Approximate Quaternary Addition with the Fast Carry Chains of FPGAs

Sina Boroumand
University of Tehran
Tehran, Iran
s_boroumand@ut.ac.ir

Hadi P. Afshar
Qualcomm Research
San Diego, USA
hpafshar@qti.qualcomm.com

Philip Brisk
University of California, Riverside
Riverside, USA
philip@cs.ucr.edu

***Abstract*—A heuristic is presented to efficiently synthesize approximate adder trees on Altera and Xilinx FPGAs using their carry chains. The mapper constructs approximate adder trees using an approximate quaternary adder as the fundamental building block. The approximate adder trees are smaller than exact adder trees, allowing more operators to fit into a fixed-area device, trading off arithmetic accuracy for higher throughput.**

***Index Terms*—Approximate Arithmetic, Adder tree, Ternary Adder, Quaternary Adder, FPGA**

I. Introduction

For the past few years, deep learning has been one of the foremost engines for growth in the computing industry. With the emergence of FPGA-accelerated deep learning platforms, an urgent need for tools to enhance the arithmetic performance of deep learning applications on FPGAs has emerged. In this paper, we take advantage of two key factors that differentiate deep learning models from other application domains. First, from an arithmetic standpoint, a typical deep learning model is an interconnected network of multiply-accumulate (MAC) operations. Second, deep learning models are inherently tolerant to error, and can therefore benefit from approximate arithmetic operators that sacrifice accuracy in order to improve performance, power consumption, or area utilization.

This paper describes an efficient method to implement approximate MAC operations on FPGA programmable logic. Although modern FPGAs contain hardened MAC blocks, a typical deep learning model will saturate the supply, and can therefore achieve higher throughput by configuring a portion of the programmable logic as MAC operators; throughput can be further increased by implementing approximate, rather than exact, arithmetic operators. The heuristic mapper proposed here takes advantage of the carry chains that are presently available on commercial FPGAs sold today by Xilinx and Altera (Intel Product Solutions Group).

Specifically, we efficiently implement approximate quaternary (4-input) adders on FPGA programmable logic, based on the rationale that they are useful building blocks for larger operators such as MACs. The approximate quaternary adders proposed here are no larger than the exact ternary (3-input) adders that can be realized naturally using FPGA carry chains. The higher compression ratio reduces the number of stages required to build the large adder trees that form the internal structure of the MAC operators, which reduces critical path delay and area; moreover, this allows for a greater number of MAC blocks to be synthesized into a fixed-size region of programmable logic, further increasing throughput.

II. Related Work

A number of papers in recent years have focused on the problem of synthesizing exact adder trees on FPGAs, exploiting carry chains when possible [1]–[4]. This paper is similar, but systematically introduces approximation into the adder tree designs to further increase the compression ratio.

On the other hand, there have been a number of approximate arithmetic component designs proposed for standard-cell design flows. Many of these techniques, such as truncation of the least significant bits [5] are equally valid and applicable to FPGAs as well. The drawback of truncation is that the error rate increases exponentially with each additional truncated bit.

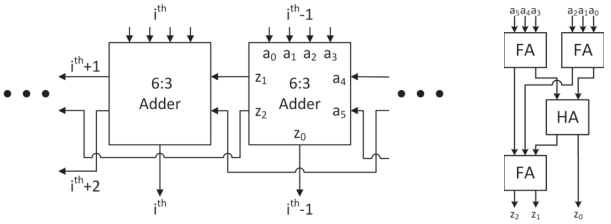
Liu et al. introduced an approximate 4:1 compressor that also produces an error signal [6]. They construct an approximate adder tree using these 4:1 compressors and add an accumulated vector to the adder tree output to reduce the error. One of our LUT configurations is similar to the 4:1 compressor introduced here, but our design eschews the additional error logic and is tailored to map efficiently onto FPGAs.

Venkatachalam and Ko [7] take pairs of partial product bits and ORs and ANDs them to respectively form propagate (P) and generate (G) signals. The P signals are compressed using approximate half-adders, full-adders, and 4:2 compressors (which can be approximate, if desired [8]), while the G signals are compressed using OR gates. Similarly, Ref. [9] generates the P signals, but not G signals, thereby reducing the number of bits to compress by half, which can be viewed as an approximate form of radix-4 Booth encoding. These circuits have informed our approach, but do not map well onto FPGAs.

