal out_s_01: std_logic;
signal out_s0: std_logic;
signal out_s1: std_logic;

signal cin0 : std_logic;
signal sign0 : std_logic;
signal cin_f : std_logic;
signal sign_f : std_logic;
signal cin_s_0: std_logic;
signal sign_s_0 : std_logic;
signal cin_s : std_logic;
```

```
 signal sign_s : std_logic;

 signal total : std_logic;


begin

-- about number 9
-------------------------------
addf_0 : ARCH2_CSD4_9
    port map(clock, enable, tin, f_out0);

flip_flop_f0 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>f_out0, Q=>f_out1);

flip_flop_f1 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>f_out1, Q=>f_out2);

flip_flop_f2 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>f_out2, Q=>f_out3);

add_f : ADD2
    port map(f_out0, f_out3, cin_f, out_f0, sign_f);

flip_flop_f3 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign_f, Q=>cin_f);

flip_flop_f4 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>out_f0, Q=>out_f1);

flip_flop_f5 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>out_f1, Q=>out_f2);


-- about number 11
-----------------------------------
adds_0 : ARCH2_CSD4_11
    port map(clock, enable, tin, s_out0);

flip_flop_s0 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>s_out0, Q=>s_out1);

add_s_0 : ADD2
    port map(s_out0, s_out1, cin_s_0, out_s_00, sign_s_0);

flip_flop_s1 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign_s_0, Q=>cin_s_0);
```

```
flip_flop_s2 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>out_s_00, Q=>out_s_01);


flip_flop_s3 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>s_out1, Q=>s_out2);

flip_flop_s4 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>s_out2, Q=>s_out3);

flip_flop_s5 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>s_out3, Q=>s_out4);

add_s : ADD2
    port map(out_s_01, s_out4, cin_s, out_s0, sign_s);

flip_flop_s6 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign_s, Q=>cin_s);

flip_flop_s7 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>out_s0, Q=>out_s1);

-- end adding

add_0 : ADD2
    port map(out_f2, out_s1, cin0, total, sign0);

flip_flop_0 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign0, Q=>cin0);

flip_flop_1 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>total, Q=>tout);

end Behavioral;
```

Add13And15.vhdl

```
   -- Created by Guri Kristine Birkeland 2005

 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
```

```
library UNISIM;
use UNISIM.VComponents.all;

entity add13and15 is
    Port ( clock : in std_logic;
                enable :  in std_logic;
                tin : in std_logic_vector(3 downto 0);
           tout : out std_logic
                );
end add13and15;

architecture Behavioral of add13and15 is

component ARCH2_CSD4_13
    Port ( clock : in std_logic;
                enable :  in std_logic;
                tin : in std_logic_vector(3 downto 0);
           tout : out std_logic
                );
end component;

component ARCH2_CSD4_15
    Port ( clock : in std_logic;
                enable :  in std_logic;
                tin : in std_logic_vector(3 downto 0);
           tout : out std_logic
                );
end component;

component SUB2
   port (
     a:  in std_logic;
     b:  in std_logic;
     bin : in std_logic;
     q: out std_logic;
     bout : out std_logic
   );
end component;

component ADD2
  port
  (
     a:  in std_logic;
     b:  in std_logic;
     cii : in std_logic;
     s: out std_logic;
     coo : out std_logic
  );
```

```
 end component;

 signal f_out0: std_logic;
 signal f_out1: std_logic;
 signal f_out2: std_logic;
 signal f_out3: std_logic;
 signal f_out4: std_logic;
 signal out_f_00: std_logic;
 signal out_f_01: std_logic;
 signal out_f0: std_logic;
 signal out_f1: std_logic;

 signal s_out0: std_logic;
 signal s_out1: std_logic;
 signal s_out2: std_logic;
 signal s_out3: std_logic;
 signal s_out4: std_logic;
 signal out_s0: std_logic;
 signal out_s1: std_logic;

 signal cin0 : std_logic;
 signal sign0 : std_logic;
 signal cin_f_0: std_logic;
 signal sign_f_0 : std_logic;
 signal cin_f : std_logic;
 signal sign_f : std_logic;
 signal cin_s : std_logic;
 signal sign_s : std_logic;

 signal total : std_logic;


begin

-- about number 13
---------------------------------------
addf_0 : ARCH2_CSD4_13
    port map(clock, enable, tin, f_out0);

flip_flop_f0 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>f_out0, Q=>f_out1);

flip_flop_f1 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>f_out1, Q=>f_out2);

add_f_0 : ADD2
    port map(f_out0, f_out2, cin_f_0, out_f_00, sign_f_0);
```

```
flip_flop_f2 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign_f_0, Q=>cin_f_0);

flip_flop_f3 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>out_f_00, Q=>out_f_01);

flip_flop_f4 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>f_out2, Q=>f_out3);

flip_flop_f5 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>f_out3, Q=>f_out4);

add_f_1 : ADD2
    port map(out_f_01, f_out4, cin_f, out_f0, sign_f);

flip_flop_f6 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign_f, Q=>cin_f);

flip_flop_f7 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>out_f0, Q=>out_f1);

-- about number 15
----------------------------------------
adds_0 : ARCH2_CSD4_15
    port map(clock, enable, tin, s_out0);

flip_flop_s0 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>s_out0, Q=>s_out1);

flip_flop_s1 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>s_out1, Q=>s_out2);

flip_flop_s2 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>s_out2, Q=>s_out3);

flip_flop_s3 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>s_out3, Q=>s_out4);

sub_s : SUB2
    port map(s_out4, s_out0, cin_s, out_s0, sign_s);

flip_flop_s4 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign_s, Q=>cin_s);

flip_flop_s5 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>out_s0, Q=>out_s1);
```

```
-- end adding add_0 : ADD2
    port map(out_f1, out_s1, cin0, total, sign0);

flip_flop_0 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign0, Q=>cin0);

flip_flop_1 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>total, Q=>tout);

end Behavioral;
```

AddAll.vhdl

```
   -- Created by Guri Kristine Birkeland 2005

 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
 use UNISIM.VComponents.all;

 entity addAll is
    port (
         clock : in std_logic;
         enable : in std_logic;
         tin0 : in  std_logic;
         O : out    std_logic
         );
 end addAll;

 architecture Behavioral of addAll is

 component add2
   port
   (
      a:  in std_logic;
      b:  in std_logic;
      cii : in std_logic;
      s: out std_logic;
      coo : out std_logic
   );
 end component;
```

```
 begin

 O<=tin0;

 end Behavioral;
```

AddAndFlipFlop.vhdl

```
   -- Created by Guri Kristine Birkeland 2005

 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
 use UNISIM.VComponents.all;

entity add_ff is
   port ( clock   : in    std_logic;
          CE  : in    std_logic;
          CLR : in    std_logic;
            afi0 : in std_logic_vector(5 downto 0);
        afso : out std_logic);
end add_ff;

architecture BEHAVIORAL of add_ff is

component add16bit
    port ( A   : in    std_logic_vector (15 downto 0);
           B   : in    std_logic_vector (15 downto 0);
           CI  : in    std_logic;
           CO  : out   std_logic;
           OFL : out   std_logic;
           S   : out   std_logic_vector (15 downto 0));
end component;

component flipflop
   port ( C   : in    std_logic;
          CE  : in    std_logic;
          CLR : in    std_logic;
          D   : in    std_logic_vector (6 downto 0);
          Q   : out   std_logic_vector (5 downto 0);
```

```
        so : out std_logic);
end component;

signal outadder : std_logic_vector(6 downto 0); signal inadder :
std_logic_vector(5 downto 0);

signal dummy0 : std_logic; signal dummy1 : std_logic_vector(8
downto 0); signal dummy2 : std_logic;

begin

  adder : add16bit
    port map ( A(5 downto 0) => afi0, A(15 downto 6)=>"0000000000",
              B(5 downto 0) => inadder, B(15 downto 6)=>"0000000000",
              CI => '0', CO=> dummy0, OFL => dummy2, S(6 downto 0)=> outadder,
              S(15 downto 7)=> dummy1);

  ff : flipflop
    port map (C => clock, CE => CE, CLR => CLR, D => outadder, Q => inadder,
             so => afso);

 end BEHAVIORAL;
```

ARCH2_CSD4.vhdl

```
   -- Created by Guri Kristine Birkeland 2005

 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
 use UNISIM.VComponents.all;

entity ARCH2_CSD4 is
     Port ( clock : in std_logic;
          enable :  in std_logic;
          tin : in std_logic_vector(3 downto 0);
        tout : out std_logic
          );
end ARCH2_CSD4;

architecture Behavioral of ARCH2_CSD4 is
```

```
component add5and7
    Port ( clock : in std_logic;
               enable :  in std_logic;
               tin : in std_logic_vector(3 downto 0);
           tout : out std_logic
               );
end component;

component add9and11
    Port ( clock : in std_logic;
               enable :  in std_logic;
               tin : in std_logic_vector(3 downto 0);
           tout : out std_logic
               );
end component;

component add13and15
    Port ( clock : in std_logic;
               enable :  in std_logic;
               tin : in std_logic_vector(3 downto 0);
           tout : out std_logic
               );
end component;

component ADD2
   port (
     a:  in std_logic;
     b:  in std_logic;
     cii : in std_logic;
     s: out std_logic;
     coo : out std_logic
   );
end component;

signal partsum0: std_logic; signal partsum01: std_logic; signal
partsum1: std_logic; signal partsum2: std_logic; signal partsum21:
std_logic;

signal sum0: std_logic; signal sum01: std_logic;

signal cin0, cin1 : std_logic; signal sign0, sign1 : std_logic;
signal total : std_logic;

begin

addand_0 : add5and7
    port map(clock, enable, tin, partsum0);
```

```
addand_1 : add9and11
    port map(clock, enable, tin, partsum1);

addand_2 : add13and15
    port map(clock, enable, tin, partsum2);

flip_flop_00 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>partsum0, Q=>partsum01);

add_0 : ADD2
    port map( partsum01, partsum1, cin0, sum0, sign0);

flip_flop_0 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign0, Q=>cin0);

flip_flop_01 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sum0, Q=>sum01);

flip_flop_02 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>partsum2, Q=>partsum21);

add_1 : ADD2
    port map( partsum21, sum01, cin1, total, sign1);

flip_flop_1 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign1, Q=>cin1);

flip_flop_2 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>total, Q=>tout);

end Behavioral;
```

Arch2_CSD4_5.vhdl

```
   -- Created by Guri Kristine Birkeland 2005

 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
 use UNISIM.VComponents.all;
```

```
entity ARCH2_CSD4_5 is
    Port ( clock : in std_logic;
         enable :  in std_logic;
         tin : in std_logic_vector(3 downto 0);
       tout : out std_logic
         );
end ARCH2_CSD4_5;

architecture Behavioral of ARCH2_CSD4_5 is

component biglut_5
    Port ( clock : in std_logic;
              enable :  in std_logic;
              tin : in std_logic_vector(3 downto 0);
           posPartsum0 : out std_logic;
           negPartsum0 : out std_logic);
end component;

component endAddSub
   port (
        clock : in std_logic;
       enable :  in std_logic;
        pos0 : in std_logic;
        neg0 : in std_logic;
        sum : out std_logic
        );
end component;

signal posPartsum0 : std_logic;
signal negPartsum0 : std_logic;

begin

biglut_0 : biglut_5
   port map (clock, enable, tin, posPartsum0,
       negPartsum0);

endAddSub_0 : endAddSub
   port map (clock, enable, posPartsum0,
       negPartsum0, tout);

end Behavioral;
```

Arch2_CSD4_7.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity ARCH2_CSD4_7 is
    Port ( clock : in std_logic;
         enable :  in std_logic;
         tin : in std_logic_vector(3 downto 0);
       tout : out std_logic
         );
end ARCH2_CSD4_7;

architecture Behavioral of ARCH2_CSD4_7 is

component biglut_7
    Port ( clock : in std_logic;
              enable :  in std_logic;
              tin : in std_logic_vector(3 downto 0);
           posPartsum0 : out std_logic;
           negPartsum0 : out std_logic);
end component;

component endAddSub
   port (
        clock : in std_logic;
       enable :  in std_logic;
        pos0 : in std_logic;
        neg0 : in std_logic;
        sum : out std_logic
        );
end component;

signal posPartsum0 : std_logic;
signal negPartsum0 : std_logic;

begin

biglut_0 : biglut_7
   port map (clock, enable, tin, posPartsum0,
       negPartsum0);
```

```
endAddSub_0 : endAddSub
   port map (clock, enable, posPartsum0,
       negPartsum0, tout);

end Behavioral;
```

Arch2_CSD4_9.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity ARCH2_CSD4_9 is
    Port ( clock : in std_logic;
         enable :  in std_logic;
         tin : in std_logic_vector(3 downto 0);
       tout : out std_logic
         );
end ARCH2_CSD4_9;

architecture Behavioral of ARCH2_CSD4_9 is

component biglut_9
    Port ( clock : in std_logic;
              enable :  in std_logic;
              tin : in std_logic_vector(3 downto 0);
           posPartsum0 : out std_logic;
           negPartsum0 : out std_logic);
end component;

component endAddSub
   port (
        clock : in std_logic;
       enable :  in std_logic;
        pos0 : in std_logic;
        neg0 : in std_logic;
        sum : out std_logic
```

```
        );
end component;

signal posPartsum0 : std_logic;
signal negPartsum0 : std_logic;

begin

biglut_0 : biglut_9
   port map (clock, enable, tin, posPartsum0,
       negPartsum0);

endAddSub_0 : endAddSub
   port map (clock, enable, posPartsum0,
       negPartsum0, tout);

end Behavioral;
```

Arch2_CSD4_11.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity ARCH2_CSD4_11 is
    Port ( clock : in std_logic;
         enable :  in std_logic;
         tin : in std_logic_vector(3 downto 0);
       tout : out std_logic
         );
end ARCH2_CSD4_11;

architecture Behavioral of ARCH2_CSD4_11 is

component biglut_11
    Port ( clock : in std_logic;
              enable :  in std_logic;
              tin : in std_logic_vector(3 downto 0);
```

```
           posPartsum0 : out std_logic;
           negPartsum0 : out std_logic);
end component;

component endAddSub
   port (
        clock : in std_logic;
       enable :  in std_logic;
        pos0 : in std_logic;
        neg0 : in std_logic;
        sum : out std_logic
        );
end component;

signal posPartsum0 : std_logic;
signal negPartsum0 : std_logic;

begin

biglut_0 : biglut_11
   port map (clock, enable, tin, posPartsum0,
       negPartsum0);

endAddSub_0 : endAddSub
   port map (clock, enable, posPartsum0,
       negPartsum0, tout);

end Behavioral;
```

Arch2_CSD4_13.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity ARCH2_CSD4_13 is
    Port ( clock : in std_logic;
         enable :  in std_logic;
```

```
         tin : in std_logic_vector(3 downto 0);
       tout : out std_logic
         );
end ARCH2_CSD4_13;

architecture Behavioral of ARCH2_CSD4_13 is

component biglut_13
    Port ( clock : in std_logic;
              enable :  in std_logic;
              tin : in std_logic_vector(3 downto 0);
           posPartsum0 : out std_logic;
           negPartsum0 : out std_logic);
end component;

component endAddSub
   port (
        clock : in std_logic;
       enable :  in std_logic;
        pos0 : in std_logic;
        neg0 : in std_logic;
        sum : out std_logic
        );
end component;

signal posPartsum0 : std_logic;
signal negPartsum0 : std_logic;

begin

biglut_0 : biglut_13
   port map (clock, enable, tin, posPartsum0,
       negPartsum0);

endAddSub_0 : endAddSub
   port map (clock, enable, posPartsum0,
       negPartsum0, tout);

end Behavioral;
```

Arch2_CSD4_15.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity ARCH2_CSD4_15 is
    Port ( clock : in std_logic;
         enable :  in std_logic;
         tin : in std_logic_vector(3 downto 0);
       tout : out std_logic
         );
end ARCH2_CSD4_15;

architecture Behavioral of ARCH2_CSD4_15 is

component biglut_15
    Port ( clock : in std_logic;
              enable :  in std_logic;
              tin : in std_logic_vector(3 downto 0);
           posPartsum0 : out std_logic;
           negPartsum0 : out std_logic);
end component;

component endAddSub
   port (
        clock : in std_logic;
       enable :  in std_logic;
        pos0 : in std_logic;
        neg0 : in std_logic;
        sum : out std_logic
        );
end component;

signal posPartsum0 : std_logic;
signal negPartsum0 : std_logic;

begin

biglut_0 : biglut_15
   port map (clock, enable, tin, posPartsum0,
       negPartsum0);

endAddSub_0 : endAddSub
   port map (clock, enable, posPartsum0,
       negPartsum0, tout);
```

```
end Behavioral;
```

BigLut_5.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity biglut_5 is
    Port ( clock : in std_logic;
              enable :  in std_logic;
              tin : in std_logic_vector(3 downto 0);
           posPartsum0 : out std_logic;
           negPartsum0 : out std_logic);
end biglut_5;

architecture Behavioral of biglut_5 is

component lut_5_0
    Port ( clock : in std_logic;
              enable :  in std_logic;
              ti : in std_logic_vector(3 downto 0);
           toutpos : out std_logic;
              toutneg : out std_logic
       );
 end component;

begin

lut_00 : lut_5_0
   port map (clock, enable, tin(3 downto 0), posPartsum0, negPartsum0);

end Behavioral;
```

BigLut_7.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity biglut_7 is
    Port ( clock : in std_logic;
              enable :  in std_logic;
              tin : in std_logic_vector(3 downto 0);
           posPartsum0 : out std_logic;
           negPartsum0 : out std_logic);
end biglut_7;

architecture Behavioral of biglut_7 is

component lut_7_0
    Port ( clock : in std_logic;
              enable :  in std_logic;
              ti : in std_logic_vector(3 downto 0);
           toutpos : out std_logic;
              toutneg : out std_logic
       );
 end component;

begin

lut_00 : lut_7_0
   port map (clock, enable, tin(3 downto 0), posPartsum0, negPartsum0);

end Behavioral;
```

BigLut_9.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity biglut_9 is
    Port ( clock : in std_logic;
              enable :  in std_logic;
              tin : in std_logic_vector(3 downto 0);
           posPartsum0 : out std_logic;
           negPartsum0 : out std_logic);
end biglut_9;

architecture Behavioral of biglut_9 is

component lut_9_0
    Port ( clock : in std_logic;
              enable :  in std_logic;
              ti : in std_logic_vector(3 downto 0);
           toutpos : out std_logic;
              toutneg : out std_logic
       );
 end component;

begin

lut_00 : lut_9_0
   port map (clock, enable, tin(3 downto 0), posPartsum0, negPartsum0);

end Behavioral;
```

BigLut_11.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;
```

```
entity biglut_11 is
    Port ( clock : in std_logic;
              enable :  in std_logic;
              tin : in std_logic_vector(3 downto 0);
           posPartsum0 : out std_logic;
           negPartsum0 : out std_logic);
end biglut_11;

architecture Behavioral of biglut_11 is

component lut_11_0
    Port ( clock : in std_logic;
              enable :  in std_logic;
              ti : in std_logic_vector(3 downto 0);
           toutpos : out std_logic;
              toutneg : out std_logic
       );
 end component;

begin

lut_00 : lut_11_0
   port map (clock, enable, tin(3 downto 0), posPartsum0, negPartsum0);

end Behavioral;
```

BigLut_13.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity biglut_13 is
    Port ( clock : in std_logic;
              enable :  in std_logic;
              tin : in std_logic_vector(3 downto 0);
           posPartsum0 : out std_logic;
           negPartsum0 : out std_logic);
```

```
end biglut_13;

architecture Behavioral of biglut_13 is

component lut_13_0
    Port ( clock : in std_logic;
              enable :  in std_logic;
              ti : in std_logic_vector(3 downto 0);
           toutpos : out std_logic;
              toutneg : out std_logic
       );
 end component;

begin

lut_00 : lut_13_0
   port map (clock, enable, tin(3 downto 0), posPartsum0, negPartsum0);

end Behavioral;
```

BigLut_15.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity biglut_15 is
    Port ( clock : in std_logic;
              enable :  in std_logic;
              tin : in std_logic_vector(3 downto 0);
           posPartsum0 : out std_logic;
           negPartsum0 : out std_logic);
end biglut_15;

architecture Behavioral of biglut_15 is

component lut_15_0
    Port ( clock : in std_logic;
```

```
              enable :  in std_logic;
              ti : in std_logic_vector(3 downto 0);
           toutpos : out std_logic;
              toutneg : out std_logic
       );
 end component;

 begin

 lut_00 : lut_15_0
    port map (clock, enable, tin(3 downto 0), posPartsum0, negPartsum0);

 end Behavioral;
```

EndAddSub.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
 use UNISIM.VComponents.all;

 entity endAddSub is
    port (
             clock : in std_logic;
             enable : in std_logic;
             pos0 : in std_logic;
             neg0 : in std_logic;
             sum : out std_logic
             );
 end endAddSub;

 architecture Behavioral of endAddSub is

component SUB2
    PORT(
              A:  IN STD_LOGIC;
          B:  IN STD_LOGIC;
          bin : in std_logic;
             Q:  OUT STD_LOGIC;
```

```
          bout :   OUT STD_LOGIC
        );
END component;

 component addAll
    port (
         clock : in std_logic;
         enable : in std_logic;
         tin0 : in  std_logic;
         O : out    std_logic
         );
 end component;

 signal sign0, bin0, sum0 : std_logic;

 signal posSum : std_logic;
 signal negSum : std_logic;

 begin

 posAdd_0 : addAll
    port map (clock, enable, pos0, posSum);

 negAdd_0 : addAll
    port map (clock, enable, neg0, negSum);

 sub_0 : sub2
    port map (posSum, negSum, bin0, sum0, sign0);

 flip_flop_0 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign0, Q=>bin0);

 flip_flop_1 : FDCE
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sum0, Q=>sum);

 end Behavioral;
```

FlipFlop.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
 use UNISIM.VComponents.all;

entity flipflop is
   port ( C   : in    std_logic;
          CE  : in    std_logic;
          CLR : in    std_logic;
          D   : in    std_logic_vector (6 downto 0);
          Q   : out   std_logic_vector (5 downto 0);
         so : out std_logic);
end flipflop;

architecture BEHAVIORAL of flipflop is

   attribute INIT       : STRING ;
   attribute BOX_TYPE   : STRING ;
   component FDCE
      -- synopsys translate_off
      generic( INIT : bit :=  '0');
      -- synopsys translate_on
      port ( C   : in    std_logic;
             CE  : in    std_logic;
             CLR : in    std_logic;
             D   : in    std_logic;
             Q   : out   std_logic);
   end component;
   attribute INIT of FDCE : COMPONENT is "0";
   attribute BOX_TYPE of FDCE : COMPONENT is "BLACK_BOX";

begin

   I_Q0 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(0), Q=>so);

   I_Q1 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(1), Q=>Q(0));

   I_Q2 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(2), Q=>Q(1));

   I_Q3 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(3), Q=>Q(2));

   I_Q4 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(4), Q=>Q(3));
```

```
   I_Q5 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(5), Q=>Q(4));

   I_Q6 : FDCE
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(6), Q=>Q(5));

end BEHAVIORAL;
```

Init_ShiftRegisters.vhdl

```
   -- Created by Guri Kristine Birkeland 2005

 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

 --  Uncomment the following lines to use the declarations that are
 --  provided for instantiating Xilinx primitive components.
 library UNISIM;
 use UNISIM.VComponents.all;

 entity INIT_SHIFTREGS is
     Port ( clk, enable, load : in std_logic;
          tin : in std_logic_vector(7 downto 0);
          tout : out std_logic
          );
 end INIT_SHIFTREGS;

 architecture Behavioral of INIT_SHIFTREGS is

 component shiftreg14bit
    port ( C    : in    std_logic;        -- clock
           CE   : in    std_logic;        -- enable
           CLR  : in    std_logic;        -- clear
           D    : in    std_logic_vector (13 downto 0);  -- input
           L    : in    std_logic;        -- load
           LEFT : in    std_logic;        -- shift left
           SLI  : in    std_logic;        -- add to left 0/1
           SRI  : in    std_logic;        -- add to right 0/1
           Q    : out   std_logic_vector (13 downto 0));     -- output
 end component;

 component FDCE
      -- synopsys translate_off
      generic( INIT : bit :=  '0');
```

```
     -- synopsys translate_on
     port ( C   : in    std_logic;
            CE  : in    std_logic;
            CLR : in    std_logic;
            D   : in    std_logic;
            Q   : out   std_logic);
end component;

component ARCH2_CSD4
    Port ( clock : in std_logic;
         enable :  in std_logic;
         tin : in std_logic_vector(3 downto 0);
       tout : out std_logic
         );
end component;

signal shiftOut : std_logic_vector(3 downto 0);

signal out0 : std_logic_vector(12 downto 0);
signal out1 : std_logic_vector(12 downto 0);
signal out2 : std_logic_vector(12 downto 0);
signal out3 : std_logic_vector(12 downto 0);

signal Q0 : std_logic_vector(7 downto 0);
signal Q1 : std_logic_vector(7 downto 0);
signal Q2 : std_logic_vector(7 downto 0);

signal GND : std_logic := '0';

begin

shiftreg_0 : shiftreg14bit
   port map (C => clk, CE => enable, CLR => GND,
           D(5 downto 0) => "000000",
           D(13 downto 6) => tin,
           L => load, LEFT => GND, SLI => GND, SRI => GND,
           Q(0) => shiftOut(0), Q(13 downto 1) => out0
           );

-- flipflops to hold the value for next shiftregister.
flipflop_00 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>shiftOut(0), Q=>Q0(7));

flipflop_01 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(7), Q=>Q0(6));

flipflop_02 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(6), Q=>Q0(5));
```

```
flipflop_03 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(5), Q=>Q0(4));

flipflop_04 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(4), Q=>Q0(3));

flipflop_05 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(3), Q=>Q0(2));

flipflop_06 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(2), Q=>Q0(1));

flipflop_07 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q0(1), Q=>Q0(0));


shiftreg_1 : shiftreg14bit
   port map (C => clk, CE => enable, CLR => GND,
           D(5 downto 0) => "000000",
           D(13 downto 6) => Q0,
           L => load, LEFT => GND, SLI => GND, SRI => GND,
           Q(0) => shiftOut(1), Q(13 downto 1) => out1
           );
flipflop_10 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>shiftOut(1), Q=>Q1(7));

flipflop_11 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(7), Q=>Q1(6));

flipflop_12 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(6), Q=>Q1(5));

flipflop_13 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(5), Q=>Q1(4));

flipflop_14 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(4), Q=>Q1(3));

flipflop_15 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(3), Q=>Q1(2));

flipflop_16 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(2), Q=>Q1(1));

flipflop_17 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q1(1), Q=>Q1(0));
```

```
shiftreg_2 : shiftreg14bit
   port map (C => clk, CE => enable, CLR => GND,
           D(5 downto 0) => "000000",
           D(13 downto 6) => Q1,
           L => load, LEFT => GND, SLI => GND, SRI => GND,
           Q(0) => shiftOut(2), Q(13 downto 1) => out2
           );
flipflop_20 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>shiftOut(2), Q=>Q2(7));

flipflop_21 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(7), Q=>Q2(6));

flipflop_22 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(6), Q=>Q2(5));

flipflop_23 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(5), Q=>Q2(4));

flipflop_24 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(4), Q=>Q2(3));

flipflop_25 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(3), Q=>Q2(2));

flipflop_26 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(2), Q=>Q2(1));

flipflop_27 :  FDCE
     port map (C=>clk, CE=>enable, CLR=>GND, D=>Q2(1), Q=>Q2(0));


shiftreg_3 : shiftreg14bit
   port map (C => clk, CE => enable, CLR => GND,
           D(5 downto 0) => "000000",
           D(13 downto 6) => Q2,
           L => load, LEFT => GND, SLI => GND, SRI => GND,
           Q(0) => shiftOut(3), Q(13 downto 1) => out3
           );

arch0 : ARCH2_CSD4
    port map (clock=>clk, enable=>enable,
                  tin=>shiftOut, tout=>tout
         );

end Behavioral;
```

