Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2009, Article ID 789592, 21 pages
doi:10.1155/2009/789592

Research Article

Analysis and Design of a Context Adaptable SAD/MSE Architecture

Arvind Sudarsanam,[1] Aravind Dasu,[1] and Karthik Vaithianathan[2]

[1] *Reconfigurable Computing Group, Utah State University, Old Main Hill, UMC-4120, Logan, UT 84321, USA*
[2] *Visual Computing Group, Intel Corporation, 2501, NW 229th Avenue, Hillsboro, OR 97124, USA*

Correspondence should be addressed to Arvind Sudarsanam, arvind.sudarsanam@aggiemail.usu.edu

Received 7 February 2009; Accepted 27 May 2009

Recommended by Miriam Leeser

Design of flexible multimedia accelerators that can cater to multiple algorithms is being aggressively pursued in the media processors community. Such an approach is justified in the era of sub-45 nm technology where an increasingly dominating leakage power component is forcing designers to make the best possible use of on-chip resources. In this paper we present an analysis of two commonly used window-based operations (sum of absolute differences and mean squared error) across a variety of search patterns and block sizes (2×3, 5×5, etc.). We propose a context adaptable architecture that has (i) configurable 2D systolic array and (ii) 2D Configurable Register Array (CRA). CRA can cater to variable pixel access patterns while reusing fetched pixels across search windows. Benefits of proposed architecture when compared to 15 other published architectures are adaptability, high throughput, and low latency at a cost of increased footprint, when ported on a Xilinx FPGA.

1. Introduction

There has been an increase in flexibility and variations demanded by video processing applications. Conventional approaches to support flexibility in hardware design have centered around the design of multiple accelerators, each designed to accelerate a specific variation of the application, and switching off the accelerators selectively during run-time to save power. However, the dominance of leakage power dissipation in deep submicron technologies [1] continues to challenge VLSI architects to make the best possible use of on-chip resources, and the conventional approaches fail to provide a low-power design. As an alternative, VLSI architects are exploring designs that can offer flexibility or adaptability within and across multiple algorithms.

Video processing applications include motion estimation, deinterlacing, and optical flow analysis. Such applications are embarrassingly parallel, thereby making them suitable for hardware acceleration. An analysis of these algorithms reveals most of the compute-intensive kernels to be window-based operations such as sum of absolute differences (SADs) and mean-square error (MSE). Depending on content and compression ratios, each of these kernels in turn may need to be supported for various block sizes of $p \times q$ (2×3, 3×3, 4×4, 8×4, etc.). A high-level pseudocode is presented in Figure 1. This code illustrates the window-based operations for variable block sizes and also showcases the parallelism available amongst such operations. The focus of this paper is a video processing accelerator capable of efficiently supporting arbitrary block sizes for both SAD and MSE kernels and is based on a context adaptable architecture that builds on prior efforts which use systolic arrays.

Rest of the paper is organized as follows. Limited flexibility hardware accelerators for motion estimation that have been proposed in [2–11] are discussed in Section 2. Section 3 analyzes the compute and data access patterns of arbitrary block-sized SAD/MSE computations. Section 4 discusses the proposed context adaptable architecture and Section 5 presents results of the architecture, in comparison to peer designs in terms of an FPGA implementation.

```
Algorithm: Window based operations
Variables:
RF: Reference frame
CF: Current frame
nBlocks: Total number of blocks in the current frame that
need to be processed
nSearches: Number of candidate blocks in the reference
frame for a given block in the current frame
p, q: block sizes (horizontal and vertical)
SAD_or_MSE: Sum of absolute differences or Mean
squared error as error metric to compare two blocks
F(x, y): Per-pixel operation (absolute difference between x
and y or difference between squares of x and y)

For x = 1 to nBlocks
For y = 1 to nSearches
For y = 1 to nSearches
    (a, b) = location of anchor pixel for current search.
    SAD_or_MSE = 0;
    For i = 0 to p − 1
    For j = 0 to q − 1
    SAD_or_MSE = SAD_or_MSE + F(RF(i + a, j + b),CF(i, j))
    End j
    End i
    Analyze SAD_or_MSE
End y
End x
```

Figure 1: High-level pseudocode to illustrate the window-based operations that are targeted for acceleration.

2. Literature Review

This section provides a review of motion estimation architectures. Motion estimation is an embarrassingly parallel application that can be accelerated using a set of vector processing elements. However, massively parallel vector processors result in the requirement for exceedingly high memory or I/O bandwidth, thereby making such designs infeasible to be realized in actual hardware. The concept of systolic arrays was first proposed in the year 1980 by Kung and Leiserson [12] and it provided a gateway to execute operations concurrently, while data is fed into the system in a pipelined fashion. I/O resources are transformed into local connections between processing elements. Fixed Size Block Motion Estimation (FSBME) is observed to possess the characteristic of data being reused in a highly regular manner, and this can be exploited to develop a systolic array. Previous works have utilized systolic arrays to realize efficient motion estimation architectures and have shown 100% (or near 100%) resource utilization while attempting to limit the memory bandwidth. In the year 1989, there was a plethora of research efforts [2–4] towards developing such architectures.