III. Proposed Quaternary Adder

A quaternary adder adder sums 4 bits (all in the same bit position) and generates a sum and a carry bit. Figure 1 shows a structure of an exact quaternary adder: in each bit position the quaternary adder in position i receives two carry input bits and generates two carry output bits in bits positions $i+1$ and $i+2$, respectively. Thus, an exact quaternary adder requires two carry chains, but cannot be realized on modern FPGAs which only have one carry chain; however, it is possible to build an

978-3-9819263-0-9/DATE18/©2018 EDAA



(a) Mapping (b) Logic

Fig. 1. Structure of an exact quaternary adder.

approximate quaternary adder using the one available carry chain: the carry bit is forwarded to the next bit position through the carry chain, which creates an approximate quaternary ripple carry adder. Due to architectural differences between the Altera ALM and Xilinx Slice, implementations of this quaternary adder have lower accuracy on Altera devices. The next two subsections introduce and analyze the proposed structure for the quaternary adder on both targets.

A. Altera

We configure each half-ALM in arithmetic mode as a 3-LUT and a 2-LUT respectively with five independent inputs. In actuality, we only need 4 input bits, so one of the inputs to the 3-LUTs is not used. This yields the quaternary adder architecture shown in Figure 2. The 2-LUTs are configured as OR gates, each of which reduces two bits to one; the bits produced by the 2-LUTs are then summed using the carry chain, which generates sum and carry outputs.

The quaternary adder produces an exact result for nine of the sixteen possible 4-bit input combinations. The seven input bit combinations that generate approximate results are 0011, 0111, 1011, 1100, 1101, 1110, and 1111. The absolute error difference for input case 1111 is 2, and 1 for all other six input combinations. To the best of our knowledge, this is the most accurate quaternary adder implementation that can be achieved using Altera's current carry chains. Higher accuracy could be achieved by extending the "shared arithmetic mode" of the carry chain so that the fracturable LUTs can be configured as parallel 4-LUTs with independent inputs; at present, shared arithmetic mode only supports 3-LUTs.

B. Xilinx

The LUTs in the Xilinx Slice architecture have fewer restrictions on input sharing compared to the Altera ALM in arithmetic mode. We configure the two LUTs in a slice as 4-LUTs with shared inputs, with logic functions shown in Figure 3a for two consecutive bit positions; Figure 3c shows the actual mapping and its relation to the carry chain. This configuration only produces an incorrect output in one of sixteen possible input bit combinations: 1111. In this case, the result sums to 3, rather than 4.

IV. Mapping Algorithm

The objective of the mapping algorithm is to produce an approximate adder tree using approximate quaternary adders as building blocks. The general strategy is to reduce 4 rows of bits to one row of bits, four rows at a time. Each stage of the reduction tree may require several parallel quaternary adders, and multiple stages may be needed to reduce the inputs to a single row. If at some point the reduction tree produces three rows, then they will be summed by a ternary adder, rather than undergoing another stage of compression.

When targeting Altera devices, we are given some leeway in situations where a given bit position has less than four input bits; the unused bits are set to 0. For example, suppose that only two bits are available. If we put them in positions 00bb, then the quaternary adder will produce an incorrect output when both bits are 1; on the other hand, if we put them in positions 0b0b, then the quaternary adder will produce the correct output when both bits are 1.

Since different applications may vary in terms of their accuracy requirements, we allow the user to specify a *pivot*, which is the bit position that separates approximate from exact compression. Bit positions more significant than the pivot are compressed exactly, using ternary adders (which have a lower compression ratio than our proposed quaternary adders), while bits at the pivot position or lower are compressed using approximate quaternary adders. By varying the pivot position, the user can trade-off accuracy for performance.

Adder trees are a fairly general structure, although they are mostly commonly used for partial product reduction in multipliers and MACs. In principle, the mapper described here can be used to implement an approximate adder tree for any purpose, not just multipliers or MACs.

