A Tag Based Random Order Vector Reduction Circuit

YIHUA HUANG, WENJIN HUANG, RUI CHEN, HUANGTAO WU, AND MING WEI
School of Electronics and Information Technology, Sun Yat-sen University, Guangzhou 510006, China
Corresponding author: Wenjin Huang (hwenjin@mail2.sysu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61771494, and in part by the Science and Technology Planning Project of Guangdong Province under Grant 2016A010104007.

ABSTRACT Vector reduction is a very common operation to reduce a vector into a single scalar value in many scientific and engineering application scenarios. Therefore a fast and efficient vector reduction circuit has great significance to the real-time system applications. Usually the pipeline structure is widely adopted to increase the throughput of the vector reduction circuit and achieve maximum efficiency. In this paper, to deal with multiple vectors of variable length in random input sequence, a novel tag based fully pipelined vector reduction circuit is firstly proposed, in which a cache state module is used to queer and update the cache state of each vector. However, when the quantity of the input vector becomes large, a larger cache state module is required, which consumes more combinational logic and lower the operating frequency. To solve this problem, a high speed circuit is proposed in which the input vectors will be divided into several groups and sent into the dedicated cache state circuits, which can improve the operating frequency. Compared with other existing work, the prototype circuit and the improved circuit based on the prototype circuit can achieve the smallest Slices×us (<80% of the state-of-the-art work) for different input vector lengths. Moreover, both circuits can provide simple and efficient interface whose access timing is similar to that of a RAM. Therefore the circuits can be applied in a greater range.

INDEX TERMS Field-programmable gate arrays, fully pipelined, vector reduction.

I. INTRODUCTION
Vector reduction is a common operation used to reduce a vector into a single scalar value, which exists in many scientific and engineering application scenarios including the inference of convolution neural network, video coding and decoding, etc. Perhaps the most common example is to calculate the accumulation of a vector’s elements, which is the critical step in performing inner product operations in many matrix computations [1]. Other vector reduction operations include vector chain product and searching for a maximum or a minimum element. The vector reduction can be characterized by multiple-step computations. Usually, the later steps need the computed results of the former steps.

In some real-time embedded system applications, the vector reduction operation needs to be implemented in hardware, such as FPGA or ASIC [2], [3]. So the construction of the vector reduction circuit is closely related to the latency of the adopted hard-wired operator. If the latency of the operator is only 1 clock cycle, the operator itself is a reduction circuit. But for most of the complex operations, such as double-precision floating-point additions and multiplications, a deep pipeline structure has to be adopted in the hard-wired operator to achieve high clock rate, the latency will be generally greater than one clock cycle [4]. In this case, the implementation of the reduction circuit will be quite complicated, e.g., using an accumulator with p-stage pipeline to calculate the vector summation directly, then each new element of the vector has to wait p-1 clocks before entering the accumulator [5]. To achieve high throughput, the design of the hardware operator with multiple pipeline stages has to be carried out very carefully, which includes the arrangement of each step of the operation and the storage and dispatch of the intermediate results, so that the pipeline in the operator can be fully utilized. Further, in order to improve the hardware resource utilization ratio and to obtain high performance, the vector reduction circuit should consume fewer hardware operators and meanwhile can concurrently process multiple independent vectors.
Vector reduction circuit has been studied for more than two decades. Kogge [6] proposed a divide-by-half method to deal with a fixed-length vector reduction. The method was modified by Ni and Hwang to support the reduction for the vector of variable length [7]. However, the circuits are not suitable to deal with multiple vectors because of the data conflicts from different vectors. To solve the data conflicts, the blocked MA method [8], fully compacted binary tree, dual strides adder [9], and several other methods [10]–[12] were proposed. But all the aforementioned methods required that the vectors should be inputted in order. Obviously, this requirement limits the applications of the circuit, such as Network on Chip in which all the elements of different vectors are mixed together and are disordered. Therefore, in [13], we proposed a novel vector reduction circuit which can deal with multiple vectors of variable lengths inputted in random order. The experimental results have shown that the proposed circuit can reach the highest operating frequency with the least area (slices) consumptions compared with other work, but how to design the circuit in detail and how to evaluate its latency and storage requirement in theory remain unknown. So in this paper, the detailed design process is presented and the related analysis theory is established and verified.

The key contributions are as follows.

(1) Propose a novel vector reduction circuit to handle multiple vectors of variable length inputted in random order. Compared with other work, the circuit has the smallest Slices×us (<80% of the state-of-the-art work).

(2) Based on the novel circuit, an improved circuit is also proposed, which can achieve higher operating frequency than the novel circuit.

(3) The key performance of both circuits including the least depth of the required buffer and the maximum clock cycles required for the output process is analyzed.

This paper has been arranged as follows: Section II shows the background and the related work. The novel tag-based circuit is shown in Section III. The detailed design and implementation process of the novel tag-based circuit is shown in Section IV. The detailed design process of the improved circuit for high speed is also presented in Section IV. In Section V, the detailed theoretical analysis process is presented, the storage space and the latency of the proposed circuit are derived, respectively. The hardware consumptions of the proposed circuits are compared with that of other work. In Section VI, we conclude the paper.

II. BACKGROUND AND RELATED WORK

Since several decades ago, the vector reduction problem has been studied. Kogge in [6] proposed the concept of the divide-by-half, the d elements of a vector are divided into two halves, d/2 pairs of elements are pushed into the first operator to obtain the first group of the intermediate results, and then the intermediate results are again split into two halves, d/4 elements each, and pushed into the second operator to obtain the second group of intermediate results. Following the similar way and after \( \log_2 d \) steps, the final scalar result can be obtained. It is easy to see that \( \log_2 d \) operators are needed in this method. Obviously, such kind of vector reduction circuit can’t handle variable length vector. Then, Ni and Hwang in [7] proposed the symmetric method (SM) and the asymmetric method (AM) where one operator is required. Assume that the length of the vector is \( d \) and the pipeline stage number of the operator is \( p \), then the reduction latency, the number of clock cycles between the last input and the completion of reduction, of the SM is

\[
T_{SM}(d, p) = \begin{cases} 
\left\lceil \frac{\log_2 d}{2} \right\rceil + \left\lceil \frac{\log_2 p}{2} \right\rceil + 1 & d < p \\
\left\lceil \frac{\log_2 d}{2} \right\rceil + \left\lceil \frac{\log_2 p}{2} \right\rceil + 2 & d \geq p 
\end{cases}
\]

The reduction latency of the AM is

\[
T_{AM}(d, p) = \begin{cases} 
\left\lceil \frac{\log_2 d}{2} \right\rceil + \left\lceil \frac{\log_2 p}{2} \right\rceil + 1 & d < p \\
\left\lceil \frac{\log_2 d}{2} \right\rceil + 2 & d \geq p 
\end{cases}
\]

where, the ceiling function “\( \lceil \cdot \rceil \)” is used to denote the nearest integers to the real number from above. For the reduction circuit with only one operator, when the length of the vector is long enough, the reduction time will not less than that of the AM circuit. Such a law is determined by the essence of the reduction operation carried out by the operator with multiple stage pipelines.

Some work focused on processing short vectors with high efficiency. In [14], Sips and Lin proposed the modified symmetric (MS) method and the modified asymmetric (MA) method, in which the input feeding phase are overlapped with the merging phases to achieve lower latency for short vectors. The reduction methods in [7] and [14] are appropriate to handle single input vector. However, the reduction methods cannot handle multiple input vectors efficiently because these methods need dummy stages to be inserted into the pipeline structure. Besides, these methods have to satisfy the assumption that all elements from different vectors are already stored in memory and can be retrieved in an interleaving order. But in real applications, this assumption can’t be easily satisfied, especially for a large number of vectors. Therefore, in [8] the blocked MA method (MAB) was proposed to lower the requirement of the memory. In this method, the number of the sets processed within a block (a batch) is limited. In addition, the block size of the MAB is determined by the pipeline length. However, such buffer size is still related to the length of the longest set in a block, which limits the application of the MAB method.

Some researchers tried to enhance the performance of the reduction circuit through designing special operator. In [15], a self-alignment technique has been developed to improve the performance of floating-point accumulation. And then the technique is modified to implement a single-precision floating-point multiply accumulator [15]. However, the accumulator needs to stall internally to handle the overflow under the control of complicated control logic because the
self-alignment technique is not suitable for the fully pipelined structure.

He et al. [16] focused on the correctness and accuracy of these techniques. In order to design an accurate floating-point accumulator, they proposed a group alignment algorithm. However, the algorithm still has the same disadvantage as that of the self-alignment technique, i.e., a pipeline stall signal has to be inserted between the reduction processes of two consecutive vectors. In order to simplify the control logic, Nagar and Bakos [17] proposed a method to integrate a coalescing reduction circuit within the low-level design of a base-converting floating-point adder. However, the method requires a minimum set size and is only feasible on some specific types of FPGA, which limited the application of the method.

