The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

IMPLEMENTATION OF PIPELINING MECHANISM IN RE-PROGRAMMABLE LOGIC STRUCTURES WITH VHDL LANGUAGE USAGE

Maciej MICHALCZAK, Zbigniew SKOWROŃSKI

Computer Engineering and Electronics Institute, Technical University of Zielona Góra,
ul. Podgórna 50, 65-246 Zielona Góra, POLAND,
M.Michalczak@willow.iie.pz.zgora.pl, Z.Skowronski@iie.pz.zgora.pl

Abstract. *Using re-programmable logic components along with HDL languages encompasses wider and wider areas of practical applications, becoming a standard of complex digital system design. One of the basic tasks, which are to be carried out in the process of design, is obtaining the highest efficiency of the solution under design. Thereby designers are still looking for methods making it possible to speed up design processing time. Pipelining mechanism is one of these methods. It helps to speed up some dedicated operations. This paper contains an example of practical application of multiplier system of floating - point numbers described in VHDL language (model in the shape of a net - structure), while using operands described by the format compatible with the IEEE 754 standard of writing the floating - point numbers. In the early stage of design, a given unit described by high level language, is divided into some independent parts, which are synchronized with each other via intermediate registers and synchronization signal (pipelining mechanism). The main goal of this paper is to present practical aspects of designing an advanced and complex digital system while using pipelining mechanism in re-programmable logic structures with using VHDL language.*

Key Words. *Pipelining Mechanism, Re-programmable Logic Devices, FPGA, PLD, ASIC, Hardware Description Language, VHDL*

1. INTRODUCTION

Digital units can be found in almost every domain of the surrounding world. Modern inventions like cellular telephony or digital television use digital signal processing for communications and couldn't exist without such units. So it is very important to ensure effective and unfailing design methods. In the last years we can notice a rapid development of the design methods. For example: hardware description languages (HDL), logical synthesis or design implementation methods in, still improving, re-programmable devices such as FPGA, CPLD or ASIC. Now, the design path with HDL and with a high level logical synthesis is an

industry standard. Advanced re-programmable units are larger and larger (in respect of gate number) so it enables the implementation of complicated digital units like SoC (System on chip), processors or specialized controllers. One of the important aspects of the design is its speed.

The main, functional part of a computer and other electronic devices are arithmetic blocks. The operation of multiplication is one of the basic arithmetic operations possible to carry out by digital units. Practically, it is a partial product accumulation of operator A by the value of the next digit of the operator Xi with its weight [4]:

$$P_{i+1} = P_i + AX_i\beta^i = A\sum_{s=-m}^{i+1} X_s\beta^s, \qquad i = -m,...,0,1,...,k-1, \qquad P_{-m} = 0 \tag{1}$$

what in normalized result $p_i = \beta^{-i}P_i$ is:

$$p_{i+1} = \beta^{-1}(p_i + X_i A), \qquad i = -m,...,0,1,...,k-1, \qquad p_{-m} = 0 \tag{2}$$

The scaled end result $P_k = \beta^k p_k$ is equal to the value of multiplication A*X.

In binary system, partial product may be the operator A or the value 0, so the multiplication algorithm can be realized by the add-and-shift method (Figure 1).