In one of the earlier works towards FSBME acceleration engines, Komarek and Pirsch [4] presented a set of four systolic arrays of differing flavors—1D, 2D, and varying scheduling vectors. Overall data flow for FSBME algorithm was modeled by combining the three individual data flows, those of the reference pixels, current pixels, and the partial SAD values. In each of the four architectures, partial SAD value is computed in each Processing Element (PE) and is passed to the bottom PE or accumulator. Dependence graph for the search operation of a single Macroblock (MB) is a 4D graph. This 4D graph is projected along two of the axes onto a 2D plane, thereby resulting in a 2D systolic array. Another level of projection would result in a 1D systolic array. Regular block-type systolic array architecture is realized by passing columns of data either between two rows or two columns of PEs (in case of 2D array) or between two PEs (in case of 1D array) in a regular fashion (not diagonal). AS-1 and AS-2 are systolic array architectures that involve data transfer along non-uniform delay units and along the diagonal paths between PEs in adjacent rows or columns respectively. Amount of time to compute each motion vector is also proposed. Time for computing best SAD for a $N \times N$ block with search area of size P for AS-2 is proportional to P^2, and the compute time for AB-2 (another 2D systolic array proposed by Komarek et al.) is proportional to $P \times N$. AS-2 architecture proposed in this paper has a high throughput and a relatively low resource count and is used as the motivation for the context-adaptable architecture proposed in our paper.

Ou et al. [5] propose an SAD accelerator that exploits data reuse at multiple levels and keeps I/O bandwidth to a minimum, while maintaining a high throughput. Ou proposes a Variable Block Size Motion Estimation (VBSME) engine that is capable of generating a set of 41 motion vectors for every 256 clock cycles, and the engine runs at 123 MHz. This set of 41 motion vectors are calculated for the various block sizes as required by H.264-based motion estimation. Each MB in the current frame is split into sixteen 4×4 subblocks, and the best match is identified inside a search area of size 7×7 in the reference block. SAD is used as the metric to determine the best match. SADs are calculated directly using pixel values from all the 4×4 sub-blocks and resultant 4×4 SADs are combined to generate the SADs for larger blocks. This architecture contains 256 PEs, with each PE containing a subtract unit, absolute value calculation unit, and an accumulator, and requires a total bandwidth of 1024 bits per clock cycle. Latency is computed to be 256 clock cycles. A configurable mode processor is used to combine the 4×4 SADs (and eventually the best motion vector) and compute SADs for larger blocks.

Chen et al. [6] propose a motion estimation architecture that isolates input data flow from the 2D compute array. Input data is maintained using a pipeline of registers, and data is allowed to flow through this pipeline thereby reducing I/O bandwidth. Jen et al. [7] explore the properties of data reuse that are found in an FSBME, and the associated memory bandwidth requirements and each of these properties are formulated. Also, multiple modes of scheduling data flow are identified and formulated. Other systolic array-based architectures are proposed in [8–11]. While some of these approaches cater to the need for limited flexibility in window sizes and focused only on one kernel (SAD), we extend some of these approaches to cater to a larger set of window sizes across both SAD and MSE kernels. Approaches in [2–11] were designed to support motion estimation based on Full Search (FS). Remainder of this section discusses prior work [13–15] in developing architectures that can support data flow for various search patterns other than Full Search.

Ruchika and Akoglu [13] propose a reconfigurable architecture to perform VBSME. A soft reconfigurable

router is used to support data flow for multiple search patterns, including full, diamond, and hexagon searches. A 2-dimensional array of PEs is used to compute the SAD for 4 $\times$4 blocks and 5 other PEs are used to compute the larger SADs (4×8, 8×4, 8×8, etc.). Each PE is connected to four nearest neighboring PEs and is associated with a router to send/receive packets of data using handshaking protocols. The paper claims that there is an overhead of 8 clock cycles per search point for diamond and hexagon search patterns. Our research does not use a router-based design, thus removing the overhead due to handshaking. We intend to use a 2D Configurable Register Array to support multiple data flows required by multiple search patterns. In our approach, timing overhead is found to be 2 clock cycles per search point for diamond and hexagon searches.

Marcelo Porto et al. [14] propose a high-throughput and low-cost architecture for performing diamond search based motion estimation for a block size of 16×16. This architecture is dedicated to diamond search and uses a set of memory banks to hold the required data. Pixel data from the search area in reference frame and current frame is preloaded into a monolithic memory bank (higher level of hierarchy), which is then copied into smaller and parallel memory banks (lower level of hierarchy). Apart from the initial latency to fill the data into the monolithic memory bank, the architecture is found to provide valid data to the compute engine during every clock cycle. However, such memory architecture is dedicated to a single search pattern and cannot be reused to support any other search pattern.