Lut6Bits.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity lut is
    generic (              INIT0 : bit_vector := x"16";
               INIT1 : bit_vector := x"16";
               INIT2 : bit_vector := x"16";
               INIT3 : bit_vector := x"16";
               INIT4 : bit_vector := x"16";
               INIT5 : bit_vector := x"16");
   Port ( ui0 : in std_logic;
           ui1 : in std_logic;
           ui2 : in std_logic;
           ui3 : in std_logic;
           uo : out std_logic_vector(5 downto 0));
end lut;

architecture Behavioral of lut is
       -- Component Declaration for LUT4 should be placed
-- after architecture statement but before begin keyword
component LUT4
   generic (INIT : bit_vector := x"16");
   port (O : out STD_LOGIC;
           I0 : in STD_LOGIC;
           I1 : in STD_LOGIC;
           I2 : in STD_LOGIC;
           I3 : in STD_LOGIC);
end component;

-- Component Attribute specification for LUT4
-- should be placed after architecture declaration but
-- before the begin keyword
attribute INIT : string;
begin
LUT4_p0 : LUT4 generic map (INIT => INIT0)
   port map (O => uO(0),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
```

```
LUT4_p1 : LUT4 generic map (INIT => INIT1)
   port map (O => uO(1),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p2 : LUT4 generic map (INIT => INIT2)
   port map (O => uO(2),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p3 : LUT4 generic map (INIT => INIT3)
   port map (O => uO(3),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p4 : LUT4 generic map (INIT => INIT4)
   port map (O => uO(4),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);
LUT4_p5 : LUT4 generic map (INIT => INIT5)
   port map (O => uO(5),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3);

end Behavioral;
```

Lut_50.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity lut_5_0 is
    Port ( clock : in std_logic;
           enable :  in std_logic;
           ti : in std_logic_vector(3 downto 0);
           toutpos : out std_logic;
           toutneg : out std_logic
           );
end lut_5_0;

architecture Behavioral of lut_5_0 is

component lut
    generic (
            INIT0 : bit_vector := x"16";
            INIT1 : bit_vector := x"16";
            INIT2 : bit_vector := x"16";
            INIT3 : bit_vector := x"16";
            INIT4 : bit_vector := x"16";
            INIT5 : bit_vector := x"16");
```

```
   port (
           UI0   : in    std_logic;
           UI1   : in    std_logic;
           UI2   : in    std_logic;
           UI3   : in    std_logic;
           UO    : out   std_logic_vector(5 downto 0)
       );
end component;

component add_ff
  port ( clock   : in    std_logic;
         CE  : in    std_logic;
         CLR : in    std_logic;
            afi0 : in std_logic_vector(5 downto 0);
            afso : out std_logic
            );
end component;

signal outpos : std_logic_vector(5 downto 0);
signal outneg : std_logic_vector(5 downto 0);

begin

-- lut
poslut_0 : lut
   generic map (
           INIT0=>"0000000010000000",
           INIT1=>"0000000000100000",
           INIT2=>"1000000000000000",
           INIT3=>"0000000000000000",
           INIT4=>"0001000001000000",
           INIT5=>"0000000000000000")
   port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),
             UO=>outpos);

neglut_0 : lut
   generic map (
           INIT0=>"0000000001000000",
           INIT1=>"0000000100000000",
           INIT2=>"0000000000000000",
           INIT3=>"0000000000000000",
           INIT4=>"0000000000000000",
           INIT5=>"0000000000000000")
   port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),
             UO=>outneg);

 add_ff_0: add_ff
   port map (clock, enable, '0', outpos, toutpos);
```

```
 add_ff_1: add_ff
   port map (clock, enable, '0', outneg, toutneg);

end Behavioral;
```

Lut_70.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity lut_7_0 is
    Port ( clock : in std_logic;
           enable :  in std_logic;
           ti : in std_logic_vector(3 downto 0);
           toutpos : out std_logic;
           toutneg : out std_logic
           );
end lut_7_0;

architecture Behavioral of lut_7_0 is


component lut
    generic (
            INIT0 : bit_vector := x"16";
            INIT1 : bit_vector := x"16";
            INIT2 : bit_vector := x"16";
            INIT3 : bit_vector := x"16";
            INIT4 : bit_vector := x"16";
            INIT5 : bit_vector := x"16");
   port (
           UI0   : in    std_logic;
           UI1   : in    std_logic;
           UI2   : in    std_logic;
           UI3   : in    std_logic;
           UO   : out   std_logic_vector(5 downto 0)
```

```
      );
end component;

component add_ff
  port ( clock   : in    std_logic;
         CE  : in    std_logic;
         CLR : in    std_logic;
            afi0 : in std_logic_vector(5 downto 0);
            afso : out std_logic
            );
end component;

signal outpos : std_logic_vector(5 downto 0);
signal outneg : std_logic_vector(5 downto 0);

begin

-- lut
poslut_0 : lut
   generic map (
           INIT0=>"0000000000000000",
           INIT1=>"0000000000000000",
           INIT2=>"0000000000000010",
           INIT3=>"0000000000000000",
           INIT4=>"0010110010000000",
           INIT5=>"0000000000000000")
   port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),
             UO=>outpos);

neglut_0 : lut
   generic map (
           INIT0=>"0000000000000010",
           INIT1=>"0000000000001000",
           INIT2=>"0000000000000000",
           INIT3=>"0000000000000000",
           INIT4=>"0000000000000000",
           INIT5=>"0000000000000000")
   port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),
             UO=>outneg);

 add_ff_0: add_ff
   port map (clock, enable, '0', outpos, toutpos);

 add_ff_1: add_ff
   port map (clock, enable, '0', outneg, toutneg);

end Behavioral;
```

Lut_90.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity lut_9_0 is
    Port ( clock : in std_logic;
           enable :  in std_logic;
           ti : in std_logic_vector(3 downto 0);
           toutpos : out std_logic;
           toutneg : out std_logic
           );
end lut_9_0;

architecture Behavioral of lut_9_0 is


component lut
    generic (
            INIT0 : bit_vector := x"16";
            INIT1 : bit_vector := x"16";
            INIT2 : bit_vector := x"16";
            INIT3 : bit_vector := x"16";
            INIT4 : bit_vector := x"16";
            INIT5 : bit_vector := x"16");
   port (
           UI0   : in    std_logic;
           UI1   : in    std_logic;
           UI2   : in    std_logic;
           UI3   : in    std_logic;
           UO   : out   std_logic_vector(5 downto 0)
       );
end component;

component add_ff
  port ( clock   : in    std_logic;
         CE  : in    std_logic;
         CLR : in    std_logic;
```

```
            afi0 : in std_logic_vector(5 downto 0);
            afso : out std_logic
            );
end component;

signal outpos : std_logic_vector(5 downto 0);
signal outneg : std_logic_vector(5 downto 0);

begin

-- lut
poslut_0 : lut
   generic map (
           INIT0=>"0000000000000000",
           INIT1=>"0000000000000000",
           INIT2=>"0001000000000000",
           INIT3=>"0000000000010000",
           INIT4=>"1100000000000000",
           INIT5=>"0000000000000000")
   port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),
             UO=>outpos);

neglut_0 : lut
   generic map (
           INIT0=>"0000000000000000",
           INIT1=>"0100000000000000",
           INIT2=>"0000010000000000",
           INIT3=>"0000000000000000",
           INIT4=>"0000000000000000",
           INIT5=>"0000000000000000")
   port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),
             UO=>outneg);

 add_ff_0: add_ff
   port map (clock, enable, '0', outpos, toutpos);

 add_ff_1: add_ff
   port map (clock, enable, '0', outneg, toutneg);

end Behavioral;
```

Lut_110.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity lut_11_0 is
    Port ( clock : in std_logic;
                enable :  in std_logic;
                ti : in std_logic_vector(3 downto 0);
              toutpos : out std_logic;
                toutneg : out std_logic
                );
end lut_11_0;

architecture Behavioral of lut_11_0 is


component lut
    generic (
             INIT0 : bit_vector := x"16";
             INIT1 : bit_vector := x"16";
             INIT2 : bit_vector := x"16";
             INIT3 : bit_vector := x"16";
             INIT4 : bit_vector := x"16";
             INIT5 : bit_vector := x"16");
   port (
             UI0   : in    std_logic;
             UI1   : in    std_logic;
             UI2   : in    std_logic;
             UI3   : in    std_logic;
             UO    : out   std_logic_vector(5 downto 0)
       );
end component;

component add_ff
  port ( clock   : in    std_logic;
          CE  : in    std_logic;
          CLR : in    std_logic;
             afi0 : in std_logic_vector(5 downto 0);
             afso : out std_logic
             );
end component;

signal outpos : std_logic_vector(5 downto 0);
```

```
signal outneg : std_logic_vector(5 downto 0);

begin

-- lut
poslut_0 : lut
   generic map (
           INIT0=>"0001000000000000",
           INIT1=>"0000000000010000",
           INIT2=>"0000000001000000",
           INIT3=>"0000000000100000",
           INIT4=>"0000000000000000",
           INIT5=>"0000000000000000")
   port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),
             UO=>outpos);

neglut_0 : lut
   generic map (
           INIT0=>"0010000000000000",
           INIT1=>"0000101000000000",
           INIT2=>"0000000000000000",
           INIT3=>"0000000000000000",
           INIT4=>"0000000000000000",
           INIT5=>"0000000000000000")
   port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),
             UO=>outneg);

 add_ff_0: add_ff
   port map (clock, enable, '0', outpos, toutpos);

 add_ff_1: add_ff
   port map (clock, enable, '0', outneg, toutneg);

end Behavioral;
```

Lut_130.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
```

```
library UNISIM;
use UNISIM.VComponents.all;

entity lut_13_0 is
    Port ( clock : in std_logic;
                enable :  in std_logic;
                ti : in std_logic_vector(3 downto 0);
           toutpos : out std_logic;
                toutneg : out std_logic
                );
end lut_13_0;

architecture Behavioral of lut_13_0 is


component lut
    generic (
             INIT0 : bit_vector := x"16";
             INIT1 : bit_vector := x"16";
             INIT2 : bit_vector := x"16";
             INIT3 : bit_vector := x"16";
             INIT4 : bit_vector := x"16";
             INIT5 : bit_vector := x"16");
   port (
            UI0   : in    std_logic;
            UI1   : in    std_logic;
            UI2   : in    std_logic;
            UI3   : in    std_logic;
            UO    : out   std_logic_vector(5 downto 0)
       );
end component;

component add_ff
  port ( clock   : in    std_logic;
         CE  : in    std_logic;
         CLR : in    std_logic;
            afi0 : in std_logic_vector(5 downto 0);
            afso : out std_logic
            );
end component;

signal outpos : std_logic_vector(5 downto 0);
signal outneg : std_logic_vector(5 downto 0);

begin

-- lut
poslut_0 : lut
```

```
   generic map (
           INIT0=>"0000000000000000",
           INIT1=>"0000000000000000",
           INIT2=>"0000000000000000",
           INIT3=>"0000000100000100",
           INIT4=>"0000000000000000",
           INIT5=>"0000000000000000")
   port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),
             UO=>outpos);

neglut_0 : lut
   generic map (
           INIT0=>"1000000000000000",
           INIT1=>"0000000000000000",
           INIT2=>"0000000000000000",
           INIT3=>"0000000000000000",
           INIT4=>"0000000000000000",
           INIT5=>"0000000000000000")
   port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),
             UO=>outneg);

 add_ff_0: add_ff
   port map (clock, enable, '0', outpos, toutpos);

 add_ff_1: add_ff
   port map (clock, enable, '0', outneg, toutneg);

end Behavioral;
```

Lut_150.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity lut_15_0 is
    Port ( clock : in std_logic;
               enable :  in std_logic;
```

```
              ti : in std_logic_vector(3 downto 0);
           toutpos : out std_logic;
              toutneg : out std_logic
              );
end lut_15_0;

architecture Behavioral of lut_15_0 is


component lut
    generic (
             INIT0 : bit_vector := x"16";
             INIT1 : bit_vector := x"16";
             INIT2 : bit_vector := x"16";
             INIT3 : bit_vector := x"16";
             INIT4 : bit_vector := x"16";
             INIT5 : bit_vector := x"16");
   port (
            UI0   : in    std_logic;
            UI1   : in    std_logic;
            UI2   : in    std_logic;
            UI3   : in    std_logic;
            UO    : out   std_logic_vector(5 downto 0)
       );
end component;

component add_ff
  port ( clock   : in    std_logic;
         CE  : in    std_logic;
         CLR : in    std_logic;
            afi0 : in std_logic_vector(5 downto 0);
            afso : out std_logic
            );
end component;

signal outpos : std_logic_vector(5 downto 0);
signal outneg : std_logic_vector(5 downto 0);

begin

-- lut
poslut_0 : lut
   generic map (
            INIT0=>"0000000000000000",
            INIT1=>"0000000000000000",
            INIT2=>"0000000000000000",
            INIT3=>"0000001000001000",
            INIT4=>"0000000000000000",
```

```
            INIT5=>"0000000000000000")
   port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),
             UO=>outpos);

neglut_0 : lut
   generic map (
            INIT0=>"0000000000000000",
            INIT1=>"0000000000000000",
            INIT2=>"0010000010000000",
            INIT3=>"0000000000000000",
            INIT4=>"0000000000000000",
            INIT5=>"0000000000000000")
   port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),
             UO=>outneg);

 add_ff_0: add_ff
   port map (clock, enable, '0', outpos, toutpos);

 add_ff_1: add_ff
   port map (clock, enable, '0', outneg, toutneg);

end Behavioral;
```

ShiftReg14.vhdl

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity M2_1_MXILINX_test is
   port ( D0 : in    std_logic;
          D1 : in    std_logic;
          S0 : in    std_logic;
          O  : out   std_logic);
end M2_1_MXILINX_test;

architecture BEHAVIORAL of M2_1_MXILINX_test is
   attribute BOX_TYPE   : STRING ;
```

```
   signal M0 : std_logic;
   signal M1 : std_logic;
   component AND2B1
      port ( I0 : in    std_logic;
             I1 : in    std_logic;
             O  : out   std_logic);
   end component;
   attribute BOX_TYPE of AND2B1 : COMPONENT is "BLACK_BOX";

   component OR2
      port ( I0 : in    std_logic;
             I1 : in    std_logic;
             O  : out   std_logic);
   end component;
   attribute BOX_TYPE of OR2 : COMPONENT is "BLACK_BOX";

   component AND2
      port ( I0 : in    std_logic;
             I1 : in    std_logic;
             O  : out   std_logic);
   end component;
   attribute BOX_TYPE of AND2 : COMPONENT is "BLACK_BOX";

begin
   I_36_7 : AND2B1
      port map (I0=>S0, I1=>D0, O=>M0);

   I_36_8 : OR2
      port map (I0=>M1, I1=>M0, O=>O);

   I_36_9 : AND2
      port map (I0=>D1, I1=>S0, O=>M1);

end BEHAVIORAL;


  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;
```

```
entity shiftreg14bit is
   port ( C    : in    std_logic;   -- clock
          CE   : in    std_logic;   -- enable
          CLR  : in    std_logic;   -- clear
          D    : in    std_logic_vector (13 downto 0);   -- input
          L    : in    std_logic;   -- load
          LEFT : in    std_logic;   -- shift left
          SLI  : in    std_logic;   -- add to left 0/1
          SRI  : in    std_logic;   -- add to right 0/1
          Q    : out   std_logic_vector (13 downto 0));  -- output
end shiftreg14bit;

architecture BEHAVIORAL of shiftreg14bit is
   attribute HU_SET     : STRING ;
   attribute INIT       : STRING ;
   attribute BOX_TYPE   : STRING ;
   signal L_LEFT  : std_logic;
   signal L_OR_CE : std_logic;
   signal MDL0    : std_logic;
   signal MDL1    : std_logic;
   signal MDL2    : std_logic;
   signal MDL3    : std_logic;
   signal MDL4    : std_logic;
   signal MDL5    : std_logic;
   signal MDL6    : std_logic;
   signal MDL7    : std_logic;
   signal MDL8    : std_logic;
   signal MDL9    : std_logic;
   signal MDL10    : std_logic;
   signal MDL11    : std_logic;
   signal MDL12    : std_logic;
   signal MDL13    : std_logic;
   signal MDR0    : std_logic;
   signal MDR1    : std_logic;
   signal MDR2    : std_logic;
   signal MDR3    : std_logic;
   signal MDR4    : std_logic;
   signal MDR5    : std_logic;
   signal MDR6    : std_logic;
   signal MDR7    : std_logic;
   signal MDR8    : std_logic;
   signal MDR9    : std_logic;
   signal MDR10    : std_logic;
   signal MDR11    : std_logic;
   signal MDR12    : std_logic;
   signal MDR13    : std_logic;
   signal Q_DUMMY : std_logic_vector (13 downto 0);
   component M2_1_MXILINX_test
```

```
      port ( D0 : in    std_logic;
             D1 : in    std_logic;
             S0 : in    std_logic;
             O  : out   std_logic);
   end component;

   component FDCE
      -- synopsys translate_off
      generic( INIT : bit :=  '0');
      -- synopsys translate_on
      port ( C   : in    std_logic;
             CE  : in    std_logic;
             CLR : in    std_logic;
             D   : in    std_logic;
             Q   : out   std_logic);
   end component;
   attribute INIT of FDCE : COMPONENT is "0";
   attribute BOX_TYPE of FDCE : COMPONENT is "BLACK_BOX";

   component OR2
      port ( I0 : in    std_logic;
             I1 : in    std_logic;
             O  : out   std_logic);
   end component;
   attribute BOX_TYPE of OR2 : COMPONENT is "BLACK_BOX";


begin
   Q(13 downto 0) <= Q_DUMMY(13 downto 0);
   I_MDL0 : M2_1_MXILINX_test
      port map (D0=>SLI, D1=>D(0), S0=>L, O=>MDL0);

   I_MDL1 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(0), D1=>D(1), S0=>L, O=>MDL1);

   I_MDL2 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(1), D1=>D(2), S0=>L, O=>MDL2);

   I_MDL3 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(2), D1=>D(3), S0=>L, O=>MDL3);

   I_MDL4 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(3), D1=>D(4), S0=>L, O=>MDL4);

   I_MDL5 : M2_1_MXILINX_test
      port map (D0=>Q_DUMMY(4), D1=>D(5), S0=>L, O=>MDL5);

   I_MDL6 : M2_1_MXILINX_test
```

```
   port map (D0=>Q_DUMMY(5), D1=>D(6), S0=>L, O=>MDL6);

I_MDL7 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(6), D1=>D(7), S0=>L, O=>MDL7);

I_MDL8 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(7), D1=>D(8), S0=>L, O=>MDL8);

I_MDL9 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(8), D1=>D(9), S0=>L, O=>MDL9);

I_MDL10 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(9), D1=>D(10), S0=>L, O=>MDL10);

I_MDL11 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(10), D1=>D(11), S0=>L, O=>MDL11);

I_MDL12 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(11), D1=>D(12), S0=>L, O=>MDL12);

I_MDL13 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(12), D1=>D(13), S0=>L, O=>MDL13);

I_MDR0 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(1), D1=>MDL0, S0=>L_LEFT, O=>MDR0);

I_MDR1 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(2), D1=>MDL1, S0=>L_LEFT, O=>MDR1);

I_MDR2 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(3), D1=>MDL2, S0=>L_LEFT, O=>MDR2);

I_MDR3 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(4), D1=>MDL3, S0=>L_LEFT, O=>MDR3);

I_MDR4 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(5), D1=>MDL4, S0=>L_LEFT, O=>MDR4);

I_MDR5 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(6), D1=>MDL5, S0=>L_LEFT, O=>MDR5);

I_MDR6 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(7), D1=>MDL6, S0=>L_LEFT, O=>MDR6);

I_MDR7 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(8), D1=>MDL7, S0=>L_LEFT, O=>MDR7);

I_MDR8 : M2_1_MXILINX_test
```

```
   port map (D0=>Q_DUMMY(9), D1=>MDL8, S0=>L_LEFT, O=>MDR8);

I_MDR9 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(10), D1=>MDL9, S0=>L_LEFT, O=>MDR9);

I_MDR10 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(11), D1=>MDL10, S0=>L_LEFT, O=>MDR10);

I_MDR11 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(12), D1=>MDL11, S0=>L_LEFT, O=>MDR11);

I_MDR12 : M2_1_MXILINX_test
   port map (D0=>Q_DUMMY(13), D1=>MDL12, S0=>L_LEFT, O=>MDR12);

I_MDR13 : M2_1_MXILINX_test
   port map (D0=>SRI, D1=>MDL13, S0=>L_LEFT, O=>MDR13);

I_Q0 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR0, Q=>Q_DUMMY(0));

I_Q1 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR1, Q=>Q_DUMMY(1));

I_Q2 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR2, Q=>Q_DUMMY(2));

I_Q3 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR3, Q=>Q_DUMMY(3));

I_Q4 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR4, Q=>Q_DUMMY(4));

I_Q5 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR5, Q=>Q_DUMMY(5));

I_Q6 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR6, Q=>Q_DUMMY(6));

I_Q7 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR7, Q=>Q_DUMMY(7));

I_Q8 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR8, Q=>Q_DUMMY(8));

I_Q9 : FDCE
   port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR9, Q=>Q_DUMMY(9));

I_Q10 : FDCE
```

```
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR10, Q=>Q_DUMMY(10));

   I_Q11 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR11, Q=>Q_DUMMY(11));

   I_Q12 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR12, Q=>Q_DUMMY(12));

   I_Q13 : FDCE
      port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR13, Q=>Q_DUMMY(13));

   I_36_145 : OR2
      port map (I0=>L, I1=>CE, O=>L_OR_CE);

   I_36_161 : OR2
      port map (I0=>LEFT, I1=>L, O=>L_LEFT);

end BEHAVIORAL;
```

InitShift_Test.vhd

```
  -- Created by Guri Kristine Birkeland 2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;


LIBRARY ieee; USE ieee.std_logic_1164.ALL; USE
ieee.std_logic_unsigned.all; USE ieee.numeric_std.ALL;

ENTITY init_shift_test_vhd IS END init_shift_test_vhd;

ARCHITECTURE behavior OF init_shift_test_vhd IS

    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT init_shiftregs
    PORT(
        clk : IN std_logic;
        enable : IN std_logic;
```

```
        load : IN std_logic;
        tin : IN std_logic_vector(7 downto 0);
        tout : OUT std_logic
        );
    END COMPONENT;

    --Inputs
    SIGNAL clk :  std_logic := '0';
    SIGNAL enable :  std_logic := '0';
    SIGNAL load :  std_logic := '0';
    SIGNAL tin :  std_logic_vector(7 downto 0) := (others=>'0');

    --Outputs
    SIGNAL tout :  std_logic;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: init_shiftregs PORT MAP(
        clk => clk,
        enable => enable,
        load => load,
        tin => tin,
        tout => tout
    );

   PROCESS -- Process for clock
    BEGIN
        CLOCK_LOOP : LOOP
        clk <= transport '0';
        WAIT FOR 10 ns;
        clk <= transport '1';
        WAIT FOR 10 ns;
        WAIT FOR 40 ns;
        clk <= transport '0';
        WAIT FOR 40 ns;
        END LOOP CLOCK_LOOP;
    END PROCESS;

    tb : PROCESS
    BEGIN

        -- Wait 100 ns for global reset to finish
        wait for 100 ns;

        -- 1st
        -- --------------------
        enable <= transport '1';
```

```
load <= transport '0';
tin <= transport std_logic_vector'("00000001");
-- --------------------
WAIT FOR 100 ns; -- Time=100 ns
load <= transport '1';
-- --------------------
WAIT FOR 100 ns; -- Time=200 ns
load <= transport '0';
-- --------------------
--
WAIT FOR 1600 ns; -- Time=1700 ns
load <= transport '1';
tin <= transport std_logic_vector'("00111111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11111100");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("01111111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11111110");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11111100");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
```

```
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11111001");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("10011111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00111111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("01111111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11100111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00000011");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
```

```
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00000110");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("01100000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11000000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("10000000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00011000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00100100");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
```

```
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("01000010");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("10000001");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00000000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11100000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("00000111");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
-- --------------------
--
WAIT FOR 1700 ns;
load <= transport '1';
tin <= transport std_logic_vector'("11110000");
-- --------------------
WAIT FOR 100 ns;
load <= transport '0';
```

```
        -- --------------------
        --
        WAIT FOR 1700 ns;
        load <= transport '1';
        tin <= transport std_logic_vector'("00001111");
        -- --------------------
        WAIT FOR 100 ns;
        load <= transport '0';
        -- --------------------
        --
        WAIT FOR 1700 ns;
        load <= transport '1';
        tin <= transport std_logic_vector'("00011111");
        -- --------------------
        WAIT FOR 100 ns;
        load <= transport '0';
        -- --------------------
        --
        WAIT FOR 1700 ns;
        load <= transport '1';
        tin <= transport std_logic_vector'("11111111");
        -- --------------------
        WAIT FOR 100 ns;
        load <= transport '0';
        -- --------------------
        --
        WAIT FOR 1700 ns;
        load <= transport '1';
        tin <= transport std_logic_vector'("11111000");
        -- --------------------
        WAIT FOR 100 ns;
        load <= transport '0';
        -- --------------------
        --
        WAIT FOR 1700 ns;
        -- --------------------
        -- end waiting
        WAIT FOR 1300 ns;
        -- --------------------

-- accumulated runtime is: 51800 ns

    END PROCESS;

END;
```

APPENDIX D: DAT SOURCE CODE

This appendix contains the full source code of the design automation tool.