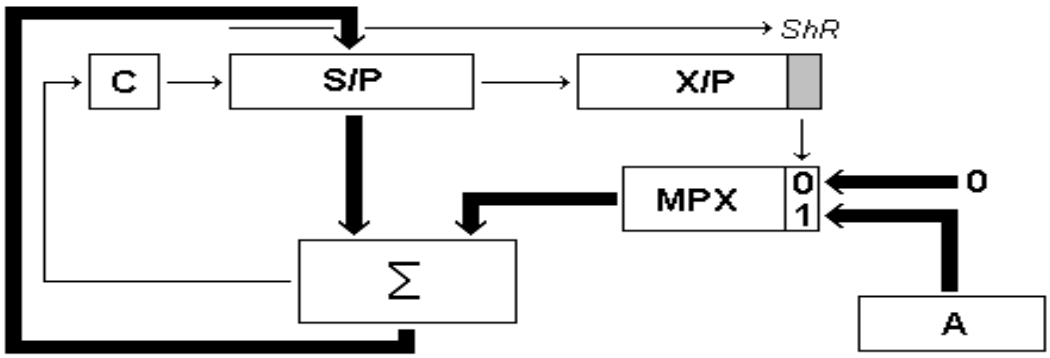

Figure 1. Algorithm of accumulation multiplication in the natural binary system

The main goal of this work is presenting a practical method of speeding up the operation of multiplication. The paper shows the example solution of a multiplication module. Operators used in the module are consistent with IEEE 754 floating-point number notation standard. The module includes a constant-number-multiplication module, which will be sped up.

2. RULES OF USE OF PIPELINING MECHANISM

Pipelining is a technique increasing the function speed of a design [6]. While pipeline implementation, the design is divided into individual pipelining levels, creating a line of matrix of elementary processing modules. Each matrix level may be modeled individually and independently. It allows choosing the optimal method of realization. In the case of sequential operation unit, when each stage is implemented in a different functional block of a design, other blocks remain idle. When individual parts of a unit are separated by latches (thus locking partial products), it becomes possible to process simultaneously each data stream. That is what pipelining is. In such a solution the results can be produced in each cycle of the synchronization signal, triggering given levels, not only once per several cycles. The frequency of producing results is limited by the delay time of the longest processing

matrix level. Compared to combinational units, it is often possible to create the situation where the delay time of the whole unit (propagation time) is significantly larger than the period between the synchronization signals in the unit with implemented pipelining. Assuming that results of a unit with pipelining are produced with each cycle of the synchronization signal [4], it becomes possible to increase significantly the frequency of results produced, in comparison with combinational units.

3. THE DESCRIPTION OF FUNCTIONAL ASSUMPTIONS

The presented unit is an attempt of implementation of floating-point number multiplier, based on IEEE 754 binary standard numbering notation. Interface has been restricted to support only one mode of that standard. It is the simple mode with single precision, called *real*. A number written in that notation consists of three components (Figure 2):

- sign: 1 bit;
- exponent: 8 bits;
- mantissa: 23 bits + one hidden bit used for normalization.

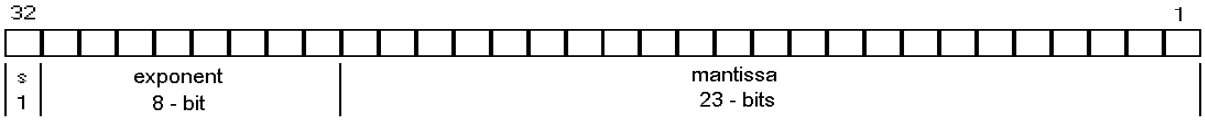

Figure 2. The representation of the floating-point number in IEEE 754 notation

The implementation was based on the floating-point–number-multiplication algorithm [7]. The first step is the operation of mantissa multiplication (23 bits + 1 hidden bit. For the next operation it is necessary to round the result to 24-bit number using only the top 24 bits of the 48-bit result. After that this number should be normalized if overflow occurs on the hidden bit (the most important bit) by shifting the mantissa to the right along with the exponent value increment. That is the outcome value of the mantissa. The exponent in IEEE 754 standard is represented in biased-127 code. The next step is the addition of exponents, which prior to that, are converted to the value in 2's complementary code through adding the value of 127. After the addition, the result should be corrected by subtracting the value of 127. The overflow occurs when the value of the addition result is bigger than 127. In that case the exponent is set to the value of 128, and the mantissa is set to the 0 value. Underflow occurs when the value of the exponent addition is smaller than minus 126. In that case the exponent is set to the value of minus 127. The last step is to determine a sign value through XOR operation on the signs of operands.