Zhang and Gao [15] propose a methodology to derive a reusable architecture that can support a predefined set of search patterns. Best motion vectors for four different block sizes (16×16, 16×8, 8×16, and 8×8) are computed using a set of 9 processing units (PUs). Each unit is dedicated for the computation of the cost of motion vector (which is based on SAD) for a pair of 16×16 pixel blocks from the reference frame (candidate block) and current frame (current block), respectively. During each clock cycle, two rows of pixels from two blocks (one from reference frame and one from current frame) are fed into each PU which updates the cost of motion vector for that input. A data array is used to buffer the input data from the reference frames before feeding it to nine PUs. Each row of the data array is long enough to hold the maximum number of pixels in a single row of current set of nine candidate blocks. Rows are introduced from the top of the data array and piped through to the bottom. Based on the search patterns selected and the control step, specific pixels are selected using multiplexers and fed to appropriate PUs. Also, the data array has provisions to calculate the subpixel values for fractional motion estimation. This paper fails to provide a detailed discussion of data array architecture. While the architecture does exploit data reuse among candidate blocks, it is effective only if size of the search window is limited. Also, the set of search patterns need to be predefined.

In this paper, we propose a 2D Configurable Register Array (CRA) that can support multiple search patterns, while exploiting the property of data reuse among adjacent candidate blocks in the reference frames. Amongst the prior approaches, only [13] and [15] have the capability to support multiple search patterns. While [13] suffers from the overhead associated with the router (8 clock cycles per search operation), [15] requires the search area size to be limited. Proposed CRA does not suffer from either of these limitations.

3. Compute Flow and Data Access Analysis of Arbitrary Block SAD/MSE Operations

3.1. Compute Flow Analysis. An analysis of video processing applications such as motion estimation, deinterlacing, and motion flow analysis reveals that most of the compute-intensive kernels are window-based operations [6, 16, 17]. Such operations include SAD and MSE computations for arbitrary block sizes. In this discussion, the terms "window" and "block" are used interchangeably. For example, (1–4) are commonly found in motion adaptive deinterlacing algorithms [13] and (5) is often used in motion estimation algorithms [4]:

$$\text{SAD} = \sum_{j=0}^{1}\sum_{i=0}^{2} \left|\,\text{CFB}(i,j) - \text{RFB}(i,j-2)\,\right|, \tag{1}$$

$$\text{SAD} = \sum_{j=-1}^{1}\sum_{i=-1}^{1} \left|\,\text{CFB}(i,j) - \text{RFB}(i+1,j)\,\right|, \tag{2}$$

$$\text{SAD} = \left(\sum_{j=-2}^{2} \left|\,\text{CFB}(i,j) - \text{RFB}(i+1,j)\,\right|\right) \times \text{vecmMul}, \tag{3}$$

$$\text{SAD} = \sum_{j=-1}^{2}\sum_{i=-2}^{2} \left|\,\text{CFB}(i,j) - \text{RFB}(i,j)\,\right|, \tag{4}$$

$$\text{SAD} = \sum_{j=1}^{p}\sum_{i=1}^{q} \left|\,\text{CFB}(i,j) - \text{RFB}(i+a,j+b)\,\right|. \tag{5}$$

In these equations, CFB(i,j) and RFB(i,j) represent the intensity of pixel in the (i,j) location inside pixel blocks in current and reference frames, respectively. a,b represent the Manhattan distance between the anchor point location of current block and anchor point location of reference block. Size of these blocks varies based on the operation being performed. For instance, in (1) size of the block is 3×2, and in (5) size of the block is $p\times q$, where p,q can be 4, 8, and 16, if the reference standard is H.264. In (3), *vecmMul* is a scaling factor and is a constant number. In motion flow analysis algorithms, unusual blocks of size 5×5 are used in many compute-intensive kernels [14]. Due to the wide range of window/block sizes that are used for video processing, we will analyze their data flow by decomposing such window-based operations into two suboperations:

(i) per-pixel operations,

(ii) reduction operations.