In [9], Zhou proposed three architectures including fully compacted binary tree (FCBT), dual striped adder (DSA), and single striped adder (SSA). In FCBT, the maximum size of the input elements must be known in advance, limiting the application of FCBT. In both the DSA and the SSA structure, the results are out-of-order and not easy to be used in hardware design when handling the vectors of variable sizes. In [10], the FCBT was extended to support multiple inputs per cycle and reduce the area required to identify the end of a group. The DSA was also modified to address the out-of-order output problem and solve the stalling problems that were neither considered nor documented in [9]. Specifically, they proposed a floating-Point accumulator based on the FCBT and DSA reduction circuit. The accumulator included the reduction circuits and an adder tree, which was introduced to eliminate out-of-order outputs and reduce buffering requirements of the reduction circuits. Wayne [11] proposed an open-source library for Dataflow acceleration on FPGAs, in which the partially compacted binary reduction tree was introduced. And the state machine was used to enable the PCBT to stall but preserve the intermediate results if necessary.

A delayed buffering (DB) method, which only requires one adder and O(p) storage size, is proposed in [12]. The method can achieve a better performance when handling the vectors with certain range of size. Huang and Andrews in [18] presented a flexible module architecture to design the reduction circuit. By using p pairs of FIFO and 2 operators, their design can achieve the same performance when a chain of p operators is used. Almabrok implemented the Big Bang-Big Crunch optimization algorithm on FPGA, in which the reduction circuit [18] was utilized to replace the full binary tree for fewer hardware resources [19].

In fact, the basic architectures of the reduction circuits [10], [11], [18] were not modified, but the additional circuits were introduced to improve the functionality of the reduction circuit.

All the proposed architectures in [9], [12] and [18] except the FCBT are fully pipelined and can deal with multiple vectors of variable length, but they are all limited by the precondition that the input vectors must be pushed into the reduction circuit in turn. Therefore, these methods are not suitable to deal with the statistic-based algorithms of Network on Chip, in which all the elements of different vectors are mixed together and are disordered. So stalling the input data and using caches to realign the statistic data sequentially are the necessary operations, resulting in extra hardware and time consumptions. Obviously, to deal with multiple independent vectors of variable lengths input in random order simultaneously, a novel vector reduction circuit is required.

III. DESIGN IDEA

In the vector reduction circuit, a special control logic is required to handle those multiple independent vectors simultaneously. Specifically, the logic is related to the pipeline stage number of the binary operator and is used for dispatching the intermediate results exactly. Generally, the dispatch process is very complicated when pushing the elements of the multiple vectors into the circuit in arbitrary order. Hence, a tag-based approach is proposed to ease the complexity of the control logic.

Generally the vector reduction circuit is used to carry out the computations that satisfy the communicative and associative law, such as the multiplication, addition, maximum, and minimum. In this paper, a data set which includes the data of a vector and the computational intermediate results related to the vector is defined, so multiple vectors can be processed concurrently in the vector reduction circuit as follows: design a container to store the current input data and the current output from the operator at every clock cycle, then the two data from the same vector are retrieved from the container and form a data pair, which are sent to the operator. The vector reduction is completed when satisfying the following three conditions: (1) the data of all the vectors have already been fed into the vector reduction circuit. (2) Any two data in the container cannot form a data pair. (3) The pipeline registers of the operator are empty. When the vector reduction is completed, all the results are stored in the container. Obviously, the input order of the vectors is of no importance in this method. To determine which vector the data in the container belongs to is the essential part of the method. Therefore, by using a unique tag for each specific vector, it is easy to tell which vector the data belongs to. The tag can be the signal that the data source uses to distinguish the different vectors. Because only the operands with the same tag can be sent to the operator, the intermediate results generated by the operator are also attached with the same tag as that of the operands. Hence, both the data of the same vector and the intermediate results calculated from these data have the same tag. In addition, an invalid tag indicating the invalid data is also introduced in this paper. For instance, the outputs generated by the operator in the first few initial clock cycles are invalid. With this invalid tag, we can ensure that all the invalid data will not be stored in the container so that the function of the circuit will not be affected. Based on the preceding description, the architecture of the tag-based vector reduction circuit, which includes the Container,
the Buffer, the MUX (multiplexer) and the Operator, is shown in Fig. 1.

In Fig. 1, all the input and output data of each module are tagged data or tagged data pair. In every clock cycle, the Container receives two different tagged data. One is the input data from the external data source and the other is the output of the Operator. Meanwhile, the Container module compares all the tags of the data, pairs any two data with the same tag, and output the data pairs. Therefore, all the data in the Container have different tags inevitably, which are attached by the data sources. It means that the Container only requires no more than m storage size when processing m different vectors. The authors in [9] declared that the O(m) storage complexity is unacceptable. However, here the O(m) storage complexity of the Container is determined by the requirement of the application scenarios in which the vectors are inputted in arbitrary sequence. As a matter of fact, if the internal storage size of the reduction circuit is less than m, there is no way to deal with m vectors input in random order simultaneously without stalling the input data or using the external memory. More specifically, assume that the internal storage size of the vector reduction circuit is n (n<m), the number of the states of the circuit is finite no matter what kind of control strategy is used. So there will be a moment that all the storage space of n size is totally filled by the input data and the outputs from the Operators. In that case, when the element of the vector that is not stored in the Container is fed into the Container, the data overflow will happen. Therefore, to process m vectors in arbitrary sequence, at least O(m) storage complexity is required. If the vectors are inputted sequentially into the vector reduction circuit, regardless of how many vectors are waiting for processing, the number of the data pairs under processing cannot exceed the total number of the pipeline stage p. In this situation, the storage size of the Container in the vector reduction circuit is equal to p.

Obviously, the number of the tagged data pairs generated by the Container can only be 0, 1 or 2 based on the function analysis of the Container. Based on the different number of the tagged data pairs, the Buffer will take different actions to make sure that the Container can output continuously. When the Container only generates one tagged data pair, this data pair will be transmitted to the Operator directly through the MUX, or be pushed into the Buffer which will pop up another tagged data pair to the Operator through the MUX. When two tagged data pairs are generated, one will be transmitted to the Operator through the MUX, and the other will be fed into the Buffer. When no valid data pair is generated, the Buffer will pop up one valid data pair and send it to the Operator through the MUX in the meanwhile. In addition, if the Buffer has no valid tagged data pair to pop up, it will output the data pair with the invalid tag. Such the invalid tagged data pair will still be transmitted to the Operator through the MUX to ensure that the pipeline of the Operator keeps running. In addition, the invalid data pair through the Operator will never be stored in the Container. In section V, we will discuss the least depth of the Buffer and prove that the least depth is min {m/2, p-1}. Only when the depth of the Buffer is not less than min{m/2, p-1}, the overflow will not happen in the Buffer for the vectors of any length or the elements arriving in arbitrary order, which ensures the reliability of the proposed vector reduction circuit.

The Operator module includes a binary operator and a tag delay. The Operator will only receive one tagged data pair from the MUX in each clock cycle. And then the input data pair will be split into two parts, the tag and the data (two operands). The tag will be pushed into the tag delay and the data pair will be pushed into the binary operator in the meanwhile. To ensure that the output from the binary operator has the same tag as that of the two operands, the tag delay should have the same latency as that of the operator, which is equal to the pipeline stage number of the operator. The output of the binary operator and the delayed tag will constitute the tagged data which will be pushed back to the Container.

As aforementioned, the Container and Operator will never stop running when the size of the Buffer is not less than the least depth min{m/2, p-1}. Furthermore, the MUX itself cannot affect the pipeline. Therefore, we can have the conclusion that our hardware design is fully pipelined.

IV. DETAILED DESIGN
A. PROTOTYPE CIRCUIT DESIGN

Fig.2 depicts the architecture of the prototype circuit. It contains the detailed design about the Container module, the Buffer module, the MUX module, and the Operator module. The tagged data is the input from three ports, i.e., stb_x, tag_x, and dat_x, while the final reduction result is the output from the dat_o port. And then the tag of the final reduction result will be released under the control of the ctl_read signal. The unique tag \{stb_*, tag_*\} (where * is the wildcard character) is attached to the data or the data pair. In the tag, the stb_* is the most significant bit of the tag, which is used to represent the validity of the tagged data or the tagged data pair, 1 for valid data and 0 for invalid data. And the tag_* signal represents the value of the tag with the bit-width w, whose value range is [0, m-1].

In the proposed prototype circuit, no matter what kind of the binary operator it is, it can be applied as long as it
meets the commutative and associative law, and there are no constraints on the pipeline stage number \( p_o \). In this paper, D flip-flops are used to implement all delayers including the tag layer in the Operator.

