RESEARCH ARTICLE OPEN ACCESS

Implementation of Pipeline Architecture for High-Speed Computation of the Discrete Wavelet Transform Using HDL

G. Pavan Kumar[1], T. Vishnu Murthy[2], David Solomon Raju Y[3]
[1]PG Student, M. Tech (VLSI) Pragathi Engineering College, Kakinada, East Godavari Dt.-533 437 A.P
[2]Assoc.Prof.ECE, Pragathi Engineering College, Kakinada, East Godavari Dt.-533 437 A.P
[3]Assoc. Prof. ECE, Holy Mary Institute of Technology & Science, Keesara, R. R. Dt.- 501 301, A.P

Abstract
In this paper, a scheme for the design of a high-speed pipeline VLSI architecture for the computation of the 2-D discrete wavelet transform (DWT) is proposed. The main focus in the development of the architecture is on providing a high operating frequency and a small number of clock cycles along with efficient hardware utilization by maximizing the inter-stage and intra-stage computational parallelism for the pipeline. The inter-stage parallelism is enhanced by optimally mapping the computational task of multi decomposition levels to the stages of the pipeline and synchronizing their operations. The intra-stage parallelism is enhanced by dividing the 2-D filtering operation into four subtasks that can be performed independently in parallel and minimizing the delay of the critical path of bit-wise adder networks for performing the filtering operation. To validate the proposed scheme, a circuit is designed, simulated, and implemented in FPGA for the 2-D DWT computation. The results of the implementation show that the circuit is capable of operating with a maximum clock frequency of 80.749MHz and processing 1022 frames of size 512 × 512 per second with this operating frequency. It is shown that the performance in terms of the processing speed of the architecture designed based on the proposed scheme is superior to those of the architectures designed using other existing schemes, and it has similar or lower hardware consumption.
Keywords - Computational parallelism, Discrete Wavelet Transform FPGA implementation, Image Processing, Multi-resolution filtering, Non-separable approach, Parallel architecture.

I. INTRODUCTION

The 2-D discrete wavelet transforms (DWT) have been widely used in many engineering applications because of their multi-resolution decomposition capability [1]. However, processing large volumes of data of various decomposition levels of the transform makes their computation computationally very intensive. In the past, many architectures have been proposed aimed at providing high-speed 2-D DWT computation with the requirement of utilizing a reasonable amount of hardware resources. These architectures can be broadly classified into separable and non-separable architectures. In a separable architecture, a 2-D filtering operation is divided into two 1-D filtering operations, one for processing the data row-wise and the other column-wise. In the previous papers it was proposed a low-storage short-latency separable architecture in which the row-wise operations are performed by systolic filters and the column-wise operations by parallel filters. This architecture requires complex control units to facilitate the interleaved operations of the output samples of different decomposition levels by employing a recursive pyramid algorithm (RPA). Architecture has been introduced in which each of the row- and column-wise filtering operations are decomposed using the so called lifting operations into a cascade of sub-filtering operations. The scheme leads to low-complexity architecture with a large latency. The separable architectures, in which a 1-D filtering structure is used to perform the 2-D DWT, have an additional requirement of transposing the intermediate data between the two 1-D filtering processes. This increases the memory size as well as the latency of the architectures. The non-separable architectures do not have this problem, since in these architectures, the 2-D transforms are computed directly by using 2-D filters. It has been proposed that two non-separable architectures, one using parallel 2-D filters and the other an SIMD 2-D array, both based on a modified RPA. In the former architecture, a high degree of computational parallelism is achieved at the expense of less efficient hardware utilization, whereas the latter architecture requires a reconfigured organization of the array as the processing moves on to higher decomposition levels. But the processing speed of this architecture is low in view of the fact that the same architecture is utilized recursively to perform the tasks of successive decomposition levels. As the processing units employed in this architecture differ from one another, the complexity of the hardware resources is high and the design of the architecture is complicated. Most existing non-separable architectures aim at providing fast computation of the DWT by using pipeline structures and a large number of parallel filters. However, these existing architectures have not exploited the computational parallelism inherent in the DWT operation to the extent possible in order to provide a high speed.

