Efficient Sum of Absolute Difference Computation on FPGAs

Martin Kumm, Marco Kleinlein and Peter Zipf
University of Kassel, Germany
Digital Technology Group
Email: kumm@uni-kassel.de, kleinlein@uni-kassel.de, zipf@uni-kassel.de

Abstract

An improved architecture for efficiently computing the sum of absolute differences (SAD) on FPGAs is proposed in this work. It is based on a configurable adder/subtractor implementation in which each adder input can be negated at runtime. The negation of both inputs at the same time is explicitly allowed and used to compute the sum of absolute values in a single adder stage. The architecture can be mapped to modern FPGAs from Xilinx and Altera. An analytic complexity model as well as synthesis experiments yield an average look-up table (LUT) reduction of 17.4% for an input word size of 8 bit compared to state-of-the-art. As the SAD computation is a resource demanding part in image processing applications, the proposed circuit can be used to replace the SAD core of many applications to enhance their efficiency.

1. Introduction

The computation of the sum of absolute differences (SAD) is an important core operation that is found in numerous image and video processing applications. It is the most commonly used metric to measure the *distance* between two blocks of an image and is used, e. g., in motion estimation [1], [2] or stereo matching [3]. Although it is one of the lowest cost distance measure, the SAD operation is a resource and time consuming part due to the multitude of absolute value computations and additions. In some applications, it is even the most complex operation [2]. Typical block sizes range from 3×3 to 11×11 in stereo matching [3], 16×16 for H.264/AVC and up to 64×64 for HEVC [2].

Formally, the SAD computation of two $R \times C$ matrices $\mathbf{A}$ and $\mathbf{B}$ can be written as

$$\mathrm{SAD}(\mathbf{A}, \mathbf{B}) = \sum_{i=1}^{R} \sum_{j=1}^{C} |a_{i,j} - b_{i,j}| \ , \qquad (1)$$

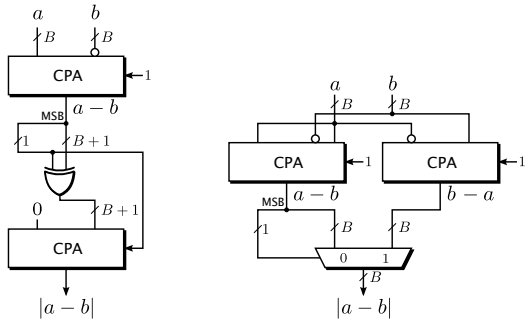
where $a_{i,j}$ and $b_{i,j}$ denote the element in the i'th row and the j'th column of matrix $\mathbf{A}$ and $\mathbf{B}$, respectively. Due to the inherent parallelism in the SAD scheme, its computation can be accelerated by parallel architectures like field programmable gate arrays (FPGAs).

The main contribution of this paper is a novel architecture computing SAD which is more efficient than the previous approaches and is not limited to a single FPGA vendor. Our focus is on the very fundamental SAD computation in a parallel and (possibly) pipelined way. How to embed the SAD operation in an image processing application can be found in several previous publications [1]–[4]. The proposed SAD architecture can be used in nearly any of these image processing methods cited above to improve their efficiency on FPGAs.

2. Previous Work

A lot of research effort has been performed in the last decade on SAD computation on FPGAs [1]–[7]. Typically, the SAD operation is divided into first computing the absolute difference $|a_{i,j} - b_{i,j}|$ and, second, to sum up these in a multi-input addition. Figure 1 shows two common ways to compute the absolute difference $|a - b|$ (the indices were removed for clarity). In Figure 1(a), the difference $a - b$ is computed by using a carry propagate adder (CPA) and the result is negated in case of a negative result by a bit-wise XOR operation, followed by a conditional incrementation [8], [9]. Figure 1(b) shows another common but probably faster way to compute the same operation [2], [7]. Here, both differences $a - b$ and $b - a$ are computed in parallel and the positive result is selected by the most significant bit (MSB) (sign bit) of one of the results using a multiplexer (MUX). This reduces the critical path while having a slight increase in complexity (in terms of gate count).

In [1], on-line (digit-by-digit) arithmetic was used to reduce the complexity of SAD. A high performance



(a) Sequential AD [8], [9] (b) Parallel AD [2], [7]

Figure 1: Common absolute difference (AD) circuits

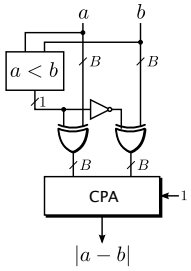


Figure 2: FPGA optimized absolute difference circuits due to Perri et. al. [6]

SAD architecture using carry-save arithmetic targeting application specific integrated circuits (ASICs) is considered in [9]. An efficient scheme of re-using SAD results from previous candidate block computations is proposed in [3]. The computation of large SADs using smaller SAD cores at different levels of parallelism which scales well to the 64×64 requirements of HEVC is proposed in [2]. An FPGA implementation of a 16×1 SAD was proposed in [5] which is based on a SAD architecture that was previously proposed by the same group [10]. Their main idea is to detect the smaller operand in the absolute difference computation $|a - b|$ and to subtract it from the larger operand [5], [10].