The Container consists of three parts, i.e., the caching module Cache, the cache state queering and updating module CacheStatQAU_PT, and the tag comparison circuit (the box with an equal sign inside). The pipeline stage number \( p_c = \max\{p_c-c, p_{c-pt}\} \), where \( p_{c-c} \) is the reading latency of the Cache and \( p_{c-pt} \) is the latency of the CacheStatQAU_PT. The Container will receive two tagged data, one is the element \( \{\text{stb}_x, \text{tag}_x, \text{dat}_x\} \) from external multiple vectors and the other is the output \( \{\text{stb}_r, \text{tag}_r, \text{dat}_r\} \) from the Operator, in every clock cycle. Because in the Container any two data with the same tag will form a data pair and be outputted immediately, it means that there will be three different kinds of data pairs: First, when \( \text{tag}_x = \text{tag}_r \), the data from external multiple vectors and the output data from the Operator will form a data pair, represented as \( \{\text{stb}_x, \text{tag}_x, \text{dat}_x, \text{dat}_x\} \). Second, when \( \text{tag}_x \neq \text{tag}_r \) and there exists a data with the tag \( \text{tag}_x \) in the Cache, the data from external multiple vectors and the data from the Cache will form a data pair, represented as \( \{\text{stb}_x, \text{tag}_x, \text{dat}_x, \text{dat}_x\} \). Third, when \( \text{tag}_x \neq \text{tag}_r \) and there exists a data with the tag \( \text{tag}_r \) in the Cache, the output data from the Operator and the data from the Cache, represented as \( \{\text{stb}_r, \text{tag}_r, \text{dat}_r, \text{dat}_r\} \). The strobe signals \( \text{stb}_{xr}, \text{stb}_{xc}, \text{and stb}_{rc} \) are respectively used to indicate the validity of these three possible kinds of data pair. The MUX and the Buffer will take different operations according to the strobe signals. Later we will show how the MUX and Buffer work.

The strobe signal \( \text{stb}_{xr} \) is generated by the tag comparison circuit. When \( \{\text{stb}_x, \text{tag}_x\} = \{\text{stb}_r, \text{tag}_r\} \) and \( \text{stb}_x=1 \), the data \( \text{dat}_x \) and the data \( \text{dat}_r \) belong to the same vector and can be paired up. In this case, the \( \text{stb}_{xr} \) equals to 1. When the strobe signal \( \text{stb}_{xr} = 1 \), the tagged data pair is a valid data pair, otherwise the data pair is invalid.

The strobe signals \( \text{stb}_{xc} \) and \( \text{stb}_{rc} \) are generated by the CacheStatQAU_PT module. After receiving the signals \( \{\text{stb}_x, \text{tag}_x\} \) and \( \{\text{stb}_r, \text{tag}_r\} \), this module will query and update the data stored in the Cache according to the addresses \( \text{tag}_x \) and \( \text{tag}_r \). Obviously the storage state of the corresponding data in the Cache can be indicated by a 1-bit data. When the 1-bit data is 1, the corresponding data exists in the Cache, otherwise it does not exist. Therefore, a m-bit data \( \text{cache\_stat} \), which is stored in the m-bit register of the CacheStatQAU_PT module, is introduced to record the storage state of the m data. By using the cache_stat, this module can generate the strobe signal \( \text{stb}_{xc} \) and \( \text{stb}_{rc} \) for the tagged data pair \( \{\text{stb}_x, \text{tag}_x, \text{dat}_x, \text{dat}_x\} \) and \( \{\text{stb}_r, \text{tag}_r, \text{dat}_r, \text{dat}_r\} \). The Algorithm 1 describes the specific behavior of the CacheStatQAU_PT module.

Where ‘\( \wedge \)’ represents the logic exclusive operator. The cache_stat is updated every clock cycle. Because the signals \( \text{stb}_{xc} \) and \( \text{stb}_{rc} \) are latched before they are outputted, the latency \( p_{c-pt} \) of the CacheStatQAU_PT is 1 clock cycle.

A dual port read first RAM of size \( m \) is used to implement the Cache module and the reading latency is \( p_{c-c} \). The aforementioned \( m \) is the maximum number of the vectors that the vector reduction circuit can handle simultaneously. Take the tag signal \( \text{tag}_x \) and \( \text{tag}_r \) as address. The data of the corresponding vector can be retrieved from the Cache. Based on the address \( \text{tag}_x \), the Cache will fetch the \( \text{dat}_x\) to pair with the \( \text{dat}_x\) in every clock cycle and then the CacheStatQAU module will check the validity of this data

---

Algorithm 1 Behavior of the CacheStatQAU_PT Module

1: //Read \( \text{stb}_x, \text{tag}_x, \text{stb}_r, \text{tag}_r \) signals from external vectors and the output of the operator, respectively:
2: //Input \( \text{stb}_x, \text{tag}_x, \text{stb}_r, \text{tag}_r \)
3: //Queer the cache state to find whether the data with tag \( \text{tag}_x \) and \( \text{tag}_r \) exists in the Container or not. And then generate and output the \( \text{stb}_{xc} \) and \( \text{stb}_{rc} \) signal
4: \( \text{stb}_{xc} = \text{cache\_stat}[\text{tag}_x] \& \text{stb}_x \)
5: \( \text{stb}_{rc} = \text{cache\_stat}[\text{tag}_r] \& \text{stb}_r \)
6: //If \( \text{stb}_{xc}=1 \), the container includes the data with tag \( \text{tag}_x \), otherwise not include. If \( \text{stb}_{rc}=1 \), the container includes the data with tag \( \text{tag}_r \), otherwise not include.
7: //Update the cache_stat
8: \( \text{cache\_stat}[\text{tag}_x] = \text{cache\_stat}[\text{tag}_x] \wedge \text{stb}_x \)
9: \( \text{cache\_stat}[\text{tag}_r] = \text{cache\_stat}[\text{tag}_r] \wedge \text{stb}_r \)
pair and generate the strobe signal stb_xc. When the strobe signal stb_x = 1 and the \{stb_x, tag_x\} ≠ \{stb_r, tag_r\}, the Cache will store the dat_x in the storage unit whose address is tag_x. Because the RAM works in the read first mode, the data_x_cache will be read from the tag_x memory unit before writing the data_x into the same memory. Similarly, these operations can also be taken for the data dat_r. To avoid the data hazard, for example, handling a data repetitively or erasing a new valid data before reading, the Cache will always work in the Read First mode and meanwhile the CacheStatQAU_PT module will record the state of the input tagged data.

The design of the Buffer module is based on a FIFO which works in the First-Word-Fall-Through mode, and the MUX is a 3-to-1 logic. The Algorithm 2 describes the specific behaviors of the Buffer and the MUX.

From the Algorithm 2 and Fig. 2, it can be known that when the input tagged data \{stb_x, tag_x, dat_x\} and the intermediate result \{stb_r, tag_r, dat_r\} share the same valid tag, they will be paired together to constitute the new valid data pair \{stb_x, tag_x, dat_x, dat_r\}. In the case that the data with the same tag has been stored in the Cache, one can see that the tagged data pairs \{stb_xc, tag_x, dat_x, dat_x_cache\} and \{stb_rc, tag_r, dat_r, dat_r_cache\} will also be indicated as valid based on the Algorithm 1 of the CacheStatQAU_PT module. In this case, the MUX seems to receive three valid data pairs concurrently. But actually only one indeed valid tagged data pair \{stb_xr, tag_x, dat_x, dat_r\} will be sent to the MUX, and the other two data pairs will be ignored and discarded. And as aforementioned, the dat_x and the dat_r will not be written into the Cache therefore to overwrite the original data. The corresponding bit of the storage state register in the CacheStatQAU_PT will be inverted twice, so its value will not be changed. Further, the Buffer will take no action. Therefore, by all these designs, the operator can get the right data pair to avoid data conflict and error.

In Fig. 3, a simple example is given to illustrate the work process of the Container, Buffer, and Adder. In this example, the length of the vector is \(d = 4\), and the quantity of vectors is \(m = 3\). The three vectors are \(\{V11, V12, V13, V14\}\), \(\{V21, V22, V23, V24\}\), \(\{V31, V32, V33, V34\}\), respectively. Each vector has a unique tag and the data of a vector share the same tag. The pipeline depth of the Adder is 3. According to the above analysis, the capacity of the Container should be 3, and the depth of the Buffer should be 1. At each clock, a datum from the vectors is sent into the circuit. And the work process of the Adder and Buffer is described in Algorithm 2. The states of the stb_xr, stb xc, stb rc and cache_stat signals are described in Algorithm 1.

If the signal \(ctl_read = 1\), it means that the reduction result of a specific vector should be outputted and the CacheStatQAU_PT module should update the corresponding bit of the cache_stat to release the tag of this specific vector. If the signal \(ctl_read = 0\), it means that the reduction result is not ready or needed. Through the calculation of the Operator, the vector is eventually reduced into a single scalar value, which is the only data related to this vector, and stored in the Container. In the cache_stat, the bit of this vector must be 1. By sending the valid tag \{stb_x, tag_x\} of the vector and setting the signal \(ctl_read\) high, the final reduction result of the vector can be obtained, which is the data dat_x_cache. And then the dat_x_cache will be sent to the port dat_o and the corresponding bit in the cache_stat will be set to 0. While the \(ctl_read = 1\), the MUX will stop sending the

**Algorithm 2 Behaviors of the Buffer and the MUX**

1: switch (stb_xr, stb_xc&(!ctl_read), stb_rc)
2: case 1xx:
3: //’x’ means ‘don’t care’
4: //when \{stb_xr, tag_x, dat_x, dat_r\} is the only valid tagged data pair.
5: \{stb_xr, tag_x, dat_x, dat_r\} is selected by MUX and sent to the Operator
6: Buffer will do nothing;
7: case 011:
8: //Both \{stb_xc, tag_x, dat_x, dat_x_cache\} and \{stb_rc, tag_r, dat_r, dat_r_cache\} are valid tagged data pairs.
9: The tagged data pair \{stb_xc, tag_x, dat_x, dat_x_cache\} is selected by the MUX and sent to the operator;
10: The tagged data pair \{stb_rc, tag_r, dat_r, dat_r_cache\} is pushed into the Buffer;
11: case 010:
12: // \{stb_xc, tag_x, dat_x, dat_x_cache\} is the only valid tagged data pair.
13: The tagged data pair \{stb_xc, tag_x, dat_x, dat_x_cache\} is selected by the MUX and sent to the operator;
14: Buffer will do nothing;
15: case 001:
16: // \{stb_rc, tag_r, dat_r, dat_r_cache\} is the only valid tagged data pair.
17: The tagged data pair \{stb_rc, tag_r, dat_r, dat_r_cache\} is pushed into the Buffer;
18: //The tagged data pair previously stored in the Buffer will be sent to the operator through the MUX.
19: The tagged data pair appears on the output port of the Buffer is selected by the MUX and sent to the operator;
20: Buffer pop up a tagged data pair;
21: case 000:
22: The tagged data pair appears on the output port of the Buffer is selected by the MUX and sent to the operator;
23: Buffer pop up a tagged data pair;
24: // If there is no tagged data pair in the Buffer, an invalid tagged data pair will be pop up.
25: end switch
selected data pair \{stb_xc, tag_x, dat_x, dat_x_cache\} to the Operator. Hence, the tag of the vector that has finished the vector reduction process will be released to ensure the normal operation of the other vectors.