In this paper, non-separable pipeline architecture for fast computation of the 2-D DWT with a reasonable low cost for the hardware resources is proposed. The high-speed computation is achieved by efficiently distributing the task of the computations of multiple decomposition levels among the stages of the pipeline, and by optimally configuring the data and synchronizing the operations of pipeline so as to maximize the inter-stage and intra-stage computational parallelism. The paper is organized as follows. In Section II, a mathematical formulation of the 2-D DWT computation necessary for the development of the proposed architecture is presented. In Section III, a study is conducted to determine the number of stages of a pipeline necessary for optimally mapping the task of the DWT computation onto the stages of the pipeline. Based on this study, in Section IV, three-stage pipeline architecture is developed with an efficient structure of the 2-D input data and an optimal organization of the processing units in each of the stages. In Section V, the performance results is assessed and shown in the form of synthesis by an FPGA implementation. Section VI summarizes the work of this paper by highlighting the salient features of the proposed architecture.

II. Formulations for the Computation of the 2-D DWT

The 2-D DWT is an operation through which a 2-D signal is successively decomposed in a spatial multi resolution domain by low pass and high pass FIR filters along each of the two dimensions. The four FIR filters, denoted as high pass-high pass (HH), high pass-low pass (HL), low pass-high pass (LH) and low pass-low pass (LL) filters, produce, respectively, the HH, HL, LH and LL sub band data of the decomposed signal at a given resolution level. The samples of the four sub bands of the decomposed signal at each level are decimated by a factor of two in each of the two dimensions. For the operation at the first level of decomposition, the given 2-D signal is used as input, whereas for the operations of the succeeding levels of decomposition, the decimated LL sub band signal from the previous decomposition level is used as input.

A. Formulation for the 2-D DWT Computation

Let a 2-D signal be represented by an $N_o \times N_o$ matrix $S^{(o)}$, with its *(m, n)* th element denoted by $S^{(o)}$ (m, n) $(0 \le m, n \le N_o-1)$, where N_o is chosen to be 2^J, J being an integer. Let the coefficients of a 2-D FIR filter P (P = HH, HL, LH, LL), be represented by an L×M matrix $H^{(P)}$. The $(k, i)^{th}$ coefficient of the filter P is denoted by $H^{(P)}$ (k, i) $(0 \le k \le L-1)$, $(0 \le i \le M-1)$. The decomposition at a given level j = 1, 2, ….. J can be expressed as

$$A^{(j)}(m,n) = \sum_{k=0}^{L-1}\sum_{i=0}^{M-1} H^{(\mathrm{HH})}(k,i) \cdot S^{(j-1)}(2m-k, 2n-i),$$

$$B^{(j)}(m,n) = \sum_{k=0}^{L-1}\sum_{i=0}^{M-1} H^{(\mathrm{HL})}(k,i) \cdot S^{(j-1)}(2m-k, 2n-i),$$

$$C^{(j)}(m,n) = \sum_{k=0}^{L-1}\sum_{i=0}^{M-1} H^{(\mathrm{LH})}(k,i) \cdot S^{(j-1)}(2m-k, 2n-i),$$

$$S^{(j)}(m,n) = \sum_{k=0}^{L-1}\sum_{i=0}^{M-1} H^{(\mathrm{LL})}(k,i) \cdot S^{(j-1)}(2m-k, 2n-i),$$

Each 2-D convolution can be seen as a sum of the products of L×M the filter coefficients and the elements contained in an L×M window sliding on a 2-D data. The decimation by a factor of two in both the horizontal and vertical dimensions can be accomplished by sliding the L×M window by two positions horizontally and vertically for the computation of two successive samples. Only the LL sub band data of decomposition are used as input for the decomposition at the next level. After iterations, the 2-D signal $S^{(o)}$ is transformed into *J* resolution levels, with HH, HL and LH sub bands from each of the first *J-1* levels and HH, HL, LH and LL sub bands from the last J^{th} level. Since $N_j = N_0/2^j$, the number of samples that need to be processed at each level is one quarter of that at the preceding level.

B. Formulation for a Four-Channel Filtering Operation

In order to facilitate parallel processing for the 2–D DWT computation, the L×M filtering operation needs to be divided into multi-channel operations, each channel processing one part of the 2-D data. It is seen from (1) that the even and odd indexed elements are always operated on the even and odd indexed filter coefficients, respectively. The matrix $S^{(o)}$ representing the LL sub band at j^{th} the level can, therefore, be divided into four $N_j = (N_j/2+L/2) \times (N_j/2+M/2)$ sub-matrices, and the elements are given by as shown below

$$S_{ee}^{(j)}(m,n) = S^{(j)}(2m, 2n)$$
$$S_{oe}^{(j)}(m,n) = S^{(j)}(2m+1, 2n)$$
$$S_{eo}^{(j)}(m,n) = S^{(j)}(2m, 2n+1)$$
$$S_{oo}^{(j)}(m,n) = S^{(j)}(2m+1, 2n+1)$$

taking into consideration the periodic padding samples at the boundary [30]. It is seen from (2) that the data at any decomposition level are divided into four channels for processing by first separating the even and odd indexed rows of $S^{(j)}$, and then separating the even and odd indexed columns of the resulting two sub-matrices. The data in each channel can then be computed by an *(L/2×M/2)* - tap filtering operation. In order to facilitate such a 4-channel filtering operation, the filter coefficients, as used in (1), need to be decomposed appropriately.

