© Journal of the Technical University - Sofia
Plovdiv branch, Bulgaria
"Fundamental Sciences and Applications"
Vol. 28, 2022

Ultimate Execution Speed of FPGA based Edge Detection: Parallel Addition

Dimitre Kromichev

Department of Marketing and International Economic Relations,
University of Plovdiv, 24 Tzar Asen Street,
Plovdiv 4000, Bulgaria

dkromichev@yahoo.com

Abstract. In the ultimate execution speed domain, parallel addition is critical for all FPGA based edge detection methods which rely on the Gaussian weighted average function as the filtering stage. With these methods, the first parameter of ultimate execution speed - maximum operating frequency, is defined by the integer arithmetic in Gaussian filtering. The second parameter of ultimate execution speed - minimum number of clock cycles taken to obtain accurate result, requires that a filtered image pixel must be available at the output of Gaussian filtering stage every clock cycle, irrespective of the size of Gaussian filter and the magnitude of its coefficients. Parallel addition has a very important role in the fulfillment of this requirement. The paper focuses on exploring the capabilities of parallel addition in FPGA based edge detection which uses Gaussian filtering to contribute to the goal of securing a Gaussian filtered pixel every clock cycle at the maximum operating frequency, and the analysis of the impact of parallel addition on the organization of Gaussian filtering computations. The obtained exploration results are based on ten Intel (Altera) FPGA families.

1. INTRODUCTION

In all FPGA based edge detection methods which rely on the Gaussian weighted average function as the filtering stage, the efficiency of pipelining in the entire method, and therefore the minimum number of clock cycles taken to obtain the edge detected image, depends on the number of clock cycles required to obtain a filtered image pixel at the output of Gaussian filter. Hence, in FPGA edge detection which uses Gaussian filtering the limit of ultimate execution speed in terms of its second parameter – minimum number of clock cycles, is: a filtered image pixel must be available at the output of Gaussian filter every clock cycle at the maximum possible operating frequency. Because both the size of Gaussian filter and the magnitude of its coefficients are variables, achieving a filtered pixel per clock cycle depends on the organization of Gaussian filtering computations and the indispensable integer arithmetic operations in the weighted average function. In this respect parallel addition has a very important role.

So far, in the studies of FPGA edge detection which uses Gaussian filtering the problem of parallel addition has never been addressed with respect to the maximum operating frequency and the minimum number of clock cycles. In [1] n-bit parallel adder is designed using Xilinx IP core function and implemented in Virtex-5. In [2] studied are approximate adders for Gaussian filter which are designed in hardware using only shifts and additions. The adder tree is implemented using a ripple carry adder based approximate adder and an error tolerant adder. The approximate architectures are compared with the precise implementation of Gaussian filter. In [3] the carry chains of hard adders are analyzed. It is pointed out that for simple adders carry chains increase performance by a factor of four or more, but for larger designs the overall performance improvement is roughly 15%. In [4] designed and simulated is a multi-operand carry save adder using carry look-ahead adder instead of the usual ripple carry adder. It is found that

speed increases by 10%. In [5] presented are different approaches to the implementation of generic carry-save compressor trees in FPGAs. They are used to design parallel multi-operand redundant adders. Analyzed is a novel linear array structure, which efficiently uses the fast carry-chain resources. A detailed study is provided for a wide range of bit widths and large number of operands. On the basis of comparison with binary and ternary carry propagate adder trees it is argued that a considerable increase in speed is achieved for 16-bit and 64-bit input data widths. In [6] proposed are high-radix parallel prefix adders and modular adders which utilize the fast carry chains in FPGA. In [7] proposed is an approach in which the carry chain length is compressed to N/2. It is found that the proposed adder is faster than a normal adder for a word length larger than 64 bits in Virtex-6. In [8] multi-operand adder delay is analyzed in terms of various methods of optimizations in multi-operand addition. It is pointed out that multi-operand adders are generally implemented as array adders or adder tree structure. Different multi-operand adders are assessed in terms of propagation delay, power consumption and resource utilization in Virtex-6. It is found that Wallace tree adder is the fastest adder and consumes the least amount of power and resources. In [9] it is argued that carry look ahead adder, carry select adder, carry save adder and carry skip adder are the most popular high speed adders. Proposed is a carry save adder with multi-operand addition in which the carry is saved and propagated down to the next stage. The final sum calculation stage in the carry save adder is implemented in this study using ripple carry adder, carry look ahead adder and carry select adder. Comparison in terms of power, area and delay is conducted for addition of four 4-bit numbers, four 8-bit numbers, four 16-bit numbers and four 32-bit numbers.