Based on the foregoing analysis, it is obvious that the proposed circuit is fully pipelined, which means that the reduction circuit will never stop. In addition, the proposed vector reduction circuit is quite similar to a RAM of \(m\) size. Because the tag_x works like the address port, the dat_x works like the input data port, the dat_o works like the output data port, the \{ctl_read, stb_x\} works like the read/write enable port, and the reading latency is equal to the Container pipeline stage number \(p_{c}\). The total pipeline length \(p\) of the proposed circuit is \(p = p_{c} + p_{o}\).

In every clock, only one bit of the \(m\)-bit cache_stat is operated by the CacheStatQAU_PT module. And the operation is carried out by using the multiplexer and demultiplexer. Unfortunately, the delays of the multiplexer and demultiplexer in the CacheStatQAU_PT module are proportional to \(m\). Therefore, when the \(m\) becomes larger, the delays of the multiplexer and demultiplexer will also become unacceptable. In addition, for some signals including the stb_x and the stb_r, their fan-outs are also proportional to \(m\). So when the \(m\) becomes too large, the fan-out of these signals will become so large that extra driving buffers are required. However, the introduced driving buffers will result in larger path delay. These two problems will lower the operating frequency if \(m\) becomes too large in the proposed circuit. So it is necessary to propose a high speed circuit design that can operate with high clock rate if \(m\) becomes large.

### B. HIGH SPEED CIRCUIT DESIGN

The divide-and-conquer method is introduced to design the high speed circuit as follows: \(m\) tags are split into \(M\) domains and each domain has \(m/M\) tags. \(M\) CacheStatQAU_PT modules are designed to implement the querying and updating function for the tags in each domain, respectively. Obviously, choosing a suitable \(M\) can minimize the scale of each CacheStatQAU_PT module to avoid the large logic delay and path delay in the large scale circuit. Fig 4 shows the detailed design of the cache storage state querying and updating module in the proposed high speed circuit, which called CacheStatQAU_HS.

The \(w\)-width tag signal tag_* can be divided into two parts: the domain identifier tag_*_id and the domain tag signal tag_*_sub. The tag_*_id is the high \(w_{id}\) bits of the tag_* while the tag_*_sub is the rest low \(w_{sub}\) bits, which satisfies \(w_{sub} = w - w_{id}\). In this case, \(m = 2^{w}\) tags are divided into \(M = 2^{w_{id}}\) domains and then the tags in each domain are processed by the corresponding CacheStatQAU_PT module. In every domain, based on the signals \{stb_x, tag_x_id\} and \{stb_r, tag_r_id\}, two domain identifier comparison circuits are designed to check the validity of the input signals \{tag_x, tag_r\} and find which domain the signals belong to. These outputs of the comparison circuits will be sent to the stb_x port and the stb_r port of the CacheStatQAU_PT module as the strobe signal for the tag_x_sub and tag_r_sub, respectively. In each domain, the CacheStatQAU_PT module has the same function as that of the CacheStatQAU_PT module in the prototype circuit. Specifically, for a certain input tag signal, the corresponding

### FIGURE 3. Snapshot of the prototype circuit.
small CacheStatQAU_PT module will work and meanwhile the other small CacheStatQAU_PT module will be left idle. In the CacheStatQAU_HS, the output signals stb_xc and stb_rc are obtained through implementing the logic OR calculation of the output signals stb_xc and stb_rc from M small CacheStatQAU_PT modules, respectively.

After the signal \{stb_*, tag_*\} is fed into the CacheStatQAU_HS module, it will be fanned out to the M domains by a FanoutTree module. Implemented with D flip-flops, the node of the Fanout Tree module latches the signal from its parent node and fans out this signal to no more than \(d_f\) child nodes. It is obvious that the \(d_f\) is the degree of the fan-out tree. Because the excessively high fan-out will drag down the operating frequency, the value of \(d_f\) should be determined based on the driving capacity of the related logic gates. The pipeline stage number \(p_c\) of the FanoutTree is equal to the level number of the tree, which means \(p_c \propto \log_2(N) \propto \Theta(\log(m))\).

In the proposed circuit, there are \(N\) pairs of the stb_xc and stb_rc signals which are processed by the Boolean operations. The when the M is huge, the Boolean operations will have to process a large number of input signals and require a large scale combinational circuit in the hardware design, which results in large logic delay. To avoid the large delay, the Boolean operations need to be implemented with pipelined tree-shaped circuits including the CmpTree and OrTree. The latencies of both trees are \(p_{c_{-{}ct}} \propto [\log_{d_f}(w_{id})] \propto \Theta(\log(\log(m)))\) and \(p_{c_{or}} \propto [\log_{d_f}(N)] \propto \Theta(\log(m))\), respectively.

In the high speed circuit, the CacheStatQAU_HS module has the same structure as that of the aforementioned CacheStatQAU_PT except the latency \(p_{c_{-{}hs}} = p_{c_{-{}f}} + p_{c_{-{}ct}} + p_{c_{-{}or}} + p_{c_{-or}} \propto \Theta(\log(m))\).

Fig.5 shows the structure of the Container in the high speed circuit. In the Container, the comparison of the domain identifiers \{stb_x, tag_x\} and \{stb_r, tag_r\} is a multi-input and single-output logic calculation, which is implemented with the CmpTree_2i module. Specifically, this module is of a pipelined and tree-shaped structure with the degree \(d_{ct}2\).

Moreover, in order to align to the comparison result, the input signals of the Cache including stb_x, tag_x, dat_x, stb_r, tag_r, and dat_r have to be delayed \(p_{c_{-{}ct}2}\) clock cycles through their respective delayers. The delay \(p_{c_{-{}ct}2}\) satisfies that \(p_{c_{-{}ct}2} \propto [\log_{d_{ct}2}(w_{id})] \propto \Theta(\log(\log(m)))\).

We can conclude that the pipeline stage number of the Container is \(p_c = \max\{p_{c_{-{}ct}2} + p_{c_{-{}ct}} + p_{c_{-{}hs}}\} \propto \Theta(\log(m))\) and the total pipeline length is still \(p = p_c + p_o\) in the high speed circuit. If \(M = 1\), the proposed high speed circuit can be regarded as the prototype circuit. For the high speed circuit, the improvement of the performance is at the cost of the extending pipeline length with \(\Theta(\log(m))\).

V. EVALUATION AND DISCUSSION
A. CHARACTERISTICS ANALYSIS AND COMPARISON

The storage modules in the proposed reduction circuit are the Container and the Buffer. As aforementioned, the storage size of the Container is \(m\), and \(m\) is equal to the number of vector that the reduction circuit can simultaneously process. Therefore, this section will mainly focus on the analysis of the least depth of the Buffer to avoid overflow when multiple vectors are inputted in arbitrary sequence. The analysis of the latency of the reduction circuit will also be carried out.

In order to facilitate our analysis, in this section, the input elements and the data or data pairs existing in the circuit are collectively called “item”.

As aforementioned, pipelines exist in the Operator and Container. The pipeline length of the Operator is \(p_o\). As to the Container, it has two pipelines which are with the same length \(p_c\) and are parallel to each other. The first pipeline of the Container connects the input port of the proposed circuit with the input port of the MUX. The second pipeline of the Container connects the output port of the Operator with the Buffer. The second pipeline of the Container and the pipeline of the Operator constitute the total pipeline of the proposed circuit, so the total pipeline length of the proposed circuit is \(p = p_c + p_o\). In this section, if we take the first stage of the pipeline of the Operator as the first stage of the total pipeline of the proposed circuit, then the input port of the MUX is the entrance of the total internal pipeline. Accordingly, the outside elements emerging on the input port of the proposed circuit will arrive in the total pipeline after \(p_c\) clock cycles.