Accordingly, the matrix $H^{(P)}$ needs to be decomposed into four *(L/2×M/2)* sub-matrices, and elements are given by

$$H^{(\mathrm{P})}_{ee}(k,i) = H^{(\mathrm{P})}(2k,2i)$$
$$H^{(\mathrm{P})}_{oe}(k,i) = H^{(\mathrm{P})}(2k+1,2i)$$
$$H^{(\mathrm{P})}_{eo}(k,i) = H^{(\mathrm{P})}(2k,2i+1)$$
$$H^{(\mathrm{P})}_{oo}(k,i) = H^{(\mathrm{P})}(2k+1,2i+1)$$

respectively. By using (2) and (3) in (1), any of the four sub band signals $\mathbf{A}^{(j)}$, $\mathbf{B}^{(j)}$, $\mathbf{C}^{(j)}$, and $\mathbf{S}^{(j)}$, at the *j th* decomposition level, can be computed as a sum of four convolutions using *(L/2×M/2)* -tap filters. For example, the LL sub band given by (1d) can now be expressed as

$$\begin{aligned} S^{(j)}(m,n) = & \sum_{k=0}^{L/2-1}\sum_{i=0}^{M/2-1} H^{(\mathrm{LL})}_{ee}(k,i) \cdot S^{(j-1)}_{ee}(m+k,n+i) \\ & + \sum_{k=0}^{L/2-1}\sum_{i=0}^{M/2-1} H^{(\mathrm{LL})}_{oe}(k,i) \cdot S^{(j-1)}_{oe}(m+k,n+i) \\ & + \sum_{k=0}^{L/2-1}\sum_{i=0}^{M/2-1} H^{(\mathrm{LL})}_{eo}(k,i) \cdot S^{(j-1)}_{eo}(m+k,n+i) \\ & + \sum_{k=0}^{L/2-1}\sum_{i=0}^{M/2-1} H^{(\mathrm{LL})}_{oo}(k,i) \cdot S^{(j-1)}_{oo}(m+k,n+i) \end{aligned}$$

At any decomposition level, the separation of the sub band processing corresponding to even and odd indexed data as given by (4) is consistent with the requirement of decimation of the data in each dimension by a factor of two in the DWT computation. It is also seen from (4) that the filtering operations in the four channels are independent and identical, which can be exploited in the design of efficient pipeline architecture for the 2-D DWT computation.