The objective of this paper is to present a research into parallel addition with respect to the concept of ultimate execution speed of FPGA edge detection which relies on the Gaussian weighted average function as the filtering stage. The focus is on the two parameters of ultimate execution speed: maximum operating frequency $F_{\max}$ and minimum number of clock cycles required to guarantee mathematically accurate result $nTclk_{\min}$. Due to the computational specifics of weighed average function, in FPGA edge detection which uses Gaussian filtering the adding of all convolution results in parallel is critical to the goal of achieving ultimate execution speed. Thus the tasks are: (1) explore the capabilities of parallel addition to contribute to the goal of securing a Gaussian filtered pixel every clock cycle at $F_{\max}$; (2) define the impact of parallel addition on the organization of Gaussian filtering computations. The explorations are conducted on the basis of ten Intel (Altera) FPGA families. Used tools: Scilab, Intel (Altera) Quartus, TimeQuest Timing Analyzer, ModelSim. The hardware description language is VHDL. Relevant to the analyses and conclusions arrived at are gray scale images.

2. PARALLEL ADDITION IN FPGA BASED EDGE DETECTION WHICH USES GAUSSIAN FILTERING

The computation of Gaussian weighted average function depends on these integer arithmetic operations: multiplication, addition, division. $F_{\max}$ of FPGA based edge detection which uses Gaussian filtering is defined by the maximum operating frequency of multiplication executed by hard multiplier $F_{\max}(hardMult)$. It is device dependent. Therefore, for parallel addition it is required that

$$F_{\max}(Paralladd) > F_{\max}(hardMult) \tag{1}$$

where

$F_{\max}(parallAdd)$ is the maximum operating frequency of parallel addition.

Because pipelining efficiency of FPGA based edge detection which uses Gaussian filtering is a function of $nTclk_{\min}$ of weighted average function, in order to obtain a filtered image pixel at the output of Gaussian filter every clock cycle the following basic assumptions must be fulfilled:

1. Multiplication is executed within a single clock cycle
2. All image pixels in the square neighborhood defined by Gaussian filter are accessible within a single clock cycle
3. Division is executed within a single clock cycle.

Assumptions #1 and #3 depend entirely on the computational mechanism of multiplication and division algorithms used in the weighted average function. Assumption # 2 depends on the organization of computations in Gaussian filtering. $nTclk_{\min}$ of FPGA based edge detection is device independent. It is impacted only by the organization of computations in each edge detection module and $nTclk_{\min}$ of each concrete integer arithmetic operation. Therefore, with respect to parallel addition in Gaussian filtering the goal of ultimate execution speed requires that:

$$nTclk_{\min}(parallAdd) = 1 \tag{2}$$

where

$nTclk_{\min}(parallAdd)$ is the minimum number of clock cycles required for parallel addition to secure mathematically accurate result.

With respect to the order of execution of integer arithmetic operations, there are two general approaches to FPGA based weighted average computations:

- Distributive law is used

$$\sum_{i=1}^{N_C} \frac{R_{C_i}}{S_{coeff}} \tag{3}$$

where

N_C is the number of convolutions,

R_{C_i} is a convolution result,

S_{coeff} is the sum of filter's coefficients.

- Distributive law is not used

$$\frac{\sum_{i=1}^{N_C} R_{C_i}}{S_{coeff}} \tag{4}$$

Thus FPGA based parallel addition must be explored for satisfying expressions (1) and (2) under the test conditions determined by approaches (3) and (4).

3. EXPLORING $F_{\max}(parallAdd)$ AND $nTclk_{\min}(parallAdd)$ IN FPGA

Methodology:

- $F_{\max}(hardMult)$ of 18x18 multiplier is explored and defined for the targeted Intel (Altera) FPGA families.

Lpm_mult megafunction is used to implement the hard multiplier in Cyclone II-V and Stratix I-V. There are no hard multipliers in Cyclone FPGA family. In Cyclone family multipliers are implemented with Lpm_mult megafunction using logic resources only. Used are the hard multipliers (embedded and DSP based) with symmetric inputs according to their availability in each particular FPGA family. All obtained results are for the highest speed grade in a particular FPGA family. All tests are performed for the entire range of possible input values defined by the inputs of 18x18 hard of multiplier.

- Lpm_parallel_add megafunction is used to implement a purely combinational parallel adder
- Parallel addition is executed within a single clock cycle
- The number of inputs is defined by the Gaussian filter size $z * z$, z is an odd number and $z \geq 3$
- Maximum input data width for (3) is defined by the largest size of a Gaussian filtered pixel. Therefore, maximum input data width for (3) is 8 bits.

