A codesign case study: implementing arithmetic functions in FPGAs

Klotchkov, I. V.; Pedersen, Steen

Published in:
Proceedings of the IEEE Symposium and Workshop on Engineering of Computer-Based Systems

Link to article, DOI:
10.1109/ECBS.1996.494565

Publication date:
1996

Document Version
Publisher's PDF, also known as Version of record

A Codesign Case Study: Implementing Arithmetic Functions in FPGA's

I.V. Klotchkov and S. Pedersen*
Department of Information Technology†
Technical University of Denmark
DK-2800 Lyngby, Denmark

Abstract

Different way of implementing and designing arithmetic functions for 16/32 bit integers in FPGA technology are studied. This also includes a comparison of four different design methods.

The results are used to increase the overall system performance in a dedicated 3D image analysis prototype system by moving a vector length calculation from software to hardware.

The conclusion is that by adding one relatively simple board containing two FPGA's in the prototype setup, the total computing time is reduced by 30 %. The total amount of image data, in this case 300 Mbyte which has to be transmitted via network, is reduced by a factor of two, and the required network bandwidth is reduced similarly.

1 Introduction

This paper describes and analyses different ways of designing and implementing the arithmetic functions $X^2 + Y^2$ and $\sqrt{Z}$, where X and Y are 16 bit integers and Z is a 32 bit integer, in FPGA[1] technology.

In a prototype setup, there is 2.5 μs available for calculating the length of one vector (X,Y) i.e. $\sqrt{X^2 + Y^2}$, and a number of implementation alternatives, which can meet this timing constraint, are studied. This also includes a novel implementation that is highly optimised for a speed efficient realization in the chosen FPGA technology. Some of the implementations are demonstrated using an experimental setup with one Altera epf81188 chip, and speed measurements are compared to simulated values.

It turns out, that the traditional trade-offs in ASIC circuit design regarding optimisation for area or speed may also be applied in utilising the resources of the FGPA. The primary difference is the hard limit, i.e. the Altera epf81188 chip has 1008 basic logic building blocks, LE's, and a fixed amount of routing capabilities.

For example, the specially developed fast implementation of a 16 bit squaring unit, which utilises the considered FPGA technology optimally, requires 30 % of the available LE's in one epf81188 chip. But, at the same time it occupies all the routing resources of this chip, and hereby leaving more than two third of the logic cells unused. On the other hand, the use of a high level input description, (in fact an arithmetic equation in VHDL), and a commercial logic synthesis tool will lead to an implementation with much better fitting ability. Three copies of this considerable less efficient synthesised implementation of the 16 bit squaring unit will fit in one chip, and hereby provide more total computational power by using 99 % of the LE's.

The work presented here is a part of a larger codesign case study[2], which is performed at the Department of Information Technology in collaboration with the Department of Mathematical Modelling, both at the Technical University of Denmark. This case study deals with implementing a combined hardware/software prototype system for an advanced image analysis method in Optical Flow analysis called *In Betweening*, [1, 9]. In this paper the codesign aspects are discussed further in section 4.

The vector length calculation, which is the topic of this paper, is initially situated right on the hardware/software boundary, as the first calculation in software. The aim is to investigate the price/performance relation for the overall hardware/software system by implementing this vector length calculation in hardware. Besides the detailed hardware implementation considerations, this also includes hardware/software communication aspects.

*E-mail: ik@it.dtu.dk and sp@it.dtu.dk
†The name was previously: Department of Computer Science

[1] In this context the term FPGA, Field Programmable Gate Array, primarily refers to the FLEX8000 series of devices from Altera [3].