The suggested solution here is a structure of three modules [10]:

- adder of exponents;
- multiplier of mantissas;
- and the main module, realizing all the necessary corrections and setting the sign of the result value.

The main part of the entire unit is the multiplier module, which is used for realizing the operation of multiplication of two given operands. Different conceptions of realization of that module have been tested. After that it has been decided to present two solutions:

- behavioral description, with using the multiplication function from standard library;
- multiplication with using multiplying matrix, with implemented pipelining mechanism.

4. PRESENTED DESIGN IMPLEMENTATION

The design has been realized in the programming environment of Aldec's Active-HDL (compilation & simulation code), and by using the program of Foundation 2.1i from Xilinx [2] (logic synthesis & hardware implementation). The whole design has been created in VHDL language [3, 8, 9]. Functionality has been checked during functional simulations and during timing simulations using the pseudo-hardware time delays of the target device.

The realization has been divided into two modules. The realized adder module with the result correction in overflow case on the hidden bit position during mantissas multiplication and with the estimation of the outcome overflow of the exponent:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
--
entity a_der is
port (
   INC_E      : in STD_LOGIC;
   EXP_A      : in STD_LOGIC_VECTOR(7 downto 0);
   EXP_B      : in STD_LOGIC_VECTOR(7 downto 0);
   OV         : out STD_LOGIC;
   EXP_Q      : out STD_LOGIC_VECTOR(7 downto 0)
     );
end a_der;
--
architecture ADDER_ARCH of adder is
-- -127 value for conversion
constant CONST : STD_LOGIC_VECTOR(7 downto 0) := "10000001";
-- temporary for overflow
signal TEMP_OV : STD_LOGIC;
-- temporary for underflow
signal TEMP_UN : STD_LOGIC;
-- result in 2's complement code
signal TEMP : STD_LOGIC_VECTOR(7 downto 0);
-- biased-127 result representation
signal TEMP_Q : STD_LOGIC_VECTOR(7 downto 0);
--
begin
--
TEMP <= EXP_A + EXP_B + INC_E;                      -- addition
TEMP_Q <= TEMP + CONST;                             -- conversion from
                                                    -- 2's to biased-127
TEMP_OV <= TEMP(7) and E_A(7) and E_B(7);           -- set if overflow occurs
TEMP_UN <= not (TEMP(7) or E_A(7) or E_B(7));       -- set if underflow occurs

E_Q <= (others => '1') when TEMP_OV='1'
       else (others => '0') when TEMP_UN='1'
       else TEMP_Q;
OV <= TEMP_OV;

end ADDER_ARCH;
```

The main part of the design is a constant-number multiplication module. It ensures the realization of the multiplication operation of two 23-bit operands. In the overflow case multiplication results are set to 0 value.

The main goal of that paper is to present the possibility of realization of fast multiplication module. Multiplication modules are usually the source of the longest delays in projected units. Two different solutions have been worked out:

- functional description (operation based on standard library);
- multiplication matrix with pipelining (description in the form of a structure) [5].

4.1. Functional description

The description of the module has been based on the following construction:

```
RESULTS <= MANTISSA_A * MANTISSA_B;
```

The '*' operator is defined in IEEE standard library. The style of realization of this module is the same as, the formerly presented, adder module.

4.2. Matrix multiplier with pipelining

The whole operation has been divided into stages. The matrix of elementary adding modules has been defined. Each of them is a full 1-bit adder (FA) [10]. The operation of multiplication is carried out using 24-bit operands, so it is necessary to use the matrix of 23x23 adders + one last level of 23 adders for counting the end results of 24 top bits of the result. Each matrix level is composed of 23 adders counting independently introducing the delay equaled to the delay of one elementary module. So in that solution, the whole delay consists of all stages delays plus the addition of 23 adders of the last stage. The unit has been modified by introducing some more memorizing elements called flip-flops, to separate each matrix stage [10]. This pipelining allows for synchronized data flow between each matrix levels. This solution makes it possible that independent data is processed at different stages simultaneously. Data exchange between stages is realized during the active edge of the synchronizing signal. At the output of the unit we obtain results with every synchronizing signal cycle. Also, the input data should be fed into the unit while observing the synchronizing cycle. Synchronization signal period cycle must be bigger than the largest delay of unit stages of the given matrix.

5. COMPARISON OF SPEED

The final stage of realizing each of presented solutions was the unit implementation while using the FPGA re-programmable structure. During work, device from Xilinx's families Virtex have been used. Table 1 presents the results of the implementation process and hardware time delay.

Table 1. Comparison of results for the implementation of example multiplying units

VIRTEX – V100PQ240		
	Behavioral	***Matrix (with pipelining)***
Maximal combinational delay [ns]	**34.419**	39.316
Maximal path delay [ns]	7.095	7.424
Maximal frequency [MHz]	-	**103.681**
Minimal clock period time [ns]	-	**9.45**
SLICE	320	1200

Combinational delay of the behavioral unit is 34 ns for Virtex. It is the time required for producing valid results by a module at the output on the basis of the input data.

For synchronized matrix unit with implemented pipelining mechanism the most important parameter is the synchronization signal frequency. In the Virtex case it is equaled to 103.681 MHz, so the cycle period time for synchronization signal is 9.45 ns.

Synchronization signal period time determines the time between producing consecutive results. The main difference between presented solutions is the fact that for a behavioral unit the whole operation time is 34.419 ns. Matrix unit takes 39.316 ns, but this unit produced results after 9.45 ns. When the input data is fed between that times, the time of produced results is much better than in a behavioral unit.

6. SUMMARY

This paper presents a method that allows decreasing the period of time required for producing consecutive multiplication results. A practical example of floating-point number multiplier has been presented. It consists of the constant-number multiplier module for which the attempt of pipelining mechanism implementation has been presented. Two solutions have been worked out and implemented. The obtained results have been compared paying special attention to the processing speed (time between generated results) (Table 1).

Combinational delay of behavioral unit is bigger than the minimal cycle period time of the synchronization signal in the matrix with a pipelining unit. Thanks to that it is possible to achieve the situation where the pipelining unit allows to produce results every 9.45 ns in comparison to 34.419 for a behavioral unit. That is a significant difference, in the aspect of obtaining consecutive unit results. The disadvantage of that solution is its size and more complicated description. A designer must ensure suitable synchronization of partial results and correct data flow between the unit stages.

The application of the presented proposal concerns the sequential processing structures, where the most important thing is data processing time and size of system is not the main concern.

REFERENCES

[1] http://www.aldec.com

[2] http://www.xilinx.com

[3] P. J. Ashenden: *The Designer's Guide to VHDL*, Morgan Kaufmann Publisher, Inc., 1996

[4] J. Biernat: *Architecture of Computers*, Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław, 1999 (in Polish)

[5] J. Biernat: *Arithmetics of Computers*, Wydawnictwo Naukowe PWN, Warszawa, 1996 (in Polish)

[6] G. De Micheli: *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, Inc., Stanford, 1994

[7] M. Roth: Assembly Language, *http://www.cs.uaf.edu/~cs301/notes/Chapter6/node4.html*

[8] A. Rushton: *VHDL for Logic Synthesis - Second Edition*, John Wiley & Sons, 1998

[9] S. Sjoholm, L. Lindh: *VHDL for Designers*, Prentice Hall, 1997

[10] M. Michalczak, Z. Skowroński: „Practical Principles of Design of Digital Devices with Pipeline in the Programmable Logic Structures Using VHDL", *Proceedings of IV National Scientific Conference, RUC'2001,* Szczecin, May 7-8, 2001, ISBN 83-87362-35-2, pp.93-101 (in Polish)