- Maximum input data width for (4) is equal to the maximum output of 18x18 hard multiplier according to

$$L_C * L_P = 2^{17} * (2^8 - 1) \tag{5}$$

where

L_C is the largest Gaussian coefficient value

L_P is the largest gray scale image pixel value.

Therefore, maximum input data width for (4) is 25 bits.
The obtained results are shown in Table 1, Table 2, Table 3 and Table 4.

TABLE 1. $F_{max}(hardMult)$ of 18x18 hard multiplier

FPGA family	$F_{max}(hardMult)$ of 18x18 hard multiplier (in MHz)
Cyclone	**-**
Cyclone II	**248**
Cyclone III	**291**
Cyclone IV	**292**
Cyclone V	**295**
Stratix	**278**
Stratix II	**401**
Stratix III	**503**
Stratix IV	**505**
Stratix V	**507**

In Cyclone FPGA family, the maximum operating frequency of logic elements based multiplier is obtained for multiplier of size 16x16. It is 132 MHz.

TABLE 2. $F_{max}(parallAdd)$ for input data width = 8 bits and Gaussian filter of sizes: 3x3, 5x5, 7x7, 9x9

FPGA family	$F_{max}(parallAdd)$ for input width = 8 bits and Gaussian filter 3x3 (in MHz)	$F_{max}(parallAdd)$ for input width = 8 bits and Gaussian filter 5x5 (in MHz)	$F_{max}(parallAdd)$ for input width = 8 bits and Gaussian filter 7x7 (in MHz)	$F_{max}(parallAdd)$ for input width = 8 bits and Gaussian filter 9x9 (in MHz)
Cyclone	**275**	**239**	**201**	**164**
Cyclone II	**389**	**353**	**318**	**282**
Cyclone III	**462**	**422**	**379**	**338**
Cyclone IV	**470**	**426**	**375**	**334**
Cyclone V	**479**	**433**	**382**	**338**
Stratix	**417**	**389**	**351**	**316**
Stratix II	**565**	**531**	**489**	**444**
Stratix III	**712**	**664**	**610**	**561**
Stratix IV	**761**	**721**	**615**	**561**
Stratix V	**784**	**732**	**619**	**563**

TABLE 3. $F_{max}(parallAdd)$ for input data width = 8 bits and Gaussian filter of sizes: 11x11, 13x13, 15x15, 17x17

FPGA family	$F_{max}(parallAdd)$ for input width = 8 bits and Gaussian filter 11x11 (in MHz)	$F_{max}(parallAdd)$ for input width = 8 bits and Gaussian filter 13x13 (in MHz)	$F_{max}(parallAdd)$ for input width = 8 bits and Gaussian filter 15x15 (in MHz)	$F_{max}(parallAdd)$ for input width = 8 bits and Gaussian filter 17x17 (in MHz)
Cyclone	**163**	**127**	**109**	**41**
Cyclone II	**279**	**244**	**221**	**178**
Cyclone III	**334**	**289**	**266**	**207**
Cyclone IV	**329**	**290**	**268**	**209**
Cyclone V	**334**	**292**	**273**	**212**
Stratix	**310**	**275**	**255**	**192**
Stratix II	**441**	**399**	**372**	**312**
Stratix III	**549**	**501**	**473**	**430**
Stratix IV	**556**	**503**	**475**	**432**
Stratix V	**559**	**505**	**479**	**439**

TABLE 4. $F_{max}(parallAdd)$ for input data width = 25 bits and Gaussian filter of sizes: 3x3, 5x5, 7x7, 9x9

FPGA family	$F_{max}(parallAdd)$ for input width = 8 bits and Gaussian filter 3x3 (in MHz)	$F_{max}(parallAdd)$ for input width = 8 bits and Gaussian filter 5x5 (in MHz)	$F_{max}(parallAdd)$ for input width = 8 bits and Gaussian filter 7x7 (in MHz)	$F_{max}(parallAdd)$ for input width = 8 bits and Gaussian filter 9x9 (in MHz)
Cyclone	**161**	**122**	**81**	**39**
Cyclone II	**262**	**221**	**169**	**121**
Cyclone III	**314**	**261**	**214**	**172**
Cyclone IV	**316**	**266**	**217**	**174**
Cyclone V	**318**	**269**	**219**	**178**
Stratix	**294**	**253**	**212**	**171**
Stratix II	**422**	**372**	**338**	**295**
Stratix III	**513**	**479**	**435**	**388**
Stratix IV	**534**	**485**	**441**	**392**
Stratix V	**545**	**491**	**446**	**399**