[2] WWW: http://www.it.dtu.dk/~case3d

0-8186-7355-9/96 $05.00 © 1996 IEEE

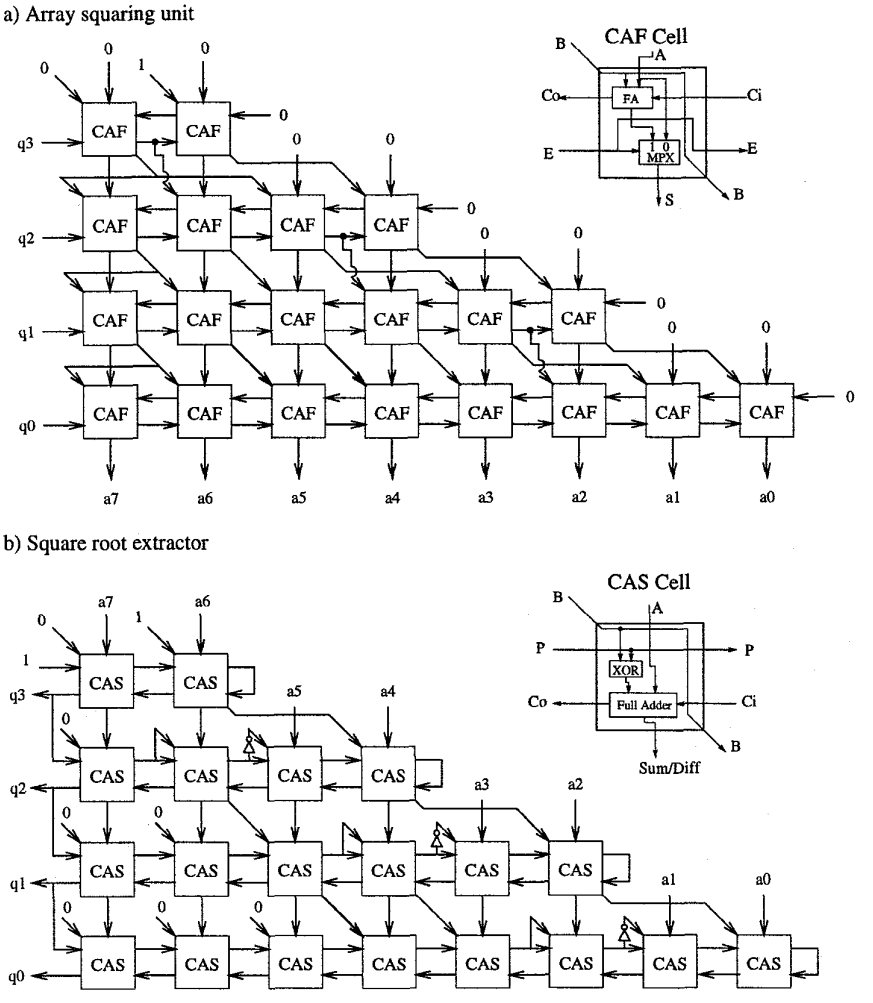

Figure 1: Array block diagrams. a) Squaring unit. b) Square root extractor.

2 The Arithmetic Functions

Different ways of implementing a squaring function for 16 bit integers and a square root function for a 32 bit integer are presented in this section, and a number of designs are compared in the following section.

2.1 The Array Squaring Unit

A matrix based squaring unit, which is described in [6], is chosen first. This array, Figure 1a, requires $N = n^2 + n$ cells, where n is the number of bits in the input. It accepts $Q = q_{n-1}..q_1q_0$ as input and calculates $A = Q^2 = a_{2n-1}..a_1a_0$. This structure is very similar to an array multiplier [10], where the two inputs are connected together.

The basic building block in the squaring unit, (CAF), which is a combination of a full adder and a multiplexer, is also shown in Figure 1a. In this structure each bit of the input operand is broadcast in parallel to all cells in a row. The carry propagation starts at the imaginary upper right corner and proceeds towards the lower left corner, and a_7 is the last computed output bit.