Arch1.c

```
// This file contains all functions which are used by two among binary, CSD2 and CSD4 designs.
// Special files used only in the CSD4-architecture are put in the "CSD4Arch1.c"-file etc.

#include <direct.h>

#include "extIncl.h"

#include "main.h"
#include "arch1.h"
#include "bin.h"
#include "CSD4.h"
#include "csd.h"
#include "comFiles.h"

    // failure and alloc rutines
#define oops(s) { perror((s)); exit(EXIT_FAILURE); }
#define MALLOC(s,t) if(((s) = malloc(t)) == NULL) { oops("error: malloc() "); }



/**
* Calls all the functions which creates the different architectures.
*
* @param taps <code>int</code> number of taps in the initial file.
* @param groups <code>int</code> number of groups with 4 taps in each group
* @param tapGroups <code>double**</code> The value of the taps are stored here. 4 taps in each group
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int createArch1(int taps, int groups, double** tapGroups, int BINdigits, int CSD2digits, int CSD4digits)
{
      // creating directories
    mkdir("./ARCH1");
    mkdir("./ARCH1/BIN");
    mkdir("./ARCH1/CSD2");
    mkdir("./ARCH1/CSD4");

    createBINArch1(taps, groups, tapGroups, BINdigits);
    createCSD2Arch1(taps, groups, tapGroups, CSD2digits);
```

```
    createCSD4Arch1(taps, groups, tapGroups, CSD4digits);

    return 0;
}



/**
* Calls all the functions needed to create architecture1 Binary
*
* @param taps <code>int</code> number of taps in the initial file.
* @param groups <code>int</code> number of groups with 4 taps in each group
* @param tapGroups <code>double**</code> The value of the taps are stored here. 4 taps in each group
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int createBINArch1(int taps, int groups, double** tapGroups, int BINdigits)
{
    createAddFile("ARCH1", "BIN");
//  createSubFile("ARCH1", "BIN");
    createAdd16File("ARCH1", "BIN");
    createShiftRegFile(BINdigits, "ARCH1", "BIN");
    createTestFile(BINdigits, groups, "ARCH1", "BIN");

    createInitShiftRegFile(taps, BINdigits, "ARCH1", "BIN");
    createArch1File(taps, groups, "BIN", "");
    createNBitLutFile(BINdigits, "ARCH1", "BIN");
    createAddFFFile(BINdigits, "ARCH1", "BIN");
    createFlipFlopFile(BINdigits, "ARCH1", "BIN");

    createBINLutFiles(groups, BINdigits, tapGroups, "ARCH1");   // found in binArch1.c

    return 0;
}


/**
* Calls all the functions needed to create architecture1 CSD2
*
* @param taps <code>int</code> number of taps in the initial file.
* @param groups <code>int</code> number of groups with 4 taps in each group
* @param tapGroups <code>double**</code> The value of the taps are stored here. 4 taps in each group
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int createCSD2Arch1(int taps, int groups, double** tapGroups, int CSD2digits)
```

```
{
    createAddFile("ARCH1", "CSD2");
    createSubFile("ARCH1", "CSD2");
    createAdd16File("ARCH1", "CSD2");
    createShiftRegFile(CSD2digits, "ARCH1", "CSD2");
    createTestFile(CSD2digits, groups, "ARCH1", "CSD2");

    createInitShiftRegFile(taps, CSD2digits, "ARCH1", "CSD2");
    createArch1File(taps, groups, "CSD2", "");
    createA1LutSubFiles(groups, CSD2digits, tapGroups, "CSD2", "", 2);
    createNBitLutFile(CSD2digits, "ARCH1", "CSD2");
    createAddFFFile(CSD2digits, "ARCH1", "CSD2");
    createFlipFlopFile(CSD2digits, "ARCH1", "CSD2");

    return 0;
}


/**
* Calls all the functions needed to create architecture1 CSD4
*
* @param taps <code>int</code> number of taps in the initial file.
* @param groups <code>int</code> number of groups with 4 taps in each group
* @param tapGroups <code>double**</code> The value of the taps are stored here. 4 taps in each group
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int createCSD4Arch1(int taps, int groups, double** tapGroups, int CSD4digits)
{

    int i=0;

    char* ex[6];

    ex[0] = "_5";
    ex[1] = "_7";
    ex[2] = "_9";
    ex[3] = "_11";
    ex[4] = "_13";
    ex[5] = "_15";

    createAddFile("ARCH1", "CSD4");
    createSubFile("ARCH1", "CSD4");
    createAdd16File("ARCH1", "CSD4");
    createShiftRegFile(CSD4digits, "ARCH1", "CSD4");
    createTestFile(CSD4digits, groups, "ARCH1", "CSD4");
```

```
    createInitShiftRegFile(taps, CSD4digits, "ARCH1", "CSD4");

    for(i=0;i<6;i++)
    {
      createArch1File(taps, groups, "CSD4", ex[i]);
      createA1LutSubFiles(groups, CSD4digits, tapGroups, "CSD4", ex[i], 4);
    }

    createNBitLutFile(CSD4digits, "ARCH1", "CSD4");
    createAddFFFile(CSD4digits, "ARCH1", "CSD4");
    createFlipFlopFile(CSD4digits, "ARCH1", "CSD4");

    createCSD4ArchFile(taps, "ARCH1");
    createCSD4EndAddFile(taps, 5, 7, "ARCH1");
    createCSD4EndAddFile(taps, 9, 11, "ARCH1");
    createCSD4EndAddFile(taps, 13, 15, "ARCH1");

    return 0;

}


///////// HERE STARTS THE FUNCTIONS THAT CREATES VHDL-FILES /////////




/**
* Creates the architecture-file
*
* @param taps <code>int</code> number of taps in the initial file.
* @param groups <code>int</code> number of groups with 4 taps in each group
* @param base <code>char*</code> Base is "BIN" for binary numbers, "CSD2" for CSD/CSD2, "CSD4" for CSD4
* @param extention <code>char*</code> For CSD4 this function is called several times with a different extention each
*                                    For the other bases, extention can be given the value "".
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int createArch1File(int taps, int groups, char* base, char* extention)
{
  int i=0, j=0, k=0;

  int noAdds=0, groups_temp=0, noSum=0, newSum=0, noGroups=0;
  int* order;
  int* adds;
```

```
 char s[4][20];
 char a[15], b[15];

 char sub[5] = "_";

 FILE *VHDLfile;
 char VHDLfilename[35];

   // allocate memory space of the size of groups to array "order"
 MALLOC(order, sizeof(int) * groups);
 MALLOC(adds, sizeof(int) * groups);

 for(i=0;i<groups;i++)
 {
   order[i] = 0;
   adds[i] = 0;
 }

 sprintf(VHDLfilename, "./ARCH1/%s/Arch1_%s%s.vhdl", base, base, extention);

 if ((VHDLfile = fopen(VHDLfilename, "w")) == NULL)
        fprintf(stderr, "Cannot open %s\n", VHDLfilename);

 fprintf(VHDLfile, returnHeader());

 fprintf(VHDLfile, "\
entity ARCH1_%s%s is \n\
    Port ( clock : in std_logic; \n\
         enable :  in std_logic; \n\
         tin : in std_logic_vector(%d downto 0); \n", base, extention, taps-1);
 fprintf(VHDLfile, "\
       tout : out std_logic\n\
         );      \n\
end ARCH1_%s%s; \n\
 \n\
architecture Behavioral of ARCH1_%s%s is \n\
\n", base, extention, base, extention);

   // if the base is not binary, there should be added a subtraction many places.
 if( strcmp("BIN", base) )
   strcpy(sub,"sub");

 for(i=0;i<groups;i++)
 {
   fprintf(VHDLfile, "\
component lut%s%s%d \n\
    Port ( clock : in std_logic; \n\
         enable :  in std_logic; \n\
```

```
        ti : in std_logic_vector(3 downto 0); \n\
      tout : out std_logic); \n\
end component; \n\n", sub, extention, i);
 }

 fprintf(VHDLfile, "\
component add2 \n\
  port\n\
  (\n\
     a:  in std_logic;\n\
     b:  in std_logic;\n\
     cii : in std_logic;\n\
     s: out std_logic;\n\
     coo : out std_logic\n\
  );\n\
end component; \n\n");

   // find the number of additions needed.
 groups_temp = groups;
 noAdds = 0;
 i=0;
 while(groups_temp > 1)
 {
   order[i] = roof(groups_temp, 2);
   adds[i] = groups_temp/2;
   i++;

   newSum = groups_temp/2;
   noAdds += newSum;
   groups_temp -= newSum;
 }

 for(i=0;i<groups;i++)
 {
   fprintf(VHDLfile, "\
signal partsum%d : std_logic; \n", i);

   if(order[0] != adds[0] && i == order[0]+1)
   {
     fprintf(VHDLfile, "\
signal partsum%d_d : std_logic; \n", i);
   }

 }

 fprintf(VHDLfile, "\n");

 for(i=0;i<noAdds;i++)
```

```
  {
    fprintf(VHDLfile, "\
 signal sum%d0 : std_logic; \n\
 signal sum%d1 : std_logic; \n\
 signal sum%d1_d : std_logic; \n\
", i, i, i);
  }

  fprintf(VHDLfile, "\n");

  for(i=0;i<noAdds;i++)
  {
    fprintf(VHDLfile, "\
 signal cin%d : std_logic; \n\
 signal sign%d : std_logic; \n", i, i);
  }

  fprintf(VHDLfile, "\
 \n\
 begin \n\
\n");

  for(i=0;i<groups;i++)
  {
    if(i*4+3 < taps)    //normal case
    {
      fprintf(VHDLfile, "\
 LUT%s0%d : lut%s%s%d \n\
    port map (clock=>clock, enable=>enable, ti=>tin(%d downto %d), tout=>partsum%d); \n\n",
      sub, i, sub, extention, i, i*4+3, i*4, i);
    }
    else                //last lutsub-entity
    {
      j = 0;
      for(k=i*4;k<i*4+4;k++)
      {
        if(i*4+j<taps)
        {
          sprintf(s[j], "tin(%d)", i*4+j);
        }
        else
          strcpy(s[j],"'0'");

        j++;
      }

      fprintf(VHDLfile, "\
 LUT%s0%d : lut%s%s%d \n\
```

```
   port map (clock=>clock, enable=>enable, ti(0)=>%s, ti(1)=>%s, \n\
               ti(2)=>%s, ti(3)=>%s, \n\
               tout=>partsum%d); \n\n", sub, i, sub, extention, i, s[0] , s[1], s[2], s[3], i);
   }
 }

   // in the general case adds > 2 is assumed. Otherwise, special case.
 if(noAdds > 2)
 {
   noGroups = groups;
   noSum = noAdds;
   j = 0;
   for(i=0;i<noAdds;i++)
   {
     while(order[i] > 0)
     {
       if(noGroups > 1)
       {
         sprintf(a, "partsum%d", groups - noGroups);
         noGroups --;
         sprintf(b, "partsum%d", groups - noGroups);
         noGroups --;
       }
       else if(noGroups == 1)
       {
         sprintf(a, "partsum%d", groups - noGroups);
         noGroups --;
         sprintf(b, "sum01");
         noSum --;
       }
       else
       {
         sprintf(a, "sum%d1", noAdds - noSum);
         noSum --;
         sprintf(b, "sum%d1", noAdds - noSum);
         noSum --;
       }

       // insert a flipflop if addition has to be delayed
       if(order[i] == 1 && adds[i] == 0)
       {
         fprintf(VHDLfile, "\
 -- extra flipflop to syncronize the result\n\
flip_flop_d_%d : FDCE\n\
   port map (C=>clock, CE=>enable, CLR=>'0', D=>%s, Q=>%s_d); \n", j, a, a);

         fprintf(VHDLfile, "\n\
add_%d : add2 \n\
```

```
   port map (a=>%s_d, \n\
                b=>%s, \n\
                cii=>cin%d,\n\
                s=>sum%d0, \n\
                coo=>sign%d); \n\n", j, a, b, j, j, j);

         order[i+1] --;
         adds[i+1] --;
       }

       else
       {
         fprintf(VHDLfile, "\
add_%d : add2 \n\
   port map (a=>%s, \n\
                b=>%s, \n\
                cii=>cin%d,\n\
                s=>sum%d0, \n\
                coo=>sign%d); \n\n", j, a, b, j, j, j);
       }

       fprintf(VHDLfile, "\
flip_flop_c_%d : FDCE\n\
   port map (C=>clock, CE=>enable, CLR=>'0', D=>sign%d, Q=>cin%d); \n", j, j, j);

       if(j!=noAdds-1)
       {
         fprintf(VHDLfile, "\n\
flip_flop_s_%d : FDCE\n\
   port map (C=>clock, CE=>enable, CLR=>'0', D=>sum%d0, Q=>sum%d1); \n\n", j, j, j);
       }

       else
       {
         fprintf(VHDLfile, "\n\
flip_flop_s_%d : FDCE\n\
   port map (C=>clock, CE=>enable, CLR=>'0', D=>sum%d0, Q=>tout); \n\n", j, j);
       }

       order[i] --;
       adds[i] --;
       j++;
     } // end of while(order[i]>0)
   } //  end of for(i=0;i<noAdds;i++)
 } // end of if adds > 2

 else if(noAdds == 2)
 {
```

```
    fprintf(VHDLfile, "\
 add_0 : add2 \n\
    port map (a=>partsum0, \n\
                b=>partsum1, \n\
                cii=>cin0,\n\
                s=>sum00, \n\
                coo=>sign0); \n\
\n\
 flip_flop_0 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign0, Q=>cin0);\n\
  \n\
 flip_flop_1 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sum00, Q=>sum01);\n\
  \n\
  -- extra flipflop to syncronize the result\n\
 flip_flop_2 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>partsum20, Q=>partsum21);\n\
  \n\
 add_1 : add2 \n\
    port map (a=>partsum21, \n\
                b=>sum01, \n\
                cii=>cin1,\n\
                s=>sum10, \n\
                coo=>sign1); \n\
\n\
 flip_flop_3 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign1, Q=>cin1);\n\
\n\
 flip_flop_4 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sum10, Q=>tout);\n");
  }

  else if(noAdds == 1)
  {
    fprintf(VHDLfile, "\
 add_0 : add2 \n\
    port map (a=>partsum0, \n\
                b=>partsum1, \n\
                cii=>cin0,\n\
                s=>sum00, \n\
                coo=>sign0); \n\
\n\
 flip_flop_0 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign0, Q=>cin0);\n\
\n\
 flip_flop_1 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sum00, Q=>tout);\n");
  }
```

```
  else if(noAdds == 0)
  {
    fprintf(VHDLfile, "\
 tout<=partsum0;\n");
  }

    fprintf(VHDLfile, "\n\
 end Behavioral; \n");

  fclose(VHDLfile);

  return 0;

}



/**
* Creates the LutSub-files for CSD2 and CSD4 numbers
*
* @param groups <code>int</code> number of groups with 4 taps in each group
* @param digits <code>int</code> number of digits in the CSD2 or CSD4 number
* @param tapGroups <code>double**</code> The value of the taps are stored here. 4 taps in each group
* @param base <code>char*</code> Base is "CSD2" for CSD/CSD2, "CSD4" for CSD4
* @param extention <code>char*</code> For CSD4 this function is called several times with a different extention each
*                                    For the other bases, extention can be given the value "".
* @param i_base <code>int</code> i_base is 2 for CSD2 and 4 for CSD4
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int createA1LutSubFiles(int groups, int digits, double** tapGroups, char* base, char* extention, int i_base)
{

  int i=0, j=0, k=0;
  int pos=0, neg=0;

  FILE *VHDLfile;
  char VHDLfilename[35];

  int*** csdArray; //[GROUPS][16][DIGITS];  // store the CSD numbers, 16 is 4^2, i.e. number of combinations of taps

  // allocate memory space for csdArray
  // csdArray[GROUPS][16][DIGITS]
  MALLOC(csdArray, sizeof(int**) * groups);
  for (i = 0; i < groups; i++)
  {
    MALLOC(csdArray[i], sizeof(int*) * 16);
```

```
   for(j = 0; j < 16; j++)
   {
     MALLOC(csdArray[i][j], sizeof(int) * digits);
   }
 }

   // put values into csdArray
 findCSDArray(groups, digits, tapGroups, i_base, csdArray);

   // for each file
 for(i=0;i<groups;i++)
 {
       // create a filename for each file
   sprintf(VHDLfilename, "./ARCH1/%s/LutSub%s%d.vhdl", base, extention, i);

   if ((VHDLfile = fopen(VHDLfilename, "w")) == NULL)
        fprintf(stderr, "Cannot open %s\n", VHDLfilename);

   fprintf(VHDLfile, returnHeader());


   fprintf(VHDLfile, "\
entity LUTsub%s%d is \n\
    Port ( clock : in std_logic; \n\
                enable :  in std_logic; \n\
                ti : in std_logic_vector(3 downto 0); \n\
           tout : out std_logic\n\
                ); \n\
end LUTsub%s%d; \n\n", extention, i, extention, i);

   fprintf(VHDLfile, "\
architecture Behavioral of LUTsub%s%d is \n\
 \n\n", extention, i);

   fprintf(VHDLfile, "\
component lut  \n\
    generic (\n");

   for(j=0;j<digits-1;j++)
   {
     fprintf(VHDLfile, "\
            INIT%d : bit_vector := x\"16\";\n", j);
   }
   fprintf(VHDLfile, "\
            INIT%d : bit_vector := x\"16\");\n", digits-1);
   fprintf(VHDLfile, "\
   port ( \n\
           UI0   : in    std_logic;  \n\
```

```
            UI1   : in    std_logic;  \n\
            UI2   : in    std_logic;  \n\
            UI3   : in    std_logic;  \n\
            UO    : out   std_logic_vector(%d downto 0)  \n\
        ); \n\
 end component; \n\n", digits-1);

    fprintf(VHDLfile, "\
 component SUB2\n\
    PORT(  A:  IN STD_LOGIC;\n\
           B:  IN STD_LOGIC;\n\
           bin : in std_logic;\n\
              Q:  OUT STD_LOGIC;\n\
           bout :   OUT STD_LOGIC\n\
        );\n\
 END component;\n\
  \n\
 component add_ff \n\
   port ( clock   : in    std_logic; \n\
          CE  : in    std_logic; \n\
          CLR : in    std_logic; \n\
             afi0 : in std_logic_vector(%d downto 0);\n\
             afso : out std_logic\n\
             );\n\
 end component;\n\n", digits-1);

    fprintf(VHDLfile, "\
 signal poslutout : std_logic_vector(%d downto 0); \n\
 signal neglutout : std_logic_vector(%d downto 0); \n\
\n\
 signal posffout : std_logic;\n\
 signal negffout : std_logic;\n\
\n\
 signal sign0 : std_logic;\n\
 signal bin0 : std_logic;\n\
 signal diff : std_logic;\n\
\n", digits-1, digits-1);

    fprintf(VHDLfile, "\
 begin \n\
  \n\
 -- lut \n\
 poslut_0 : lut \n\
    generic map (\n");

    // determine which number to look for
    if(i_base == 2)
    {
```

```
      pos = 1;
      neg = -1;
    }
    else if(i_base == 4)
    {
      pos = atoi(extention+1);
      neg = -pos;
    }
    else
      oops("Error: input i_base must be 2 or 4. Otherwise wrong function is called.");

        // writing positive numbers
    for(j=0;j<digits;j++)
    {
      fprintf(VHDLfile, "\
            INIT%d=>\"", j);
      for(k=0;k<16;k++)
      {
        // currently writing positive numbers
        if(*( csdArray[i][15-k] + (digits-1-j) ) == pos)
          fprintf(VHDLfile, "1");
        else
          fprintf(VHDLfile, "0");
      }
      if(j != digits-1)
        fprintf(VHDLfile, "\",\n");
      else
        fprintf(VHDLfile, "\")\n");
    }

    fprintf(VHDLfile, "\
    port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),\n\
              UO=>poslutout);  \n\
\n\
 neglut_0 : lut \n\
    generic map (\n");

        // writing negative numbers
    for(j=0;j<digits;j++)
    {
      fprintf(VHDLfile, "\
            INIT%d=>\"", j);
      for(k=0;k<16;k++)
      {
        // currently writing negative numbers
        if(*( csdArray[i][15-k] + (digits-1-j) ) == neg)
          fprintf(VHDLfile, "1");
        else
```

```
          fprintf(VHDLfile, "0");
      }
      if(j != digits-1)
        fprintf(VHDLfile, "\",\n");
      else
        fprintf(VHDLfile, "\")\n");
    }

    fprintf(VHDLfile, "\
    port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),\n\
              UO=>neglutout);  \n\
  \n");

    fprintf(VHDLfile, "\
  add_ff_0: add_ff\n\
    port map (clock, enable, '0', poslutout, posffout); \n\
\n\
  add_ff_1: add_ff\n\
    port map (clock, enable, '0', neglutout, negffout); \n\
\n\
  -- sub neg from pos  \n\
  sub1bit_0 : SUB2  \n\
        port map (A=>posffout, B=>negffout, bin=>bin0, Q=>diff, bout=>sign0);\n\
\n\
  -- store carry signal\n\
  flip_flop_0 : FDCE\n\
        port map (C=>clock, CE=>enable, CLR=>'0', D=>sign0, Q=>bin0);\n\
\n\
  -- store diff\n\
  flip_flop_1 : FDCE\n\
        port map (C=>clock, CE=>enable, CLR=>'0', D=>diff, Q=>tout);\n\
  \n\
 end Behavioral; \n");

    fclose(VHDLfile);

  } // end of for each group


    // free allocated memory
  for (i = 0; i < groups; i++)
  {
    for (j = 0; j < 16; j++)
    {
      free(csdArray[i][j]);
    }
    free(csdArray[i]);
  }
```

```
  free(csdArray);


  return 0;
} // end of createLutSubFiles()
```

Arch1.h

```
int createArch1(int taps, int groups, double** tapGroups, int BINdigits, int CSD2digits, int CSD4digits);

int createBINArch1(int taps, int groups, double** tapGroups, int BINdigits);
int createCSD2Arch1(int taps, int groups, double** tapGroups, int CSD2digits);
int createCSD4Arch1(int taps, int groups, double** tapGroups, int CSD4digits);

int createArch1File(int taps, int groups, char* base, char* extention);
int createA1LutSubFiles(int groups, int digits, double** tapGroups,
                        char* base, char* extention, int i_base);
```