While most of the methods discussed so far were mapped to FPGAs, there were no FPGA-specific optimizations performed. An FPGA optimized absolute difference (AD) circuit for Xilinx devices was proposed by Perri et al. [6]. Its overall structure is shown in Figure 2. First, a comparator checks for $a < b$, if true, the operation $b - a$ is performed while in the other case $a - b$ is computed. The effectiveness of this structure comes from a clever way in realizing the comparator. Instead of computing $a-b$ and evaluating the carry output, the carry-chain MUXes of the Xilinx architecture are used which saves about half of the LUTs.

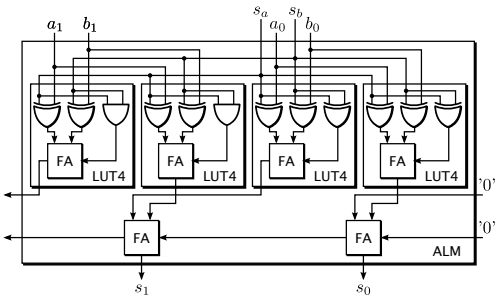


Figure 3: Altera ALM configuration of the proposed generalized adder/subtractor computing $s = (-1)^{s_a}a + (-1)^{s_b}b$

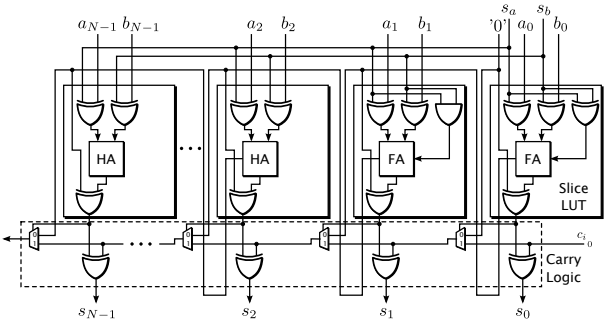


Figure 4: Xilinx Slice configuration of the proposed generalized adder/subtractor

3. Proposed SAD Architecture

3.1. Generalized Adder/Subtractor

The proposed SAD architecture is based on a novel configurable adder/subtractor where both inputs can be negated while – in contrast to other realizations – the negation of both inputs at the same time is explicitly allowed. This generalized adder/subtractor can be described by

$$(-1)^{s_a}a + (-1)^{s_b}b \ , \tag{2}$$

i. e., the inputs a and/or b are negated when s_a and/or s_b are set, respectively. The circuit performing the first two bits of this operation when mapped to an Altera adaptive logic module (ALM) in shared arithmetic mode (available on Arria I,II,V and Stratix II-V families) is shown in Figure 3. The mapping to a Xilinx Slice (Virtex 5/6, Spartan 6 and 7 Series) is shown in Figure 4. In these figures, a_i and b_i denote the i'th bit of a and b, respectively. A bit-wise inversion of the inputs a and b is performed in case s_a or s_b are set by using two XOR gates per LUT, similar to the architecture in Figure 2. To finalize the complement, an additional '1' has to be added for each non-zero s_a and s_b. This is done by adding an additional vector using an additional stage of full adders (FAs) realized

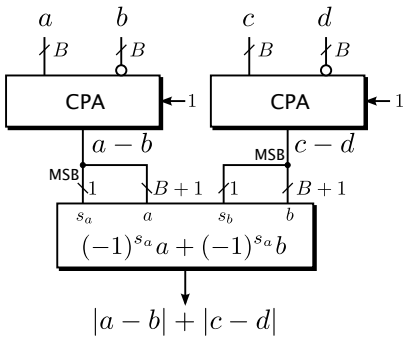


Figure 5: Proposed 1×2 SAD architecture

in the LUTs, similar to the realization of a ternary (3-input) adder [11], [12]. This additional vector is set to "00...000" in case that both s_a and s_b are zero, it is set to "00...001" in case that only one out of s_a and s_b is '1' and it is set to "00...010" in case that both s_a and s_b are '1'. The lowest two bits of this vector are obtained from an XOR (LSB position) and an AND gate (LSB+1 position), respectively, while the bits of higher significance are always zero. This is why FAs in the higher stages are replaced by half adders (HAs). Each carry produced by the FAs realized in the LUTs has to be routed to the next adjacent LUT. On Altera FPGAs, this is done by using the shared arithmetic mode of the ALM while on Xilinx FPGAs, a local routing connection is required. Note that this carry only depends on the inputs and, thus, does not ripple through the stages of the circuit. Like a common two-input adder, the generalized adder/subtractor requires exactly B LUTs for computing a B bit output word.