2.2 The Sliced Squaring Unit

The array squaring unit does however not lead to an efficient FPGA-based solution, as it will be seen in section 3. Therefore, we consider an alternative based on the following equations for an 8-bit integer $A = a_7..a_0$:

$$A^2 = (A_1 + 16A_2)^2 = A_1^2 + 32(A_1A_2) + 256A_2^2 \quad (1)$$

where $A_1 = a_3..a_0$ and $A_2 = a_7..a_4$ are high and low nibbles of the original 8-bit integer A. To implement

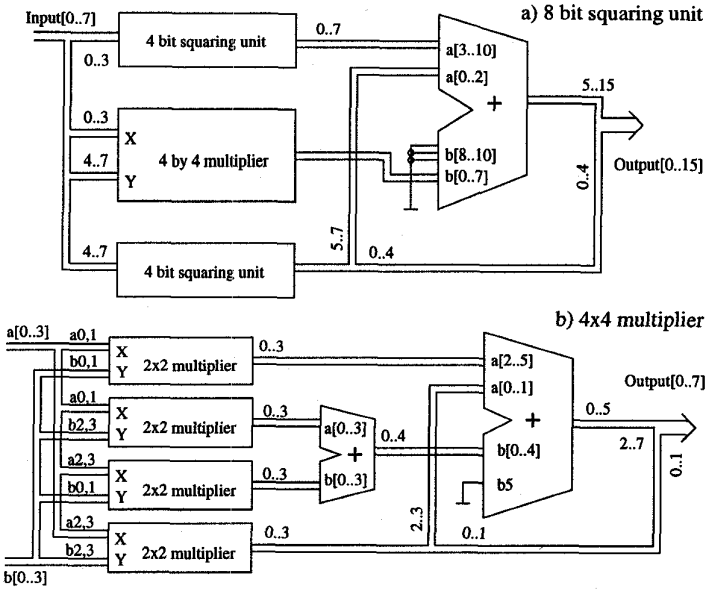

Figure 2: a) An 8-bit sliced squaring unit. b) 4x4 bit sliced multiplier.

(1), we need 4-bit squaring, 4x4-bit multiplication and addition. Due to the FPGA architecture, 4-bit squaring is realized in one level of logic as well as 2x2-bit multiplication. A 4x4-bit multiplication could then be split according to the following equation:

$$\begin{aligned} A_1A_2 &= (X_1 + 4X_2)(Y_1 + 4Y_2) \qquad (2) \\ &= X_1Y_1 + 4(X_1Y_2 + X_2Y_1) + 16X_2Y_2 \end{aligned}$$

where we assume $A_1 = X_1 + 4X_2$, $A_2 = Y_1 + 4Y_2$, and X_1, X_2, Y_1, and Y_2 are 2-bit integers.

Figure 2 shows a block diagram of the sliced squaring unit with 8-bit input.

2.3 The Square Root Extractor

A non-restoring square root extractor array, see [6], is shown in Figure 1b. It accepts $A = a_{n-1}..a_1a_0$ as input and calculates $Q = \sqrt{A} = q_{(n/2)-1}..q_1q_0$. This block diagram is corrected compared with the original figure in [6], by adding the triangle with six cells situated in the lower left corner of the matrix. The main building block is a "Controlled Adder-Subtracter" cell, (CAS), as shown in the figure, and $N = (n/2)^2 + (n/2)$ cells are required.

The data flow in this calculation is sequential. The input bits are applied simultaneously, one bit to each column, and the computation starts from the rightmost cell in the upper row. When the first output bit, q_3, is produced, this value is also broadcast to all the cells in the next row, and the computation proceeds from the rightmost cell towards the left. This procedure is repeated until the last output bit, q_0, is calculated.

3 FPGA Implementation Results

The two arithmetic functions described above are implemented in FPGA technology using four different hardware design strategies, which is described briefly here and in more detail together with the obtained results in the following section.

Synthesised	The high-level VHDL description is synthesised using Synopsys [7] and transfered to the Altera software [4], which does the placement and routing for the FPGA.
Altera-optimised	The squaring array is compiled and optimised by the Altera software [2].
Altera-direct	The squaring array is compiled without optimisation.
Sliced design	Manually sliced design according to Figure 2.

3.1 Results from the Squaring Unit

The squaring unit is implemented using all of the above mentioned methods, and a comparison is shown in Table 1.