Per-pixel operations can include

(a) absolute difference between two-pixels or

(b) square of the difference between two pixels. In some of the motion flow applications, MSE based on (b) is used as an error metric instead of SAD based on (a) and is shown in (6):

$$\text{MSE} = \sum_{j=1}^{p}\sum_{i=1}^{q}\left(\text{CFB}(i,j) - \text{RFB}(i+a, j+b)\right)^2. \quad (6)$$

Reduction operations may include: (a) simple accumulation of per-pixel results or (b) scaled accumulation of per-pixel results. To support multiple per-pixel (SAD and MSE) and reduction operations (simple and scaled accumulation), each operation is realized in hardware, and routing resources (multiplexers) can be used to select between the outputs of these operations. In the proposed architecture only accumulation of results of per-pixel operations (SAD and MSE) is supported. A single window-based operation for block size $p \times q$ is illustrated using a dependence graph shown in Figure 2. Each node in this figure is a computation node represented by (i, j) coordinate, and inputs to this node are pixels from current and reference frames, CFB (i, j) and RFB (i, j), respectively. For purpose of illustration, computation shown in (5) is transformed into two stages as shown in (7) and (8):

$$\text{SAD}_j = \sum_{i=1}^{p}\left|\,\text{CFB}(i,j) - \text{RFB}(i+a, j+b)\,\right|, \quad (7)$$

$$\text{SAD} = \sum_{j=1}^{q}\text{SAD}_j. \quad (8)$$

In Figure 2, "dark" nodes represent computation of a single step in (7) (One per-pixel operation and one reduction operation), and two "light" nodes (last nodes in each column of the dependence graph) represent computation of a single step of (7) for the last pair of pixels in each column and a single step of (8) (one reduction operation) respectively. Operation of each node is illustrated in Figure 3. Section 4 discusses the design of a configurable PE array that can support parallel execution of multiple arbitrary sized block operations.

3.2. Data Access Analysis. This section discusses analysis of access patterns of reference frame pixels. Proposed architecture is aimed to support multiple search algorithms while exploiting data reuse and hence is required to support multiple data access patterns.

Figure 4 illustrates the data access inside a search area for a particular search pattern. In this illustration, a variation of the diamond search pattern is considered. Block SAD/MSE computations are performed over $p \times q$ blocks. Following notations are used in this discussion:

(i) $m \times n$: number of contiguous blocks in the current frame involved in block-based computation,

(ii) $p \times q$: size of each block,

(iii) CFB_r : rth current frame block,

(iv) nSearches : number of searches (4 in this illustration),

(v) RFB_r^s : rth reference frame block corresponding to sth search,

(vi) $(x\text{hop}_n, y\text{hop}_n)$: Manhattan distance between anchor pixels of RFB_1^n and RFB_1^{n+1},

(vii) (i, j) : pixel location.

Concurrency amongst such block-based computations can be exploited in two different ways. Concurrency is found in set of block computations between different blocks in reference frame, and different blocks in current frame for example, SAD computation between RFB_1^1 and CFB_1 and SAD computation between RFB_2^1 and CFB_2. Group of RFBs during the ith and $(i+1)$th search steps are shown in (9) and (10), respectively:

$$\text{RFBSet}_\alpha(i) = \left\{\text{RFB}_i^j \mid 1 \le j \le m \times n\right\}, \quad (9)$$

$$\text{RFBSet}_\alpha(i+1) = \left\{\text{RFB}_{i+1}^j \mid 1 \le j \le m \times n\right\}. \quad (10)$$

From Figure 4, it can be observed that there is considerable number of pixels common between the two sets ($\text{RFBSet}_\alpha(i)$ and $\text{RFBSet}_\alpha(i+1)$) that can be reused. This approach assumes that RFBs belonging to $\text{RFBSet}_\alpha(i)$ are contiguous; that is, motion vectors of contiguous CFBs are highly correlated. In case of motion involving affine or perspective warping or CFBs being processed concurrently belong to different visual objects (MPEG4 advanced profile [18]), correlation between the motion vectors of CFBs is weak. This leads to non-contiguous RFBs in $\text{RFBSet}_\alpha(i)$ and the proposed approach cannot be applied.

Concurrency is also found in set of block computations between different blocks in reference frame, and same block in current frame that is, SAD computation between RFB_1^1 and CFB_1 and SAD computation between RFB_1^2 and CFB_1. Groups of RFBs during the ith and $(i+1)$th search steps are shown in (11) and (12), respectively:

$$\text{RFBSet}_\beta(j) = \left\{\text{RFB}_i^j \mid 1 \le i \le n\text{Searches}\right\}, \quad (11)$$

$$\text{RFBSet}_\beta(j+1) = \left\{\text{RFB}_i^{j+1} \mid 1 \le i \le n\text{Searches}\right\}. \quad (12)$$

From Figure 4, it can be observed that there are no pixels common between these two sets ($\text{RFBSet}_\beta(j)$ and $\text{RFBSet}_\beta(j+1)$) that can be reused. Thus, option (a) is used to exploit concurrency and data reuse. A set of $m \times n$ blocks in the reference frame is involved in block computation with a set of $m \times n$ blocks in the current frame during a single search step. This is repeated sequentially till all searches are completed (*nSearches*). We observed earlier that there is possible data reuse between reference block sets in consecutive searches. This property can be exploited to reduce the amount of data fetched from memory at a higher level of hierarchy (e.g., $L1$ cache) that stores the reference frame. Each set of reference frame blocks consist of $m \times n \times p \times q$ pixels. Instead of fetching a new set of such pixels for every new search, it is sufficient to fetch a specific number of new rows and columns and reuse some of the pixels from the earlier set of reference frame blocks. For the ith search step, number of new rows and columns that need to be fetched