3.2. Overall SAD Architecture

The configurable adder/subtractor circuits in figures 3 and 4 can be used to compute $|a| + |b|$ by simply connecting the MSB (sign bit) of a and b to s_a and s_b, respectively. Now, a 1×2 SAD can be realized by the structure shown in Figure 5. First, the differences of the inputs are computed and, next, the absolute values of the differences are added using the configurable adder/subtractor. To realize SAD circuits with larger input size, a multi-input addition is used to sum up the results from several 1×2 SAD units.

4. Results

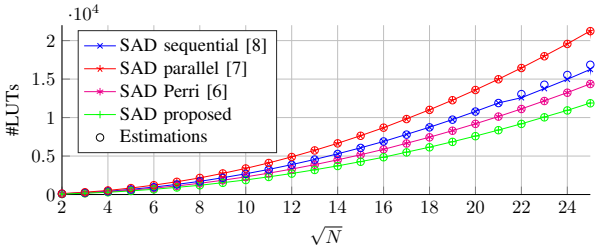
The proposed and previous SAD circuits are predictable in their LUT count. Table 1 provides a summary of the number of LUTs that are required for each method in dependence of the number of input pairs N and the word size B using a common adder tree for the multi-input addition. It can be observed that the complexity of the proposed SAD circuit grows with only $2NB$ while the best circuit reported so far grows with $2.5NB$.

Table 1: Analytical LUT comparison of SAD circuits computing N input pairs of B-bit each

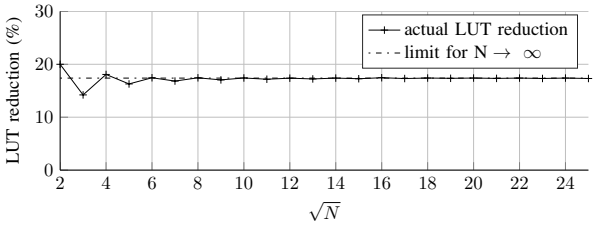
Architecture	#LUTs
[8], [9] (Figure 1(a))	$3NB - B + 3N - \log_2 N - 2$
[2], [7] (Figure 1(b))	$4NB - B + 2N - \log_2 N - 2$
[6] (Figure 2)	$2.5NB - B + 3N - \log_2 N - 2$
proposed	$2NB - B + 3N - \log_2 N - 2$

To be able to compare the resource usage and speed as well as a validation for the LUT estimations above, synthesis experiments were performed. For that, the proposed SAD architecture, the common SAD architectures based on the AD circuits of Figure 1 as used in [2], [7]–[9] as well as the state-of-the-art architecture of Perri [6] were implemented. A VHDL code generator was realized for all four architectures using the open-source arithmetic core generator framework FloPoCo [14]. It supports different implementation styles for the multi-input addition including a common adder tree and an optimized compressor tree [13]. The code generator is also made available as open-source in the "uni_ks" branch of the FloPoCo git repository. In the experiment, the input word size was set to 8 bit and the number of inputs was varied for rectangular matrices between $N = 2 \times 2 \ldots 25 \times 25$, which covers most practical applications. All implementations are deeply pipelined, i. e., a register is used after each processing stage. For a fair speed comparison, registers were placed at inputs and outputs. All multi-input additions were realized using a common adder tree. However, more sophisticated compression methods like [13] would further improve the results.

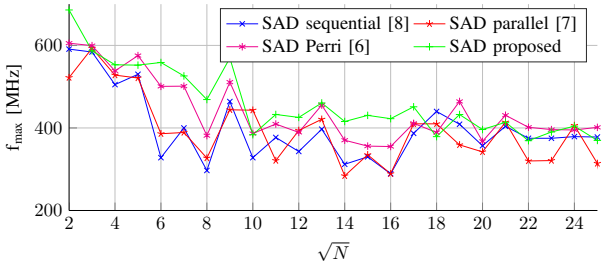
The synthesis was performed for a Xilinx Virtex 6 FPGA (XC6VSX475T-1) using ISE v.13.4 with design goal 'speed'. The synthesis results after place & route are shown in Figure 6. It can be observed in Figure 6(a) that the LUT estimations of Table 1 (marked with circles) are very close to the actual LUT usage. The relative LUT reduction compared to [6] is illustrated in Figure 6(b). It can be seen that it quickly converges to a constant average of 17.36%. This limit can be obtained from the analytic formula in Table 1 for $B = 8$ bit and $N \to \infty$, leading to $\frac{8}{46} = 17.39\%$. The resulting Slice resources show a very similar trend and an average reduction of 14% compared to [6]. The maximum clock frequencies are plotted in Figure 6(c). The average speeds are 374.5 MHz, 373.8 MHz, 431.7 MHz and 449.0 MHz for the sequential, parallel, Perri and the proposed SAD architecture, respectively.