V. Results

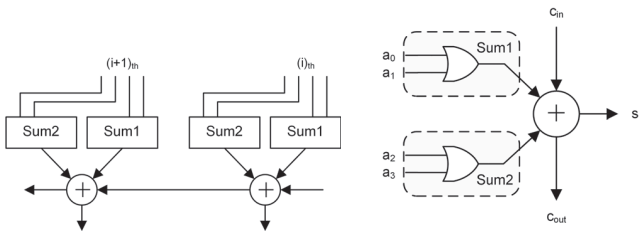
We implemented the mapping algorithm described in Section IV in C++, targeting both Altera and Xilinx FPGAs. The user provides the number of inputs to sum, their bitwidth, and the location of the pivot. The mapped circuit is fed into an error analysis engine, discussed next, to measure the accuracy of the resulting design. This process exhausts all input combinations for small adder trees, and a subset of input combinations for larger designs, and compares the approximate result with exact results produced by exact adder tree implementations.

A. Error Analysis Metrics

We use standard metrics for error analysis that have been published by others [6], [10]. For a given input pair (x, y), the Error Distance (ED) is the arithmetic difference between the accurate product $P_{exact}(x, y)$ produced by an exact multiplier and the approximate product $P_{approx}(x, y)$ produced by the approximate multiplier. The Relative Error Distance (RED) is the ratio of ED to the accurate product. The Mean Relative Error Distance (MRED) is the average of ED values across all possible input combinations.

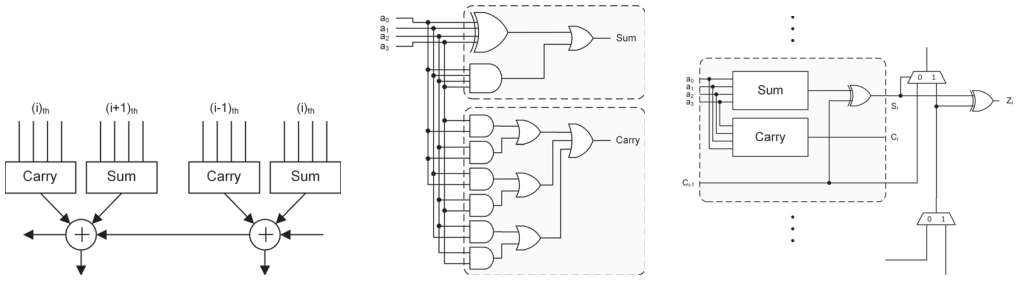
B. Design Cost Evaluation Methodology

As shown in Figures 2 and 3, each quaternary adder bit slice is mapped onto one logic cell (Altera half-ALM or Xilinx Slice). In other words, an approximate adder tree that requires



(a) Proposed Structure (b) Mapping

Fig. 2. The proposed structure to build an approximate quaternary adder on Altera FPGAs.



(a) Proposed Structure (b) Adder Logic (c) Mapping

Fig. 3. The proposed structure to build an approximate quaternary adder on Xilinx FPGAs.

Q quaternary adders and T ternary adders will require $Q+T$ ALMs/Slices. Q and T can be computed analytically when adding N B-bit unsigned integers with pivot position P. For bits in position P or larger, the number of adder tree stages is $S_G = \log_3 N$, since bits are compressed using ternary adders; for bits in position lower than P, the number of adder tree stages is $S_L = \log_4 N$, since bits are compressed using approximate quaternary adders. If we assume that the first stage of the adder tree is indexed by 0, then the bitwidth of each adder at stage i is $B + 2i$, to account for overflow.

To evaluate the impact using approximate quaternary adders in lieu of exact ternary adders on the critical path delay, we compare the total number of stages required to compress the user-provided inputs. The critical path first goes through the stages of the adder tree, and then continues through the carry-propagation chain of the final adder from the least to the most significant bit position. The delay of the carry-propagate component will be present regardless of whether ternary or quaternary adders are employed; therefore, it suffices to compare the number of levels of the adder tree.

It is also important to note that the wires that connect one stage of the adder tree to the next go through the FPGA routing network, whose critical path delays are generally greater than carry chain delays; the exact critical path delay cannot be known until after routing completes. Our mapper does not attempt to predict or optimize for routing delays.