III. PIPELINE FOR THE 2-D DWT COMPUTATION

In a pipeline structure for the DWT computation, multiple stages are used to carry out the computations of the various decomposition levels of the transform. The computation corresponding to each decomposition level needs to be mapped to a stage or stages of the pipeline. It is seen from the formulation in Section II that the task of computing the *jth* decomposition level in a -level DWT computation consists of computing samples. The computation of each sample actually performs an *(L×M)*-tap HH, HL, LH or LL FIR filtering operation that comprises the operations of *(L×M)* multiplications followed by *(L×M)* accumulations. Assuming that these operations for the computation of one sample are carried out by a unit of filter processor, the overall task of the DWT computation would require a certain number of such filter units. In order to design a pipeline structure capable of performing a fast computation of the DWT with low expense on hardware resources and low design complexity, an optimal mapping of the overall task of the DWT computation to the various stages of the pipeline needs to be determined. Any distribution of the overall task of the DWT computation to stages must consider the inherent nature of the sequential computations of the decomposition levels that limit the computational parallelism of the pipeline stages, and consequently the latency of the pipeline. The key factors in the distribution of the task to the stages are the maximization of the inter-stage and intra-stage computational parallelism and the synchronization of the stages within the constraint of the sequential nature of the computation of the decomposition levels. The feature of identical operations associated with the computations of all the output samples irrespective of the decomposition levels in a DWT computation can be exploited to maximize the intra-stage parallelism of the pipeline. Further, in order to minimize the expense on the hardware resources of the pipeline, the number of filter units used by each stage ought to be minimum and proportional to the amount of the task assigned to the stage. A straightforward mapping of the overall task of the DWT computation to a pipeline is one-level to one-stage mapping, in which the tasks of decomposition levels are distributed to *J* stages of the pipeline. In this mapping, the amount of hardware resources used by a stage should be one-quarter of that used by the preceding stage. Thus, the ratio of the hardware resource used by the last stage to that used by the first stage has a value of $1/4^{J-1}$. For images of typical size, this parameter would assume a very small value. Hence, for a structure of the pipeline that uses identical filter units, the number of these filters units would be very large. Further, since the number of such filter units employed by the stages would decrease exponentially from one stage to the next in pipeline, it will make their synchronization very difficult. The solution to such a difficult synchronization problem, in general, requires more control units, multiplexers and registers, which results in a higher complexity of the hardware resources. A reasonably large value of $\lambda < 1$would be more attractive for synchronization. In this respect, the parameter can be seen as a measure of difficulty in that a smaller value of this parameter implies a greater design effort and more hardware resources for the pipeline. The parameter can be increased from its value of $1/4^{J-1}$. in the one-level to one-stage pipeline structure by dividing the large-size stages into a number of smaller stages or merging the small-size stages into larger ones. However, dividing a stage of the one-level to one-stage pipeline into multiple stages would require a division of the task associated with the corresponding decomposition level into sub-tasks, which in turn, would call for a solution of even a more complex problem of synchronization of the sub-tasks associated with divided stages. On the other hand, merging multiple small-size stages of the pipeline into one stage would not create any additional synchronization problem. As a matter of fact, such a merger could be used to reduce the overall number of filter units of the pipeline. In view of the above discussion, the synchronization parameter λ can be increased by merging a number of stages at tail end of the pipeline. Fig. shows the structure of a pipeline in which the stages *I* to *J* of the one-level to one-stage pipeline have been merged. In this structure, the tasks of the decomposition level from *j = 1to j=I - 1*are mapped to stage 1 to *I - 1*, respectively, whereas those of the decomposition levels *j = I....J*, are mapped all together to the I th stage. Note that the total amount of computations performed by stage *I* is

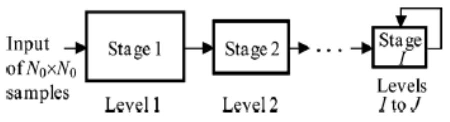


Fig Pipeline structure with stages for -level computation

IV. DESIGN OF THE ARCHITECTURE

In the previous section, we advocated a three-stage pipeline structure for the computation of the 2-D DWT to realize an optimal combination of the parameters for the hardware utilization and pipeline synchronization. In this three-stage structure, like in any pipeline architecture, the operations in a given stage depend on the data produced by the preceding stage. However, because of the way that the computational load of the various decomposition levels of the 2-D DWT computation has been distributed among the three stages, the operations in the first and second stages of the pipeline do not depend on the data produced by them, whereas that in stage 3 does depend on the data produced by itself. The operations of the three stages need to be synchronized in a manner so that the three stages perform the computation of multiple decomposition levels within a minimum possible time period while using the available hardware resources maximally. In this section, we present the design of the proposed 3-stage pipeline architecture, starting with the synchronization of the operations of the stages, and then focusing on the details of the intra-stage design so as to provide an optimal performance.

A. Synchronization of Stages

The distribution of the computational load among the three stages, and the hardware resources made available to them are in the ratio 8:2:1. Accordingly, the synchronization of the operations between the stages needs to be carried out under this constraint of the distribution of the computational load and hardware resources. According to the nature of the DWT, the computation of a decomposition level *j* depends on the data computed at its previous level *j - 1*, in which the number of computations is four times of that at the decomposition level. Therefore, the stages of pipeline need to be synchronized in such a way that each stage starts the operation at an earliest possible time when the required data become available for its operation. Once the operation of a stage is started, it must continue until the task assigned to it is fully completed. Consider the timing diagram given in Fig. below for the operations of the three stages, where t_1, t_2 and t_3 are the times taken individually by stages 1, 2 and 3, respectively, to complete their assigned tasks, t_a and t_b are the times elapsed between the starting points of the tasks by stages 1 and 2, and that by stages 2 and 3, respectively.