The Synthesised implementation of the squaring unit without splitting was carried out from a VHDL expressions $\boldsymbol{X} := \boldsymbol{Y} * \boldsymbol{Y}$, where $\boldsymbol{X}$ and $\boldsymbol{Y}$ are vari-

Input width	Unit	Design methodology											
		Synthesised						Altera-optimised		Altera-direct		Sliced design	
		no splitting		split in two		split in four							
4	Area	7	(1.0)	-	-	-	-	7	(1.0)	28	(4.0)	7	(1.0)
	Delay	21	(1.0)	-	-	-	-	21	(1.0)	68	(3.2)	21	(1.0)
8	Area	111	(2.1)	64	(1.2)	75	(1.4)	79	(1.5)	120	(2.3)	52	(1.0)
	Delay	105	(1.8)	101	(1.8)	150	(2.6)	91	(1.6)	137	(2.4)	57	(1.0)
16	Area	-	-	422	(1.5)	333	(1.22)	422	(1.5)	496	(1.8)	274	(1.0)
	Delay	-	-	211	(2.1)	217	(2.2)	252	(2.5)	338	(3.5)	101	(1.0)

Table 1: The result of implementing a squaring unit using six different design approaches and three different word sizes. The area is in used number of logic elements and the delay is in ns. The numbers in parentheses are relative to the results obtained by the Sliced design method.

ables of type `natural`, constrained to the desired data range. The splitting operation, mentioned in Table 1, means splitting of the input port, so that the equation will look like (1) when splitting in two is desired. Splitting in four is then assuming the input value to be $X = (X_1 + 16X_2 + 256X_3 + 4096X_4)$ with 16-bit input width.

In the Altera-optimised approach each CAF block is represented as a set of equations only. This definition lets the Altera compiler eliminate the array structure and consider the squaring unit as a set of boolean equations without predefined structure. This means, that the compiler is allowed to insert logic elements.

In the Altera-direct implementation the design specification is followed more directly. The resulting structure is very close to the source definition.

It is clear from Table 1, that the Sliced design, which is optimised to utilise the LE structure of the FLEX device, is the most efficient implementation.

It is also obvious that the Altera-direct definition produces a less efficient design, than the pure logical equations used in the Altera-optimised design. This means, that the original block diagram is not well suited for the FLEX architecture. This is because the requirements of the CAF-block realization do not match the basic LE. The CAF block is too large for one LE, but too small for two LE's, so some resources in each LE are still unused.

The Synthesised solutions have two nice features: The highest level of input description, (in fact, the arithmetic equation), and a good fitting ability. For instance, though the Sliced design is less area expensive, due to the use of carry chains, it occupies the entire epf81188 chip while using only 30 % of the available LE's. On the other hand, three copies of the considerable less efficient Synthesised (split in four design) could fit in one chip and provide more total computational power by using 99 % of the LE's.

Input width	Altera-optimised				Altera-direct			
	Area		Delay		Area		Delay	
8	19	(1.0)	79	(1.0)	37	(1.9)	106	(1.3)
16	93	(1.0)	368	(1.0)	141	(1.5)	275	(0.7)
32	409	(1.0)	1200	(1.0)	541	(1.3)	734	(0.6)

Table 2: The results from implementing the square root function. The area is in number of logic elements and the delay is in ns. The numbers in parentheses are relative to the results obtained by the Altera-optimised method.

3.2 Results from the Square Root Unit

We were not able to produce a working solution to the square root function from a high level VHDL specification using the Synthesised method. This is due to the more complicated algorithm of the square root extraction. The synthesis tool does not have any directives for implementing this function in an efficient way. Also, the Sliced design method, which was developed especially for the squaring function, is not considered here.

Table 2 presents six implementations of the square root function. It is seen from the table, that there is some correlation between input width and design area. The Altera-optimised implementation is still more area efficient due to the redundancy of the Altera-direct implementation. On the other hand, the opposite relationship is found for the delay, as the Altera-direct design gives a better speed performance for large square root extractors.