C. Multipliers

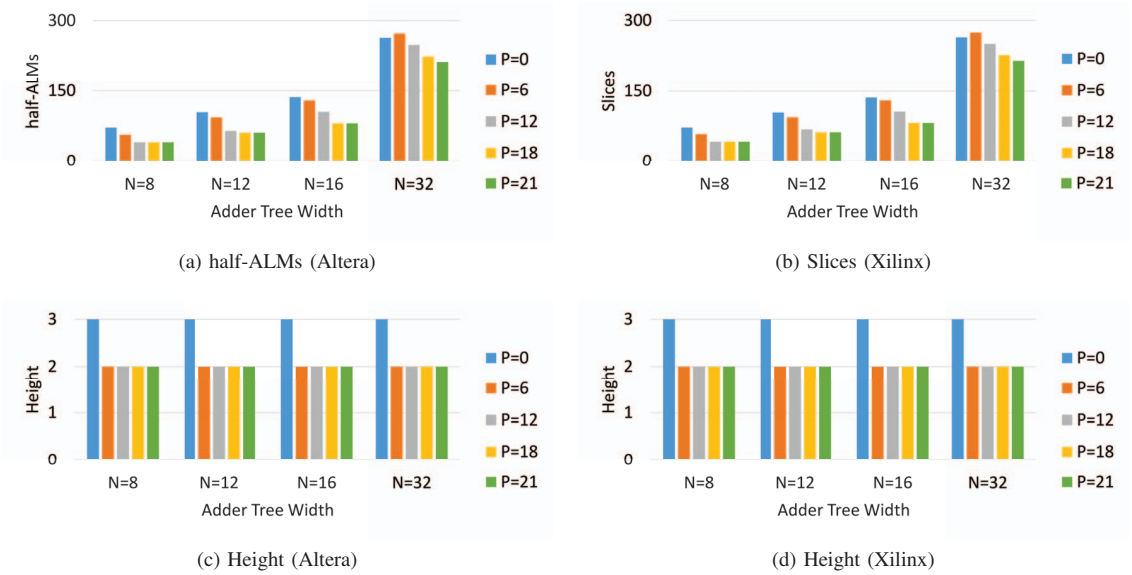
Table I reports the impact of pivot position on accuracy and resource usage when designing the adder tree for an 8×8-bit multiplier. As discussed earlier, the approximate multipliers achieve greater accuracy (lower ED and MRED percentages) for Xilinx compared to Altera. The exact adders (pivot = 0)

TABLE I
ERROR AND DESIGN COST ANALYSIS OF AN 8-BIT MULTIPLIER WHEN PIVOT VARIES FROM 0 TO 14.

N	P	Altera				Xilinx			
		MRED%	ER%	3:1	4:1	MRED%	ER%	3:1	4:1
8	14	3.57	55.43	13	20	0.044	2.29	13	20
8	12	3.57	54.90	14	18	0.044	2.29	14	18
8	10	2.04	52.60	21	16	0.031	2.11	21	16
8	8	1.00	46.88	32	12	0.015	1.73	32	12
8	6	0.31	34.76	47	8	0.005	0.97	47	8
8	4	0.08	21.87	44	4	0.002	0.58	44	4
8	2	0.00	0.00	48	0	0.000	0.00	48	0
8	**0**	**0.00**	**0.00**	**47**	**0**	**0.000**	**0.00**	**47**	**0**

TABLE II
DESIGN OF DIFFERENT MULTIPLIERS WITH DIFFERENT BIT-WIDTHS WHEN THE PIVOT BIT (P) POSITION VARIES.

N	P	Altera				Xilinx			
		Stages	3:1	4:1	Total	Stages	3:1	4:1	Total
32	32	3	48	359	407	3	48	361	409
32	16	3	331	186	517	3	336	186	522
32	**0**	**4**	**583**	**0**	**583**	**4**	**583**	**0**	**583**
16	16	2	1	99	100	2	0	100	100
16	8	2	78	53	131	2	79	53	132
16	**0**	**3**	**154**	**0**	**154**	**3**	**154**	**0**	**154**
12	12	2	19	42	61	2	20	42	62
12	6	2	59	24	83	2	60	24	84
12	**0**	**3**	**88**	**0**	**88**	**3**	**88**	**0**	**88**
8	8	2	11	20	31	2	12	20	32
8	4	2	22	12	34	2	23	12	35
8	**0**	**2**	**37**	**0**	**37**	**2**	**37**	**0**	**37**



(a) half-ALMs (Altera)

(b) Slices (Xilinx)

(c) Height (Altera)

(d) Height (Xilinx)

Fig. 4. The number of logic blocks for 16-bit adder trees synthesized on Altera (a) and Xilinx (b) FPGAs, as the number of inputs N and pivot position P varies; the height of the adder trees for Altera (c) and Xilinx (d).

consume 47 ALMs/Slices, while the maximally inexact adders in this table (pivot = 14) consume 33 ALMs (Altera) and 33 Slices (Xilinx), a savings of 30%. It is interesting to note that the largest overall designs require 55 ALMs for Altera (pivot = 6) and 55 Slices for Xilinx (pivot = 6). This occurs because the choice of pivot leaves some adders underutilized in the earliest stages of the reduction tree; thus, increasing the pivot position does not always lower resource usage.