Denote \(m\) valid tags as 1, 2, \ldots, \(m\), invalid tag as 0. Define three functions \(c_n(v), q_n(v),\) and \(l_n(v)\) to represent the number of the items with tag \(v\) in the Cache of the Container, in the Buffer, and on the pipeline, at clock \(n\), respectively. Based on the mechanism of the proposed circuit, it is known that in the Container, for each tag, the number of the associated items is not greater than 1, so \(c_n(v)\) only can be 0 or 1. The total pipeline length of the circuit is \(p\), so \(l_n(v)\) ranges from 0 to \(p\). The item with the invalid tag will never be stored by the Buffer and Container, so for the invalid tag 0, at any clock \(n\), \(c_n(v) = 0, q_n(v) = 0\). Denote the time when the circuit is reset as clock 0. After resetting, the Container and Buffer are empty, all the items on the pipeline are with the invalid tag 0,
then we have \( l_0(0) = p \) for the invalid tag 0 and \( c_0(v) = q_0(v) = l_0(v) = 0 \) for any valid tag \( v \).

Define the following sets to represent where the tag is at clock \( n \).

\[
\begin{align*}
C(n) &= \{ v \mid \text{At clock } n, \text{tag } v \text{ satisfies } c_n(v) \neq 0 \} \\
Q(n) &= \{ v \mid \text{At clock } n, \text{tag } v \text{ satisfies } q_n(v) \neq 0 \} \\
L(n) &= \{ v \mid \text{At clock } n, \text{tag } v \text{ satisfies } l_n(v) \neq 0 \}
\end{align*}
\]

Define the following disjoint sets to classify the tags.

\[
\begin{align*}
P1(n) &= \{ v \mid v \notin C(n), v \notin L(n), v \notin Q(n) \} \\
P2(n) &= \{ v \mid v \in C(n), v \notin L(n), v \notin Q(n) \} \\
P3(n) &= \{ v \mid v \notin C(n), v \in L(n), v \notin Q(n) \} \\
P4(n) &= \{ v \mid v \in C(n), v \in L(n), v \notin Q(n) \} \\
P5(n) &= \{ v \mid v \notin C(n), v \notin L(n), v \in Q(n) \} \\
P6(n) &= \{ v \mid v \in C(n), v \notin L(n), v \in Q(n) \} \\
P7(n) &= \{ v \mid v \notin C(n), v \in L(n), v \in Q(n) \} \\
P8(n) &= \{ v \mid v \in C(n), v \in L(n), v \in Q(n) \}
\end{align*}
\]

Next, we will prove that the least depth of the Buffer in the proposed circuit is equal to \( \min[m/2,p-1] \) through Lemma 1 and the reduction latency is not greater than \( T_{AM}(p,p) + [p/2] \) through Lemma 2.

**Lemma 1:** The least depth of the Buffer is equal to \( \min[m/2,p-1] \). Whatever the length and the input sequence of the vectors are, as soon as the depth of the Buffer is not less than \( \min[m/2,p-1] \), the Buffer will never overflow.

**Proof:**

Proving Lemma 1 is equivalent to proving that the following inequality is valid at any clock \( n \).

\[
\sum_{v \in Q(n)} q_n(v) \leq \min\left\{ \left\lfloor \frac{m}{2} \right\rfloor, p - 1 \right\} \tag{3}
\]

First, we prove \( \sum_{v \in Q(n)} q_n(v) \leq p - 1 \).

From the definitions about the sets and functions defined in the fore part of this section, we have

\[
\begin{align*}
\forall v \in Q(n), q_n(v) &\geq 1 \\
\forall v \in L(n), l_n(v) &\geq 1 \\
Q(n) &= P5(k) \cup P6(k) \cup P7(k) \cup P8(k) \\
L(n) &= P3(k) \cup P4(k) \cup P7(k) \cup P8(k)
\end{align*}
\]

\[
\sum_{v \in L(n)} l_n(v) = p
\]

Then

\[
\begin{align*}
|v|v \in P3(n), v \neq 0 | + |P7(n)| + l_n(0) - 1 \\
\leq \sum_{v \in L(n)} l_n(v) - 1 = p - 1 \tag{4}
\end{align*}
\]

\[
|P5(n)| \leq |Q(n)| \leq \sum_{v \in Q(n)} q_n(v) \tag{5}
\]

From the Theorem 1 proved in APPENDIX, we have

\[
2 \sum_{v \in Q(n)} q_n(v) \leq |v|v \in P3(n), v \neq 0 | + |P5(n)| \\
+ |P7(n)| + l_n(0) - 1 \tag{6}
\]

From (4), (5) and (6), we have

\[
\sum_{v \in Q(n)} q_n(v) \leq p - 1 \tag{7}
\]

Second, we prove \( \sum_{v \in Q(n)} q_n(v) \leq |m/2| 

When the Buffer is empty at any clock \( n \),

\[
\sum_{v \in Q(n)} q_n(v) = 0 \leq \left\lfloor \frac{m}{2} \right\rfloor \tag{8}
\]

When the Buffer is not empty, following the similar proof process as that of the Theorem 1, we have:

If case “1xx” occurs at clock \( n \),

\[
|C(n)| = |C(n - 1)| \tag{9}
\]

If case “011” occurs at clock \( n \),

\[
|C(n)| = |C(n - 1)| - 2 \tag{10}
\]

If case “010” occurs at clock \( n \),

\[
|C(n)| \leq |C(n - 1)| \tag{11}
\]

If case “001” occurs at clock \( n \),

\[
|C(n)| \leq |C(n - 1)| \tag{12}
\]

If case “000” occurs at clock \( n \),

\[
|C(n)| \leq |C(n - 1)| + 2 \tag{13}
\]

Choose arbitrary time \( k \) when the Buffer changes from empty state to nonempty state. Accordingly suppose that after \( l \) clock cycle, the Buffer will become empty again. In the time interval \( [k, k + l] \), choose arbitrary time \( k + i, i \in [0, l] \). Based on the mechanism of the proposed circuit, we know that when the case “011” occurs, the number of the items in the Buffer increases by 1. When the case “000” occurs and the Buffer is not empty, the number of the items in the Buffer decreases by 1. In other cases, the number of the items in the Buffer remains the same. Suppose that during the clocks \( k, k + 1, \ldots, k + i \), the case “011” occurred for \( a \) times, the case “000” occurred for \( b \) times. Because during this time interval the Buffer is not empty, at clock \( k + i \) we have

\[
\sum_{v \in Q(k+i)} q_{k+i}(v) = a - b > 0 \tag{14}
\]
From (9), (10), (11), (12), (13) and (14), we have
\[ |C(k+i)| = |(C(k+i)| - |C(k+i-1)| + \cdots + |C(k+1)| - |C(k)| + |C(k)| - 2a + 2b \]
\[ = |C(k)| - 2 \sum_{v \in Q(k+i)} q_{k+i}(v) \quad (15) \]

Then
\[ 2 \sum_{v \in Q(k+i)} q_{k+i}(v) \leq |C(k)| - |C(k+i)| \leq |C(k)| \leq m \quad (16) \]

That is
\[ \sum_{v \in Q(k+i)} q_{k+i}(v) \leq \left\lfloor \frac{m}{2} \right\rfloor \quad (17) \]

Because the \( i \) is selected randomly, then at any clock \( n \) when the Buffer is not empty, \( \sum_{v \in Q(n)} q_{n}(v) \leq \left\lfloor \frac{m}{2} \right\rfloor \) is valid.

Based on (7), (8) and (17), it is easy to know that the Lemma 1 is valid.

**Lemma 2:** When the last input element appears on the input of the pipeline, all the reduction operations will be completed after \( T_{AM}(p,p) + [p/2] \) clock cycles at most.

**Proof:**

1) In the following part, we will prove that all the items stored in the Buffer will be emptied within \( p \) clock cycles after the last input element appears on the input of the buffer.

Suppose that the last input element appears on the input of the pipeline at clock \( n \). After that, only the case “001” and the case “000” will occur. When the case “000” occurs and the Buffer is not empty, the number of the items in the Buffer decreases by 1. So we only need to prove that the number of the items in the Buffer, \( \sum_{v \in Q(n)} q_{n}(v) \), is not greater than the total times that the case “000” occurs in the following \( p \) clock cycles.

At clock \( n \), the number of the items of the vector with the valid tag \( v \) on the pipeline is equal to \( c_{n}(v) + l_{n}(v) \).

In the following \( p \) clock cycles they form \( \lfloor (c_{n}(v) + l_{n}(v))/2 \rfloor \) times with the valid tag \( v \), which makes the case “001” occur for \( \lfloor (c_{n}(v) + l_{n}(v))/2 \rfloor \) times. Therefore, the total times that the case “001” occurs in the following \( p \) clock cycles is equal to \( \sum_{v \in L(n)} \lfloor (c_{n}(v) + l_{n}(v))/2 \rfloor \). In the remaining \( p - \sum_{v \in L(n)} \lfloor (c_{n}(v) + l_{n}(v))/2 \rfloor \) clock cycles, the case “000” occurs.