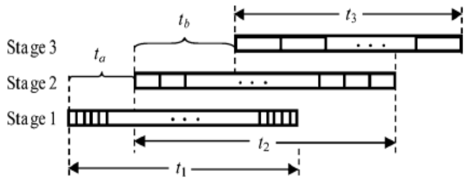


Fig Timing diagram for the operations of three stages

Note that the lengths of the times t_1, t_2 and t_3 to complete the tasks by individual stages are approximately the same, since the ratios of the tasks assigned and the resources made available to the three stages are the same. The average times to compute one output sample by stages 1, 2 and 3 are in the ratio 1:4:8. In Fig. above relative widths of the slots in the three stages are shown to reflect this ratio. Our objective is to minimize the total computation time t_a+ t_b+ t_3 by minimizing t_a, t_b and t_3, and individually. Assume that 2-D output samples for a decomposition level are computed row-by-row starting from the upper-left corner sample. Since the operations in stage 1 are independent of those in the other two stages, it can operate continuously to compute all the samples of level 1. The value of t_1 is equal to T_8 N^2_1, where T_8 is the average time taken by stage 1 to compute one output sample. Since the operations of stages 2 and 3 require the output data computed by stages 1 and 2, respectively, their operations must be delayed by certain amount of times so that they can operate continuously with the data required by them becoming available. We now give the lowest bound on t_a and t_b so that once stages 2 and 3 start their operations they could continue their operations uninterruptedly. Assume that stage 3 computes all the output samples of all remaining levels (i.e., level 3 to level) in a sequential manner. We only need to consider the requirement of the data availability for the computation of level-3, which uses the level-2 samples computed by stage 2. Then, in a way similar to that obtaining $t_{a\ min}$, by imposing the condition that at the time instant of starting the calculation of a level-3 output sample by stage 3, all the samples in the window of the level-2 output samples are available, it can be shown that the minimum value of is given by $t_{a\ min} = T_8\ [N_1\ (L-1) +M]$

B. Design of Stages

The three-stage architecture, stages 1 and 2 perform the computations of levels 1 and 2, respectively, and stage 3 that of all the remaining levels. Since the basic operation of computing each output sample, regardless of the decomposition level or the sub band, is the same, the computation blocks in the three stages can differ only in the number of identical processing units employed by them depending on the amount of the computations assigned to the stages. As seen, an *(L×M)*-tap filtering operation is decomposed into four independent *(L/2×M/2)*-tap filtering operations, each operating on the 2-D *L/2×M/2*data resulting from the even or odd numbered rows and even or odd numbered columns of an *L×M* window of an LL-sub band data. A unit consisting of *L/2×M/2*MAC cells can now be regarded as the basic *processing unit* to carry out an *(L/2×M/2)*-tap filtering operation. An *L×M* window of the raw 2-D input data or that of an LL-sub band data must be decomposed into four distinct *L/2×M/2*sub-windows in accordance with the four decomposed terms given by the right side of (4). This decomposition of the data in an *L×M* window can be accomplished by designing for each stage an appropriate data scanning unit (DSU) based on the way the raw input or the LL-sub band data is scanned. The stages would also require memory space (buffer) to store the raw input data or the LL-sub band data prior to scanning. Since stages 1 and 2 need to store only part of a few rows of raw input or LL-sub band data at a time, they require a buffer of size of O (N), whereas since stage 3 needs to store the entire LL-sub band data of a single decomposition level, it has a buffer of size of O (N)^2. Fig. gives the block diagram of the pipeline showing all the components required by the three stages. Note that the data flow shown in this figure comprises only the LL-sub band data necessary for the operations of the stages. The HH, HL and LH sub band data are outputted directly to an external memory. Now, we give details on the structure of the data scanning unit to scan the 2-D data and establish four distinct *L/2×M/2* sub-windows, as well as on the distribution of the filtering operations to the processing units in each stage.