Arch2.c

```
// This file contains all functions which are used by two among binary, CSD2 and CSD4 designs.
// Special files used only in the CSD4-architecture are put in the "CSD4Arch1.c"-file etc.

#include <direct.h>

#include "extIncl.h"

#include "main.h"
#include "arch2.h"
#include "bin.h"
#include "CSD4.h"
#include "csd.h"
#include "comFiles.h"

    // failure and alloc rutines
#define oops(s) { perror((s)); exit(EXIT_FAILURE); }
#define MALLOC(s,t) if(((s) = malloc(t)) == NULL) { oops("error: malloc() "); }


/**
* Calls all the functions which creates the different architectures.
*
* @param taps <code>int</code> number of taps in the initial file.
```

```
* @param groups <code>int</code> number of groups with 4 taps in each group
* @param tapGroups <code>double**</code> The value of the taps are stored here. 4 taps in each group
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int createArch2(int taps, int groups, double** tapGroups, int BINdigits, int CSD2digits, int CSD4digits)
{
      // creating directories
    mkdir("./ARCH2");
    mkdir("./ARCH2/BIN");
    mkdir("./ARCH2/CSD2");
    mkdir("./ARCH2/CSD4");

    createBINArch2(taps, groups, tapGroups, BINdigits);
    createCSD2Arch2(taps, groups, tapGroups, CSD2digits);
    createCSD4Arch2(taps, groups, tapGroups, CSD4digits);

    return 0;
}



/**
* Calls all the functions needed to create architecture2 Binary
*
* @param taps <code>int</code> number of taps in the initial file.
* @param groups <code>int</code> number of groups with 4 taps in each group
* @param tapGroups <code>double**</code> The value of the taps are stored here. 4 taps in each group
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int createBINArch2(int taps, int groups, double** tapGroups, int BINdigits)
{
    createAddFile("ARCH2", "BIN");
//  createSubFile("ARCH2", "BIN");
    createAdd16File("ARCH2", "BIN");
    createShiftRegFile(BINdigits, "ARCH2", "BIN");
    createTestFile(BINdigits, groups, "ARCH2", "BIN");

    createInitShiftRegFile(taps, BINdigits, "ARCH2", "BIN");

    createBINArch2File(taps, groups);
    createBINA2BigLutFile(taps, groups);
    createBINLutFiles(groups, BINdigits, tapGroups, "ARCH2");

    createNBitLutFile(BINdigits, "ARCH2", "BIN");
```

```
    createAddFFFile(BINdigits, "ARCH2", "BIN");
    createFlipFlopFile(BINdigits, "ARCH2", "BIN");

    createA2AddAllFile(groups, "BIN");

    return 0;
}


/**
* Calls all the functions needed to create architecture2 CSD2
*
* @param taps <code>int</code> number of taps in the initial file.
* @param groups <code>int</code> number of groups with 4 taps in each group
* @param tapGroups <code>double**</code> The value of the taps are stored here. 4 taps in each group
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int createCSD2Arch2(int taps, int groups, double** tapGroups, int CSD2digits)
{
    createAddFile("ARCH2", "CSD2");
    createSubFile("ARCH2", "CSD2");
    createAdd16File("ARCH2", "CSD2");
    createShiftRegFile(CSD2digits, "ARCH2", "CSD2");
    createTestFile(CSD2digits, groups, "ARCH2", "CSD2");

    createInitShiftRegFile(taps, CSD2digits, "ARCH2", "CSD2");
    createArch2File(taps, groups, "CSD2", "");

    createA2BigLutFile(taps, groups, "CSD2", "");

    createA2LutFiles(groups, CSD2digits, tapGroups, "CSD2", "", 2);
    createNBitLutFile(CSD2digits, "ARCH2", "CSD2");
    createAddFFFile(CSD2digits, "ARCH2", "CSD2");
    createFlipFlopFile(CSD2digits, "ARCH2", "CSD2");

    createA2EndAddSubFile(groups, "CSD2");
    createA2AddAllFile(groups, "CSD2");

    return 0;
}


/**
* Calls all the functions needed to create architecture2 CSD4
*
* @param taps <code>int</code> number of taps in the initial file.
```

```
* @param groups <code>int</code> number of groups with 4 taps in each group
* @param tapGroups <code>double**</code> The value of the taps are stored here. 4 taps in each group
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int createCSD4Arch2(int taps, int groups, double** tapGroups, int CSD4digits)
{
    int i=0;

    char* ex[6];

    ex[0] = "_5";
    ex[1] = "_7";
    ex[2] = "_9";
    ex[3] = "_11";
    ex[4] = "_13";
    ex[5] = "_15";

    createAddFile("ARCH2", "CSD4");
    createSubFile("ARCH2", "CSD4");
    createAdd16File("ARCH2", "CSD4");
    createShiftRegFile(CSD4digits, "ARCH2", "CSD4");
    createTestFile(CSD4digits, groups, "ARCH2", "CSD4");

    createInitShiftRegFile(taps, CSD4digits, "ARCH2", "CSD4");

    for(i=0;i<6;i++)
    {
      createArch2File(taps, groups, "CSD4", ex[i]);
      createA2BigLutFile(taps, groups, "CSD4", ex[i]);
      createA2LutFiles(groups, CSD4digits, tapGroups, "CSD4", ex[i], 4);
    }

    createNBitLutFile(CSD4digits, "ARCH2", "CSD4");
    createAddFFFile(CSD4digits, "ARCH2", "CSD4");
    createFlipFlopFile(CSD4digits, "ARCH2", "CSD4");

    createA2EndAddSubFile(groups, "CSD4");
    createA2AddAllFile(groups, "CSD4");


    createCSD4ArchFile(taps, "ARCH2");
    createCSD4EndAddFile(taps, 5, 7, "ARCH2");
    createCSD4EndAddFile(taps, 9, 11, "ARCH2");
    createCSD4EndAddFile(taps, 13, 15, "ARCH2");

    return 0;
```

```
}

///////// HERE STARTS THE FUNCTIONS THAT CREATES VHDL-FILES /////////



/**
* Creates the architecture-file
*
* @param taps <code>int</code> number of taps in the initial file.
* @param groups <code>int</code> number of groups with 4 taps in each group
* @param base <code>char*</code> Base is "BIN" for binary numbers, "CSD2" for CSD/CSD2, "CSD4" for CSD4
* @param extention <code>char*</code> For CSD4 this function is called several times with a different extention each
*                                    For the other bases, extention can be given the value "".
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int createArch2File(int taps, int groups, char* base, char* extention)
{
  int i=0;

  FILE *VHDLfile;
  char VHDLfilename[35];

  sprintf(VHDLfilename, "./ARCH2/%s/Arch2_%s%s.vhdl", base, base, extention);

  if ((VHDLfile = fopen(VHDLfilename, "w")) == NULL)
        fprintf(stderr, "Cannot open %s\n", VHDLfilename);

  fprintf(VHDLfile, returnHeader());

  fprintf(VHDLfile, "\
 entity ARCH2_%s%s is \n\
    Port ( clock : in std_logic; \n\
          enable :  in std_logic; \n\
          tin : in std_logic_vector(%d downto 0); \n", base, extention, taps-1);
  fprintf(VHDLfile, "\
       tout : out std_logic\n\
         );      \n\
 end ARCH2_%s%s; \n\
  \n\
 architecture Behavioral of ARCH2_%s%s is \n\
 \n", base, extention, base, extention);
```

```
 fprintf(VHDLfile, "\
component biglut%s\n\
    Port ( clock : in std_logic; \n\
           enable :  in std_logic; \n\
           tin : in std_logic_vector(%d downto 0); \n", extention, taps-1);

 for(i=0;i<groups;i++)
 {
   fprintf(VHDLfile, "\
         posPartsum%d : out std_logic;\n", i);
 }
 for(i=0;i<groups;i++)
 {
   if(i!=groups-1)
     fprintf(VHDLfile, "\
         negPartsum%d : out std_logic;\n", i);
   else
     fprintf(VHDLfile, "\
         negPartsum%d : out std_logic);\n", i);
 }

 fprintf(VHDLfile, "\
end component; \n\
\n\
component endAddSub\n\
   port (\n\
        clock : in std_logic; \n\
       enable :  in std_logic; \n");

 for(i=0;i<groups;i++)
 {
   fprintf(VHDLfile, "\
        pos%d : in std_logic;\n", i);
 }
 for(i=0;i<groups;i++)
 {
   fprintf(VHDLfile, "\
        neg%d : in std_logic;\n", i);
 }

 fprintf(VHDLfile, "\
       sum : out std_logic\n\
       );\n\
end component;\n\
\n");

 for(i=0;i<groups;i++)
 {
```

```
    fprintf(VHDLfile, "\
 signal posPartsum%d : std_logic; \n", i);
  }

  for(i=0;i<groups;i++)
  {
    fprintf(VHDLfile, "\
 signal negPartsum%d : std_logic; \n", i);
  }

  fprintf(VHDLfile, "\n\
 begin \n\
 \n\
 biglut_0 : biglut%s\n\
    port map (clock, enable, tin, ", extention);

  for(i=0;i<groups;i++)
    fprintf(VHDLfile, "posPartsum%d, ", i);

  fprintf(VHDLfile, "\n\t\t");

  for(i=0;i<groups;i++)
  {
    if(i!=groups-1)
      fprintf(VHDLfile, "negPartsum%d, ", i);
    else
      fprintf(VHDLfile, "negPartsum%d);\n", i);
  }

  fprintf(VHDLfile, "\n\
 endAddSub_0 : endAddSub\n\
    port map (clock, enable, ");

  for(i=0;i<groups;i++)
    fprintf(VHDLfile, "posPartsum%d, ", i);

  fprintf(VHDLfile, "\n\t\t");

  for(i=0;i<groups;i++)
    fprintf(VHDLfile, "negPartsum%d, ", i);

  fprintf(VHDLfile, "tout);\n\
\n\
 end Behavioral; \n");

  fclose(VHDLfile);

  return 0;
```

```
}  // end createArch2File()



/**
* Creates the BigLutFile
*
* @param taps <code>int</code> number of taps in the initial file.
* @param groups <code>int</code> number of groups with 4 taps in each group
* @param base <code>char*</code> Base is "BIN" for binary numbers, "CSD2" for CSD/CSD2, "CSD4" for CSD4
* @param extention <code>char*</code> For CSD4 this function is called several times with a different extention each
*                                    For the other bases, extention can be given the value "".
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int createA2BigLutFile(int taps, int groups, char* base, char* extention)
{
  int i=0, j=0, k=0;

  char s[4][20];

  FILE *VHDLfile;
  char VHDLfilename[35];

  sprintf(VHDLfilename, "./ARCH2/%s/BigLut%s.vhdl", base, extention);

  if ((VHDLfile = fopen(VHDLfilename, "w")) == NULL)
         fprintf(stderr, "Cannot open %s\n", VHDLfilename);

  fprintf(VHDLfile, returnHeader());

  fprintf(VHDLfile, "\
 entity biglut%s is\n", extention);

  fprintf(VHDLfile, "\
     Port ( clock : in std_logic; \n\
                enable :  in std_logic; \n\
                tin : in std_logic_vector(%d downto 0); \n", taps-1);

  for(i=0;i<groups;i++)
  {
    fprintf(VHDLfile, "\
           posPartsum%d : out std_logic;\n", i);
  }
  for(i=0;i<groups;i++)
  {
    if(i!=groups-1)
```

```
     fprintf(VHDLfile, "\
           negPartsum%d : out std_logic;\n", i);
   else
     fprintf(VHDLfile, "\
           negPartsum%d : out std_logic);\n", i);
 }

 fprintf(VHDLfile, "\
end biglut%s;\n\n", extention);

 fprintf(VHDLfile, "\
architecture Behavioral of biglut%s is \n\n", extention);

 for(i=0;i<groups;i++)
 {
   fprintf(VHDLfile, "\
component lut%s_%d \n\
    Port ( clock : in std_logic; \n\
              enable :  in std_logic; \n\
              ti : in std_logic_vector(3 downto 0); \n\
           toutpos : out std_logic; \n\
              toutneg : out std_logic\n\
       ); \n\
 end component; \n\n", extention, i);
 }

 fprintf(VHDLfile, "\
begin \n\
 \n");

 for(i=0;i<groups;i++)
 {
   if(i*4+3 < taps)    //normal case
   {
     fprintf(VHDLfile, "\
lut_0%d : lut%s_%d\n\
   port map (clock, enable, tin(%d downto %d), posPartsum%d, negPartsum%d); \n\n",
     i, extention, i, i*4+3, i*4, i, i);
   }
   else                //last lutsub-entity
   {
     j = 0;
     for(k=i*4;k<i*4+4;k++)
     {
       if(i*4+j<taps)
       {
         sprintf(s[j], "tin(%d)", i*4+j);
       }
```

```
        else
          strcpy(s[j],"'0'");

        j++;
      }

      fprintf(VHDLfile, "\
 lut_0%d : lut%s_%d\n\
    port map (clock=>clock, enable=>enable, ti(0)=>%s, ti(1)=>%s, \n\
                ti(2)=>%s, ti(3)=>%s, \n\
                toutpos=>posPartsum%d, toutneg=>negPartsum%d); \n\n",
      i, extention, i, s[0] , s[1], s[2], s[3], i, i);
    }
  }
  fprintf(VHDLfile, "\
 end Behavioral; \n");


  return 0;
}


/**
* Creates the Lut-files for CSD2 and CSD4 numbers
*
* @param groups <code>int</code> number of groups with 4 taps in each group
* @param digits <code>int</code> number of digits in the CSD2 or CSD4 number
* @param tapGroups <code>double**</code> The value of the taps are stored here. 4 taps in each group
* @param base <code>char*</code> Base is "CSD2" for CSD/CSD2, "CSD4" for CSD4
* @param extention <code>char*</code> For CSD4 this function is called several times with a different extention each
*                                     For the other bases, extention can be given the value "".
* @param i_base <code>int</code> i_base is 2 for CSD2 and 4 for CSD4
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int createA2LutFiles(int groups, int digits, double** tapGroups, char* base, char* extention, int i_base)
{

  int i=0, j=0, k=0;
  int pos=0, neg=0;

  FILE *VHDLfile;
  char VHDLfilename[35];

  int*** csdArray; //[GROUPS][16][DIGITS];  // store the CSD numbers, 16 is 4^2, i.e. number of combinations of taps

  // allocate memory space for csdArray
```

```
// csdArray[GROUPS][16][DIGITS]
MALLOC(csdArray, sizeof(int**) * groups);
for (i = 0; i < groups; i++)
{
  MALLOC(csdArray[i], sizeof(int*) * 16);
  for(j = 0; j < 16; j++)
  {
    MALLOC(csdArray[i][j], sizeof(int) * digits);
  }
}

  // put values into csdArray
findCSDArray(groups, digits, tapGroups, i_base, csdArray);

  // for each file
for(i=0;i<groups;i++)
{
      // create a filename for each file
  sprintf(VHDLfilename, "./ARCH2/%s/Lut%s%d.vhdl", base, extention, i);

  if ((VHDLfile = fopen(VHDLfilename, "w")) == NULL)
       fprintf(stderr, "Cannot open %s\n", VHDLfilename);

  fprintf(VHDLfile, returnHeader());


  fprintf(VHDLfile, "\
entity lut%s_%d is \n\
    Port ( clock : in std_logic; \n\
                enable :  in std_logic; \n\
                ti : in std_logic_vector(3 downto 0); \n\
           toutpos : out std_logic; \n\
                toutneg : out std_logic \n\
                ); \n\
end lut%s_%d; \n\n", extention, i, extention, i);

  fprintf(VHDLfile, "\
architecture Behavioral of lut%s_%d is \n\
 \n\n", extention, i);

  fprintf(VHDLfile, "\
component lut  \n\
    generic (\n");

  for(j=0;j<digits-1;j++)
  {
    fprintf(VHDLfile, "\
           INIT%d : bit_vector := x\"16\";\n", j);
```

```
    }
    fprintf(VHDLfile, "\
             INIT%d : bit_vector := x\"16\");\n", digits-1);
    fprintf(VHDLfile, "\
    port ( \n\
            UI0   : in    std_logic;  \n\
            UI1   : in    std_logic;  \n\
            UI2   : in    std_logic;  \n\
            UI3   : in    std_logic;  \n\
            UO    : out   std_logic_vector(%d downto 0)  \n\
        ); \n\
 end component; \n\n", digits-1);

    fprintf(VHDLfile, "\
 component add_ff \n\
   port ( clock   : in    std_logic; \n\
          CE  : in    std_logic; \n\
          CLR : in    std_logic; \n\
             afi0 : in std_logic_vector(%d downto 0);\n\
             afso : out std_logic\n\
             );\n\
 end component;\n\n", digits-1);

    fprintf(VHDLfile, "\
 signal outpos : std_logic_vector(%d downto 0); \n\
 signal outneg : std_logic_vector(%d downto 0); \n\
\n", digits-1, digits-1);

    fprintf(VHDLfile, "\
 begin \n\
  \n\
 -- lut \n\
 poslut_0 : lut \n\
    generic map (\n");

    // determine which number to look for
    if(i_base == 2)
    {
      pos = 1;
      neg = -1;
    }
    else if(i_base == 4)
    {
      pos = atoi(extention+1);
      neg = -pos;
    }
    else
      oops("Error: input i_base must be 2 or 4. Otherwise wrong function is called.");
```

```
        // writing positive numbers
    for(j=0;j<digits;j++)
    {
      fprintf(VHDLfile, "\
            INIT%d=>\"", j);
      for(k=0;k<16;k++)
      {
        // currently writing positive numbers
        if(*( csdArray[i][15-k] + (digits-1-j) ) == pos)
          fprintf(VHDLfile, "1");
        else
          fprintf(VHDLfile, "0");
      }
      if(j != digits-1)
        fprintf(VHDLfile, "\",\n");
      else
        fprintf(VHDLfile, "\")\n");
    }

    fprintf(VHDLfile, "\
    port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),\n\
              UO=>outpos);  \n\
\n\
 neglut_0 : lut \n\
    generic map (\n");

        // writing negative numbers
    for(j=0;j<digits;j++)
    {
      fprintf(VHDLfile, "\
            INIT%d=>\"", j);
      for(k=0;k<16;k++)
      {
        // currently writing negative numbers
        if(*( csdArray[i][15-k] + (digits-1-j) ) == neg)
          fprintf(VHDLfile, "1");
        else
          fprintf(VHDLfile, "0");
      }
      if(j != digits-1)
        fprintf(VHDLfile, "\",\n");
      else
        fprintf(VHDLfile, "\")\n");
    }

    fprintf(VHDLfile, "\
    port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),\n\
```

```
              UO=>outneg);  \n\
  \n");

    fprintf(VHDLfile, "\
  add_ff_0: add_ff\n\
    port map (clock, enable, '0', outpos, toutpos); \n\
\n\
  add_ff_1: add_ff\n\
    port map (clock, enable, '0', outneg, toutneg); \n\
\n\
 end Behavioral; \n\n");

    fclose(VHDLfile);

  } // end of for each group


    // free allocated memory
  for (i = 0; i < groups; i++)
  {
    for (j = 0; j < 16; j++)
    {
      free(csdArray[i][j]);
    }
    free(csdArray[i]);
  }
  free(csdArray);


  return 0;
} // end of createLutSubFiles()



/**
* Create EndAddSub-File
*
* @param groups <code>int</code> number of groups with 4 taps in each group
* @param base <code>char*</code> Base is "CSD2" for CSD/CSD2, "CSD4" for CSD4
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int createA2EndAddSubFile(int groups, char* base)
{
  int i=0;

  FILE *VHDLfile;
```

```
  char VHDLfilename[35];

  sprintf(VHDLfilename, "./ARCH2/%s/EndAddSub.vhdl", base);

  if ((VHDLfile = fopen(VHDLfilename, "w")) == NULL)
         fprintf(stderr, "Cannot open %s\n", VHDLfilename);

  fprintf(VHDLfile, returnHeader());

  fprintf(VHDLfile, "\
 entity endAddSub is\n\
    port ( \n\
              clock : in std_logic;\n\
              enable : in std_logic;\n");

  for(i=0;i<groups;i++)
  {
    fprintf(VHDLfile, "\
              pos%d : in std_logic;\n", i);
  }

  for(i=0;i<groups;i++)
  {
    fprintf(VHDLfile, "\
              neg%d : in std_logic;\n", i);
  }

  fprintf(VHDLfile, "\
              sum : out std_logic\n\
              );\n\
 end endAddSub;\n\
\n\
 architecture Behavioral of endAddSub is \n\
\n\
component SUB2 \n\
    PORT(   \n\
               A:  IN STD_LOGIC;\n\
           B:  IN STD_LOGIC;\n\
           bin : in std_logic;\n\
              Q:  OUT STD_LOGIC;\n\
           bout :   OUT STD_LOGIC\n\
        );\n\
END component;\n\
\n\
 component addAll\n\
    port (\n\
         clock : in std_logic;\n\
         enable : in std_logic;\n");
```

```
  for(i=0;i<groups;i++)
  {
    fprintf(VHDLfile, "\
         tin%d : in std_logic;\n", i);
  }
  fprintf(VHDLfile, "\
         O : out    std_logic\n\
         );\n\
 end component;\n\
\n\
 signal sign0, bin0, sum0 : std_logic;\n\
\n\
 signal posSum : std_logic;\n\
 signal negSum : std_logic;\n\
\n\
 begin \n\
 \n\
 posAdd_0 : addAll\n\
    port map (clock, enable, ");

  for(i=0;i<groups;i++)
  {
    fprintf(VHDLfile, "pos%d, ", i);
  }

  fprintf(VHDLfile, "posSum);\n\
  \n\
 negAdd_0 : addAll \n\
    port map (clock, enable, ");

  for(i=0;i<groups;i++)
  {
    fprintf(VHDLfile, "neg%d, ", i);
  }

  fprintf(VHDLfile, "negSum);\n\
  \n\
 sub_0 : sub2 \n\
    port map (posSum, negSum, bin0, sum0, sign0);\n\
\n\
 flip_flop_0 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign0, Q=>bin0);\n\
\n\
 flip_flop_1 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sum0, Q=>sum);\n\
\n\
 end Behavioral;\n");
```

```
  fclose(VHDLfile);

  return 0;
}



/**
* Create AddAll-File
*
* @param groups <code>int</code> number of groups with 4 taps in each group
* @param base <code>char*</code> Base is "CSD2" for CSD/CSD2, "CSD4" for CSD4
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int createA2AddAllFile(int groups, char* base)
{
  int i=0, j=0;

  int noAdds=0, groups_temp=0, noSum=0, newSum=0, noGroups=0;
  int* order;
  int* adds;

  char a[15], b[15];

  FILE *VHDLfile;
  char VHDLfilename[35];

    // allocate memory space of the size of groups to array "order"
  MALLOC(order, sizeof(int) * groups);
  MALLOC(adds, sizeof(int) * groups);

  for(i=0;i<groups;i++)
  {
    order[i] = 0;
    adds[i] = 0;
  }

  sprintf(VHDLfilename, "./ARCH2/%s/AddAll.vhdl", base);

  if ((VHDLfile = fopen(VHDLfilename, "w")) == NULL)
         fprintf(stderr, "Cannot open %s\n", VHDLfilename);

  fprintf(VHDLfile, returnHeader());

  fprintf(VHDLfile, "\
```

```
 entity addAll is \n\
    port (\n\
         clock : in std_logic;\n\
         enable : in std_logic;\n");

  for(i=0;i<groups;i++)
  {
    fprintf(VHDLfile, "\
         tin%d : in std_logic;\n", i);
  }

  fprintf(VHDLfile, "\
         O : out    std_logic\n\
         );\n\
 end addAll;\n\
\n\
 architecture Behavioral of addAll is \n\
\n\
 component add2 \n\
   port\n\
   (\n\
      a:  in std_logic;\n\
      b:  in std_logic;\n\
      cii : in std_logic;\n\
      s: out std_logic;\n\
      coo : out std_logic\n\
   );\n\
 end component; \n\
\n");

    // find the number of additions needed.
  groups_temp = groups;
  noAdds = 0;
  i=0;
  while(groups_temp > 1)
  {
    order[i] = roof(groups_temp, 2);
    adds[i] = groups_temp/2;
    i++;

    newSum = groups_temp/2;
    noAdds += newSum;
    groups_temp -= newSum;
  }

  for(i=0;i<noAdds;i++)
  {
    fprintf(VHDLfile, "\
```

```
 signal sum%d0 : std_logic; \n\
 signal sum%d1 : std_logic; \n\
 signal sum%d1_d : std_logic; \n\
", i, i, i);
  }

  fprintf(VHDLfile, "\n");

  for(i=0;i<noAdds;i++)
  {
    fprintf(VHDLfile, "\
 signal cin%d : std_logic; \n\
 signal sign%d : std_logic; \n", i, i);
  }

    // in the general case noAdds > 2 is assumed. Otherwise, special case.
  if(noAdds > 2)
  {
    fprintf(VHDLfile, "\
\n\
 begin\n\
\n");

    noGroups = groups;
    noSum = noAdds;
    for(i=0;i<noAdds;i++)
    {
      while(order[i] > 0)
      {
        if(noGroups > 1)
        {
          sprintf(a, "tin%d", groups - noGroups);
          noGroups --;
          sprintf(b, "tin%d", groups - noGroups);
          noGroups --;
        }
        else if(noGroups == 1)
        {
          sprintf(a, "tin%d", groups - noGroups);
          noGroups --;
          sprintf(b, "sum01");
          noSum --;
        }
        else
        {
          sprintf(a, "sum%d1", noAdds - noSum);
          noSum --;
          sprintf(b, "sum%d1", noAdds - noSum);
```

```
        noSum --;
      }

      // insert a flipflop if addition has to be delayed
      if(order[i] == 1 && adds[i] == 0)
      {
        fprintf(VHDLfile, "\
 -- extra flipflop to syncronize the result\n\
flip_flop_d_%d : FDCE\n\
   port map (C=>clock, CE=>enable, CLR=>'0', D=>%s, Q=>sum%d1_d); \n", j, a, j);

        fprintf(VHDLfile, "\n\
add_%d : add2 \n\
   port map (a=>sum%d1_d, \n\
             b=>%s, \n\
             cii=>cin%d,\n\
             s=>sum%d0, \n\
             coo=>sign%d); \n\n", j, j, b, j, j, j);

        order[i+1] --;
        adds[i+1] --;
      }

      else
      {
        fprintf(VHDLfile, "\
add_%d : add2 \n\
   port map (a=>%s, \n\
             b=>%s, \n\
             cii=>cin%d,\n\
             s=>sum%d0, \n\
             coo=>sign%d); \n\n", j, a, b, j, j, j);
      }
      fprintf(VHDLfile, "\
flip_flop_%d : FDCE\n\
   port map (C=>clock, CE=>enable, CLR=>'0', D=>sign%d, Q=>cin%d); \n\n", j, j, j);

      if(j!=noAdds-1)
      {
        fprintf(VHDLfile, "\n\
flip_flop_s_%d : FDCE\n\
   port map (C=>clock, CE=>enable, CLR=>'0', D=>sum%d0, Q=>sum%d1); \n\n", j, j, j);
      }

      else
      {
        fprintf(VHDLfile, "\n\
flip_flop_s_%d : FDCE\n\
```

```
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sum%d0, Q=>O); \n\n", j, j);
        }

        order[i] --;
        adds[i] --;
        j++;
      } // end of while(order[i]>0)
    } //  end of for(i=0;i<noAdds;i++)
  } // end of if adds > 2

  else if(noAdds == 2)
  {
    fprintf(VHDLfile, "\
 signal partsum21 : std_logic; \n\
\n\
 begin\n\
\n");

    fprintf(VHDLfile, "\
 add_0 : add2 \n\
    port map (a=>tin0, \n\
                b=>tin1, \n\
                cii=>cin0,\n\
                s=>sum00, \n\
                coo=>sign0); \n\
\n\
 flip_flop_0 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign0, Q=>cin0);\n\
  \n\
 flip_flop_1 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sum00, Q=>sum01);\n\
  \n\
  -- extra flipflop to syncronize the result\n\
 flip_flop_2 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>tin2, Q=>partsum21);\n\
  \n\
 add_1 : add2 \n\
    port map (a=>partsum21, \n\
                b=>sum01, \n\
                cii=>cin1,\n\
                s=>sum10, \n\
                coo=>sign1); \n\
\n\
 flip_flop_3 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign1, Q=>cin1);\n\
\n\
 flip_flop_4 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sum10, Q=>O);\n");
```

```
  }
  else if(noAdds == 1)
  {
    fprintf(VHDLfile, "\
\n\
 begin\n\
\n");
    fprintf(VHDLfile, "\
 add_0 : add2 \n\
    port map (a=>tin0, \n\
                b=>tin1, \n\
                cii=>cin0,\n\
                s=>sum00, \n\
                coo=>sign0); \n\
\n\
 flip_flop_0 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign0, Q=>cin0);\n\
\n\
 flip_flop_1 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sum00, Q=>O);\n");
  }
  else if(noAdds == 0)
  {
    fprintf(VHDLfile, "\
\n\
 begin\n\
\n");
    fprintf(VHDLfile, "\
 O<=tin0;\n");
  }

  fprintf(VHDLfile, "\n\
 end Behavioral; \n");

  fclose(VHDLfile);

  return 0;
}
```

Arch2.h

```
int createArch2(int taps, int groups, double** tapGroups, int BINdigits, int CSD2digits, int CSD4digits);

int createBINArch2(int taps, int groups, double** tapGroups, int BINdigits);
int createCSD2Arch2(int taps, int groups, double** tapGroups, int CSD2digits);
int createCSD4Arch2(int taps, int groups, double** tapGroups, int CSD4digits);
```

```
int createArch2File(int taps, int groups, char* base, char* extention);
int createA2BigLutFile(int taps, int groups, char* base, char* extention);
int createA2LutFiles(int groups, int digits, double** tapGroups,
                     char* base, char* extention, int i_base);
int createA2EndAddSubFile(int groups, char* base);
int createA2AddAllFile(int groups, char* base);
```