For any \( v \neq 0 \) and \( v \in P3(n) \cup P7(n) \), we have \( c_{n}(v) = 0 \) and \( l_{n}(v) \geq 1 \), then
\[ \lfloor (c_{n}(v) + l_{n}(v))/2 \rfloor \leq l_{n}(v) - 1 \]
where, the left side of the expression equals the right side of the expression when \( l_{n}(v) = 2 \). Both sides of the expression carry out the summation operations, yields
\[ \sum_{v \in P3(n) \cup P7(n), v \neq 0} \lfloor (c_{n}(v) + l_{n}(v))/2 \rfloor \leq \sum_{v \in P3(n) \cup P7(n), v \neq 0} l_{n}(v) \]
\[ - |\{v|v \in P3(n), v \neq 0\}| - |P7(n)| \quad (18) \]

For any \( v \in P4(n) \cup P8(n) \), we have \( c_{n}(v) = 1 \) and \( l_{n}(v) \geq 1 \), then
\[ \lfloor (c_{n}(v) + l_{n}(v))/2 \rfloor \leq l_{n}(v) \]
where the left side of the expression equals the right side of the expression when \( l_{n}(v) = 1 \). Both sides of the expression carry out the summation operations, yields
\[ \sum_{v \in P4(n) \cup P8(n), v \neq 0} \lfloor (c_{n}(v) + l_{n}(v))/2 \rfloor \leq \sum_{v \in P4(n) \cup P8(n), v \neq 0} l_{n}(v) \quad (19) \]

From the definitions about the sets and functions shown in the first part of this section, we have
\[ L(n) = P3(n) \cup P4(n) \cup P7(n) \cup P8(n) \quad (20) \]

From (18), (19) and (20), we have
\[ \sum_{v \in P(n) \cup P(n) \cup P(n) \cup P(n), v \neq 0} \lfloor (c_{n}(v) + l_{n}(v))/2 \rfloor \leq \sum_{v \in P(n) \cup P(n) \cup P(n) \cup P(n), v \neq 0} l_{n}(v) \quad (21) \]

From the Theorem 1 and (21), we have
\[ 2 \sum_{v \in Q(n)} q_{n}(v) - |P5(n)| + \sum_{v \in L(n) \cup V(n)} \lfloor (c_{n}(v) + l_{n}(v))/2 \rfloor \leq \sum_{v \in L(n)} l_{n}(v) - 1 = p - 1 < p \quad (22) \]

Because \( \sum_{v \in Q(n)} q_{n}(v) > |Q(n)| > |P5(n)| \), from (22), we have
\[ \sum_{v \in Q(n)} q_{n}(v) < p - \sum_{v \in L(n) \cup V(n)} \lfloor (c_{n}(v) + l_{n}(v))/2 \rfloor \]

So the statement 1) of Lemma 2 is true.

2) In the following content, we will prove the proposition that at the \( p \)th clock cycle after the last item is pushed into the circuit, the total number of the items for any vector will not greater than \( \lfloor p/2 \rfloor \).

Assume that the last item appears on the input of the pipeline at clock \( n \). From 1), we know that at the \( n + p \) clock cycle, the Buffer is empty, then for any vector \( v \), the number of the items of any vector in the circuit is
\[ c_{n+p}(v) + l_{n+p}(v) = q_{n}(v) + \lfloor (c_{n}(v) + l_{n}(v))/2 \rfloor \]

Based on the mechanism of the proposed circuit, we know that for any tag \( v \), when \( l_{k}(v) = p, v \notin C(k) \). So at any clock \( k \), \( c_{n}(v) + l_{n}(v) \leq p \).

If \( v \in P1(n) \cup P2(n) \cup P3(n) \cup P4(n) \), \( q_{n}(v) = 0 \). Obviously
\[ c_{n+p}(v) + l_{n+p}(v) = q_{n}(v) + \lfloor (c_{n}(v) + l_{n}(v))/2 \rfloor = \lfloor (c_{n}(v) + l_{n}(v))/2 \rfloor \leq \lfloor p/2 \rfloor \quad (23) \]
From the Theorem 1, we have
\[ 2 \sum_{v \in Q(n)} q_n(v) - |P5(n)| - |P7(n)| \leq |v|v \]
\[ \in P5(n), v \neq 0]\} + l_n(0) - 1 \quad (24) \]

Both sides of (24) add \( \sum_{v \in P7(n)} l_n(v) + \sum_{v \in P8(n)} l_n(v) \)

yields
\[ 2 \sum_{v \in Q(n)} q_n(v) - |P5(n)| - |P7(n)| + \sum_{v \in P7(n)} l_n(v) \]
\[ + \sum_{v \in P8(n)} l_n(v) \leq |v|v \in P3(n), v \neq 0\} ] + \sum_{v \in P7(n)} l_n(v) \]
\[ + \sum_{v \in P8(n)} l_n(v) + l_n(0) - 1 \leq \sum_{v \in L(n)} l_n(v) - 1 = p - 1 \quad (25) \]

From the definitions about the sets and functions shown in the fore part of this section, we have
\[ Q(n) = P5(n) \cup P6(n) \cup P7(n) \cup P8(n) \quad (26) \]
\[ |P5(n)| = \sum_{v \in P5(n)} 1 \quad (27) \]
\[ |P7(n)| = \sum_{v \in P7(n)} 1 \quad (28) \]

From (25), (26), (27) and (28), we have
\[ \sum_{v \in P5(n)} \frac{2q_n(v)}{v} + \sum_{v \in P6(n)} 2q_n(v) + \sum_{v \in P7(n)} (2q_n(v) \]
\[ + l_n(v) - 1) + \sum_{v \in P8(n)} (2q_n(v) + l_n(v)) \leq p - 1 \quad (29) \]

For \( v \in P5(n) \subseteq Q(n), q_n(v) \geq 1, \) then
\[ \forall v \in P5(n), 2q_n(v) - 1 > 0 \quad (30) \]

For \( v \in P7(n) \subseteq L(n), l_n(v) \geq 1, \) then
\[ \forall v \in P7(n), l_n(v) - 1 \geq 0 \quad (31) \]

From (29), (30) and (31), we have
\[ \forall v \in P5(n), 2q_n(v) - 1 \leq p - 1 \quad (32) \]
\[ \forall v \in P6(n), 2q_n(v) - p \leq 1 \quad (33) \]
\[ \forall v \in P7(n), 2q_n(v) + l_n(v) - 1 \leq p - 1 \quad (34) \]
\[ \forall v \in P8(n), 2q_n(v) + l_n(v) \leq p - 1 \quad (35) \]

If \( v \in P5(n) \cup P6(n), \) then \( l_n(v) = 0, \) from (32) and (33) we have
\[ c_{n+p}(v) + l_{n+p}(v) = q_n(v) + \frac{c_n(v) + l_n(v)}{2} \leq \frac{p}{2} \]
\[ = q_n(v) + [c_n(v)/2] \leq \frac{p}{2} \quad (36) \]

If \( v \in P7(n) \cup P8(n), \) from (34) and (35) obtains
\[ c_{n+p}(v) + l_{n+p}(v) = q_n(v) + [c_n(v) + l_n(v)]/2 \leq \frac{p}{2} \]
\[ \quad (37) \]

Then from (23), (36) and (37), it is clear the statement 2) of Lemma 2 is true.

3) Assume that the last item appears at the entrance of the pipeline at clock \( n. \) The case “001” and “000” will occur after clock \( n \) and the Buffer is empty at clock \( n+p. \) Therefore, from clock \( n+p, \) the valid items outputted from the Container will fall through the Buffer and enter the Operator directly, which means that the reduction processes of different vectors will not affect each other anymore.

If at clock \( n+p, \) \( h \) items with the tag \( v \) are distributed continuously from the first stage of the pipeline, then the reduction process is the same as that of the AM method. The clock number needed to obtain the final reduction result is equal to \( T_p(h) = p(\log_2 h) + 1 + h - 2^{[\log_2 h]} \).

If the same \( h \) items with tag \( v \) don’t distribute on the continuous positions from the first stage of the pipeline, it is easy to see that in this situation, the position of the only item left in the last round of combination will be closer to the terminal of the pipeline than that of the continuous distribution, or the required combination rounds are less than that of the continuous distribution. So under the condition of non-continuous distribution, the reduction time \( T' \) with tag \( v \) meets \( T' \leq T_p(h) \). In fact, the above analysis is identical to that of the AM method. More details about the derivation can be found in [7].

From the statement 2) of Lemma 2 we know that at clock \( n+p, \) if the number of the items with the tag \( v \) in the circuit is not greater than \( \lceil p/2 \rceil, \) then the number of the needed clock cycles to finish the reduction is not greater than \( T_p(\lceil p/2 \rceil) \).

Based on the proofs in 1), 2) and 3) of Lemma 2, it is easy to know that for any vector \( v, \) after the last input element is fed into the first stage of the pipeline, the latency \( T \) of the reduction satisfies
\[ T \leq p + T_p(\lceil p/2 \rceil) = p(\lceil \log_2(\lceil p/2 \rceil) \rceil + 2) + \frac{p}{2} \]
\[ - 2^{[\log_2(\lceil \phi/2 \rceil)]} \leq p(\lceil \log_2 p \rceil + 2) + \frac{p}{2} - 2^{[\log_2 p]} \]
\[ = T_{AM}(p, p) + \lceil p/2 \rceil \quad (38) \]

So Lemma 2 is true.