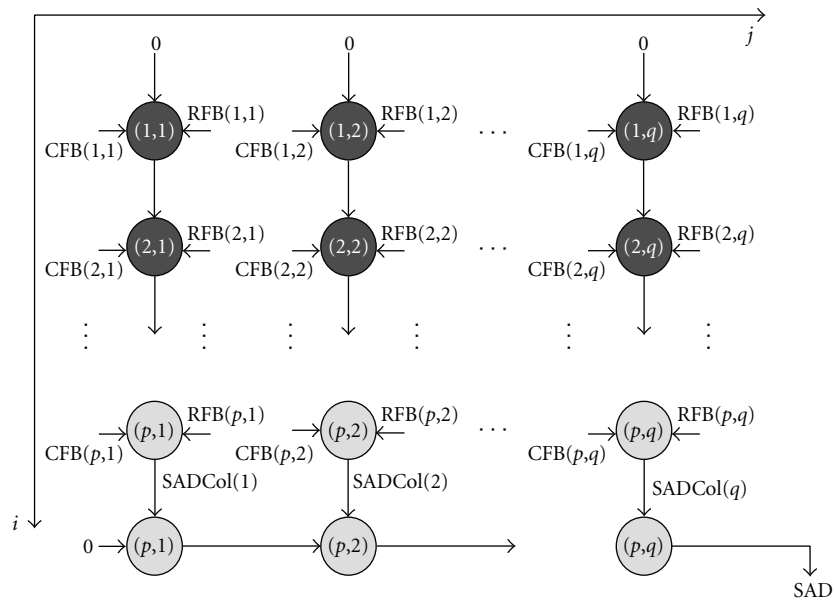

Figure 2: Dependence graph illustrating the SAD computation for block of size $p \times q$. CFB(i, j) and RFB(i, j) are reference and current block pixels fed into the system. "Dark" nodes represent computation belonging to (7), and "light" nodes represent computation belonging to (7) and (8).

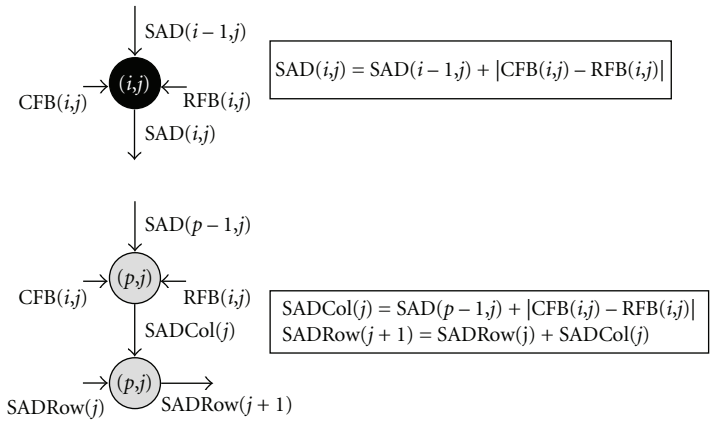

Figure 3: Operation to be performed inside each node shown in the dependence graph in Figure 2. SADCol(j) is the output of the final "dark" nodes in the jth column.

depends on $(x\text{hop}_i - 1, y\text{hop}_i - 1)$. For instance, consider the 2nd search step in Figure 4. In search step 1, set of pixels shown in (13) is fetched. Equation (14) shows the number of pixels fetched. In this discussion, RF represents reference frame, and $(x\text{hop}_i, y\text{hop}_i)$ represents the manhattan distance between the anchor pixel locations of RFB_x^i and RFB_x^{i+1}:

$$\text{RFPixelSet}(1) = \{\text{RF}(i, j) \mid i_1 \le i \le i_1 + p \times m - 1; j_1 \le j \le j_1 + q \times n - 1\}, \tag{13}$$

$$n(\text{RFPixelSet}(1)) = m \times n \times p \times q. \tag{14}$$

For fetching the pixels for next search step, new rows of pixels from north (with respect to RFPixelSet(1)) are fetched, while older rows of pixels from south are discarded. Set of pixels shown in (15) is fetched. Equation (16) shows the number of pixels fetched:

$$\text{NewRowPixelSet}(1 \longrightarrow 2) = \{\text{RF}(i, j) \mid i_1 + 1 \le i \le i_1 + x\text{hop}_1; j_1 \le j \le j_1 + q \times n - 1\}, \tag{15}$$

$$n(\text{NewRowPixelSet}(1 \rightarrow 2)) = x\text{hop}_1 \times n \times q\,. \tag{16}$$