Bin.c

```
#include "extIncl.h"

#include "bin.h"
#include "main.h"
#include "comFiles.h"

#define oops(s) { perror((s)); exit(EXIT_FAILURE); }
#define MALLOC(s,t) if(((s) = malloc(t)) == NULL) { oops("error: malloc() "); }


/**
* Creates the Lut-files for binary case
*
* @param groups <code>int</code> number of groups with 4 taps in each group
* @param digits <code>int</code> number of digits in the binary number
* @param tapGroups <code>double**</code> The value of the taps are stored here. 4 taps in each group
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int createBINLutFiles(int groups, int digits, double** tapGroups, char* arch)
{

  int i=0, j=0, k=0;
  int pos=0, neg=0;

  FILE *VHDLfile;
  char VHDLfilename[35];

  int*** binArray;

  // allocate memory space for binArray
  // binArray[GROUPS][16][DIGITS]
  MALLOC(binArray, sizeof(int**) * groups);
  for (i = 0; i < groups; i++)
  {
```

```
   MALLOC(binArray[i], sizeof(int*) * 16);
   for(j = 0; j < 16; j++)
   {
     MALLOC(binArray[i][j], sizeof(int) * digits);
   }
 }

 findBINArray(groups, digits, tapGroups, binArray);

   // for each file
 for(i=0;i<groups;i++)
 {
       // create a filename for each file
   sprintf(VHDLfilename, "./%s/BIN/Lut_%d.vhdl", arch, i);

   if ((VHDLfile = fopen(VHDLfilename, "w")) == NULL)
        fprintf(stderr, "Cannot open %s\n", VHDLfilename);

   fprintf(VHDLfile, returnHeader());

   fprintf(VHDLfile, "\
entity lut_%d is \n\
    Port ( clock : in std_logic; \n\
               enable :  in std_logic; \n\
               ti : in std_logic_vector(3 downto 0); \n\
          tout : out std_logic\n\
               ); \n\
end lut_%d; \n\n", i, i);

   fprintf(VHDLfile, "\
architecture Behavioral of lut_%d is \n\
 \n\n", i);

   fprintf(VHDLfile, "\
component lut  \n\
    generic (\n");

   for(j=0;j<digits-1;j++)
   {
     fprintf(VHDLfile, "\
           INIT%d : bit_vector := x\"16\";\n", j);
   }
   fprintf(VHDLfile, "\
           INIT%d : bit_vector := x\"16\");\n", digits-1);
   fprintf(VHDLfile, "\
   port ( \n\
           UI0   : in    std_logic;  \n\
           UI1   : in    std_logic;  \n\
```

```
            UI2   : in    std_logic;  \n\
            UI3   : in    std_logic;  \n\
            UO    : out   std_logic_vector(%d downto 0)  \n\
        ); \n\
end component; \n\n", digits-1);

   fprintf(VHDLfile, "\
component add_ff \n\
  port ( clock   : in    std_logic; \n\
         CE  : in    std_logic; \n\
         CLR : in    std_logic; \n\
            afi0 : in std_logic_vector(%d downto 0);\n\
            afso : out std_logic\n\
            );\n\
end component;\n\n", digits-1);

   fprintf(VHDLfile, "\
signal poslutout : std_logic_vector(%d downto 0); \n\n", digits-1);

   fprintf(VHDLfile, "\
begin \n\
 \n\
-- lut \n\
poslut_0 : lut \n\
   generic map (\n");

       // writing numbers
   for(j=0;j<digits;j++)
   {
     fprintf(VHDLfile, "\
           INIT%d=>\"", j);
     for(k=0;k<16;k++)
     {
       fprintf(VHDLfile, "%d", *( binArray[i][15-k] + (digits-1-j) ) );
     }
     if(j != digits-1)
       fprintf(VHDLfile, "\",\n");
     else
       fprintf(VHDLfile, "\")\n");
   }

   fprintf(VHDLfile, "\
   port map (UI0 => ti(0), UI1 => ti(1), UI2 => ti(2), UI3 => ti(3),\n\
             UO=>poslutout);  \n\
\n");

   fprintf(VHDLfile, "\
 add_ff_0: add_ff\n\
```

```
    port map (clock, enable, '0', poslutout, tout); \n\
\n\
 end Behavioral; \n");

    fclose(VHDLfile);

  } // end of for each group

    // free allocated memory
  for (i = 0; i < groups; i++)
  {
    for (j = 0; j < 16; j++)
    {
      free(binArray[i][j]);
    }
    free(binArray[i]);
  }
  free(binArray);


  return 0;
} // end of createLutFiles()



/**
* Creates the BigLutFile for Binary case
*
* @param taps <code>int</code> number of taps in the initial file.
* @param groups <code>int</code> number of groups with 4 taps in each group
* @param base <code>char*</code> Base is "BIN" for binary numbers, "CSD2" for CSD/CSD2, "CSD4" for CSD4
* @param extention <code>char*</code> For CSD4 this function is called several times with a different extention each
*                                    For the other bases, extention can be given the value "".
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int createBINA2BigLutFile(int taps, int groups)
{
  int i=0, j=0, k=0;

  char s[4][20];

  FILE *VHDLfile;
  char VHDLfilename[35];

  sprintf(VHDLfilename, "./ARCH2/BIN/BigLut.vhdl");
```

```
 if ((VHDLfile = fopen(VHDLfilename, "w")) == NULL)
        fprintf(stderr, "Cannot open %s\n", VHDLfilename);

 fprintf(VHDLfile, returnHeader());

 fprintf(VHDLfile, "\
entity biglut is\n");

 fprintf(VHDLfile, "\
    Port ( clock : in std_logic; \n\
               enable :  in std_logic; \n\
               tin : in std_logic_vector(%d downto 0); \n", taps-1);

 for(i=0;i<groups;i++)
 {
   if(i!=groups-1)
     fprintf(VHDLfile, "\
           tout%d : out std_logic;\n", i);
   else
     fprintf(VHDLfile, "\
           tout%d : out std_logic);\n", i);
 }

 fprintf(VHDLfile, "\
end biglut;\n\n");

 fprintf(VHDLfile, "\
architecture Behavioral of biglut is \n\n");

 for(i=0;i<groups;i++)
 {
   fprintf(VHDLfile, "\
component lut_%d \n\
    Port ( clock : in std_logic; \n\
               enable :  in std_logic; \n\
               ti : in std_logic_vector(3 downto 0); \n\
               tout : out std_logic\n\
       ); \n\
 end component; \n\n", i);
 }

 fprintf(VHDLfile, "\
begin \n\
 \n");

 for(i=0;i<groups;i++)
 {
   if(i*4+3 < taps)    //normal case
```

```
    {
      fprintf(VHDLfile, "\
 lut_0%d : lut_%d\n\
    port map (clock, enable, tin(%d downto %d), tout%d); \n\n",
      i, i, i*4+3, i*4, i);
    }
    else                //last lutsub-entity
    {
      j = 0;
      for(k=i*4;k<i*4+4;k++)
      {
        if(i*4+j<taps)
        {
          sprintf(s[j], "tin(%d)", i*4+j);
        }
        else
          strcpy(s[j],"'0'");

        j++;
      }

      fprintf(VHDLfile, "\
 lut_0%d : lut_%d\n\
    port map (clock=>clock, enable=>enable, ti(0)=>%s, ti(1)=>%s, \n\
                ti(2)=>%s, ti(3)=>%s, \n\
                tout=>tout%d); \n\n",
      i, i, s[0] , s[1], s[2], s[3], i);
    }
  }
  fprintf(VHDLfile, "\
 end Behavioral; \n");

  fclose(VHDLfile);

  return 0;
}


/**
* Creates the architecture-file
*
* @param taps <code>int</code> number of taps in the initial file.
* @param groups <code>int</code> number of groups with 4 taps in each group
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int createBINArch2File(int taps, int groups)
```

```
{
  int i=0;

  FILE *VHDLfile;
  char VHDLfilename[35];

  sprintf(VHDLfilename, "./ARCH2/BIN/Arch2_BIN.vhdl");

  if ((VHDLfile = fopen(VHDLfilename, "w")) == NULL)
        fprintf(stderr, "Cannot open %s\n", VHDLfilename);

  fprintf(VHDLfile, returnHeader());

  fprintf(VHDLfile, "\
 entity ARCH2_BIN is \n\
     Port ( clock : in std_logic; \n\
          enable :  in std_logic; \n\
          tin : in std_logic_vector(%d downto 0); \n",taps-1);
  fprintf(VHDLfile, "\
        tout : out std_logic\n\
          );      \n\
 end ARCH2_BIN; \n\
  \n\
 architecture Behavioral of ARCH2_BIN is \n\
 \n");

  fprintf(VHDLfile, "\
 component biglut\n\
     Port ( clock : in std_logic; \n\
               enable :  in std_logic; \n\
               tin : in std_logic_vector(%d downto 0); \n", taps-1);

  for(i=0;i<groups;i++)
  {
    if(i!=groups-1)
      fprintf(VHDLfile, "\
            tout%d : out std_logic;\n", i);
    else
      fprintf(VHDLfile, "\
            tout%d : out std_logic);\n", i);
  }

  fprintf(VHDLfile, "\
 end component; \n\
\n\
 component addAll  \n\
    port (\n\
         clock : in std_logic;\n\
```

```
        enable : in std_logic;\n");
  for(i=0;i<groups;i++)
  {
    fprintf(VHDLfile, "\
        tin%d : in std_logic;\n", i);
  }

  fprintf(VHDLfile, "\
        O : out    std_logic\n\
        );\n\
 end component;\n\
\n");

  for(i=0;i<groups;i++)
  {
    fprintf(VHDLfile, "\
 signal partsum%d : std_logic; \n", i);
  }


  fprintf(VHDLfile, "\n\
 begin \n\
 \n\
 biglut_0 : biglut\n\
    port map (clock, enable, tin, ");

  for(i=0;i<groups;i++)
  {
    if(i!=groups-1)
      fprintf(VHDLfile, "partsum%d, ", i);
    else
      fprintf(VHDLfile, "partsum%d);\n", i);
  }

  fprintf(VHDLfile, "\n\
 addAll_0 : addAll\n\
    port map (clock, enable, ");

  for(i=0;i<groups;i++)
    fprintf(VHDLfile, "partsum%d, ", i);

  fprintf(VHDLfile, "tout);\n\
\n\
 end Behavioral; \n");

  fclose(VHDLfile);

  return 0;
```

```
}  // end createArch2BINFile()
```

Bin.h

```
int createBINLutFiles(int groups, int digits, double** tapGroups, char* arch);
int createBINA2BigLutFile(int taps, int groups);
int createBINArch2File(int taps, int groups);
```