Table 1 shows the comparison of the performance of different methods. It is clear that the proposed circuits have less latency than the SSA, MFPA, Ae2MFPA, and AeMFPA. From Table 1, the required storage space in the proposed circuit can be evaluated as the summation of the least depth of the Buffer and the storage size of the Cache. When the

| Methods       | Operators | Latency | Storage Size |
|---------------|-----------|---------|--------------|
| PCBT [9]      | \([log_2 d]\) | \(p[log_2 d]\) | \(3[log_2 d]\) |
| FCBT [9]      | 2         | \((p - 1)[log_2 d]\) | \(3[log_2 d]\) |
| DSA [9]       | 2         | \(p[log_2 p + 1]\) | \(p[log_2 d + 1]\) |
| SSA [9]       | \(2^p\)   | \(2^p\) | \(2^p\) |
| DB [12]       | \((\log_2 p + 1)\) | \((\log_2 p + 2)\) | \(p\) |
| MFPA [18]     | \((\log_2 p + 1)\) | \((\log_2 p + 2)\) | \(p\) |
| AeMFPA [18]   | \((\log_2 p + 2)\) | \((\log_2 p + 2)\) | \(p\) |
| Ae2MFPA [18]  | \((\log_2 p + 2)\) | \((\log_2 p + 2)\) | \(p\) |
| Proposed      | \(T_{AM}(p, p) + \lceil p/2 \rceil\) | \(1 + m\) | \(1 + m\) |
TABLE 2. Experimental results comparison between different designs on Xilinx XC2VP30.

| Method* | Adders | Slice | BRAM | Freq. (MHz) | Latency |
|---------|--------|-------|------|-------------|---------|
|         | Vector Length = 32 | Vector Length = 64 | Vector Length = 128 |
|         | Time (us) | Slices × us | Time (us) | Slices × us | Time (us) | Slices × us |
| PCBT [9] | 7 | 6800 | 165 | 98 | 1.37 | 9327 | 1.37 | 9327 | 1.37 | 9327 |
| PCBT [9] | 2 | 2359 | 10 | 347 | 2.794 | 7988 | 2.794 | 7988 | 2.794 | 7988 |
| DSA [9] | 2 | 2215 | 3 | 70 | 0.718 | 1591 | 0.944 | 2090 | 1.394 | 3089 |
| SSA [9] | 1 | 1804 | 6 | 392 | 2.57 | 4636 | 2.764 | 4986 | 3.152 | 5685 |
| DB [12] | 1 | 1749 | 6 | 67 | 0.527 | 921 | 0.697 | 1219 | 1.037 | 1814 |
| MFPA [18] | 5 | 4991 | 2 | 84 | 0.56 | 2379 | 0.715 | 3568 | 1.024 | 5112 |
| AeMFPA [18] | 2 | 3130 | 14 | 84 | 0.569 | 1780 | 0.725 | 2271 | 1.039 | 3253 |
| Ae2MFPA [18] | 2 | 3737 | 2 | 84 | 0.806 | 3010 | 1.028 | 3841 | 1.472 | 5502 |
| Prototype | 1 | 1509 | 6 | 95 | 0.479 | 723 | 0.6 | 905 | 0.842 | 1270 |
| High speed | 1 | 1485 | 6 | 95 | 0.477 | 709 | 0.598 | 888 | 0.838 | 1245 |

* The PCBT and the FCBT are implemented with n = 128. Neither can handle the vectors of variable length, so their input are filled with zeros if the length of the vector is shorter than 128.

TABLE 3. Comparison of the proposed designs on Xilinx XC5VLX110T.

| Prototype Circuit | High Speed Circuit |
|--------------------|--------------------|
| m | Freq (MHz) | Slice | Pipeline | Freq (MHz) | Slice | Pipeline |
|-----|-------------|-------|---------|-------------|-------|---------|
| 16  | 387 | 503 | 17 | 392 | 509 | 17 |
| 32  | 387 | 503 | 17 | 386 | 527 | 17 |
| 64  | 387 | 503 | 17 | 386 | 527 | 17 |
| 128 | 387 | 503 | 17 | 380 | 578 | 19 |
| 256 | 387 | 503 | 17 | 379 | 595 | 19 |
| 512 | 387 | 503 | 17 | 378 | 1055 | 21 |
| 1024 | 387 | 503 | 17 | 359 | 1394 | 21 |
| 2048 | 387 | 503 | 17 | 356 | 2176 | 24 |

Multiple vectors are inputted sequentially, no more than p vectors can be processed simultaneously and then the required storage size of the proposed circuits is $p + \lceil p/2 \rceil$. Obviously, this storage size is less than that of the SSA and AeMFPA. In addition, when the length $d$ of the longest input vector is large enough, both the latency and the storage size of the proposed circuits will be less than those of PCBT, FCBT and DSA, which are determined by the $d$.

B. EXPERIMENTAL RESULTS AND DISCUSSION

In this paper, the proposed circuits are implemented on FPGA XC2VP30 platform through Verilog HDL on the Xilinx ISE 10.1 software platform, because most of the existing work are based on the same platform. Generally, the binary operators including the adder and the multiplier are determined according to the application scenarios. Actually, no matter what kind of binary operator is applied, only the number of the pipeline stage in the operator is related to the performance of the proposed circuit. However, the work [9], [12], [18] used the floating-point adder in their experiment. For the fair comparison, in our experiment, a double floating-point and 14-stage pipelined adder is chosen which is provided by Xilinx Core Generator. More specifically, the latency $p_c - p_{pt}$ of the CacheStatQAU_PT module is designed to be 1, and the read latency $p_c - p_{mt}$ of the Cache module is also designed to be 1. In addition, the width of the domain tag is set to be the 3-bit data. The degree of the FanoutTree is 8 and all the Boolean expression tree modules have the same degree. In our experiment, the input and output signals of the MUX and the input signals of the Buffer are latched, which will increase one pipeline stage of the Container and the Operator, respectively. The Cache size is set to 32 and the pipeline stage number of proposed circuit is set to 17. So the circuit can simultaneously deal with 32 vectors inputted in random order or arbitrary number of vectors that are inputted sequentially. There are 32 vectors contained in both two sequences, through which the correctness and accuracy of the two circuits are validated. The input order of the elements of one sequence meets the normal distribution and the other meets the even distribution. Since most of the existing methods are unable to deal with multiple vectors inputted in arbitrary order simultaneously, all input vectors of the proposed circuits should be in sequence for a fair comparison in this experiment.

Table 2 lists the adder, the consumed hardware resources, the operating frequency, and the latency of different reduction circuit. Compared with other existing work, the proposed circuits can reach the highest operating frequency with the least area (slices) consumptions. The BRAM usage of the proposed circuit is larger than that of the DSA, MFPA, and Ae2MFPA. The actual consumption of BRAM agrees with the aforementioned analysis of the storage size. The latencies of the proposed circuits are in the moderate level. However, compared with other methods, the proposed circuits achieved the smallest Slices × us for different vector lengths. The differences of index Slices × us between the methods and other methods become larger and larger with the increase of the vector length. In addition, the prototype circuit and the high speed circuit have similar performances because the m is relatively small.

However, when m becomes larger, e.g., 512, the high speed circuit provides better performance than the prototype circuit, which can be seen from the Table 3. For the high speed circuit, the consumed slices of the CacheStatQAU_HS module are linearly increased with m and the number of slice consumed by the rest of the circuit is constant. As to the prototype circuit, when m is relatively small, the number of the totally consumed slices is smaller than that of the high speed circuit, but when m becomes large, the total consumed slice number
is greater than that of the high speed circuit. This is because in the CacheStatQAU_PT module there are lots of combina-
national logic circuits where the number of output ports of
AND/OR gates are proportional to m. Without the proper par-
tion and pipeline design, such kind of circuit will consume
eormous slices. Essentially the CacheStatQAU_HS module
is the optimized design result of the CacheStatQAU_PT mod-
ule through the pipeline, which is the reason why the total
pipelines of the high speed circuit are increased in Table 3.
Therefore, when m is large, the high speed circuit consumes
fewer slices than the prototype.

Due to the additional circuits used to generate and compare
the data tags, the proposed circuits are a little bit complicated.
In addition, because of the high operating frequency, the pro-
posed circuits may consume more power than other methods.

VI. CONCLUSION
In this paper, we propose and implement a tag based random
order vector reduction circuit that can simultaneously han-
dle multiple vectors inputted in random sequence. However,
as the number of the input vectors increases, the frequency
of the circuit is lowered. To solve this problem, a high-
speed circuit is proposed by improving the control module of
the circuit. Moreover, a detailed theoretical analysis process
about the proposed circuits is also presented and verified.
Both the theoretical and experimental results have shown that
compared with other existing work, the proposed circuits can
achieve the smallest Slices×us (<80% of the state-of-the-art
work). So potentially, the proposed circuits can be used to
accelerate the inference process of the CNN (Convolutional
Neural Network) in our future work, because essentially the
convolution operation is a vector reduction operation.

APPENDIX
In this part, we will first prove that the inequality in Theo-
rem 1 can be satisfied.