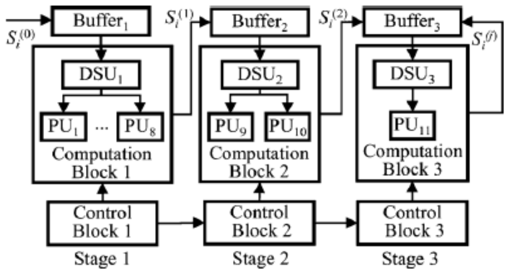


Fig Block diagram of the three-stage architecture

Structure of the Data Scanning Unit: In accordance with (4), an *L×M* window of the raw 2-D input data stored in or an LL-sub band data stored in or must be partitioned into four *L/2×M/2*sub-windows, and stored into the DSU of the corresponding stage. Further, this same equation also dictates that a 2-D input data must be scanned in a sequential manner. According to this sequence of scanning, the samples in a set of data comprising *L* rows of a 2-D input data are scanned starting from the top-left corner. Once the scanning of all the samples of *L* rows is completed, the process is repeated for another *L* rows after shifting down by two row positions. The objective is then to design a structure for a DSU so that samples scanned with this sequential mode get partitioned into the four sub-windows. In order to partition an *L×M* window into four *L/2×M/2* sub-windows, the structure of the DSU must first partition the samples of the window into two parts depending on whether a sample belongs to an even-indexed or odd-indexed row; then the samples in each part must be partitioned further into two parts depending on whether a sample belongs to an even-indexed or odd-indexed column. The first partition can be achieved by directing scanned samples alternatively to two sets of *L/2*shift registers. The second partition can be achieved by reorganizing the samples stored in the shift registers of the two sets depending on whether a sample belongs to even-indexed or odd-indexed column by employing de-multiplexers. Finally, the samples of the four sub-windows can be stored, respectively, into four units of *L/2×M/2* parallel registers. Fig. shows a structure of the DSU to accomplish this task. This data scanning scheme automatically incorporates the down sampling operations by two in the vertical and horizontal directions (as required by the transform), and thus no additional peripheral circuits and registers are required for the down sampling operations by the architecture. As a result, the data scanning scheme, in comparison to the other schemes [32], requires less hardware resources for the control units and fewer registers for the stages

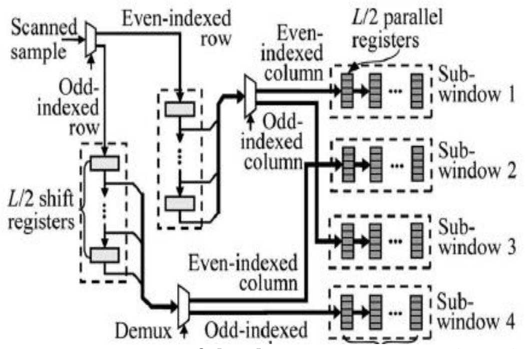


Fig Structure of the data scanning unit (DSU)

C. Design of the Processing Unit

In each stage, a processing unit carries out an *(L/2×M/2)*-tap filtering operation using the samples of an *L/2×M/2*sub-window at a time to produce the corresponding output. Since the sub-windows cannot be fed into a processing unit at a rate faster than the rate at which these sub-windows are processed by the processing unit, the processing time to process a sub-window (one time unit) is critical in determining the maximum clock frequency at which the processing units can operate. Each physical link from a given bit of the input to an output bit of the processing unit gives rise to a data path having a delay that depends on the number and the types of operations being carried out along that path. Therefore, it is crucial to aim at achieving the shortest possible delay for the critical path when designing a processing unit for our architecture [33]–[36].

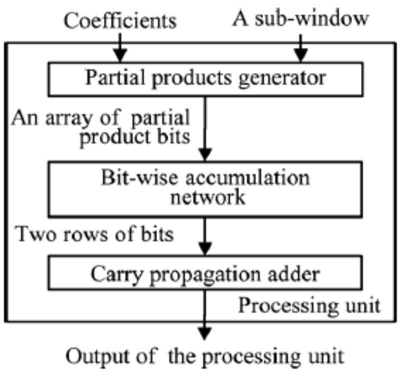
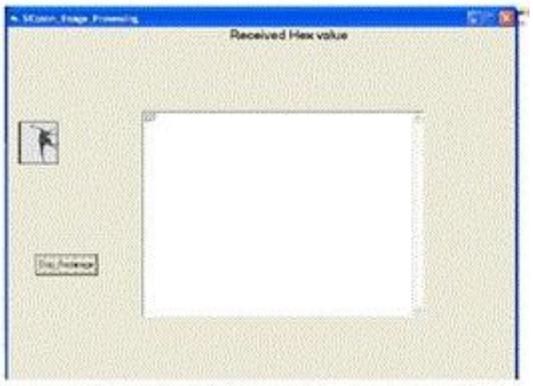

Coefficients
A sub-window
Partial products generator
An array of partial product bits
Bit-wise accumulation network
Two rows of bits
Carry propagation adder
Processing unit
Output of the processing unit