After fetching the pixels from north, new columns of pixels from east (with respect to RFPixelSet(1)) are fetched, while older columns of pixels from west are discarded. Set of pixels

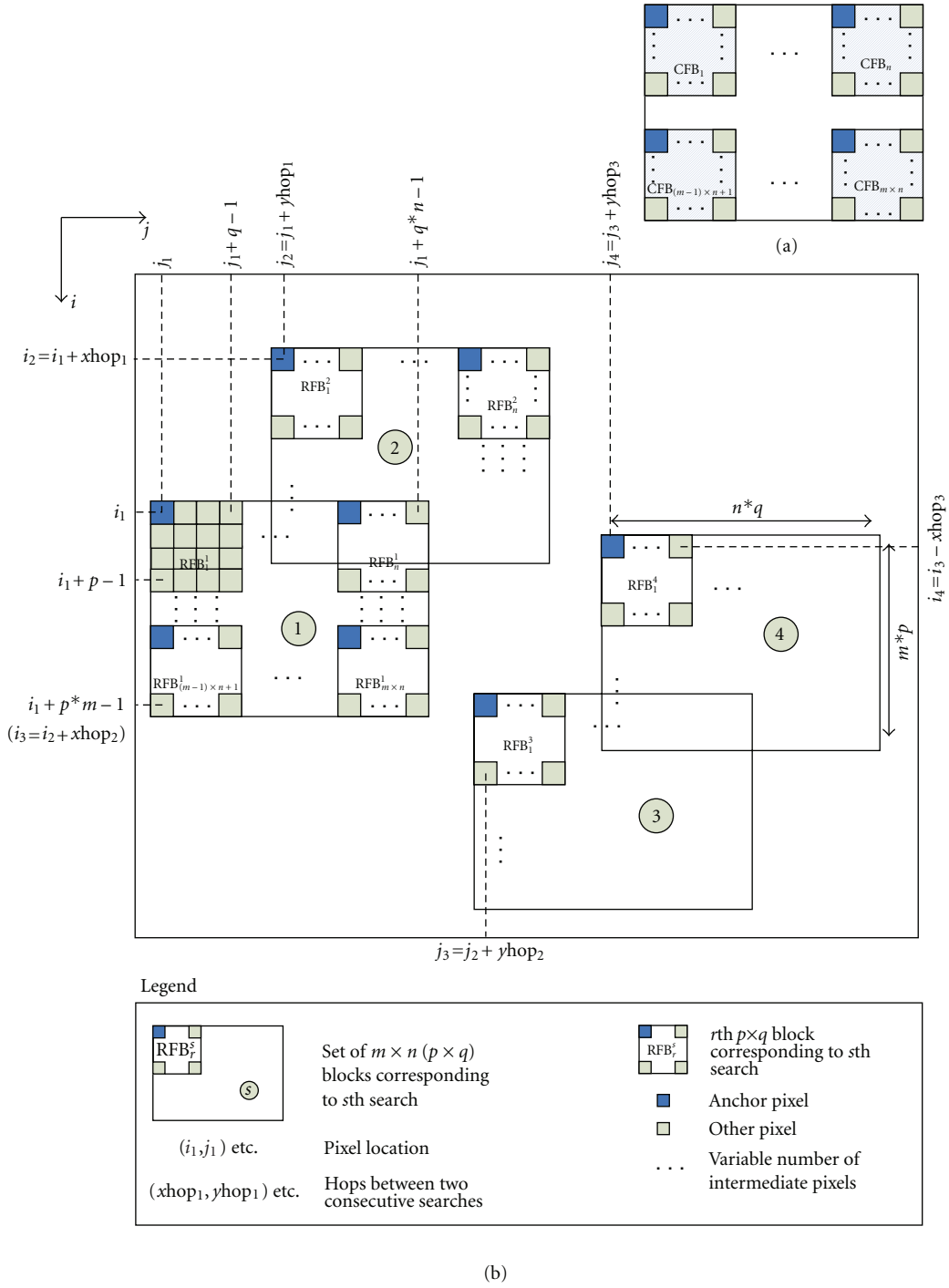

Figure 4: Illustration of data access for block size $p \times q$ and a modified diamond search pattern (a) Current frame blocks (b) Reference frame blocks inside the search area.

shown in (17) is fetched. Equation (18) shows the number of pixels fetched.

$$\begin{aligned}&\text{NewColPixelSet}(1 \longrightarrow 2)\\ &= \{\text{RF}(i,j) \mid i_1 + 1 \le i \le i_1 + p \times m - 1; j_1 + 1 \le j \le j_1 + y\text{hop}_1\},\end{aligned} \tag{17}$$

$$n(\text{NewColPixel}Set(1 \longrightarrow 2)) = \text{yhop}_1 \times m \times p. \tag{18}$$