Theorem 1: At any clock n, the following inequality is satis-
fied
\[
2 \sum_{v \in Q(n)} q_0(v) \leq |\{v | v \in P3(n), v \neq 0\}| + |P5(n)| + |P7(n)| + l_0(0) - 1
\]

Proof:
1) At clock 0, we have \(P5(0) \subseteq Q(0) = \emptyset, P7(0) \subseteq\)
\(Q(0) = \emptyset, P3(0) = \{0\}, l_0(0) = p,\) then
\[
\sum_{v \in Q(0)} q_0(v) = 0 |\{v | v \in P3(0), v \neq 0\}| + |P5(0)| + |P7(0)| + l_0(0) - 1 = p - 1
\]
Therefore, the inequality is satisfied at clock 0.
2) Assume that the inequality is satisfied at clock \(k, i.e.,\)
\[
2 \sum_{v \in Q(k)} q_k(v) \leq |\{v | v \in P3(k), v \neq 0\}| + |P5(k)| + |P7(k)| + l_k(0) - 1 \tag{39}
\]

At clock \(k+1\), the items in the Buffer, the Cache, and on
the pipeline may have been changed. According to the de-
scription of the Algorithm 2, there are five cases, i.e., case “1xx”,
case “011”, case “010”, case “001”, and case “000”. For
each case, we will prove that at clock \(k + 1\), the following
inequality can be satisfied
\[
2 \sum_{v \in Q(k+1)} q_{k+1}(v) \leq |\{v | v \in P3(k + 1), v \neq 0\}| + |P5(k + 1)| + |P7(k + 1)| + l_{k+1}(0) - 1 \tag{40}
\]

If the case “1xx” occurs at clock \(k + 1\), based on the
mechanism of the proposed circuit, we know that the item
on the pipeline has been changed. The item with the valid tag
from the terminal of the pipeline will be merged with an input
element to form a new item with the same valid tag. This new
item will be put into the first stage of the total pipeline. The
items in the Buffer and Cache will not be changed. So for
every tag \(v\), we have \(l_{k+1}(v) = l_k(v)\), \(c_{k+1}(v) = c_k(v)\), and
\(q_{k+1}(v) = q_k(v)\). Obviously, if (39) is true, then (40) is also
true.

If case “011” occurs at clock \(k + 1\), based on the
mechanism of the proposed circuit, we know that at clock
\(k\), there must be an item with \(r\) tag at the terminal of the pipeline, and in the Cache, there
must be items with tag \(x\) and \(r\), i.e., \(c_k(x) = 1, c_k(r) = 1,\) and
\(l_k(r) \geq 1\). At clock \(k + 1\), the item with the tag \(x\) in the Cache
will be merged with an input element to form an item with the
tag \(x\) and then be put into the first stage of the total pipeline,
and the item with the tag \(r\) in the Cache will be merged with
an input element to form an item with the tag \(r\) and then be
pushed into the Buffer. So, for the tag \(x, tag \, r,\) and any tag \(v\)
which satisfies \(v \neq r, v \neq x\), we have
\[
q_{k+1}(x) = q_k(x)
\]
\[
l_{k+1}(x) = l_k(x) + 1
\]
\[
c_{k+1}(x) = c_k(x) - 1 = 0
\]
\[
q_{k+1}(r) = q_k(r) + 1
\]
\[
l_{k+1}(r) = l_k(r) - 1 \geq 0
\]
\[
c_{k+1}(r) = c_k(r) - 1 = 0
\]
\[
q_{k+1}(v) = q_k(v)
\]
\[
l_{k+1}(v) = l_k(v)
\]
\[
c_{k+1}(v) = c_k(v)
\]
From (41) and (42), we have
\[
\begin{align*}
x & \in C(k) \\
r & \in C(k) \cup L(k) \\
x & \notin C(k + 1), x \in L(k + 1) \\
r & \notin C(k + 1), r \in Q(k + 1)
\end{align*}
\]
That is
\[
\begin{align*}
x & \in P2(k) \cup P4(k) \cup P6(k) \cup P8(k)
\end{align*}
\]
we have
\[ v \in P^4(k) \cup P^8(k) \]
\[ x \in P^3(k + 1) \cup P^7(k + 1) \]
\[ r \in P^5(k + 1) \cup P^7(k + 1) \]  
(45)  
(46)  
(47)  
From (44), (45), (46) and (47), \( x \neq r, x \neq 0 \), we have
\[ ||v||P^3(k + 1), v \neq 0 ||P^5(k + 1) ||P^7(k + 1) || = ||v||P^3(k), v \neq 0 ||P^5(k) ||P^7(k) || + 2 \]  
(48)  
From (41), (42) and (43), we have
\[ \sum_{v \in Q(k + 1)} q_{k+1}(v) = \sum_{v \in Q(k)} q_k(v) - 1 \]  
(49)  
\[ l_{k+1}(0) = l_k(0) \]  
(50)  
From (39), (48), (49), and (50), we know that (40) is satisfied when case “011” occurs at clock \( k + 1 \).
For case “010”, case “001”, and case “000”, following the similar way, we can also prove that equation (40) is true.
Based on 1) and 2), and from mathematical induction, we know that the Theorem 1 is always true.

**REFERENCES**

[1] I. Sayahi, M. Machhout, and R. Tourki, “FPGA implementation of matrix-vector multiplication using xilinx system generator,” in Proc. Int. Conf. Adv. Syst. Electr. Technol. (ICASET), Mar. 2018, pp. 290–295.

[2] M. Kumm, M. Hardeick, and P. Zipf, “Optimization of constant matrix multiplication with low power and high throughput,” IEEE Trans. Comput., vol. 69, no. 12, pp. 2072–2080, Dec. 2017.

[3] M. Hardeick, M. Kumm, P. Sittel, and P. Zipf, “Constant matrix multiplication with ternary adders,” in Proc. 25th IEEE Int. Conf. Electron., Circuits Syst. (ICECS), Dec. 2018, pp. 85–88.

[4] H. Zhang, D. Chen, and S.-B. Ko, “High performance and energy efficient single-precision and double-precision merged floating-point adder on FPGA,” IET Comput. Digit. Techn., vol. 12, no. 1, pp. 20–29, Jan. 2018.

[5] M. Cieplucha, “High performance FPGA-based implementation of a parallel multiplier-accumulator,” in Proc. Int. Conf. Mixed Design Integ. circuits Syst., Jun. 2013, pp. 485–489.

[6] P. M. Kogge, The Architecture of Pipelined Computers. Boca Raton, FL, USA: CRC Press, 1981.

[7] L. M. Ni and K. Huang, “Vector-reduction techniques for arithmetic pipelines,” IEEE Trans. Comput., vol. C-34, no. 5, pp. 404–411, May 1985.

[8] Z. Luo and M. Martianosi, “Accelerating pipelined integer and floating-point accumulations in configurable hardware with delayed addition techniques,” IEEE Trans. Comput., vol. 49, no. 3, pp. 208–218, Mar. 2000.

[9] L. Zhao, G. R. Morris, and V. K. Prasanna, “High-performance reduction circuits using deeply pipelined operators on FPGAs,” IEEE Trans. Parallel Distrib. Syst., vol. 18, no. 10, pp. 1377–1392, Oct. 2007.

[10] E. Wilson and G. Stitt, “The unified accumulator architecture: A configurable, portable, and extensible floating-point accumulator,” ACM Trans. Reconfigurable Technol. Syst., vol. 9, no. 3, pp. 1–23, May 2016.

[11] P. Grigoras, P. Burovskiy, J. Arram, X. Niu, K. Cheung, and J. Xie, “Dfesnippets: An Open-Source Library for Dataflow Acceleration on FPGAs,” in Proc. Int. Symp. App. Reconfigurable Comput., 2017, pp. 299–310.

[12] Y.-G. Tai, C.-T.-D. Lo, and K. Psarris, “Accelerating matrix operations with improved deeply pipelined vector reduction,” IEEE Trans. Parallel Distrib. Syst., vol. 23, no. 2, pp. 202–210, Feb. 2012.

[13] M. Wei and Y.-H. Huang, “A tag based vector reduction circuit,” in Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC), Sep. 2015, pp. 1–6.

[14] H. J. Sips and H. Lin, “An improved vector-reduction method,” IEEE Trans. Comput., vol. 40, no. 2, pp. 214–217, Feb. 1991.

[15] S. R. Vangal, Y. V. Hoskote, N. Y. Borkar, and A. Alvandpour, “A 6.2 GFlops floating-point multiply-accumulator with conditional normalization,” IEEE J. Solid-State Circuits, vol. 41, no. 10, pp. 2314–2323, Oct. 2006.

[16] C. He, G. Qin, M. Lu, and W. Zhao, “Group-alignment based accurate floating-point summation on FPGAs,” in Proc. ERSA, Jun. 2006, pp. 136–142.

[17] K. K. Nagar and J. D. Bakos, “A high-performance double precision accumulator,” in Proc. Int. Conf. Field-Programm. Technol., Dec. 2009, pp. 500–503.

[18] M. Huang and D. Andrews, “Modular design of fully pipelined reduction circuits on FPGAs,” IEEE Trans. Parallel Distrib. Syst., vol. 24, no. 9, pp. 1818–1826, Sep. 2013.

[19] A. Abdoulnasir, M. Psarakis, and A. L. Donis, “An efficient FPGA implementation of the big bang-big crunch optimization algorithm,” in Proc. Int. Symp. App. Reconfigurable Comput., 2018, pp. 166–177.