(a) Required and estimated LUTs



(b) Relative LUT reduction compared to [6]



(c) Clock frequency ($f_{\max}$)

Figure 6: Synthesis results and theoretical LUT resource usage

5. Conclusion

An improved SAD architecture is proposed which provides a significant resource reduction on current FPGAs. Its LUT complexity only grows with $2NB$ while the best state-of-the-art method grows with $2.5NB$, leading to practical resource reductions of 17.4%. As the number of inputs in the multi-input addition is halved in the proposed method, it is expected that similar reductions are obtained when using advanced compressor tree optimization methods [13]. The proposed circuit can be used to increase the efficiency of all the previously proposed applications from image processing that use the SAD metric in their core computation.

References

[1] J. Olivares, J. Hormigo, J. Villalba, and I. Benavides, "Minimum Sum of Absolute Differences Implementation in a Single FPGA Device," in *Field Programmable Logic and Applications (FPL)*, Aug. 2004, pp. 986–990.

[2] P. Nalluri, L. N. Alves, and A. Navarro, "High Speed SAD Architectures for Variable Block Size Motion Estimation in HEVC Video Coding," in *IEEE International Conference on Image Processing (ICIP)*. IEEE, 2014, pp. 1233–1237.

[3] K. Ambrosch, M. Humenberger, W. Kubinger, and A. Steininger, "SAD-Based Stereo Matching Using FPGAs," in *Embedded Computer Vision*. London: Springer London, 2009, pp. 121–138.

[4] G. Schewior, C. Zahl, H. Blume, S. Wonneberger, and J. Effertz, "HLS-Based FPGA Implementation of a Predictive Block-Based Motion Estimation Algorithm - a Field Report," *Conference on Design and Architectures for Signal and Image Processing (DASIP)*, pp. 1–8, 2014.

[5] S. Wong, B. Stougie, and S. Cotofana, "Alternatives in FPGA-based SAD Implementations," in *IEEE International Conference on Field-Programmable Technology (FPT)*. IEEE, 2002, pp. 449–452.

[6] S. Perri, P. Zicari, and P. Corsonello, "Efficient Absolute Difference Circuits in Virtex-5 FPGAs," in *IEEE Mediterranean Electrotechnical Conference (MELECON)*. IEEE, 2010, pp. 309–313.

[7] T.-A. Chirila-Rus, "Determining Sum of Absolute Differences in Parallel," Patent US 8,131,788, Mar., 2012.

[8] T. Kanoh, "Absolute Value Calculating Circuit Having a Single Adder," Patent US 4,953,115, Aug., 1990.

[9] J. Vanne, E. Aho, T. D. Hamalainen, and K. Kuusilinna, "A High-Performance Sum of Absolute Difference Implementation for Motion Estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 7, pp. 876–883, Jul. 2006.

[10] S. Vassiliadis, E. A. Hakkennes, J. S. S. M. Wong, and G. G. Pechanek, "The Sum-Absolute-Difference Motion Estimation Accelerator," in *Euromicro Conference, 1998. Proceedings. 24th*. IEEE, 1998, pp. 559–566.

[11] J. M. Simkins and B. D. Philofsky, "Structures and Methods for Implementing Ternary Adders/Subtractors in Programmable Logic Devices," *US Patent No 7274211, Xilinx Inc.*, Sep. 2007.

[12] G. Baeckler, M. Langhammer, J. Schleicher, and R. Yuan, "Logic Cell Supporting Addition of Three Binary Words," *US Patent No 7565388, Altera Coop.*, 2009.

[13] M. Kumm and P. Zipf, "Pipelined Compressor Tree Optimization Using Integer Linear Programming," in *IEEE International Conference on Field Programmable Logic and Application (FPL)*. IEEE, 2014, pp. 1–8.

[14] F. de Dinechin and B. Pasca, "Designing Custom Arithmetic Data Paths with FloPoCo," *IEEE Design & Test of Computers*, vol. 28, no. 4, pp. 18–27, 2012.