Device type	Delay (ns)	
	Simulated	Measured
16-bit squaring unit	135	100
32-bit square root unit	1200	1000

Table 3: Simulated and measured propagation delay of a 16-bit squaring and a 32-bit square root unit.

3.3 Measurements

A 16-bit Sliced design squaring unit and a 32-bit Altera-optimised square root unit are chosen for hardware implementation in a prototype system with a PCB board containing one FPGA chip (Altera epf81188) and an interface for down loading configuration data to the FPGA chip. Due to fitting problems, some additional buffers are added to the squaring unit design, which increases the expected delay by 26 %. At the same time, some timing logic is implemented as shown in the Figure 3. It contains two framing registers and a programmable delay generator, which provides a controlled delay of the clock signal for the output register relatively to the input register clock. With an oscilloscope this setup allows time interval measurements with an accuracy better than ±3 ns.

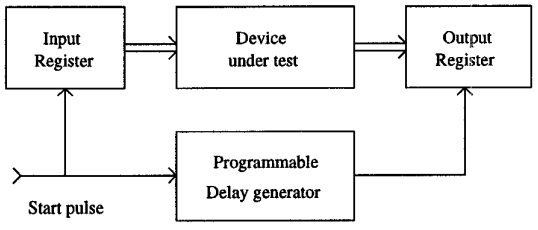

Figure 3: Additional circuit for actual speed measurement.

The prototype board is installed in a VME bus based computer system as a slave device and the FPGA implementations are tested at different speed rates. Two patterns of random data are applied to the input of the device, and by comparing the output with software generated values, the maximal operational speed of the implementation is obtained. Table 3 shows the experimental result from using a random pattern of length 10^7 numbers.

4 Codesign Aspects

The computations in the *In Betweening* case study, [1, 9], which was briefly mentioned in the introduction, falls in three separate parts.

First is the most computational intensive part by far, the 3D convolution, which is currently being implemented in a dedicated hardware prototype using ASIC's, (the 3D convolution engine), [8]. Second is an eigenvector analysis to determine a local flow vector for each pixel, [5]. This task is performed using a programmed solution on a traditional high performance workstation. Finally, the resulting local flow vectors are optimised globally by solving a large array of linear equations, also on the workstation. All in all, the combined hardware/software prototype setup gives a speed up from around seven days of CPU time in a pure software solution to around 20 minutes in the combined system.

This part of the work now considers the increase in system performance by moving the hardware/software border line one step into the eigenvector analysis by implementing the calculation of vector lengths in FPGA technology.

With the current speed requirements, (2.5 μs per vector), and 16-bit input and output data, the implementation of the vector length calculation requires two epf81188 chips. The first contains two 16-bit Synthesised squaring units, (with splitting in four), and one 32-bit ripple carry adder. Here the propagation delay will be 217 ns + 54 ns = 271 ns. The second chip contains an Altera-direct implementation of the square root extractor. The total propagation delay in the two chips will then be 271 ns + 734 ns $\simeq$ 1 μs per vector. These simulated numbers are typical, but the actual physical implementations have shown, that the FPGA chip is about 20 % faster than these simulated values.

To compare this FPGA solution to a pure software solution, a C program is written and run on different computers. It includes two 16-bit squaring operations, addition, 32-bit square root extracting and one disk access for each vector length calculation. The fastest execution was found to require 2.6 μs per vector, on a 150 MHz DEC Alpha workstation. However, a detailed comparison also has to take communication aspects into account.

There are two scenarios: A) The workstation receives data directly from the 3D convolution engine, 150 Mwords (16-bit) in three minutes, or B) The workstation receives 75 Mwords (16-bit) in three minutes from the FPGA module.

In scenario A), the workstation is fully occupied during the three minutes by receiving and storing the data, and no other processing can be performed simultaneously. When the calculation of the vector length has to be performed, it will require reading of the data form the disc and the calculation itself, which

in this case will take 150 Mwords * 2.6 μs $\simeq$ 6 minutes. Hereafter, the next steps in the image analysis, the eigenvector analysis and the global optimisation, [9], will take about 12 minutes in both situations. In scenario A) the total computation time will then be $3 + 6 + 12 = 21$ minutes.