ComFiles.c

```
#include "csd.h"
#include "comFiles.h"

#define WORDLENGTH 8

#define oops(s) { perror((s)); exit(EXIT_FAILURE); }

// This file contains the code for files which are common for all architectures and designs.



/**
* Creates the Initital shift register's file.
*
* @param taps <code>int</code> number of taps in the initial file.
* @param digits <code>int</code> number of digits in the binary, CSD2 or CSD4 number
* @param arch <code>char*</code> The current architecture made, i.e. "ARCH1", "ARCH2" etc.
* @param base <code>char*</code> Base is "BIN" for binary numbers, "CSD2" for CSD/CSD2, "CSD4" for CSD4
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int createInitShiftRegFile(int taps, int digits, char* arch, char* base)
{
  int i=0, j=0;

  FILE *VHDLfile;
  char VHDLfilename[35];

  sprintf(VHDLfilename, "./%s/%s/Init_ShiftRegisters.vhdl", arch, base);

  if ((VHDLfile = fopen(VHDLfilename, "w")) == NULL)
        fprintf(stderr, "Cannot open %s\n", VHDLfilename);
```

```
 fprintf(VHDLfile, returnHeader());

 fprintf(VHDLfile, "\
entity INIT_SHIFTREGS is \n\
    Port ( clk, enable, load : in std_logic; \n\
         tin : in std_logic_vector(7 downto 0); \n\
         tout : out std_logic\n\
         );      \n\
end INIT_SHIFTREGS; \n");

 fprintf(VHDLfile, "\
\n\
architecture Behavioral of INIT_SHIFTREGS is \n\
\n\
component shiftreg%dbit \n\
   port ( C    : in    std_logic;        -- clock \n\
          CE   : in    std_logic;        -- enable \n\
          CLR  : in    std_logic;        -- clear \n\
          D    : in    std_logic_vector (%d downto 0);  -- input  \n\
          L    : in    std_logic;        -- load \n\
          LEFT : in    std_logic;        -- shift left \n\
          SLI  : in    std_logic;        -- add to left 0/1 \n\
          SRI  : in    std_logic;        -- add to right 0/1 \n\
          Q    : out   std_logic_vector (%d downto 0));     -- output \n\
end component; \n", WORDLENGTH+digits, WORDLENGTH+digits-1, WORDLENGTH+digits-1);

 fprintf(VHDLfile, "\
\n\
component FDCE\n\
     -- synopsys translate_off\n\
     generic( INIT : bit :=  '0');\n\
     -- synopsys translate_on\n\
     port ( C   : in    std_logic; \n\
            CE  : in    std_logic; \n\
            CLR : in    std_logic; \n\
            D   : in    std_logic; \n\
            Q   : out   std_logic);\n\
end component;\n\
\n\
component %s_%s\n\
    Port ( clock : in std_logic; \n\
         enable :  in std_logic; \n\
         tin : in std_logic_vector(%d downto 0); \n", arch, base, taps-1);
 fprintf(VHDLfile, "\
       tout : out std_logic\n\
         );      \n\
end component; \n\
```

```
\n\
 signal shiftOut : std_logic_vector(%d downto 0);\n\
\n\
", taps-1);

  for(i=0;i<taps;i++)
  {
    fprintf(VHDLfile, "\
 signal out%d : std_logic_vector(%d downto 0);\n", i, WORDLENGTH+digits-2);
  }

  fprintf(VHDLfile, "\n");

  for(i=0;i<taps-1;i++)
  {
    fprintf(VHDLfile, "\
 signal Q%d : std_logic_vector(%d downto 0);\n", i, WORDLENGTH-1);
  }

  fprintf(VHDLfile, "\n\
 signal GND : std_logic := '0';\n\
\n\
 begin \n");

    // print first shift register
  fprintf(VHDLfile, "\n\
 shiftreg_0 : shiftreg%dbit  \n\
    port map (C => clk, CE => enable, CLR => GND,\n\
            D(%d downto 0) => \"", WORDLENGTH+digits, digits-1);

  for(j=0;j<digits;j++)
    fprintf(VHDLfile, "0");

  fprintf(VHDLfile, "\", \n\
            D(%d downto %d) => tin,\n\
            L => load, LEFT => GND, SLI => GND, SRI => GND, \n\
            Q(0) => shiftOut(0), Q(%d downto 1) => out0\n\
            );\n\
\n\
 -- flipflops to hold the value for next shiftregister.\n\
", WORDLENGTH+digits-1, digits, WORDLENGTH+digits-1);


    // print flipflops and the rest of the shift registers
  for(i=1;i<taps;i++)
  {

    fprintf(VHDLfile, "\
```

```
 flipflop_%d0 : FDCE\n\
      port map (C=>clk, CE=>enable, CLR=>GND, D=>shiftOut(%d), Q=>Q%d(%d));\n\n", i-1, i-1, i-1, WORDLENGTH-1);

    // the number of flipflops is dependent on wordlength.
    for(j=1;j<WORDLENGTH;j++)
    {
      fprintf(VHDLfile, "\
 flipflop_%d%d :    FDCE\n\
      port map (C=>clk, CE=>enable, CLR=>GND, D=>Q%d(%d), Q=>Q%d(%d));\n\n", i-1, j, i-1, WORDLENGTH-j, i-1, WORDLENG
    }

    fprintf(VHDLfile, "\
\n\
 shiftreg_%d : shiftreg%dbit  \n\
    port map (C => clk, CE => enable, CLR => GND,\n\
            D(%d downto 0) => \"", i, WORDLENGTH+digits, digits-1);

    for(j=0;j<digits;j++)
      fprintf(VHDLfile, "0");

    fprintf(VHDLfile, "\", \n\
            D(%d downto %d) => Q%d,\n\
            L => load, LEFT => GND, SLI => GND, SRI => GND, \n\
            Q(0) => shiftOut(%d), Q(%d downto 1) => out%d\n\
            );\n", WORDLENGTH+digits-1, digits, i-1, i, WORDLENGTH+digits-1, i);


  } // end for shiftreg + flipflops

  fprintf(VHDLfile, "\n\
 arch0 : %s_%s\n\
     port map (clock=>clk, enable=>enable, \n\
                    tin=>shiftOut, tout=>tout\n\
          );      \n\
\n\
 end Behavioral;  \n", arch, base);

  fclose(VHDLfile);

  return 0;

}


/**
* Creates the flip-flop file
*
* @param digits <code>int</code> number of digits in the binary, CSD2 or CSD4 number
```

```
* @param arch <code>char*</code> The current architecture made, i.e. "ARCH1", "ARCH2" etc.
* @param base <code>char*</code> Base is "BIN" for binary numbers, "CSD2" for CSD/CSD2, "CSD4" for CSD4
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int createFlipFlopFile(int digits, char* arch, char* base)
{
  int i=0;

  FILE *VHDLfile;
  char VHDLfilename[35];
  sprintf(VHDLfilename, "./%s/%s/FlipFlop.vhdl", arch, base);

  if ((VHDLfile = fopen(VHDLfilename, "w")) == NULL)
        fprintf(stderr, "Cannot open %s\n", VHDLfilename);

  fprintf(VHDLfile, returnHeader());

  fprintf(VHDLfile, "\
entity flipflop is\n\
   port ( C   : in    std_logic; \n\
          CE  : in    std_logic; \n\
          CLR : in    std_logic; \n\
          D   : in    std_logic_vector (%d downto 0); \n\
          Q   : out   std_logic_vector (%d downto 0);\n\
        so : out std_logic);\n\
end flipflop;\n\n", digits, digits-1);

  fprintf(VHDLfile, "\
architecture BEHAVIORAL of flipflop is\n\
\n\
   attribute INIT       : STRING ;\n\
   attribute BOX_TYPE   : STRING ;\n\
   component FDCE\n\
      -- synopsys translate_off\n\
      generic( INIT : bit :=  '0');\n\
      -- synopsys translate_on\n\
      port ( C   : in    std_logic; \n\
             CE  : in    std_logic; \n\
             CLR : in    std_logic; \n\
             D   : in    std_logic; \n\
             Q   : out   std_logic);\n\
   end component;\n\
   attribute INIT of FDCE : COMPONENT is \"0\";\n\
   attribute BOX_TYPE of FDCE : COMPONENT is \"BLACK_BOX\";\n\
   \n\
begin\n\
```

```
\n\
   I_Q0 : FDCE\n\
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(0), Q=>so);\n\n");

  for(i=1;i<digits+1;i++)
  {
  fprintf(VHDLfile, "\
   I_Q%d : FDCE\n\
      port map (C=>C, CE=>CE, CLR=>CLR, D=>D(%d), Q=>Q(%d));\n\n", i, i, i-1);
  }

   fprintf(VHDLfile, "\
end BEHAVIORAL;\n");

  fclose(VHDLfile);

  return 0;
}



/**
* Creates the needed n-bit lut's file
*
* @param digits <code>int</code> number of digits in the binary, CSD2 or CSD4 number
* @param arch <code>char*</code> The current architecture made, i.e. "ARCH1", "ARCH2" etc.
* @param base <code>char*</code> Base is "BIN" for binary numbers, "CSD2" for CSD/CSD2, "CSD4" for CSD4
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/


int createNBitLutFile(int digits, char* arch, char* base)
{
  int i=0;

  FILE *VHDLfile;
  char VHDLfilename[35];
  sprintf(VHDLfilename, "./%s/%s/Lut%dBits.vhdl", arch, base, digits);

  if ((VHDLfile = fopen(VHDLfilename, "w")) == NULL)
        fprintf(stderr, "Cannot open %s\n", VHDLfilename);

  fprintf(VHDLfile, returnHeader());

  fprintf(VHDLfile, "\
 entity lut is \n\
     generic ( ");
```

```
  for(i=0;i<digits-1;i++)
  {
    fprintf(VHDLfile, "\
                INIT%d : bit_vector := x\"16\"; \n", i);
  }
  fprintf(VHDLfile, "\
                INIT%d : bit_vector := x\"16\");\n", digits-1);
  fprintf(VHDLfile, "\
    Port ( ui0 : in std_logic; \n\
            ui1 : in std_logic; \n\
            ui2 : in std_logic; \n\
            ui3 : in std_logic; \n\
            uo : out std_logic_vector(%d downto 0)); \n\
 end lut; \n\n", digits-1);

  fprintf(VHDLfile, "\
 architecture Behavioral of lut is \n\
       -- Component Declaration for LUT4 should be placed \n\
 -- after architecture statement but before begin keyword \n\
 component LUT4 \n\
    generic (INIT : bit_vector := x\"16\"); \n\
    port (O : out STD_LOGIC; \n\
            I0 : in STD_LOGIC; \n\
            I1 : in STD_LOGIC; \n\
            I2 : in STD_LOGIC; \n\
            I3 : in STD_LOGIC); \n\
 end component; \n\
 \n\
-- attribute RLOC       : STRING ;\n");

  for(i=0;i<digits;i++)
  {
    fprintf(VHDLfile, "\
-- attribute RLOC of LUT4_p%d : LABEL is \"R%dC0.S1\";\n", i, i/2);
  }
    fprintf(VHDLfile, "\
\n\
-- attribute HU_SET       : STRING ;\n");

  for(i=0;i<digits;i++)
  {
    fprintf(VHDLfile, "\
-- attribute HU_SET of LUT4_p%d : LABEL is \"lut\";\n", i);
  }

    fprintf(VHDLfile, "\n\
 -- Component Attribute specification for LUT4 \n\
```

```
 -- should be placed after architecture declaration but \n\
 -- before the begin keyword \n\
 attribute INIT : string; \n\
 begin \n");

  for(i=0;i<digits;i++)
  {
    fprintf(VHDLfile, "\
 LUT4_p%d : LUT4 generic map (INIT => INIT%d) \n\
    port map (O => uO(%d),I0 => uI0,I1 => ui1,I2 => ui2,I3 => ui3); \n", i, i, i);
  }


  fprintf(VHDLfile, "\n\
 end Behavioral; \n");

  fclose(VHDLfile);

  return 0;
}


/**
* Creates the Add and flip-flop file
*
* @param digits <code>int</code> number of digits in the binary, CSD2 or CSD4 number
* @param arch <code>char*</code> The current architecture made, i.e. "ARCH1", "ARCH2" etc.
* @param base <code>char*</code> Base is "BIN" for binary numbers, "CSD2" for CSD/CSD2, "CSD4" for CSD4
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/


int createAddFFFile(int digits, char* arch, char* base)
{

  int i=0;

  FILE *VHDLfile;
  char VHDLfilename[35];
  sprintf(VHDLfilename, "./%s/%s/AddAndFlipFlop.vhdl", arch, base);

  if ((VHDLfile = fopen(VHDLfilename, "w")) == NULL)
         fprintf(stderr, "Cannot open %s\n", VHDLfilename);

  fprintf(VHDLfile, returnHeader());

  fprintf(VHDLfile, "\
```

```
entity add_ff is\n\
   port ( clock   : in    std_logic; \n\
          CE  : in    std_logic; \n\
          CLR : in    std_logic; \n\
            afi0 : in std_logic_vector(%d downto 0);\n\
        afso : out std_logic);\n\
end add_ff;\n\n", digits-1);

  fprintf(VHDLfile, "\
architecture BEHAVIORAL of add_ff is\n\
\n\
component add16bit\n\
    port ( A   : in    std_logic_vector (15 downto 0);  \n\
           B   : in    std_logic_vector (15 downto 0);  \n\
           CI  : in    std_logic;  \n\
           CO  : out   std_logic;  \n\
           OFL : out   std_logic;  \n\
           S   : out   std_logic_vector (15 downto 0)); \n\
end component;\n\
\n\
component flipflop\n\
   port ( C   : in    std_logic; \n\
          CE  : in    std_logic; \n\
          CLR : in    std_logic; \n\
          D   : in    std_logic_vector (%d downto 0); \n\
          Q   : out   std_logic_vector (%d downto 0);\n\
        so : out std_logic);\n\
end component;\n\n", digits, digits-1);

  fprintf(VHDLfile, "\
signal outadder : std_logic_vector(%d downto 0);\n\
signal inadder : std_logic_vector(%d downto 0);\n\n", digits, digits-1);

  if(digits<14)
  {
    fprintf(VHDLfile, "\
signal dummy0 : std_logic;\n\
signal dummy1 : std_logic_vector(%d downto 0);\n\
signal dummy2 : std_logic;\n\n", 15-digits-1);
  }
  else if(digits==14)
  {
    fprintf(VHDLfile, "\
signal dummy0 : std_logic;\n\
signal dummy1 : std_logic;\n\
signal dummy2 : std_logic;\n\n");
  }
  else if(digits==15)
```

```
  {
    fprintf(VHDLfile, "\
signal dummy0 : std_logic;\n\
signal dummy2 : std_logic;\n\n");
  }

  fprintf(VHDLfile, "\
begin\n\
\n");

  if(digits<15)
  {
    fprintf(VHDLfile, "\
  adder : add16bit\n\
    port map ( A(%d downto 0) => afi0, A(15 downto %d)=>\"", digits-1, digits);

    for(i=0;i<15-(digits-1);i++)
    {
      fprintf(VHDLfile, "0");
    }
    fprintf(VHDLfile, "\",\n");
    fprintf(VHDLfile, "\
             B(%d downto 0) => inadder, B(15 downto %d)=>\"", digits-1, digits);
    for(i=0;i<15-(digits-1);i++)
    {
      fprintf(VHDLfile, "0");
    }
    fprintf(VHDLfile, "\",\n");
    fprintf(VHDLfile, "\
             CI => '0', CO=> dummy0, OFL => dummy2, S(%d downto 0)=> outadder,\n\
             S(15 downto %d)=> dummy1);\n\n", digits, digits+1);

  } // end of if(digits<15)

  else if(digits==15)
  {
    fprintf(VHDLfile, "\
  adder : add16bit\n\
    port map ( A(14 downto 0) => afi0, A(15)=>'0',\n\
             B(14 downto 0) => inadder, B(15)=>'0',\n\
             CI => '0', CO=> dummy0, OFL => dummy2, S=> outadder);\n\n");
  }

  fprintf(VHDLfile, "\
  ff : flipflop\n\
    port map (C => clock, CE => CE, CLR => CLR, D => outadder, Q => inadder, \n\
            so => afso);\n\
\n\
```

```
 end BEHAVIORAL;\n");


  fclose(VHDLfile);

  return 0;
}



/**
* Creates the fixed add.vhdl file
*
* @param arch <code>char*</code> The current architecture made, i.e. "ARCH1", "ARCH2" etc.
* @param base <code>char*</code> Base is "BIN" for binary numbers, "CSD2" for CSD/CSD2, "CSD4" for CSD4
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int createAddFile(char* arch, char* base)
{

  FILE *VHDLfile;
  char VHDLfilename[35];
  sprintf(VHDLfilename, "./%s/%s/Add.vhdl", arch, base);

  if ((VHDLfile = fopen(VHDLfilename, "w")) == NULL)
         fprintf(stderr, "Cannot open %s\n", VHDLfilename);

  fprintf(VHDLfile, returnHeader());

  fprintf(VHDLfile, "\
entity ADD2 is\n\
   port\n\
   (\n\
      a:  in std_logic;\n\
      b:  in std_logic;\n\
      cii : in std_logic;\n\
      s: out std_logic;\n\
      coo : out std_logic\n\
   );\n\
end ADD2;\n\
\n\
architecture FULLADDER of ADD2 is\n\
\n\
signal q : std_logic;\n\
\n\
begin\n\
```

```
\n\
   q <= (A xor B); \n\
    s <= (q xor cii); \n\
   coo <= ( (A and B) or (q and cii) );\n\
\n\
end FULLADDER;\n");


  fclose(VHDLfile);

  return 0;
}


/**
* Creates the fixed sub.vhdl file
*
* @param arch <code>char*</code> The current architecture made, i.e. "ARCH1", "ARCH2" etc.
* @param base <code>char*</code> Base is "BIN" for binary numbers, "CSD2" for CSD/CSD2, "CSD4" for CSD4
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int createSubFile(char* arch, char* base)
{

  FILE *VHDLfile;
  char VHDLfilename[35];
  sprintf(VHDLfilename, "./%s/%s/Sub.vhdl", arch, base);

  if ((VHDLfile = fopen(VHDLfilename, "w")) == NULL)
        fprintf(stderr, "Cannot open %s\n", VHDLfilename);

  fprintf(VHDLfile, returnHeader());

  fprintf(VHDLfile, "\
ENTITY SUB2 IS\n\
    PORT(   \n\
              A:  IN STD_LOGIC;\n\
          B:  IN STD_LOGIC;\n\
          bin : in std_logic;\n\
             Q:  OUT STD_LOGIC;\n\
          bout :   OUT STD_LOGIC\n\
       );\n\
END SUB2;\n\
\n\
ARCHITECTURE logic OF SUB2 IS\n\
 \n\
```

```
BEGIN   \n\
   Q <= (A xor B xor bin); \n\
\n\
   bout <= (not A and B) or (not A and bin) or (B and bin);\n\
                \n\
END logic;\n\
\n");

  fclose(VHDLfile);

  return 0;
}



/**
* Creates the fixed add16.vhdl file
*
* @param arch <code>char*</code> The current architecture made, i.e. "ARCH1", "ARCH2" etc.
* @param base <code>char*</code> Base is "BIN" for binary numbers, "CSD2" for CSD/CSD2, "CSD4" for CSD4
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int createAdd16File(char* arch, char* base)
{
  int i=0;

  FILE *VHDLfile;
  char VHDLfilename[35];
  sprintf(VHDLfilename, "./%s/%s/Add16.vhdl", arch, base);

  if ((VHDLfile = fopen(VHDLfilename, "w")) == NULL)
        fprintf(stderr, "Cannot open %s\n", VHDLfilename);

  fprintf(VHDLfile, returnHeader());

  fprintf(VHDLfile, "\
 entity add16bit is \n\
    port ( A   : in    std_logic_vector (15 downto 0);  \n\
           B   : in    std_logic_vector (15 downto 0);  \n\
           CI  : in    std_logic;  \n\
           CO  : out   std_logic;  \n\
           OFL : out   std_logic;  \n\
           S   : out   std_logic_vector (15 downto 0));\n\
 end add16bit; \n\
  \n\
 architecture BEHAVIORAL of add16bit is \n\
    attribute BOX_TYPE   : STRING ; \n\
```

```
--    attribute RLOC       : STRING ; \n");

  for(i=0;i<15;i++)
  {
    fprintf(VHDLfile, "\
    signal C%d       : std_logic; \n", i);
  }

  fprintf(VHDLfile, "\
    signal C14O     : std_logic; \n\
    signal dummy    : std_logic; \n");

  for(i=0;i<16;i++)
  {
    fprintf(VHDLfile, "\
    signal I%d       : std_logic; \n", i);
  }

  fprintf(VHDLfile, "\
    signal CO_DUMMY : std_logic; \n\
    component FMAP \n\
       port ( I1 : in    std_logic;  \n\
              I2 : in    std_logic;  \n\
              I3 : in    std_logic;  \n\
              I4 : in    std_logic;  \n\
O  : in    std_logic); \n\
    end component; \n\
    attribute BOX_TYPE of FMAP : COMPONENT is \"BLACK_BOX\"; \n\
     \n\
    component MUXCY_L \n\
       port ( CI : in    std_logic;  \n\
              DI : in    std_logic;  \n\
              S  : in    std_logic;  \n\
              LO : out   std_logic); \n\
    end component; \n\
    attribute BOX_TYPE of MUXCY_L : COMPONENT is \"BLACK_BOX\"; \n\
     \n\
    component MUXCY \n\
       port ( CI : in    std_logic;  \n\
              DI : in    std_logic;  \n\
              S  : in    std_logic;  \n\
              O  : out   std_logic); \n\
    end component; \n\
    attribute BOX_TYPE of MUXCY : COMPONENT is \"BLACK_BOX\"; \n\
     \n\
    component XORCY \n\
       port ( CI : in    std_logic;  \n\
              LI : in    std_logic;  \n\
```

```
              O  : out   std_logic); \n\
    end component; \n\
    attribute BOX_TYPE of XORCY : COMPONENT is \"BLACK_BOX\"; \n\
     \n\
    component MUXCY_D \n\
       port ( CI : in    std_logic;  \n\
              DI : in    std_logic;  \n\
              S  : in    std_logic;  \n\
              LO : out   std_logic;  \n\
              O  : out   std_logic); \n\
    end component; \n\
    attribute BOX_TYPE of MUXCY_D : COMPONENT is \"BLACK_BOX\"; \n\
     \n\
    component XOR2 \n\
       port ( I0 : in    std_logic;  \n\
              I1 : in    std_logic;  \n\
              O  : out   std_logic); \n\
    end component; \n\
    attribute BOX_TYPE of XOR2 : COMPONENT is \"BLACK_BOX\"; \n\
     \n\
--    attribute RLOC of I_36_16 : LABEL is \"R3C0.S1\"; \n\
--    attribute RLOC of I_36_17 : LABEL is \"R3C0.S1\"; \n\
--    attribute RLOC of I_36_18 : LABEL is \"R2C0.S1\"; \n\
--    attribute RLOC of I_36_19 : LABEL is \"R2C0.S1\"; \n\
--    attribute RLOC of I_36_20 : LABEL is \"R1C0.S1\"; \n\
--    attribute RLOC of I_36_21 : LABEL is \"R1C0.S1\"; \n\
--    attribute RLOC of I_36_22 : LABEL is \"R0C0.S1\"; \n\
--    attribute RLOC of I_36_23 : LABEL is \"R0C0.S1\"; \n");
  fprintf(VHDLfile, "\
--    attribute RLOC of I_36_55 : LABEL is \"R3C0.S1\"; \n\
--    attribute RLOC of I_36_58 : LABEL is \"R2C0.S1\"; \n\
--    attribute RLOC of I_36_62 : LABEL is \"R2C0.S1\"; \n\
--    attribute RLOC of I_36_63 : LABEL is \"R1C0.S1\"; \n\
--    attribute RLOC of I_36_64 : LABEL is \"R0C0.S1\"; \n\
--    attribute RLOC of I_36_107 : LABEL is \"R0C0.S1\"; \n\
--    attribute RLOC of I_36_110 : LABEL is \"R1C0.S1\"; \n\
--    attribute RLOC of I_36_111 : LABEL is \"R3C0.S1\"; \n\
--    attribute RLOC of I_36_248 : LABEL is \"R4C0.S1\"; \n\
--    attribute RLOC of I_36_249 : LABEL is \"R4C0.S1\"; \n\
--    attribute RLOC of I_36_250 : LABEL is \"R5C0.S1\"; \n\
--    attribute RLOC of I_36_251 : LABEL is \"R5C0.S1\"; \n\
--    attribute RLOC of I_36_252 : LABEL is \"R6C0.S1\"; \n\
--    attribute RLOC of I_36_253 : LABEL is \"R6C0.S1\"; \n\
--    attribute RLOC of I_36_254 : LABEL is \"R7C0.S1\"; \n\
--    attribute RLOC of I_36_255 : LABEL is \"R7C0.S1\"; \n\
--    attribute RLOC of I_36_272 : LABEL is \"R7C0.S1\"; \n\
--    attribute RLOC of I_36_275 : LABEL is \"R7C0.S1\"; \n\
--    attribute RLOC of I_36_279 : LABEL is \"R6C0.S1\"; \n\
```

```
--    attribute RLOC of I_36_283 : LABEL is \"R6C0.S1\"; \n\
--    attribute RLOC of I_36_287 : LABEL is \"R5C0.S1\"; \n\
--    attribute RLOC of I_36_291 : LABEL is \"R5C0.S1\"; \n\
--    attribute RLOC of I_36_295 : LABEL is \"R4C0.S1\"; \n\
--    attribute RLOC of I_36_299 : LABEL is \"R4C0.S1\"; \n\
 begin \n\
    CO <= CO_DUMMY; \n\
    I_36_16 : FMAP \n\
       port map (I1=>A(8), I2=>B(8), I3=>dummy, I4=>dummy, O=>I8); \n\
     \n\
    I_36_17 : FMAP \n\
       port map (I1=>A(9), I2=>B(9), I3=>dummy, I4=>dummy, O=>I9); \n\
     \n\
    I_36_18 : FMAP \n\
       port map (I1=>A(10), I2=>B(10), I3=>dummy, I4=>dummy, O=>I10); \n\
     \n\
    I_36_19 : FMAP \n\
       port map (I1=>A(11), I2=>B(11), I3=>dummy, I4=>dummy, O=>I11); \n\
     \n\
    I_36_20 : FMAP \n\
       port map (I1=>A(12), I2=>B(12), I3=>dummy, I4=>dummy, O=>I12); \n\
     \n\
    I_36_21 : FMAP \n\
       port map (I1=>A(13), I2=>B(13), I3=>dummy, I4=>dummy, O=>I13); \n\
     \n\
    I_36_22 : FMAP \n\
       port map (I1=>A(14), I2=>B(14), I3=>dummy, I4=>dummy, O=>I14); \n\
     \n");
  fprintf(VHDLfile, "\
    I_36_23 : FMAP \n\
       port map (I1=>A(15), I2=>B(15), I3=>dummy, I4=>dummy, O=>I15); \n\
     \n\
    I_36_55 : MUXCY_L \n\
       port map (CI=>C8, DI=>A(9), S=>I9, LO=>C9); \n\
     \n\
    I_36_58 : MUXCY_L \n\
       port map (CI=>C10, DI=>A(11), S=>I11, LO=>C11); \n\
\n\
    I_36_62 : MUXCY_L \n\
       port map (CI=>C9, DI=>A(10), S=>I10, LO=>C10); \n\
     \n\
    I_36_63 : MUXCY_L \n\
       port map (CI=>C11, DI=>A(12), S=>I12, LO=>C12); \n\
     \n\
    I_36_64 : MUXCY \n\
       port map (CI=>C14, DI=>A(15), S=>I15, O=>CO_DUMMY); \n\
     \n\
    I_36_73 : XORCY \n\
```

```
   port map (CI=>C7, LI=>I8, O=>S(8)); \n\
 \n\
I_36_74 : XORCY \n\
   port map (CI=>C8, LI=>I9, O=>S(9)); \n\
 \n\
I_36_75 : XORCY \n\
   port map (CI=>C10, LI=>I11, O=>S(11)); \n\
 \n\
I_36_76 : XORCY \n\
   port map (CI=>C9, LI=>I10, O=>S(10)); \n\
 \n\
I_36_77 : XORCY \n\
   port map (CI=>C12, LI=>I13, O=>S(13)); \n\
 \n\
I_36_78 : XORCY \n\
   port map (CI=>C11, LI=>I12, O=>S(12)); \n\
 \n\
I_36_80 : XORCY \n\
   port map (CI=>C14, LI=>I15, O=>S(15)); \n\
 \n\
I_36_81 : XORCY \n\
   port map (CI=>C13, LI=>I14, O=>S(14)); \n\
 \n\
I_36_107 : MUXCY_D \n\
   port map (CI=>C13, DI=>A(14), S=>I14, LO=>C14, O=>C14O); \n\
 \n\
I_36_110 : MUXCY_L \n\
   port map (CI=>C12, DI=>A(13), S=>I13, LO=>C13); \n\
 \n\
I_36_111 : MUXCY_L \n\
   port map (CI=>C7, DI=>A(8), S=>I8, LO=>C8); \n\
 \n\
I_36_226 : XORCY \n\
   port map (CI=>CI, LI=>I0, O=>S(0)); \n\
 \n\
I_36_227 : XORCY \n\
   port map (CI=>C0, LI=>I1, O=>S(1)); \n\
 \n\
I_36_228 : XORCY \n\
   port map (CI=>C2, LI=>I3, O=>S(3)); \n\
 \n\
I_36_229 : XORCY \n\
   port map (CI=>C1, LI=>I2, O=>S(2)); \n\
 \n\
I_36_230 : XORCY \n\
   port map (CI=>C4, LI=>I5, O=>S(5)); \n\
 \n\
I_36_231 : XORCY \n\
```

```
       port map (CI=>C3, LI=>I4, O=>S(4)); \n\
     \n\
    I_36_233 : XORCY \n\
       port map (CI=>C6, LI=>I7, O=>S(7)); \n\
     \n\
    I_36_234 : XORCY \n\
       port map (CI=>C5, LI=>I6, O=>S(6)); \n\
     \n\
    I_36_248 : MUXCY_L \n\
       port map (CI=>C6, DI=>A(7), S=>I7, LO=>C7); \n\
     \n\
    I_36_249 : MUXCY_L \n");
  fprintf(VHDLfile, "\
       port map (CI=>C5, DI=>A(6), S=>I6, LO=>C6); \n\
     \n\
    I_36_250 : MUXCY_L \n\
       port map (CI=>C4, DI=>A(5), S=>I5, LO=>C5); \n\
     \n\
    I_36_251 : MUXCY_L \n\
       port map (CI=>C3, DI=>A(4), S=>I4, LO=>C4); \n\
     \n\
    I_36_252 : MUXCY_L \n\
port map (CI=>C2, DI=>A(3), S=>I3, LO=>C3); \n\
     \n\
    I_36_253 : MUXCY_L \n\
       port map (CI=>C1, DI=>A(2), S=>I2, LO=>C2); \n\
     \n\
    I_36_254 : MUXCY_L \n\
       port map (CI=>C0, DI=>A(1), S=>I1, LO=>C1); \n\
     \n\
    I_36_255 : MUXCY_L \n\
       port map (CI=>CI, DI=>A(0), S=>I0, LO=>C0); \n\
     \n\
    I_36_272 : FMAP \n\
       port map (I1=>A(1), I2=>B(1), I3=>dummy, I4=>dummy, O=>I1); \n\
     \n\
    I_36_275 : FMAP \n\
       port map (I1=>A(0), I2=>B(0), I3=>dummy, I4=>dummy, O=>I0); \n\
     \n\
    I_36_279 : FMAP \n\
       port map (I1=>A(2), I2=>B(2), I3=>dummy, I4=>dummy, O=>I2); \n\
     \n\
    I_36_283 : FMAP \n\
       port map (I1=>A(3), I2=>B(3), I3=>dummy, I4=>dummy, O=>I3); \n\
     \n\
    I_36_287 : FMAP \n\
       port map (I1=>A(4), I2=>B(4), I3=>dummy, I4=>dummy, O=>I4); \n\
     \n\
```

```
  I_36_291 : FMAP \n\
     port map (I1=>A(5), I2=>B(5), I3=>dummy, I4=>dummy, O=>I5); \n\
    \n\
  I_36_295 : FMAP \n\
     port map (I1=>A(6), I2=>B(6), I3=>dummy, I4=>dummy, O=>I6); \n\
    \n\
  I_36_299 : FMAP \n\
     port map (I1=>A(7), I2=>B(7), I3=>dummy, I4=>dummy, O=>I7); \n\
    \n\
  I_36_354 : XOR2\n\
     port map (I0=>A(0), I1=>B(0), O=>I0); \n\
    \n\
  I_36_355 : XOR2 \n\
     port map (I0=>A(1), I1=>B(1), O=>I1); \n\
    \n\
  I_36_356 : XOR2 \n\
     port map (I0=>A(2), I1=>B(2), O=>I2); \n\
    \n\
  I_36_357 : XOR2 \n\
     port map (I0=>A(3), I1=>B(3), O=>I3); \n\
    \n\
  I_36_358 : XOR2 \n\
     port map (I0=>A(4), I1=>B(4), O=>I4); \n\
    \n\
  I_36_359 : XOR2 \n\
     port map (I0=>A(5), I1=>B(5), O=>I5); \n\
    \n\
  I_36_360 : XOR2 \n\
     port map (I0=>A(6), I1=>B(6), O=>I6); \n\
    \n\
  I_36_361 : XOR2 \n\
     port map (I0=>A(7), I1=>B(7), O=>I7); \n\
    \n\
  I_36_362 : XOR2 \n\
     port map (I0=>A(8), I1=>B(8), O=>I8); \n\
    \n\
  I_36_363 : XOR2 \n\
     port map (I0=>A(9), I1=>B(9), O=>I9); \n\
    \n");
fprintf(VHDLfile, "\
  I_36_364 : XOR2 \n\
     port map (I0=>A(10), I1=>B(10), O=>I10); \n\
    \n\
  I_36_365 : XOR2 \n\
     port map (I0=>A(11), I1=>B(11), O=>I11); \n\
    \n\
  I_36_366 : XOR2 \n\
     port map (I0=>A(12), I1=>B(12), O=>I12); \n\
```

```
     \n\
    I_36_367 : XOR2 \n\
port map (IO=>A(13), I1=>B(13), O=>I13); \n\
     \n\
    I_36_368 : XOR2 \n\
       port map (IO=>A(14), I1=>B(14), O=>I14); \n\
     \n\
    I_36_369 : XOR2 \n\
       port map (IO=>A(15), I1=>B(15), O=>I15); \n\
     \n\
    I_36_375 : XOR2 \n\
       port map (IO=>C14O, I1=>CO_DUMMY, O=>OFL); \n\
     \n\
 end BEHAVIORAL; \n");

  fclose(VHDLfile);

  return 0;
}


/**
* Creates the fixed ShiftReg16.vhdl file
*
* @param digits <code>int</code> number of digits in the binary, CSD2 or CSD4 number
* @param arch <code>char*</code> The current architecture made, i.e. "ARCH1", "ARCH2" etc.
* @param base <code>char*</code> Base is "BIN" for binary numbers, "CSD2" for CSD/CSD2, "CSD4" for CSD4
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int createShiftRegFile(int digits, char* arch, char* base)
{
  int i=0;

  FILE *VHDLfile;
  char VHDLfilename[35];
  sprintf(VHDLfilename, "./%s/%s/ShiftReg%d.vhdl", arch, base, WORDLENGTH+digits);

  if ((VHDLfile = fopen(VHDLfilename, "w")) == NULL)
        fprintf(stderr, "Cannot open %s\n", VHDLfilename);

  fprintf(VHDLfile, returnHeader());

  fprintf(VHDLfile, "\
 entity M2_1_MXILINX_test is \n\
   port ( D0 : in    std_logic;  \n\
          D1 : in    std_logic;  \n\
```

```
          S0 : in    std_logic;  \n\
          O  : out   std_logic); \n\
end M2_1_MXILINX_test; \n\
 \n\
architecture BEHAVIORAL of M2_1_MXILINX_test is \n\
   attribute BOX_TYPE   : STRING ; \n\
   signal M0 : std_logic; \n\
   signal M1 : std_logic; \n\
   component AND2B1 \n\
      port ( I0 : in    std_logic;  \n\
             I1 : in    std_logic;  \n\
             O  : out   std_logic); \n\
   end component; \n\
   attribute BOX_TYPE of AND2B1 : COMPONENT is \"BLACK_BOX\"; \n\
    \n\
   component OR2 \n\
      port ( I0 : in    std_logic; \n\
             I1 : in    std_logic; \n\
             O  : out   std_logic); \n\
   end component; \n\
   attribute BOX_TYPE of OR2 : COMPONENT is \"BLACK_BOX\"; \n\
    \n\
   component AND2 \n\
      port ( I0 : in    std_logic; \n\
             I1 : in    std_logic;  \n\
             O  : out   std_logic); \n\
   end component; \n\
   attribute BOX_TYPE of AND2 : COMPONENT is \"BLACK_BOX\"; \n\
    \n\
begin \n\
   I_36_7 : AND2B1 \n\
      port map (I0=>S0, I1=>D0, O=>M0); \n\
    \n\
   I_36_8 : OR2 \n\
      port map (I0=>M1, I1=>M0, O=>O);\n\
    \n\
   I_36_9 : AND2 \n\
      port map (I0=>D1, I1=>S0, O=>M1); \n\
    \n\
end BEHAVIORAL; \n\
 \n");

 fprintf(VHDLfile, returnHeader());

 fprintf(VHDLfile, "\
entity shiftreg%dbit is \n\
   port ( C    : in    std_logic;   -- clock \n\
          CE   : in    std_logic;   -- enable \n\
```

```
          CLR  : in    std_logic;   -- clear \n\
          D    : in    std_logic_vector (%d downto 0);   -- input \n\
          L    : in    std_logic;   -- load \n\
          LEFT : in    std_logic;   -- shift left \n\
          SLI  : in    std_logic;   -- add to left 0/1 \n\
          SRI  : in    std_logic;   -- add to right 0/1 \n\
          Q    : out   std_logic_vector (%d downto 0));  -- output \n\
end shiftreg%dbit; \n", WORDLENGTH+digits, WORDLENGTH+digits-1, WORDLENGTH+digits-1, WORDLENGTH+digits);

 fprintf(VHDLfile, "\
 \n\
architecture BEHAVIORAL of shiftreg%dbit is \n\
   attribute HU_SET     : STRING ; \n\
   attribute INIT       : STRING ; \n\
   attribute BOX_TYPE   : STRING ; \n\
   signal L_LEFT  : std_logic; \n\
   signal L_OR_CE : std_logic; \n", WORDLENGTH+digits);

 for(i=0;i<WORDLENGTH+digits;i++)
 {
   fprintf(VHDLfile, "\
   signal MDL%d    : std_logic; \n", i);
 }
 for(i=0;i<WORDLENGTH+digits;i++)
 {
   fprintf(VHDLfile, "\
   signal MDR%d    : std_logic; \n", i);
 }

 fprintf(VHDLfile, "\
   signal Q_DUMMY : std_logic_vector (%d downto 0); \n\
   component M2_1_MXILINX_test \n\
      port ( D0 : in    std_logic;  \n\
             D1 : in    std_logic;  \n\
             S0 : in    std_logic;  \n\
             O  : out   std_logic); \n\
   end component; \n", WORDLENGTH+digits-1);

 fprintf(VHDLfile, "\
    \n\
   component FDCE \n\
      -- synopsys translate_off \n\
      generic( INIT : bit :=  '0'); \n\
      -- synopsys translate_on \n\
      port ( C   : in    std_logic;  \n\
             CE  : in    std_logic;  \n\
             CLR : in    std_logic;  \n\
             D   : in    std_logic;  \n\
```

```
            Q   : out   std_logic); \n\
    end component; \n\
    attribute INIT of FDCE : COMPONENT is \"0\"; \n\
    attribute BOX_TYPE of FDCE : COMPONENT is \"BLACK_BOX\"; \n\
     \n\
    component OR2 \n\
       port ( I0 : in    std_logic;  \n\
              I1 : in    std_logic;  \n\
              O  : out   std_logic); \n\
    end component; \n\
    attribute BOX_TYPE of OR2 : COMPONENT is \"BLACK_BOX\"; \n\
     \n");


  fprintf(VHDLfile, "\
\n\
-- attribute RLOC       : STRING ; \n");

  for(i=0;i<WORDLENGTH+digits;i++)
  {
    fprintf(VHDLfile, "\
-- attribute RLOC of I_Q%d : LABEL is \"R%dC0.S1\"; \n", i, 3 + i/2);
  }

  for(i=0;i<WORDLENGTH+digits;i++)
  {
    fprintf(VHDLfile, "\
-- attribute HU_SET of I_Q%d : LABEL is \"fdc\"; \n", i);
  }


  fprintf(VHDLfile, "\
\n\
 begin \n\
    Q(%d downto 0) <= Q_DUMMY(%d downto 0); \n\
    I_MDL0 : M2_1_MXILINX_test \n\
       port map (D0=>SLI, D1=>D(0), S0=>L, O=>MDL0); \n\
     \n", WORDLENGTH+digits-1, WORDLENGTH+digits-1 );

  for(i=1;i<WORDLENGTH+digits;i++)
  {
    fprintf(VHDLfile, "\
    I_MDL%d : M2_1_MXILINX_test \n\
       port map (D0=>Q_DUMMY(%d), D1=>D(%d), S0=>L, O=>MDL%d); \n\
     \n", i, i-1, i, i);
  }

  for(i=0;i<WORDLENGTH+digits-1;i++)
```

```
  {
    fprintf(VHDLfile, "\
    I_MDR%d : M2_1_MXILINX_test \n\
       port map (D0=>Q_DUMMY(%d), D1=>MDL%d, S0=>L_LEFT, O=>MDR%d); \n\
     \n", i, i+1, i, i);
  }

  fprintf(VHDLfile, "\
    I_MDR%d : M2_1_MXILINX_test \n\
       port map (D0=>SRI, D1=>MDL%d, S0=>L_LEFT, O=>MDR%d); \n\
     \n", WORDLENGTH+digits-1, WORDLENGTH+digits-1, WORDLENGTH+digits-1);

  for(i=0;i<WORDLENGTH+digits;i++)
  {
    fprintf(VHDLfile, "\
    I_Q%d : FDCE \n\
       port map (C=>C, CE=>L_OR_CE, CLR=>CLR, D=>MDR%d, Q=>Q_DUMMY(%d)); \n\
     \n", i, i, i);
  }

  fprintf(VHDLfile, "\
    I_36_145 : OR2 \n\
       port map (I0=>L, I1=>CE, O=>L_OR_CE); \n\
     \n\
    I_36_161 : OR2 \n\
       port map (I0=>LEFT, I1=>L, O=>L_LEFT); \n\
     \n\
 end BEHAVIORAL; \n");

  fclose(VHDLfile);

  return 0;
}


/**
* Creates the fixed test.vhd file
*
* @param digits <code>int</code> number of digits in the binary, CSD2 or CSD4 number
* @param groups <code>int</code> number of groups with 4 taps in each group
* @param arch <code>char*</code> The current architecture made, i.e. "ARCH1", "ARCH2" etc.
* @param base <code>char*</code> Base is "BIN" for binary numbers, "CSD2" for CSD/CSD2, "CSD4" for CSD4
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int createTestFile(int digits, int groups, char* arch, char* base)
{
```

```
int i=0;

FILE *VHDLfile;
char VHDLfilename[35];

char* testSample[28]; // the test samples

int accWait = 0;      // accumulated waiting time
int ex = 0;
if(!strcmp(base,"CSD4"))
  ex = 4;

testSample[0] = "00000001";
testSample[1] = "00111111";
testSample[2] = "11111100";
testSample[3] = "01111111";
testSample[4] = "11111110";
testSample[5] = "11111100";
testSample[6] = "11111001";
testSample[7] = "10011111";
testSample[8] = "00111111";
testSample[9] = "01111111";
testSample[10] = "11100111";
testSample[11] = "00000011";
testSample[12] = "00000110";
testSample[13] = "01100000";
testSample[14] = "11000000";
testSample[15] = "10000000";
testSample[16] = "00011000";
testSample[17] = "00100100";
testSample[18] = "01000010";
testSample[19] = "10000001";
testSample[20] = "00000000";
testSample[21] = "11100000";
testSample[22] = "00000111";
testSample[23] = "11110000";
testSample[24] = "00001111";
testSample[25] = "00011111";
testSample[26] = "11111111";
testSample[27] = "11111000";

sprintf(VHDLfilename, "./%s/%s/InitShift_Test.vhd", arch, base);

if ((VHDLfile = fopen(VHDLfilename, "w")) == NULL)
       fprintf(stderr, "Cannot open %s\n", VHDLfilename);

fprintf(VHDLfile, returnHeader());
```

```
  fprintf(VHDLfile, "\
\n\
LIBRARY ieee;\n\
USE ieee.std_logic_1164.ALL;\n\
USE ieee.std_logic_unsigned.all;\n\
USE ieee.numeric_std.ALL;\n\
\n\
ENTITY init_shift_test_vhd IS\n\
END init_shift_test_vhd;\n\
\n\
ARCHITECTURE behavior OF init_shift_test_vhd IS \n\
\n\
    -- Component Declaration for the Unit Under Test (UUT)\n\
    COMPONENT init_shiftregs\n\
    PORT(\n\
        clk : IN std_logic;\n\
        enable : IN std_logic;\n\
        load : IN std_logic;\n\
        tin : IN std_logic_vector(%d downto 0);\n\
        tout : OUT std_logic\n\
        );\n\
    END COMPONENT;\n", WORDLENGTH-1);

  fprintf(VHDLfile, "\
\n\
    --Inputs\n\
    SIGNAL clk :  std_logic := '0';\n\
    SIGNAL enable :  std_logic := '0';\n\
    SIGNAL load :  std_logic := '0';\n\
    SIGNAL tin :  std_logic_vector(%d downto 0) := (others=>'0');\n", WORDLENGTH-1);

  fprintf(VHDLfile, "\
\n\
    --Outputs\n\
    SIGNAL tout :  std_logic;\n\
\n\
BEGIN\n\
\n\
    -- Instantiate the Unit Under Test (UUT)\n\
    uut: init_shiftregs PORT MAP(\n\
        clk => clk,\n\
        enable => enable,\n\
        load => load,\n\
        tin => tin,\n\
        tout => tout\n\
    );\n\
\n\
   PROCESS -- Process for clock\n\
```

```
    BEGIN\n\
        CLOCK_LOOP : LOOP\n\
        clk <= transport '0';\n\
        WAIT FOR 10 ns;\n\
        clk <= transport '1';\n\
        WAIT FOR 10 ns;\n\
        WAIT FOR 40 ns;\n\
        clk <= transport '0';\n\
        WAIT FOR 40 ns;\n\
        END LOOP CLOCK_LOOP;\n\
    END PROCESS;\n\
\n\
    tb : PROCESS\n\
    BEGIN\n\
\n\
        -- Wait 100 ns for global reset to finish\n\
        wait for 100 ns;\n\
\n\
        -- 1st\n\
        -- --------------------\n\
        enable <= transport '1';\n\
        load <= transport '0';\n");
  fprintf(VHDLfile, "\
        tin <= transport std_logic_vector'(\"%s\"); \n\
        -- --------------------\n\
        WAIT FOR 100 ns; -- Time=100 ns\n\
        load <= transport '1';\n\
        -- --------------------\n\
        WAIT FOR 100 ns; -- Time=200 ns\n\
        load <= transport '0';\n\
        -- --------------------\n\
        -- \n\
        WAIT FOR %d ns; -- Time=%d ns\n",
    testSample[0], (WORDLENGTH+digits+ex-2)*100, (WORDLENGTH+digits+ex-1)*100 );

  accWait = 300 + (WORDLENGTH+digits+ex-2)*100;

  for(i=1;i<28;i++)
  {
    fprintf(VHDLfile, "\
        load <= transport '1';\n\
        tin <= transport std_logic_vector'(\"%s\"); \n\
        -- --------------------\n\
        WAIT FOR 100 ns; \n\
        load <= transport '0';\n\
        -- --------------------\n\
        -- \n\
        WAIT FOR %d ns; \n", testSample[i], (WORDLENGTH+digits+ex-1)*100 );
```

```
    accWait += (WORDLENGTH+digits+ex)*100;
  }

    // the end waitings consists of two parts, a static part, the same for all
    //     numbers of taps, and a variable part, consisting of end additions.

    // static part of end waiting
  if(!strcmp(base,"BIN"))
    ex = 0;
  else if(!strcmp(base, "CSD2"))
    ex = 2;
  else if(!strcmp(base, "CSD4"))
    ex = 7;
  else
    oops("Error: Base must be either BIN, CSD2 or CSD4");

    // variable part of end waiting
  while(groups > 1)
  {
    ex += 1;
    groups = roof(groups, 2);
  }

  fprintf(VHDLfile, "\
        -- --------------------\n\
        -- end waiting \n\
        WAIT FOR %d ns; \n", (digits+ex)*100 );

  accWait += (digits+ex)*100;

  fprintf(VHDLfile, "\
        -- --------------------\n\
\n\
-- accumulated runtime is: %d ns \n\
\n\
    END PROCESS;\n\
\n\
END;\n", accWait);

  fclose(VHDLfile);

  return 0;
}


/**
* Returns a fixed header value used in all the VHDL-files.
```

```
*
* @return <code>char*</code> Fixed header.
*/

char* returnHeader()
{
  char* ret = "\n\
   -- Created by Guri Kristine Birkeland 2005 \n\
  \n\
 library IEEE; \n\
 use IEEE.STD_LOGIC_1164.ALL; \n\
 use IEEE.STD_LOGIC_ARITH.ALL; \n\
 use IEEE.STD_LOGIC_UNSIGNED.ALL; \n\
  \n\
 --  Uncomment the following lines to use the declarations that are \n\
 --  provided for instantiating Xilinx primitive components. \n\
 library UNISIM; \n\
 use UNISIM.VComponents.all; \n\
  \n";

  return ret;

}
```