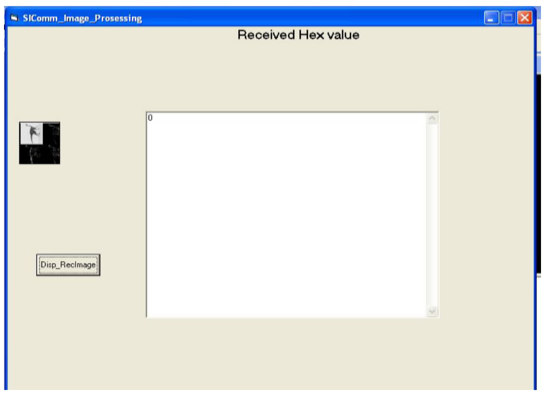
Fig Block diagram of a processing unit

The filtering operation carried out by a processing unit, as described above, can be seen as $L/2\times M/2$parallel multiplications followed by an accumulation of the $L/2\times M/2$ products. If the input samples and the filter coefficients have the word lengths of A and B bits, respectively, then the processing unit produces an array of (B*L*M/4) ×A bits simultaneously in one clock cycle. In order to obtain the output sample corresponding to a given sub-window, the bits of the partial products must be accumulated vertically downward and from right to left by taking the propagation of the carry bits into consideration. The task of this accumulation can be divided into a sequence of layers. The shortest critical data path can be achieved by minimizing the number of layers and the delay of the layers. In each layer, a number of bits consisting of the partial product bits and/or the carry bits from different rows need to be added. This can be done by employing in parallel as many bit-wise adders as needed in each layer. The idea behind using bit-wise adder is to produce to the extent possible the number of output bits from a layer is smaller than the number of input bits to that layer. This can be done by using full adders and specifically designed double adders, in which the full adder consumes 3 bits and produces 2 bits (one sum and one carry bits) whereas the double adder consumes two pairs of bits (2×2) from neighboring columns and produces 3 bits (one sum and two carry bits/two sum and one carry bits). The two types of adders have equal delay, and are efficient in generating carry bits and compressing the number of partial products [36]. With this structure of the layers, the number of layers becomes minimum possible and the delay of a layer is equal to that of a full adder or equivalently to that of a double adder, thereby providing the shortest critical path for the accumulation network. Since the two rows of bits produced by the accumulation network still remain un accumulated, they finally need to be added to produce one row of output bits in the final phase of the task of a processing unit by using a carry propagation adder. Note that tasks of the accumulation network and the carry propagation adder can be made to have some partial overlap, since the latter can start its processing as soon as the rightmost pairs of bits becomes available from the former. Fig. 9 depicts a block diagram of a processing unit based on the above discussion.

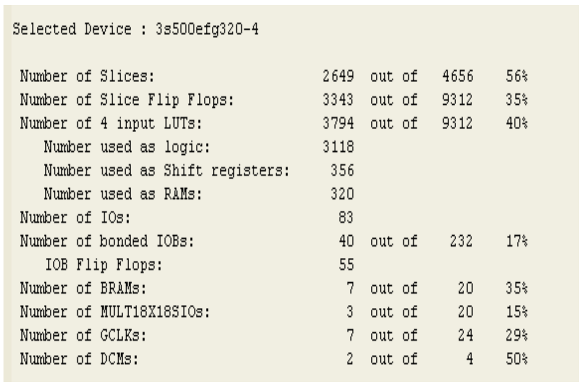
V. PERFORMANCE RESULTS

In order to evaluate the performance of a computational architecture, one needs to make use of certain metrics that characterize the architecture in terms of the hardware resources used and the computation time. In this paper, the hardware resources used for the filtering operation are measured by the number of multipliers (N_{MUL}) and the number of adders (N_{ADD}), and that used for the storage of data and filter coefficients are measured by the number of registers (N_{REG}). The computation time, in general, is technology dependent. However, a metric that is technology independent and can be used to determine the computation time T is the number of clock cycles (N_{CLK}) elapsed between the first and the last samples inputted to the architecture. Assuming that one clock period is T_c, the total computation time can then be obtained as T=N_{CLK} T_c.



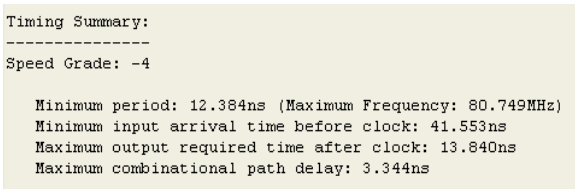


and the synthesis report is below



```
Selected Device : 3s500efg320-4

 Number of Slices:                      2649  out of   4656    56%
 Number of Slice Flip Flops:            3343  out of   9312    35%
 Number of 4 input LUTs:                3794  out of   9312    40%
    Number used as logic:               3118
    Number used as Shift registers:      356
    Number used as RAMs:                 320
 Number of IOs:                           83
 Number of bonded IOBs:                   40  out of    232    17%
    IOB Flip Flops:                       55
 Number of BRAMs:                          7  out of     20    35%
 Number of MULT18X18SIOs:                  3  out of     20    15%
 Number of GCLKs:                          7  out of     24    29%
 Number of DCMs:                           2  out of      4    50%
```



```
Timing Summary:
---------------
Speed Grade: -4

   Minimum period: 12.384ns (Maximum Frequency: 80.749MHz)
   Minimum input arrival time before clock: 41.553ns
   Maximum output required time after clock: 13.840ns
   Maximum combinational path delay: 3.344ns
```