In scenario B) it is possible to store the vector length data directly in the memory of the workstation due to the reduction by a factor of two of the total image data size. Therefore, the 6 minutes used in A) for disc operation and calculations are not required here, so the total computation time is reduced to 15 minutes.

5 Conclusions

Two main conclusions are drawn from this work. The first is about the utilisation of FPGA's in arithmetic calculations, and the second deals with codesign aspects in moving a specific computation from software to hardware.

5.1 Arithmetic Functions

This design and implementation study has shown, that for limited word size, 16/32-bit, functions of the type, X^2+Y^2 and $\sqrt{Z}$, can be implemented in the Altera FLEX FPGA in a variety of ways, leaving room for speed/area optimisation. However, if the fastest and often also smallest solution is chosen, this may lead to a low overall utilisation of the hardware resources in the FPGA, due to the high internal communication requirements. Less than 30 % utilisation is observed. If more modest speed requirements are present, a considerable larger part of the resources can be utilised. This means that the design process should include a step where the degree of parallelisation is investigated.

The considered vector length calculation with 16-bit input and output data, can be implemented in two epf81188 chips. The total simulated propagation delay is close to 1 μs, and the measured values are around 20 % faster. This result has a good margin to the available 2.5 μs in the current setup.

5.2 Codesign Aspects

System aspects are also considered in introducing the FPGA solution in the actual image analysis system; the 3D convolution engine and a high performance workstation both connected to a high speed network.

The FPGA solution is found to have some major advantages. The total amount of image data is reduced by a factor of two, to 75 Mwords, (150 Mbyte), which can be stored in the memory of the workstation. The consequence is, that one storage cycle of the complete data set is omitted. The input data rate to the workstation is hereby reduced, and the workstation is capable of doing other processing while receiving the data. All in all this will reduce the total processing time from 21 to 15 minutes.

Of course, the memory size in the workstation could be increased by around 200 Mbyte, so that the full set of image data could be stored directly in memory. However, this investment is about 20 times the price of the FPGA solution.

Finally, the investigated calculations could be made considerably faster using the same FPGA technology. A solution for real time image data, which leaves only 25 ns for the computation of one vector length, is within reach by introducing a high degree of pipelining in the considered array structures. The pipelining registers are present, one in each LE, but the overall timing and the required number of units in parallel has not been fully investigated presently.

Acknowledgements

The work presented here is partly financed by the *Codesign* research framework project, which is funded by the Danish Technical Research Council and lead by Prof. Jørgen Staunstrup. The visit by I. Klotchkov was also supported by the Jorck Foundation.

References

[1] Jens P. Brage and Steen Pedersen. A case study in architectural and technological trade-offs. *NORCHIP-94*, pages 78–85, November 1994.

[2] Altera Corp. *Max+Plus II, VHDL Version 5.0.* Altera Corp., 1994.

[3] Altera Corp. *Altera Data Book.* Altera Corp., 1995.

[4] Altera Corp. *Max+Plus II, Design Software Version 5.3.* Altera Corp., 1995.

[5] Anders Rosholm Henriksen. Analysis and realisation of in betweening algorithm. Master's thesis, Department of Computer Science, Technical University of Denmark, July 1995. In Danish.

[6] Kai Hwang. *Computer Arithmic: principles, architectures and design.* Wiley, 1979.

[7] Synopsys Inc. *Synopsys 3.1a User Guide.* Synopsys Inc., 1994.

[8] Dan C. Raun Jensen. 3D convolution VLSI ASIC. Master's thesis, Department of Computer Science, Technical University of Denmark, August 1994. In English.

[9] Rasmus Larsen. *Estimation of Visual Motion in Image Sequences.* PhD thesis, Institute for Mathematical Modelling, Technical University of Denmark, 1994.

[10] Neil H.E. Weste and Kamran Eshraghian. *Principles of CMOS VLSI Design.* Addison-Wesley, 1993.