In the proposed approach, it is sufficient to fetch n(NewRowPixelSet) + n(NewColPixelSet) pixels instead of fetching a new set of reference frame blocks for each search step. Total number of pixels that are fetched for performing all the searches (number of search steps = nSearches) is

calculated for proposed approach (np_{proposed}) as shown in (19):

$$np_{\text{proposed}} = \left(m \times p \times \sum_{i=1}^{n\text{Searches}-1} x\text{hop}_i\right) + \left(n \times q \times \sum_{i=1}^{n\text{Searches}-1} y\text{hop}_i\right) + m \times n \times p \times q. \tag{19}$$

Total number of pixels fetched for a vanilla approach that does not exploit data reuse (np_{vanilla}) is shown in (20):

$$np_{\text{vanilla}} = n\text{Searches} \times (m \times n \times p \times q). \tag{20}$$

Savings obtained in the proposed approach is computed as shown in (21). This savings reflects the reduction in bandwidth requirements of the compute unit.

$$\text{savings} = \frac{np_{\text{vanilla}} - np_{\text{proposed}}}{np_{\text{vanilla}}} \times 100. \tag{21}$$

Amount of savings is dependent on the following features of window-based operation.

(1) The larger the hops are, the lesser is the data reuse and hence the lesser is the savings.

(2) The smaller the size of the reference block set ($m \times n \times p \times q$) is, the lesser is the possibility of overlap, and hence lesser is the savings.

(3) The larger the number of searches is, the larger is the value of np_{vanilla} and hence the higher is the saving. Table 1 lists the total number of pixels fetched in the two approaches and savings obtained for some of the commonly used search patterns. Block size is assumed to be 4×4 and number of contiguous current frame blocks is also assumed to be 4×4. Number of searches depends on the search pattern and is equal to the number of candidate blocks in the reference frame for a given block in the current frame. It can be observed that a considerable amount of savings is obtained by exploiting the data reuse property.

This section analyzed the compute flow and data access patterns for SAD/MSE-based block operations with (i) arbitrary block size (ii) multiple search patterns. In this paper, we define a "context" to be a property of the block operation that is a mix of the following:

(1) per-pixel operation (SAD/MSE),

(2) block size ($p \times q$),

(3) search pattern (Full Search, Hexagon Search, etc.).

Next two sections propose an architecture template that can support multiple contexts during run-time. We also discuss the methodology to realize a prototype of this architecture on a commodity FPGA and evaluate its performance and resource overhead against fixed block size architecture.

4. Proposed Architecture

This section discusses the methodology to derive systolic array architecture for performing SAD/MSE computations (block computations) and extend it to support arbitrary block sizes. In this discussion, a control step represents a single clock period. Top-level block diagram of the proposed architecture is shown in Figure 5. It consists of two key modules: (a) a 2D Configurable Register Array (CRA), and (b) a merged PE systolic array (MPESA).

Controller and memory interface modules to facilitate the integration of this accelerator with a host processor system are also shown in Figure 5, but their discussions are beyond the scope of this paper. Remainder of this section analyzes the parallel processing of window-based operations and discusses the two key modules of the proposed architecture.

4.1. Parallel Processing of Window-based Operations and Bandwidth Requirements. In video processing applications discussed in Section 3 (motion estimation, deinterlacing, etc.), it is observed that the window-based operation is performed repeatedly to compute error metric between pairs of multiple contiguous blocks of pixels in the reference and current frame. Window-based operations for all pairs of blocks can be performed in parallel, but amount of parallelism supported by any hardware accelerator is limited by two factors.

(1) *Number of processing elements (PE) available in the proposed 2D merged array*. A single PE operates on one pair of pixels. This number is constrained by the total number of resources available on an FPGA (DSP48, LUTs, Flip Flops, etc.). Number of PEs is represented by $M \times N$.

(2) *Memory bandwidth available to the PEs*. This number depends on the implementation of the 2D CRA and is equal to output bandwidth of the CRA

For a given block size $p \times q$ and 2D merged PE array of size $M \times N$, proposed architecture can support parallel execution of window-based operations for $\lfloor M/p \rfloor \times \lfloor N/q \rfloor$ block pairs. For a block size $p \times q$, (22) lists the $\lfloor M/p \rfloor \times \lfloor N/q \rfloor$ blocks that can be processed in parallel during the first control step from the reference frame:

$$\text{RFB}_1^1 = \{\text{RFB}(i,j) 1 \le i \le p; 1 \le j \le q\},$$
$$\text{RFB}_{\text{last}}^1 = \left\{\text{RFB}(i,j) \mid \frac{M}{p} \times p - p + 1 \le i \le \frac{M}{p} \times p; \frac{N}{q} \times q - q + 1 \le j \le \frac{N}{q} \times q\right\}. \tag{22}$$

Similarly, $\lfloor M/p \rfloor \times \lfloor N/q \rfloor$ subblocks for the current frame (CFBs) CFB_1 to CFB_{last} are defined. Figure 6 illustrates the set of 5×5 reference frame blocks that are processed during the first control step for $M = N = 16$ and $p = q = 3$. In this figure, (i, j) is used to represent the relative position of the pixels in the reference frame blocks. ($i = 1$, $j = 1$) represents

Table 1: Savings obtained (in terms of pixels fetched) by exploiting data reuse.