VI. CONCLUSION

In this paper, 3-stage pipeline architecture for a real-time computation of the 2-D DWT has been implemented. The objective has been to achieve a short computation time by maximizing the operational clock frequency (1/ T_c)and minimizing the number of clock cycles (N_{CLK}) required for the DWT computation by developing a scheme for enhanced inter-stage and intra-stage computational parallelism for the pipeline architecture. The results of the FPGA implementation have shown that the circuit can process a 512×512 image in 13.840ns, which is at least two times faster than that of the other FPGA implementations, and in some instances, even with less hardware utilization. Finally, it is worth noting that the architecture designed in this paper is scalable in that its processing speed can be adjusted upward or downward by changing the number of MAC cells in each of the processing units by a factor equal to that of the reduction required in the processing speed.

REFERENCES

[1] H. Y. Liao, M. K. Mandal, and B. F. Cockburn, “Efficient architectures for 1-D and 2-D lifting-based wavelet transforms,” *IEEE Trans. Signal Process.*, vol. 52, no. 5, pp. 1315–1326, May 2004.

[2] P. Wu and L.Chen, “An efficient architecture for two-dimensional discrete wavelet transform,” *IEEE Trans. Circuits Syst. Video Technol.*,vol. 11, no. 4, pp. 536–545, Apr. 2001.

[3] S. Masud and J. V. McCanny, “Reusable silicon IP cores for discrete wavelet transform applications,” *IEEE Trans.Circuits Syst. I, Reg. Papers,* vol. 51, no. 6, pp. 1114–1124, Jun. 2004.

[4] T. Huang, P. C. Tseng, and L. G. Chen, “Generic RAM-based architectures for two-dimensional discrete wavelet transform with line-based method,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 15, no. 7, pp. 910–920, Jul. 2005.

[5] P. K. Meher, B. K. Mohanty, and J. C. Patra, “Hardware-efficient systolic- like modular design for two-dimensional discrete wavelet transform,” *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 55, no. 2, pp.151–155, Feb. 2008.

[6] A. Benkrid, D. Crookes, and K. Benkrid, “Design and implementation of a generic 2-D orthogonal discrete wavelet transform on an FPGA,” in *Proc. IEEE 9th Symp. Field-programming Custom Computing Machines (FCCM)*, Apr. 2001, pp. 190–198.

[7] P. McCanny, S. Masud, and J. McCanny, “Design and implementation of the symmetrically extended 2-D wavelet transform,” in *Proc. IEEE Int. Conf. Acoustic, Speech, Signal Process. (ICASSP)*, 2002, vol. 3, pp. 3108–3111.

[8] S. Raghunath and S. M. Aziz, “High speed area efficient multi-resolution 2-D 9/7 filter DWT processor,” in *Proc. Int. Conf. Very Large Scale Integration (IFIP)*, Oct. 2006, vol. 16–18, pp. 210–215.

[9] M. Angelo poulou, K. Masselos, P. Cheung, and Y. Andreo poulos, “A comparison of 2-D discrete wavelet transform computation schedules on FPGAs,” in *Proc. IEEE Int. Conf. Field Programmable Technology (FPT)*, Bangkok, Tailand, Dec. 2006, pp. 181–188.

[10] C. Cheng and K. K. Parhi, “High-speed VLSI implementation of 2-D discrete wavelet transform,” *IEEE Trans. Signal Process.*, vol. 56, no.1, pp. 393–403, Jan. 2008.

[11] K. C. Hung, Y. S. Hung, and Y. J. Huang, “A nonseparable VLSI architecture for two-dimensional discrete periodized wavelet transform,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 9, no. 5, pp. 565–576, Oct. 2001.

[12] I. S. Uzun and A. Amira, “Rapid prototyping—framework for FPGA based discrete biorthogonal wavelet transforms implementation,” *IEE Vision, Image Signal Process.*, vol. 153, no. 6, pp. 721–734, Dec. 2006.

[13] R. J.C. Palero, R. G. Gironez, and A. S. Cortes, “A novel FPGA architecture of a 2-D wavelet transform,” *J. VLSI Signal Process.*, vol. 42, pp. 273–284, 2006.

[14] J. Song and I. Park, “Pipelined discrete wavelet transform architecture scanning dual lines,” *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 56, no. 12, pp. 916–920, Dec. 2009.

[15] C. Zhang, C. Wang, and M. O. Ahmad, “A VLSI architecture for a fast computation of the 2-D discrete wavelet transform,” in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2007, pp. 3980–3983.