4. ANALYSIS OF RESULTS

The obtained results for implementing parallel addition by using Lpm_parallel_add megafunction show that:

- $F_{max}(parallAdd)$ is impacted by the size of Gaussian filter
- $F_{max}(parallAdd)$ is inversely proportional to the number of addends
- $F_{max}(parallAdd)$ is inversely proportional to the input data width
- For the same size of Gaussian filter $F_{max}(parallAdd)$ is higher when the organization of computations in weighted average function is based on (3)
- When the organization of computations in Gaussian filtering is based on (3)

$F_{max}(parallAdd) > F_{max}(hardMult)$ for six Gaussian filter sizes: 3x3, 5x5, 7x7, 9x9, 11x11, 13x13

- When the organization of computations in Gaussian filtering is based on (4)

$F_{max}(parallAdd) > F_{max}(hardMult)$ only for Gaussian filter of size 3x3

- In order to increase the number of Gaussian filter sizes which can be used with (3) and (4) the parallel addition must be pipelined.

Therefore, the data for parallel addition in Table 2, Table 3 and Table 4 as compared to the data in Table 1 define that it is expression (3) on which the organization of computations must be based. Even when using expression (3) the goal of ultimate execution speed in FPGA based edge detection can be achieved only for a limited number of Gaussian filter sizes. Pipelining of parallel addition cannot be applied because in this case the task of securing a filtered image pixel at the output of Gaussian filter every clock cycle cannot be fulfilled.

CONCLUSION

Presented is a research into parallel addition with respect to the goal of achieving ultimate execution speed of FPGA edge detection which uses Gaussian weighted average function as the filtering stage. With respect to the computational specifics of weighed average function, the exploration of maximum operating frequency of parallel addition in comparison with the maximum operating frequency of 18x18 hard multiplier in ten Intel (Altera) FPGA families shows that the non-pipelined parallel addition can be used with a limited number of filter sizes when Gaussian filtering is based on distributive law. If distributive law is not used, non-pipelined parallel addition can be used only with the smallest Gaussian filter size. Therefore, in terms of parallel addition, the organization of computations in Gaussian filtering must be based on distributive law.

REFERENCES

1. B. Khaleelu Rehman, Waaiz Mohammad, Mudasar Basha, Salauddin Mohammad, Hardware Implementation of Parallel adder/Subtarctor and Complex Muliplier using Xilinx IP-Core, International Journal of Technology, Management & Knowledge Processing, Vol. 1, Issue 1, 2021, pp. 6-11
2. J. de Oliveira, L. Soares, E. Costa and S. Bampi, Exploiting approximate adder circuits for power-efficient Gaussian and Gradient filters for Canny edge detector algorithm, 2016 IEEE 7th Latin American Symposium on Circuits & Systems (LASCAS), 2016, pp. 379-382
3. J. Luu et al., On Hard Adders and Carry Chains in FPGAs, 2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines, 2014, pp. 52-59
4. Komal, Gautam A. K., Design and simulation of Carry Save Adder using VHDL, International Journal of Advance Research, Ideas and Innovations in Technology, Volume 5, Issue 3, 2019, pp. 1229-1233
5. Mudasir M, Tulasi Sanath Kumar, Multioperand Redundant Adders on FPGAs, International Journal of Scientific Engineering and Technology Research Volume.03, Issue No.47, December-2014, pp. 9462-9466
6. M. Rogawski, E. Homsirikamol and K. Gaj, A novel modular adder for one thousand bits and more using fast carry chains of modern FPGAs, 2014 24th International Conference on Field Programmable Logic and Applications (FPL), 2014, pp. 1-8
7. Petter Källström and Oscar Gustafsson, Fast and Area Efficient Adder for Wide Data in Recent Xilinx FPGAs, 2016, 26th International Conference on Field-Programmable Logic and Applications, Lausanne, Switzerland August 29 - September 2, 2016, pp. 1-4
8. S. Kannappan, S. Aruna Mastani, A Survey on Multi-operand Adder, Acta Technica Corvinensis - Bulletin of Engineering, Tome XIII, Fascicule 2, April – June 2020, pp. 65–68
9. VS. Balaji, Har Narayan Upadhyay, FPGA Implementation of High Speed and Low Power Carry Save Adder, Special Issue Energy, Environment, and Engineering Section: Recent Advances in Big Data Analysis (ABDA) Vol. 7 (11), 2016, 151-159