ComFiles.h

```
int createInitShiftRegFile(int taps, int digits, char* arch, char* base);
int createFlipFlopFile(int digits, char* arch, char* base);
int createNBitLutFile(int digits, char* arch, char* base);
int createAddFFFile(int digits, char* arch, char* base);

int createAddFile(char* arch, char* base);
int createSubFile(char* arch, char* base);
int createAdd16File(char* arch, char* base);
int createShiftRegFile(int digits, char* arch, char* base);
int createTestFile(int digits, int groups, char* arch, char* base);

char* returnHeader();
```

Csd.h

```
#include <malloc.h>
#include <stdio.h>
```

```
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <ctype.h>
#include <string.h>
#include <io.h>
#include <sys/stat.h>



typedef struct
{
    double *array;
    unsigned int length;
    }d_arr;
typedef struct
{
    unsigned int *array;
    unsigned int length;
    }u_arr;


/*
i_arr   : An integer array of unsigned int length
             -----                   -------
*/
typedef struct
{
    int *array;
    unsigned int length;
    }i_arr;




typedef struct
{
    unsigned int r;
    i_arr A;
    d_arr scale;
    }csd_r_type;




#define EMPTY 5000
#define MAXINT 5000
```

```
enum err_msg{
    NO_FILE=0,  //cannot open file
    FILE_ERR,   //incorrect file format
    FILE_ERR_NODE,  //incorrect file format: node name
    FILE_ERR_NS,    //incorrect file format: node specs
    FILE_ERR_UN,    //incorrect file format: unspecified
    MEM_ERR     //insufficient memory
    };


void        err_disp(unsigned int msg,char comment[20]);

unsigned int    GCD(unsigned int a, unsigned int b);            //GCD.C

i_arr       bcd_csd(double in,csd_r_type convcsd,unsigned int b);   //BCD_CSDR.c
void        CSD_r(csd_r_type *convcsd);                             //CSD_R.c
i_arr       float_to_csd(double in,csd_r_type convcsd,
            unsigned int b);                //FL_CSD_R.C
u_arr       form_csd_alphabet(csd_r_type convcsd);          //FORMALPH.C

void program (double* a);   // GK
```

CSD4.c

```
#include "extIncl.h"

#include "CSD4.h"
#include "csd.h"
#include "main.h"
#include "comFiles.h"

#define oops(s) { perror((s)); exit(EXIT_FAILURE); }


/**
* Creates the architecture-file for the CSD4 architecture.
*
* @param taps <code>int</code> number of taps in the initial file.
* @param arch <code>char*</code> The current architecture made, i.e. "ARCH1", "ARCH2" etc.
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int createCSD4ArchFile(int taps, char* arch)
{
```

```
  FILE *VHDLfile;
  char VHDLfilename[35];

  sprintf(VHDLfilename, "./%s/CSD4/%s_CSD4.vhdl", arch, arch);

  if ((VHDLfile = fopen(VHDLfilename, "w")) == NULL)
        fprintf(stderr, "Cannot open %s\n", VHDLfilename);

  fprintf(VHDLfile, returnHeader());

  fprintf(VHDLfile, "\
entity %s_CSD4 is\n\
     Port ( clock : in std_logic; \n\
          enable :  in std_logic; \n\
          tin : in std_logic_vector(%d downto 0); \n\
         tout : out std_logic\n\
          );      \n\
end %s_CSD4;\n\
\n", arch, taps-1, arch);

  fprintf(VHDLfile, "\
architecture Behavioral of %s_CSD4 is\n\
\n\
 component add5and7\n\
     Port ( clock : in std_logic; \n\
                enable :  in std_logic; \n\
                tin : in std_logic_vector(%d downto 0); \n\
            tout : out std_logic\n\
                );    \n\
 end component; \n\
\n", arch, taps-1);

  fprintf(VHDLfile, "\
 component add9and11\n\
     Port ( clock : in std_logic; \n\
                enable :  in std_logic; \n\
                tin : in std_logic_vector(%d downto 0); \n\
            tout : out std_logic\n\
                );    \n\
 end component; \n\
\n", taps-1);

  fprintf(VHDLfile, "\
 component add13and15\n\
     Port ( clock : in std_logic; \n\
                enable :  in std_logic; \n\
                tin : in std_logic_vector(%d downto 0); \n\
            tout : out std_logic\n\
```

```
               );    \n\
 end component; \n\
\n", taps-1);

  fprintf(VHDLfile, "\
 component ADD2\n\
    port (\n\
      a:  in std_logic;\n\
      b:  in std_logic;\n\
      cii : in std_logic;\n\
      s: out std_logic;\n\
      coo : out std_logic\n\
    );\n\
 end component;\n\
\n\
signal partsum0: std_logic; \n\
signal partsum01: std_logic; \n\
signal partsum1: std_logic; \n\
signal partsum2: std_logic; \n\
signal partsum21: std_logic; \n\
\n\
signal sum0: std_logic;\n\
signal sum01: std_logic;\n\
\n\
signal cin0, cin1 : std_logic;\n\
signal sign0, sign1 : std_logic;\n\
signal total : std_logic;\n\
\n\
begin\n\
\n\
addand_0 : add5and7\n\
    port map(clock, enable, tin, partsum0);\n\
\n\
addand_1 : add9and11\n\
    port map(clock, enable, tin, partsum1);\n\
\n\
addand_2 : add13and15\n\
    port map(clock, enable, tin, partsum2);\n\
\n\
flip_flop_00 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>partsum0, Q=>partsum01);\n\
\n\
add_0 : ADD2\n\
    port map( partsum01, partsum1, cin0, sum0, sign0);\n\
\n\
flip_flop_0 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign0, Q=>cin0);\n\
\n\
```

```
flip_flop_01 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sum0, Q=>sum01);\n\
\n\
flip_flop_02 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>partsum2, Q=>partsum21);\n\
\n\
add_1 : ADD2\n\
    port map( partsum21, sum01, cin1, total, sign1);\n\
\n\
flip_flop_1 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign1, Q=>cin1);\n\
\n\
flip_flop_2 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>total, Q=>tout);\n\
\n\
end Behavioral;\n");

  fclose(VHDLfile);

  return 0;
}



/**
* Function createCSD4EndAddFile creates the endadd-files for the CSD4 architecture.
*
* @param taps <code>int</code> number of taps in the initial file.
* @param a <code>int</code> the number of the first of the CSD4-architectures to add.
* @param b <code>int</code> the number of the second of the CSD4-architectures to add.
* @param arch <code>char*</code> The current architecture made, i.e. "ARCH1", "ARCH2" etc.
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int createCSD4EndAddFile(int taps, int a, int b, char* arch)
{
  char ret[2000];

  FILE *VHDLfile;
  char VHDLfilename[35];

  sprintf(VHDLfilename, "./%s/CSD4/Add%dAnd%d.vhdl", arch, a, b);

  if ((VHDLfile = fopen(VHDLfilename, "w")) == NULL)
        fprintf(stderr, "Cannot open %s\n", VHDLfilename);

  fprintf(VHDLfile, returnHeader());
```

```
  fprintf(VHDLfile, "\
 entity add%dand%d is\n\
     Port ( clock : in std_logic; \n\
                enable :  in std_logic; \n\
                tin : in std_logic_vector(%d downto 0); \n\
            tout : out std_logic\n\
                );    \n\
 end add%dand%d; \n\
 \n", a, b, taps-1, a, b);

  fprintf(VHDLfile, "\
 architecture Behavioral of add%dand%d is\n\
\n", a, b);

  fprintf(VHDLfile, "\
 component %s_CSD4_%d\n\
     Port ( clock : in std_logic; \n\
                enable :  in std_logic; \n\
                tin : in std_logic_vector(%d downto 0); \n\
            tout : out std_logic\n\
                );    \n\
 end component; \n\
\n", arch, a, taps-1);

  fprintf(VHDLfile, "\
 component %s_CSD4_%d\n\
     Port ( clock : in std_logic; \n\
                enable :  in std_logic; \n\
                tin : in std_logic_vector(%d downto 0); \n\
            tout : out std_logic\n\
                );    \n\
 end component; \n\
\n", arch, b, taps-1);

  fprintf(VHDLfile, "\
 component SUB2\n\
    port (\n\
      a:  in std_logic;\n\
      b:  in std_logic;\n\
      bin : in std_logic;\n\
      q: out std_logic;\n\
      bout : out std_logic\n\
    );\n\
 end component;\n\
\n\
 component ADD2\n\
   port\n\
   (\n\
```

```
      a:  in std_logic;\n\
      b:  in std_logic;\n\
      cii : in std_logic;\n\
      s: out std_logic;\n\
      coo : out std_logic\n\
   );\n\
 end component;\n\
\n");

  fprintf(VHDLfile, "\
 signal f_out0: std_logic; \n\
 signal f_out1: std_logic; \n\
 signal f_out2: std_logic; \n\
 signal f_out3: std_logic; \n");

  if(a==13)
  {
    fprintf(VHDLfile, "\
 signal f_out4: std_logic; \n\
 signal out_f_00: std_logic;\n\
 signal out_f_01: std_logic;\n");
  }

  fprintf(VHDLfile, "\
 signal out_f0: std_logic;\n\
 signal out_f1: std_logic;\n");

  if(a==9)
  {
    fprintf(VHDLfile, "\
 signal out_f2: std_logic;\n");
  }

  fprintf(VHDLfile, "\
\n\
 signal s_out0: std_logic; \n\
 signal s_out1: std_logic; \n\
 signal s_out2: std_logic; \n\
 signal s_out3: std_logic; \n");

  if(b==11)
  {
    fprintf(VHDLfile, "\
 signal s_out4: std_logic; \n\
 signal out_s_00: std_logic;\n\
 signal out_s_01: std_logic;\n");
  }
  else if(b==15)
```

```
  {
    fprintf(VHDLfile, "\
 signal s_out4: std_logic;\n");
  }

  fprintf(VHDLfile, "\
 signal out_s0: std_logic;\n\
 signal out_s1: std_logic;\n\
\n\
 signal cin0 : std_logic;\n\
 signal sign0 : std_logic;\n");

  if(a==13)
  {
    fprintf(VHDLfile, "\
 signal cin_f_0: std_logic;\n\
 signal sign_f_0 : std_logic;\n");
  }

  fprintf(VHDLfile, "\
 signal cin_f : std_logic;\n\
 signal sign_f : std_logic;\n");

  if(b==11)
  {
    fprintf(VHDLfile, "\
 signal cin_s_0: std_logic;\n\
 signal sign_s_0 : std_logic;\n");
  }

  fprintf(VHDLfile, "\
 signal cin_s : std_logic;\n\
 signal sign_s : std_logic;\n\
\n\
 signal total : std_logic;\n\
\n");

  if(a==5)
  {
    writeEndFor5And7(arch, ret);
    fprintf( VHDLfile, ret );
  }
  else if(a==9)
  {
    writeEndFor9And11(arch, ret);
    fprintf( VHDLfile, ret );
  }
  else if(a==13)
```

```
  {
    writeEndFor13And15(arch, ret);
    fprintf( VHDLfile, ret );
  }
  else
    oops("Error: Wrong input to \"createEndAddFile()\". int a must be 5, 9 or 13");

  fclose(VHDLfile);

  return 0;
}



/**
* Returns a fixed ending of the add5and7.vhdl-file.
*
* @param arch <code>char*</code> The current architecture made, i.e. "ARCH1", "ARCH2" etc.
* @param ret <code>char*</code> The address where to put the return string
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int writeEndFor5And7(char* arch, char* ret)
{
  sprintf(ret, "\n\
begin\n\
\n\
-- about number 5\n\
--------------------------------------------------------------------------------------------------\n\
addf_0 : %s_CSD4_5\n\
    port map(clock, enable, tin, f_out0);\n\
\n\
flip_flop_f0 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>f_out0, Q=>f_out1);\n\
\n\
flip_flop_f1 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>f_out1, Q=>f_out2);\n\
\n\
flip_flop_f2 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>f_out2, Q=>f_out3);\n\
\n\
add_f : ADD2\n\
    port map(f_out1, f_out3, cin_f, out_f0, sign_f);\n\
\n\
flip_flop_f3 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign_f, Q=>cin_f);\n\
\n\
```

```
flip_flop_f4 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>out_f0, Q=>out_f1);\n\
\n\
-- about number 7\n\
-------------------------------------------------------------------------------------------\n\
adds_0 : %s_CSD4_7\n\
    port map(clock, enable, tin, s_out0);\n\
\n\
flip_flop_s0 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>s_out0, Q=>s_out1);\n\
\n\
flip_flop_s1 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>s_out1, Q=>s_out2);\n\
\n\
flip_flop_s2 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>s_out2, Q=>s_out3);\n\
\n\
sub_s : SUB2\n\
    port map(s_out3, s_out0, cin_s, out_s0, sign_s);\n\
\n\
flip_flop_s3 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign_s, Q=>cin_s);\n\
\n\
flip_flop_s4 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>out_s0, Q=>out_s1);\n\
\n\
-- end adding\n\
\n\
add_0 : ADD2\n\
    port map(out_f1, out_s1, cin0, total, sign0);\n\
\n\
flip_flop_0 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign0, Q=>cin0);\n\
\n\
flip_flop_1 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>total, Q=>tout);\n\
\n\
end Behavioral;\n", arch, arch);


  return 0;
}




/**
* Returns a fixed ending of the add9and11.vhdl-file.
```

```
*
* @param arch <code>char*</code> The current architecture made, i.e. "ARCH1", "ARCH2" etc.
* @param ret <code>char*</code> The address where to put the return string
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int writeEndFor9And11(char* arch, char* ret)
{
  sprintf(ret, "\n\
begin\n\
\n\
-- about number 9\n\
--------------------------------\n\
addf_0 : %s_CSD4_9\n\
    port map(clock, enable, tin, f_out0);\n\
\n\
flip_flop_f0 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>f_out0, Q=>f_out1);\n\
\n\
flip_flop_f1 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>f_out1, Q=>f_out2);\n\
\n\
flip_flop_f2 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>f_out2, Q=>f_out3);\n\
\n\
add_f : ADD2\n\
    port map(f_out0, f_out3, cin_f, out_f0, sign_f);\n\
\n\
flip_flop_f3 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign_f, Q=>cin_f);\n\
\n\
flip_flop_f4 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>out_f0, Q=>out_f1);\n\
\n\
flip_flop_f5 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>out_f1, Q=>out_f2);\n\
\n\
\n\
-- about number 11\n\
-----------------------------------\n\
adds_0 : %s_CSD4_11\n\
    port map(clock, enable, tin, s_out0);\n\
\n\
flip_flop_s0 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>s_out0, Q=>s_out1);\n\
\n\
add_s_0 : ADD2\n\
```

```
    port map(s_out0, s_out1, cin_s_0, out_s_00, sign_s_0);\n\
\n\
flip_flop_s1 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign_s_0, Q=>cin_s_0);\n\
\n\
flip_flop_s2 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>out_s_00, Q=>out_s_01);\n\
\n\
\n\
flip_flop_s3 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>s_out1, Q=>s_out2);\n\
\n\
flip_flop_s4 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>s_out2, Q=>s_out3);\n\
\n\
flip_flop_s5 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>s_out3, Q=>s_out4);\n\
\n\
add_s : ADD2\n\
    port map(out_s_01, s_out4, cin_s, out_s0, sign_s);\n\
\n\
flip_flop_s6 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign_s, Q=>cin_s);\n\
\n\
flip_flop_s7 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>out_s0, Q=>out_s1);\n\
\n\
-- end adding\n\
\n\
add_0 : ADD2\n\
    port map(out_f2, out_s1, cin0, total, sign0);\n\
\n\
flip_flop_0 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign0, Q=>cin0);\n\
\n\
flip_flop_1 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>total, Q=>tout);\n\
\n\
end Behavioral;\n", arch, arch);


  return 0;
}




/**
```

```
* Returns a fixed ending of the add13and15.vhdl-file.
*
* @param arch <code>char*</code> The current architecture made, i.e. "ARCH1", "ARCH2" etc.
* @param ret <code>char*</code> The address where to put the return string
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int writeEndFor13And15(char* arch, char* ret)
{
  sprintf(ret, "\n\
begin\n\
\n\
-- about number 13\n\
---------------------------------------\n\
addf_0 : %s_CSD4_13\n\
    port map(clock, enable, tin, f_out0);\n\
\n\
flip_flop_f0 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>f_out0, Q=>f_out1);\n\
\n\
flip_flop_f1 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>f_out1, Q=>f_out2);\n\
\n\
add_f_0 : ADD2\n\
    port map(f_out0, f_out2, cin_f_0, out_f_00, sign_f_0);\n\
\n\
flip_flop_f2 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign_f_0, Q=>cin_f_0);\n\
\n\
flip_flop_f3 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>out_f_00, Q=>out_f_01);\n\
\n\
flip_flop_f4 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>f_out2, Q=>f_out3);\n\
\n\
flip_flop_f5 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>f_out3, Q=>f_out4);\n\
\n\
add_f_1 : ADD2\n\
    port map(out_f_01, f_out4, cin_f, out_f0, sign_f);\n\
\n\
flip_flop_f6 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign_f, Q=>cin_f);\n\
\n\
flip_flop_f7 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>out_f0, Q=>out_f1);\n\
\n\
```

```
-- about number 15\n\
---------------------------------------\n\
adds_0 : %s_CSD4_15\n\
    port map(clock, enable, tin, s_out0);\n\
\n\
flip_flop_s0 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>s_out0, Q=>s_out1);\n\
\n\
flip_flop_s1 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>s_out1, Q=>s_out2);\n\
\n\
flip_flop_s2 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>s_out2, Q=>s_out3);\n\
\n\
flip_flop_s3 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>s_out3, Q=>s_out4);\n\
\n\
sub_s : SUB2\n\
    port map(s_out4, s_out0, cin_s, out_s0, sign_s);\n\
\n\
flip_flop_s4 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign_s, Q=>cin_s);\n\
\n\
flip_flop_s5 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>out_s0, Q=>out_s1);\n\
\n\
\n\
-- end adding\n\
add_0 : ADD2\n\
    port map(out_f1, out_s1, cin0, total, sign0);\n\
\n\
flip_flop_0 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>sign0, Q=>cin0);\n\
\n\
flip_flop_1 : FDCE\n\
    port map (C=>clock, CE=>enable, CLR=>'0', D=>total, Q=>tout);\n\
\n\
end Behavioral;\n", arch, arch);


  return 0;
}
```

CSD4.h

```
int createCSD4ArchFile(int taps, char* arch);
```

```
int createCSD4EndAddFile(int taps, int a, int b, char* arch);

int writeEndFor5And7(char* arch, char* ret);
int writeEndFor9And11(char* arch, char* ret);
int writeEndFor13And15(char* arch, char* ret);
```

CSD_R.c

```
//Implementation of CSD-r algorithm presented in ECCTD01
#include "csd.h"


void CSD_r(csd_r_type *convcsd)
{
    unsigned int i,j,r,*N,gv;
    int minval;
    double right;

    r=convcsd->r;

    convcsd->A.length=r*(r-1)+1;
    convcsd->scale.length=convcsd->A.length+1;

    convcsd->A.array=(int *)calloc(convcsd->A.length,sizeof(int));
    if(!convcsd->A.array)err_disp(MEM_ERR,"k");
    N=(unsigned int *)calloc(convcsd->A.length,sizeof(unsigned int));
    if(!N)err_disp(MEM_ERR,"k");
    convcsd->scale.array=(double *)calloc(convcsd->scale.length,sizeof(double));
    if(!convcsd->scale.array)err_disp(MEM_ERR,"k");

    for(i=0;i<convcsd->A.length;i++)N[i]=1;
    N[(convcsd->A.length-1)/2]=0;
    convcsd->scale.array[0]=- (double) r;
    right= (double) r;

//Find the alphabet and conversion scale

    for(i=0;i<convcsd->A.length;i++)
    {
        convcsd->A.array[i]=r*convcsd->scale.array[i]
            -convcsd->scale.array[0]/pow((double)r,(double)N[i]);
        convcsd->scale.array[i+1]=(double)(convcsd->A.array[i])/(double)r
            +right/pow((double)r,(double)(N[i]+1));
        }

//normalize alphabet and the conversion scale
```

```
    minval=MAXINT;
    for(i=0;i<convcsd->A.length-1;i++)
     if(convcsd->A.array[i]!=0)
      for(j=i+1;j<convcsd->A.length;j++)
       if(convcsd->A.array[j]!=0)
     {
        gv=GCD(abs(convcsd->A.array[i]),abs(convcsd->A.array[j]));
        if(gv<minval)minval=gv;
        }
    for(i=0;i<convcsd->A.length;i++)
     convcsd->A.array[i] /= minval;

    for(i=0;i<convcsd->scale.length;i++)
     convcsd->scale.array[i] /= (double)minval;

    free(N);
    }
```

Err_Disp.c

```
#include "csd.h"

void err_disp(unsigned int msg,char comment[20])
{
    printf("\nERROR: ");
    if(msg==NO_FILE)printf("Cannot open %s", comment);
    else if(msg==FILE_ERR)printf("Invalid format in %s", comment);
    else if(msg==FILE_ERR_NODE)printf("Invalid node %s", comment);
    else if(msg==FILE_ERR_NS)printf("Invalid, unmatching or unspecified specs for %s", comment);
    else if(msg==FILE_ERR_UN)printf("%s is unspecified",comment);
    else if(msg==MEM_ERR)printf("Insufficient memory!");
    exit(EXIT_FAILURE);
    }
```

ExtIncl.h

```
// External Inclutions

#include <malloc.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
```

```
#include <math.h>
#include <ctype.h>
#include <string.h>
#include <io.h>
#include <sys/stat.h>
```

FL_CSD_R.c

```
//Float to CSD_r Conversion for a given wordlength
#include "csd.h"

i_arr float_to_csd(double in,csd_r_type convcsd,unsigned int b)
{
    i_arr out,out2;
    unsigned int i,j,convflag;

    convflag=0;

    if(in/fabs(convcsd.scale.array[0])>1.0||
       in/fabs(convcsd.scale.array[0])<-1.0)
    {
        convflag=1;
        in /= convcsd.r;
        b++;
        }

    out.array=(int *)calloc(b,sizeof(int));
    out.length=b;

    out.array[0]=0;
    for(i=1;i<b;i++)
    {
        for(j=0;j<convcsd.A.length;j++)
         if(in>convcsd.scale.array[j]&&in<convcsd.scale.array[j+1])
         {
            out.array[i]=convcsd.A.array[j];
            in -= (0.5*(convcsd.scale.array[j]+convcsd.scale.array[j+1]));
            break;
            }
        in *= convcsd.r;
         }


    if(convflag==1)
    {
        b--;
```

```
        out2.array=(int *)calloc(b,sizeof(int));
        out2.length=b;
        for(j=1;j<out.length;j++)
         out2.array[j-1]=out.array[j];
        free(out.array);
        return(out2);
        }
    else return(out);
    }
```

GCD.c

```
#include "csd.h"

unsigned int GCD(unsigned int a, unsigned int b)
{
    while(a!=b&&a!=1&&b!=1)
    {
        if(a>b)a-=b;
        else if(a<b) b-=a;
        }
        if(b==1)a=1;
    return(a);
    }
```