Table II reports results for for multipliers with varying bitwidth (N) and pivot position (P). As discussed in Subsection V-B, the critical path tends to go through the least significant bits; in all cases, increasing the pivot position from 0 to 1 reduce the height of the tree by one logic layer; further increasing the pivot position does not yield further savings. The one exception is the 8×8-bit multiplier, where increasing the pivot position does not reduce the number of stages, due to the relatively small size of the adder tree. Greater savings is accrued when accounting for ALM and Slice counts.

D. Adder Trees

The same experiments were performed for adder trees, which add a set of equal-bitwidth unsigned integers. Figures 4a and 4b report the number of logic blocks used for different address for both target FPGAs. Increasing the pivot increases the area savings. Figures 4c and 4d report the height of the different adder trees. Increasing the pivot position from 0 to 6 reduces the height by one logic layer; further increasing the pivot position does not reduce the height of the adder trees.

VI. Conclusion and Future Work

The approximate adder trees introduced in this paper can be used by deep learning algorithms to implement efficient MAC operations on FPGA programmable logic. This paper has shown how the user can explore the approximate adder tree design space to tradeoff accuracy for reduced tree height (pipeline depth) and area. For large-scale arithmetic applications, this points to a multi-objective mapping strategy that simultaneously accounts for accuracy and throughput, including selection of the pivot point for each approximate operator. Long term, we anticipate that this research will improve the quality of deep learning models on FPGAs.

References

[1] H. Parandeh-Afshar, A. Neogy, P. Brisk, and P. Ienne, "Compressor tree synthesis on commercial high-performance FPGAs," *ACM Trans. Reconfigurable Technology and Systems*, vol. 4, no. 4, p. 39, 2011.

[2] M. Kumm, S. Abbas, and P. Zipf, "An efficient softcore multiplier architecture for Xilinx FPGAs," in *IEEE Int. Symp. on Computer Arithmetic*, June 2015, pp. 18–25.

[3] A. Kakacak, A. E. Guzel, O. Cihangir, S. Goren, and H. F. Ugurdag, "Fast multiplier generator for FPGAs with lut based partial product generation and column/row compression," *Integration, the VLSI Journal*, vol. 57, Supplement C, pp. 147 – 157, 2017.

[4] N. Brunie, F. de Dinechin, M. Istoan, G. Sergent, K. Illyes, and B. Popa, "Arithmetic core generation using bit heaps," in *Int. Conf. Field Programmable Logic and Applications*, Sept 2013, pp. 1–8.

[5] N. Petra, D. De Caro, V. Garofalo, E. Napoli, and A. G. Strollo, "Truncated binary multipliers with variable correction and minimum mean square error," *IEEE Trans. Circuits and Systems I: Regular Papers*, vol. 57, no. 6, pp. 1312–1325, 2010.

[6] C. Liu, J. Han, and F. Lombardi, "A low-power, high-performance approximate multiplier with configurable partial error recovery," in *Design, Automation & Test in Europe*, 2014, pp. 1–4.

[7] S. Venkatachalam and S.-B. Ko, "Design of power and area efficient approximate multipliers," *IEEE Trans. Very Large Scale Integration Systems*, vol. 25, no. 5, pp. 1782–1786, 2017.

[8] A. Momeni, J. Han, P. Montuschi, and F. Lombardi, "Design and analysis of approximate compressors for multiplication," *IEEE Trans. Computers*, vol. 64, no. 4, pp. 984–994, 2015.

[9] I. Qiqieh, R. Shafik, G. Tarawneh, D. Sokolov, and A. Yakovlev, "Energy-efficient approximate multiplier design using bit significance-driven logic compression," in *Design, Automation & Test in Europe*, 2017, pp. 7–12.

[10] J. Liang, J. Han, and F. Lombardi, "New metrics for the reliability of approximate and probabilistic adders," *IEEE Trans. Computers*, vol. 62, no. 9, pp. 1760–1771, 2013.