Main.c

```
#include "main.h"
#include "csd.h"
#include "extIncl.h"
#include "arch1.h"
#include "arch2.h"


#define oops(s) { perror((s)); exit(EXIT_FAILURE); }
#define MALLOC(s,t) if(((s) = malloc(t)) == NULL) { oops("error: malloc() "); }


/**
* The main()-function. From here everything starts.
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/
```

```
int main() {

  int i = 0, j = 0, k = 0;  // variables used in loops

  char s_taps[10];
  int taps = 0;                 // no of taps in file
  int groups = 0;               // no of groups with 4 taps in each group
  int groups2 = 0;              // copy of groups
  char tapValue[20];            // value of one tap
  double** tapGroups; //[GROUPS][4];    // store the tap values, 4 taps in each group

  char a_string[10]; //, b_string[10], c_string[10];
  char a_value[10]; //, b_value[10], c_value[10];

  int BINdigits=0, CSD2digits=0, CSD4digits=0;


  // take in a file containing X taps, i.e. desimal numbers below 1.
  FILE *tapInputFile;
  char tapInputFileName[100] = "./input.txt";

/*  printf("Type in input file name and location, e.g. \"./file.txt\": \n");
  scanf("%s", tapInputFileName);
  */
  // open file for reading
  tapInputFile = fopen(tapInputFileName, "r");

  if(tapInputFile==NULL) {
    printf("Error: can't open file %s.\n", tapInputFileName);
    return 1;
  }


    //find the digits used for binary, CSD2 and CSD4
  fscanf(tapInputFile, "%s", a_string);
  fscanf(tapInputFile, "%s", a_value);

/*  fscanf(tapInputFile, "%s", b_string);
  fscanf(tapInputFile, "%s", b_value);

  fscanf(tapInputFile, "%s", c_string);
  fscanf(tapInputFile, "%s", c_value);
*/
  if( !strcmp(a_string, "wordlength") )
    BINdigits = atoi(a_value);
  else
    oops("Error: first line in input-file must be \"wordlength X\", where X is the number of digits.");
```

```
/*  if( !strcmp(b_string, "CSD2") )
    CSD2digits = atoi(b_value);
  else
    oops("Error: second line in input-file must be \"CSD2 X\", where X is the number of csd2 digits.");

  if( !strcmp(c_string, "CSD4") )
    CSD4digits = atoi(c_value);
  else
    oops("Error: third line in input-file must be \"CSD4 X\", where X is the number of csd4 digits.");
*/

  // find number of taps in file
  fscanf(tapInputFile,"%s",s_taps);

  taps = atoi(s_taps);
  groups = roof(taps,4);


  // allocate memory space for tapGroups array. 4 taps in each group.
  MALLOC(tapGroups, sizeof(double*) * groups);
  for (i = 0; i < groups; i++) {
    MALLOC(tapGroups[i], sizeof(double) * 4);
  }

  groups2 = groups;                          // this algorithm does something strange, therefore this line
  for(i=0;i<groups;i++)
  {
    for(j=0;j<4;j++)
    {
        k = j;                               // this algorithm does something strange, therefore this line
        if(i*4 + j < taps)
        {
          fscanf(tapInputFile, "%s", tapValue);
          *(tapGroups[i] + j) = atof(tapValue);
        }
        else
          *(tapGroups[i] + j) = 0.00;
    }
  }

  fclose(tapInputFile);

  CSD2digits = BINdigits;
  CSD4digits = findCSD4digits( CSD2digits, groups, tapGroups );

    // start creating the architectures
  createArch1(taps, groups, tapGroups, BINdigits, CSD2digits, CSD4digits);
```

```
  createArch2(taps, groups, tapGroups, BINdigits, CSD2digits, CSD4digits);

    // free allocated memory
  for (i = 0; i < groups; i++)
  {
    free(tapGroups[i]);
  }
  free(tapGroups);

  return 0;
}


/**
* The function findCSDArray() puts values into the input-parameter "csdArray".
* Used for CSD2 and CSD4. In the case of binary numbers, findBINArray() is called.
*
* @param groups <code>int</code> number of groups with 4 taps in each group
* @param digits <code>int</code> number of digits in the CSD2 or CSD4 number
* @param tapGroups <code>double**</code> The value of the taps are stored here. 4 taps in each group
* @param base <code>int</code> is 2 for CSD2 and 4 for CSD4
* @param/output csdArray <code>int***</code> Array which to fill up with values
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int findCSDArray(int groups, int digits, double** tapGroups, int base, int*** csdArray)
{
  unsigned int j=0, k=0, i=0;   // variables used in loops
  int l=0;
  double sum = 0.0;     // the sum to find CSD numbers from

  char text[100] = "";          // holds temperary values for the resultfile

  int bin[16][4] = {{0,0,0,0}, {0,0,0,1}, {0,0,1,0}, {0,0,1,1}, {0,1,0,0}, {0,1,0,1}, {0,1,1,0}, {0,1,1,1}, {1,0,0,0}
                        // the 16 binary combinations of 4 digits

  // for debug reasons I keep this file.
  FILE *resultfile;
  char resultfilename[100];

  sprintf(resultfilename, "./TestOutputCSD%d.txt", base);

  // debug reasons
  if ((resultfile = fopen(resultfilename, "w")) == NULL)
          fprintf(stderr, "Cannot open %s\n", resultfilename);

    // ---- START ---- //
```

```
  for(l=0; l<groups; l++)
  {
    fprintf(resultfile,"\n\n\nGroup %d\t\tdes\t\tcsd%d\n", l, base);

    for(j=0; j<16; j++) {
      sum = 0.0;
      fprintf(resultfile,"\n");

      // write the binary combination of taps, and sum them
      for(k = 0; k < 4; k++) {
        fprintf(resultfile,"%d",bin[j][k]);
        if(bin[j][k] == 1)
          sum += *(tapGroups[l] + k);
                    // tapGroups[l]: the group which has been reached
                    // + k: the colomn we came to
      }
      fprintf(resultfile,"\t\t%1.9f\t",sum);

      strcpy(text, "");
      findCSD(sum, digits, base, csdArray[l][j], text);

      fprintf(resultfile, text);

    }// end for j
  }// end for l

  fclose(resultfile);

  return 0;
}



/**
* The function findCSD() finds the CSD value of one decimal number.
*
* @param dec <code>double</code> the decimal number from which to find the CSD number
* @param digits <code>int</code> number of digits, i.e. wordlength
* @param base <code>int</code> is 2 for CSD2 and 4 for CSD4
* @param/output csdValue <code>int*</code> Array where to put the CSD number
* @param/output text <code>char*</code> Some resulting text for the resultfile
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int findCSD(double dec, int digits, int base, int* csdValue, char* text)
{
```

```
char tempText[100] = "", temp[50] = "", c_coeff[50] = "";
int coeffFlag = 0;
unsigned int i=0, j=0;
unsigned int wordlength = digits;
csd_r_type convcsd;
double coeff,lval;
i_arr val;
long ival;

convcsd.r = base; // base

CSD_r(&convcsd);

coeff = dec;

  // The Blind Zone Problem is solved here
  // Search the coeffient to find out if it ends with 5. If that is the case,
  // the blind zone problem occurs. The solution is to add it with a small number.

  // coeffFlag is 0 in the beginning, becomes 1 if a coefficient ending with 5 is found,
  // and becomes 2 if a coefficient not ending with 5 is found.
coeffFlag = 0;
sprintf(c_coeff, "%1.25f", coeff);
while(coeffFlag == 0)
{
  j = strcspn(c_coeff, "5");

  if(j == strlen(c_coeff))    // no occurence of 5 found
    coeffFlag = 2;
  else                        // an occurence of 5 found, check weather it is in the end or not
  {
    coeffFlag = 1;
    for(i=j+1;i<strlen(c_coeff);i++)
    {
      if(c_coeff[i] != '0')
      {
        coeffFlag = 0;
        strcpy(c_coeff, &c_coeff[j+1]);
        break;
      }
    }
  }
}

  // add a fraction, i.e. 0.000000001 if number is ending by 5
if(coeffFlag == 1)
{
  strcpy(temp, "0.");
```

```
  for(i=0;i<15;i++)
    strcat(temp, "0");
  strcat(temp, "1");

  coeff += atof(temp);
}


  if(coeff<=1.0)
  {
    val=float_to_csd(coeff,convcsd,wordlength);
    ival=0;
    for(i=0;i<val.length;i++)
      ival += (long)((double)val.array[i]*pow((double)convcsd.r,(val.length-1-i)*(1.0)));

    ival *= convcsd.r;
    lval=(double)ival/pow((double)convcsd.r,(double)wordlength);
    ival /= convcsd.r;
    for(i=0;i<val.length;i++)
    {
      sprintf(tempText,"%d",val.array[i]);
      strcat(text, tempText);
      *(csdValue + i) = val.array[i];
      if(i==0)
      {
        strcat(text,".");
      }
    }

    free(val.array);
  } // end of if (coeff<=1)
  else
  {
  // if coeff > 1.0:  divide the value by its base and increase the wordlength by 1. After making
  //                  the quantization, shift the number to left only once. It will give you
  //                  the correct quantized number.

    coeff = coeff/base;
    val=float_to_csd(coeff,convcsd,wordlength+1);
    ival=0;
    for(i=0;i<val.length;i++)
      ival += (long)((double)val.array[i]*pow((double)convcsd.r,(val.length-1-i)*(1.0)));

    ival *= convcsd.r;
    lval=(double)ival/pow((double)convcsd.r,(double)wordlength);
    ival /= convcsd.r;
    for(i=0;i<val.length-1;i++)
    {
```

```
        sprintf(tempText,"%d",val.array[i+1]);
        strcat(text, tempText);
        *(csdValue + i) = val.array[i+1];
        if(i==0)
        {
          strcat(text,".");
        }
      }
      free(val.array);
    } //end of else (coeff<=1)

  free(convcsd.A.array);
  free(convcsd.scale.array);

  return 0;
}

/**
* The function findBINArray() puts values into the input-parameter "csdArray".
* Used for binary numbers. In the case of CSD2 or CSD4, findCSDArray() is called.
*
* @param groups <code>int</code> number of groups with 4 taps in each group
* @param digits <code>int</code> number of digits in the binary number
* @param tapGroups <code>double**</code> The value of the taps are stored here. 4 taps in each group
* @param csdArray <code>int***</code> Array which to fill up with values
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int findBINArray(int groups, int digits, double** tapGroups, int*** csdArray)
{
  unsigned int j=0, k=0, i=0;   // variables used in loops
  int l=0;
//  char error[100];
  double sum = 0.0;

  int bin[16][4] = {{0,0,0,0}, {0,0,0,1}, {0,0,1,0}, {0,0,1,1}, {0,1,0,0}, {0,1,0,1}, {0,1,1,0}, {0,1,1,1}, {1,0,0,0}
                       // the 16 binary combinations of 4 digits

  for(l=0; l<groups; l++) {
    for(j=0; j<16; j++) {

      sum = 0.0;
        // sum the binary combination
      for(k = 0; k < 4; k++) {
        if(bin[j][k] == 1)
          sum += *(tapGroups[l] + k);
                    // tapGroups[l]: the row which has been reached
```

```
                    // + k: the colomn we came to
      }

      if(sum<=0.99 && sum>=-0.99)
      {
        convertToBinary(sum, *(csdArray[l] +j), digits-1);

        for(i=digits-1;i>0;i--)
          *( csdArray[l][j] + i ) = *( csdArray[l][j] + i-1 );

      }
      else
      {
        // if coeff > 1.0:  divide the value by 2 and increase the wordlength by 1. After making
        //                  the quantization, shift the binary number to left only once. It will give you
        //                  the correct quantized number.

        convertToBinary(sum/2, *(csdArray[l] + j), digits);
      }

    } // end of for (16)
  } // end of for (groups)

  return 0;
}


/**
* Convert a desimal number to a binary one
*
* @param des <code>double</code> The desimal number to convert (less than 1)
* @param s <code>int*</code> The array where the binary number is stored
* @param digits <code>int</code> The number of digits in the binary number
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int convertToBinary(double des, int* s, int digits)
{
  int i=0, j=0;
  double newTry, diff1, diff2, abs_des;

    // suggestion 1 contains the smaller value
  double d_sug1 = 0.0;
  int* a_sug1;

    // suggestion 2 contains the bigger value
  double d_sug2 = 0.0;
```

```
int* a_sug2;

// allocate memory space for suggestion arrays
MALLOC(a_sug1, sizeof(int) * digits);
MALLOC(a_sug2, sizeof(int) * digits);

a_sug1[0] = 0;
a_sug2[0] = 0;

  // a positive number is needed.
if(des<0)
  abs_des = des - des - des;
else
  abs_des = des;

for(i=0;i<digits-1;i++)
{
  newTry = d_sug1 + (1.0/ldexp(2,i)); // (1.0/(2^i));

  if(newTry > abs_des)
  {
      // store the too big value in sug2
    d_sug2 = newTry;

      // transfer values of a_sug1 to a_sug2
    for(j=1;j<i+1;j++)
      a_sug2[j] = a_sug1[j];

    a_sug2[i+1] = 1;
    a_sug1[i+1] = 0;
  }
  else
  {
    d_sug1 = newTry;
    a_sug1[i+1] = 1;
    a_sug2[i+1] = 0;
  }
}

diff1 = abs_des - d_sug1;
diff2 = d_sug2 - abs_des;

if(diff1 < diff2)
{
  if(des<0)
  {   // find 2's complement
    find2sCompl(a_sug1, digits);
  }
```

```
    for(i=0;i<digits;i++)
      *(s + i) = a_sug1[i];
  }
  else
  {
    if(des<0)
    {   // find 2's complement
      find2sCompl(a_sug2, digits);
    }
    for(i=0;i<digits;i++)
      *(s + i) = a_sug2[i];
  }


    // free allocated memory
  free(a_sug1);
  free(a_sug2);

  return 0;
}


/**
* Modifies the pointer to array given in to contain 2's complement number of the original number.
*
* @param array <code>int*</code> The array containing the binary number to transfer by 2's complement to
*                                negative number. This array will be modified to contain the resulting number.
*
* @return <code>int</code> 0 if everything ok, otherwise 1 and errormessage.
*/

int find2sCompl(int* array, int digits)
{
  int i=0, j=0;
  int* i_num;

  MALLOC(i_num, sizeof(int) * digits);

    // put the digits into an array which I can modify
  for(i=0;i<digits;i++)
  {
    i_num[i] = array[i];
  }

    // subtract 1 from the number
  for(i=digits-1;i>=0;i--)
  {
    if(i_num[i] == 1)
```

```
    {
      i_num[i] = 0;

        // change all previous digits to 1.
      for(j=digits-1;j>i;j--)
      {
        i_num[j] = 1;
      }
      break;
    }
  }

    //turn all the digits
  for(i=0;i<digits;i++)
  {
    if(i_num[i] == 1)
      i_num[i] = 0;
    else
      i_num[i] = 1;
  }

    //put the 2's complement number back into input parameter.
  for(i=0;i<digits;i++)
  {
    array[i] = i_num[i];
  }

  free(i_num);

  return 0;
}



/**
* roof() is a small function returning the upper rounded value of a divided by b.
*
* @param a <code>int</code> devisor
* @param b <code>int</code> dividend
*
* @return <code>int</code> The upper rounded value of a/b.
*/

int roof(int a, int b)
{
    double div = 0.0;
    int result = 0;
```

```
    div = (double)a/(double)b;

    result = div;

    if((div - result) != 0.0)
        result++;

    return result;
}


/**
* findCSD4digits() is a function returning the number of digits CSD4 will be implemented with,
*     based on the number of digits CSD2 numbers are implemented with.
*
* @param CSD2digits <code>int</code> The number of CSD2 digits used as a base for this algorith.
* @param groups <code>int</code> number of groups with 4 coefficients in each group
* @param tapGroups <code>double**</code> the values of the taps in decimal format
*
* @return <code>int</code> The number of digits to use for CSD4.
*/

int findCSD4digits( int CSD2digits, int groups, double** tapGroups )
{

  double errorCSD2, errorCSD4;
  double oldError = 99.9, newError = 99.9;
  double avrErrorCSD2 = 0.0, avrErrorCSD4 = 0.0;
  double sumErrorCSD2 = 0.0, sumErrorCSD4 = 0.0;
  double decimalCSD2, decimalCSD4;
  int i=0, j=0, k=0, m=0;
  int CSD4digits;
  int oldDigits = CSD2digits, newDigits = CSD2digits;
  int* csd2Value;
  int* csd4Value;

  char dummyText[100] = "";

  // allocate memory space for csdValues
  MALLOC(csd2Value, sizeof(int) * CSD2digits);
  MALLOC(csd4Value, sizeof(int) * CSD2digits);


    // find the average error for CSD2
  m=0;
  for(j=0;j<groups;j++)
  {
    for(k=0;k<4;k++)
```

```
  {
    strcpy(dummyText,"");
    findCSD(tapGroups[j][k], CSD2digits, 2, csd2Value, dummyText);

    decimalCSD2 = convertToDecimal(csd2Value, CSD2digits, 2);

    errorCSD2 = absolute_value(decimalCSD2 - tapGroups[j][k]);
    sumErrorCSD2 += errorCSD2;

    m++;
  }
}

if(m!=0)
  avrErrorCSD2 = sumErrorCSD2/m;

i=0;
while( avrErrorCSD2 > avrErrorCSD4 )
{
  // find the average error for CSD2
  m=0;
  avrErrorCSD4 = 0.0;
  sumErrorCSD4 = 0.0;
  for(j=0;j<groups;j++)
  {
    for(k=0;k<4;k++)
    {
      strcpy(dummyText,"");
      findCSD(tapGroups[j][k], CSD2digits-i, 4, csd4Value, dummyText);
      decimalCSD4 = convertToDecimal(csd4Value, CSD2digits-i, 4);
      errorCSD4 = absolute_value(decimalCSD4 - tapGroups[j][k]);

      sumErrorCSD4 += errorCSD4;
      m++;
    }
  }

  if(m!=0)
    avrErrorCSD4 = sumErrorCSD4/m;

  oldDigits = newDigits;
  newDigits = CSD2digits - i;

  oldError = newError;
  newError = avrErrorCSD4;
  i++;
}
```

```
  if( absolute_value(oldError-errorCSD2) < absolute_value(newError-errorCSD2) )
    CSD4digits = oldDigits;
  else
    CSD4digits = newDigits;


    // free allocated memory
  free(csd2Value);
  free(csd4Value);

  return CSD4digits;
}



/**
* convertToDecimal converts a CSD2 or CSD4 number to desimal form
*
* @param csd <code> int* </code> the CSD number in a vector format
* @param digits <code> int </code> the number of digits the number consists of
* @param base <code> int </code> the base of the CSD number (2 or 4)
*
* @return <code> double </code> the desimal number of the CSD number.
*/

double convertToDecimal(int* csdValue, int digits, int base)
{
  double decimal = 0.0;
  int i=0;

  for(i=0;i<digits;i++)
  {
    decimal += ( double )csdValue[i] / (pow( base,i )) ;
  }

  return decimal;
}



/**
* absolute_value() returns the absolute value of the input x.
*/

double absolute_value(double x)
{
    if(x >= 0.0) {
        return x ;
    } else {
```

```
        return -x ;
    }
}
```

Main.h

```
int main();

int findCSD(double dec, int digits, int base, int* csdValue, char* text);
int findCSDArray(int groups, int digits, double** tapGroups, int csdBase, int*** csdArray);
int findBINArray(int groups, int digits, double** tapGroups, int*** csdArray);

int convertToBinary(double des, int* s, int digits);
int find2sCompl(int* array, int digits);

int roof(int a, int b);

int findCSD4digits( int CSD2digits, int taps, double** tapGroups );
double convertToDecimal(int* csd, int digits, int base);
double absolute_value(double x);
```

APPENDIX E: USER GUIDE

The user guide is included so that it will be easy for future students to continue this work.

The requirements for finding the power consumption of a filter is described in a step-by-step manner.

1. The file containing the tap values of the filter must have the format shown in table E.1. Table E.2 gives an example of a 4-tap filter.
2. The file of table E.1 must be named "input.txt" and put in the same folder as the project is in. Currently the project can only be run from a Microsoft Visual Studio environment.
3. Run the project from Microsoft Visual Studio (for instance by Ctrl+F5).
4. Two new directories will be created in the folder of the project: ARCH1 and ARCH2. Each of these contains three subfolders: BIN, CSD2 and CSD4, containing all the files needed to simulate the design in Xilinx Project Navigator, including a test-file called "InitShift_Test.vhd".
5. Open Xilinx Project Navigator and make a new project.
6. Add all files from one architecture to the project, right-clicking the design and choose "Add Source". All .vhdl-files are Design files and all .vhd-files are Test Bench files.

Table E.1. Format of file containing tap values

wordlength <wordlength>
<number of taps>
<tapvalue 1>
<tapvalue 2>
...
<tapvalue n>

Table E.2. Example of a file containing tap values

wordlength 12
4
-0.067371764
0.094195111
0.40580489
0.56737176

7. Set these properties:
   - While highlighting "init_shiftregs_behavioral":
     - "Synthesize" properties: "Optimization goal": Area.
   - While highlighting "init_shift_test_behavior" and keeping this file open, scroll to the bottom of it:
     - "Simulate Post-Place & Route VHDL Model" properties: "Generate VCD File": Enable, and set the "Simulation Run Time" as shown in figure E.1.
8. Highlight "init_shiftregs_behavioral" again, and under "User Constraints" run "Create Area Constraints".
9. Floorplanning: Drag the design from "Logic" to the "Architecture View" as shown in figure E.2. Put the design closest possible to the global clock situated between the red and yellow stripes in the bottom of figure E.2. Save the floorplan.
10. Highlight "init_shift_test_behavior" and run "Simulate Post-Place & Route VHDL Model".
11. Wait for the simulation to end, than close ModelSim.
12. Highlight "init_shiftregs_behavioral" and run "View XPower Report" found under "Implement Design" ⇒ "Place & Route" ⇒ "Generate Power Data".

Now do this for the binary, the CSD2 and the CSD4 implementation of an architecture and compare them. Check the power report to se if there are any warnings about inaccurate results. In that case the results cannot be considered as valid.

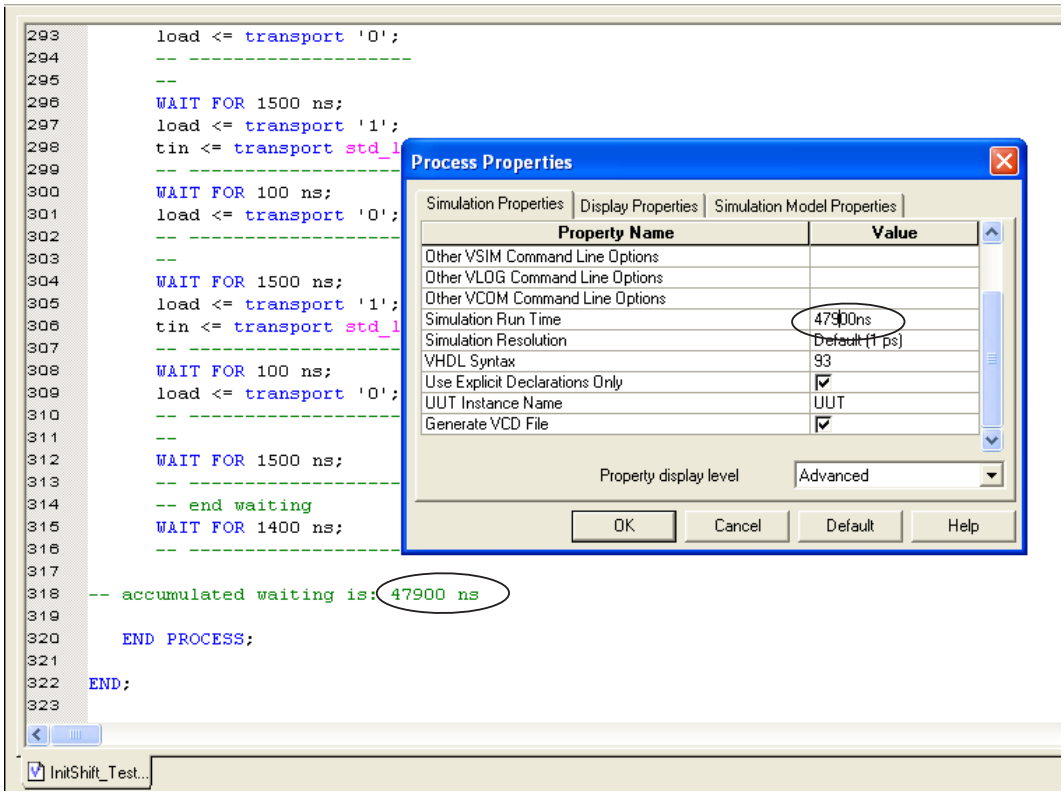

Figure E.1. Property settings for "Simulate Post-Place & Route VHDL Model"

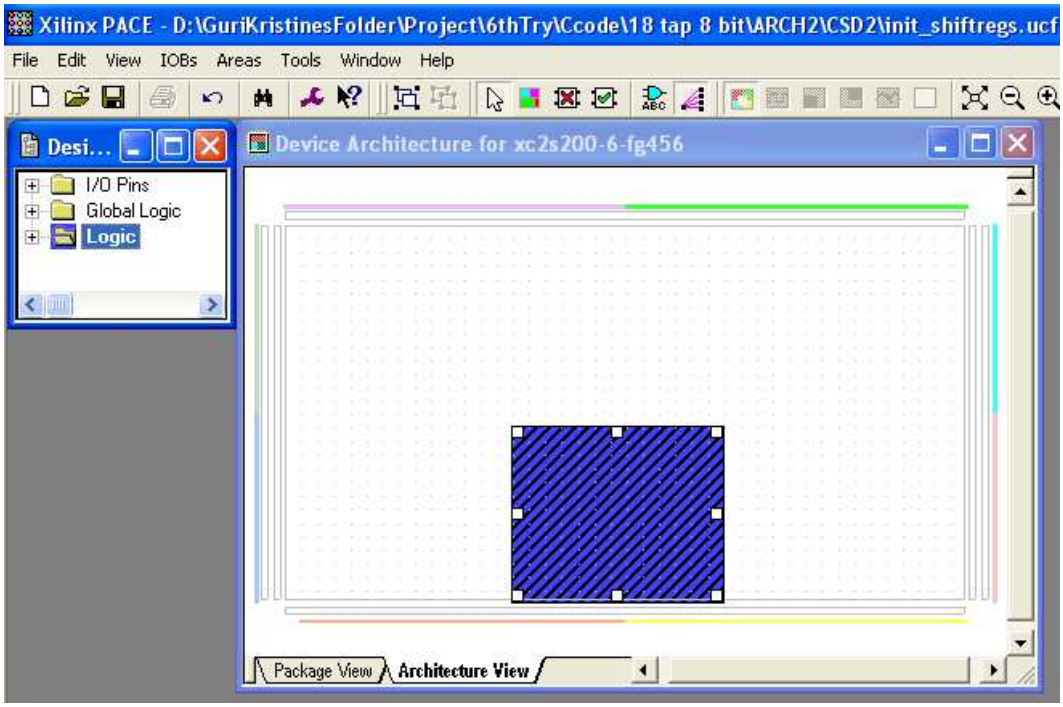

Figure E.2. Floorplanning constraints using Xilinx's PACE