Search pattern	Number of searches	np_{proposed}	np_{vanilla}	Savings (in %)
Full search [5]	16	496	4096	87.9
Diamond search [15]	8	480	2048	76.56
Hexagon search [13]	6	464	1536	68.23
Big cross search [15]	12	800	3072	73.96

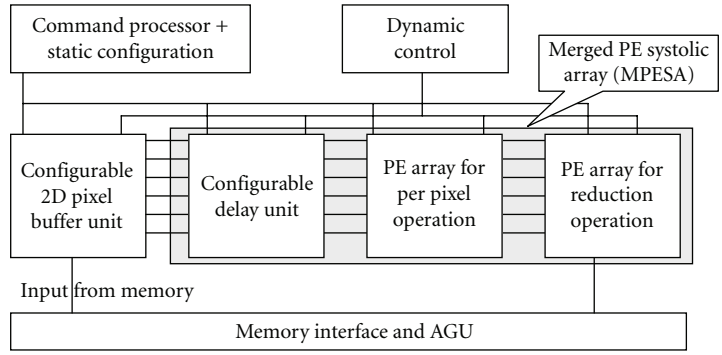

Figure 5: Top-level block diagram of the proposed context-adaptable SAD/MSE accelerator.

the top and leftmost pixel in the first 3×3 block processed during the first control step.

In many video processing applications, it is observed that window-based operations are performed between multiple blocks in the reference frame (also called candidate blocks) and a given block CFB_x (xth block in the current frame), during a single control step. Equation (23) lists one possible list of candidate blocks corresponding to CFB_1 for a Full Search pattern ($p = q = 3$), and this list of blocks is illustrated in Figure 7:

$$\begin{aligned}
\text{RFB}_1^1 &= \{\text{RFB}(i,j)1 \le i \le p; 1 \le j \le q\},\\
\text{RFB}_1^2 &= \{\text{RFB}(i,j)2 \le i \le p+1; 1 \le j \le q\},\\
\text{RFB}_1^3 &= \{\text{RFB}(i,j)2 \le i \le p+1; 2 \le j \le q+1\},\\
\text{RFB}_1^4 &= \{\text{RFB}(i,j)1 \le i \le p; 2 \le j \le q+1\}.
\end{aligned} \tag{23}$$

Number of candidate blocks in the reference frame for each current block (for a given block index x) in the current frame is represented in (24) in terms of the number of searches that need to be performed, as discussed in Section 3:

$$n\left(\text{RFB}_x^y\right) = n\text{Searches}. \tag{24}$$

In this paper, window-based operations for the following pairs of blocks are started in parallel by the proposed merged PE architecture in one control step:

$$\text{CFB}_x \text{ and } \text{RFB}_x^y \quad \forall\ 1 \le x \le \left\lfloor \frac{M}{p} \right\rfloor \times \left\lfloor \frac{N}{q} \right\rfloor. \tag{25}$$

During subsequent control steps, y is varied from 1 to *nSearches*. During each control step, a bandwidth of $\lfloor M/p \rfloor \times p \times \lfloor N/q \rfloor \times q$ is required by the PE array from the memory support system used to store the current and reference frames. Pixels from current frame do not vary between control steps, and memory support system to provide the current frame pixels is simple and is not discussed here. However, pixels from reference frame vary based on the control step y, and the memory support system is required to provide the necessary bandwidth, during every control step.

4.2. 2D Configurable Register Array. This section discusses the derivation of proposed 2D Configurable Register Array (CRA) and estimates the resource requirements.

4.2.1. Architecture Derivation. From Figure 7, it is observed that there is ample data reuse between the set of RFB_x^y and set of RFB_x^{y+1} blocks accessed in adjacent control steps. This data reuse depends on how the blocks are accessed within the search area in the reference frame. Analysis in Section 3 showed considerable data reuse for various search patterns. This paper proposes a 2D Configurable Register Array (CRA) to exploit this property of data reuse. Memory support system for reference frame comprises of external memory connected to the proposed 2D CRA that, in turn, is connected to the merged PE array. Such a design helps to reduce bandwidth that is required from external memory, while providing sufficient number of pixels to the merged PE array. Architecture of CRA is illustrated in Figures 8 and 9 illustrates the inner design of each register node. In Figure 8, three types of register nodes are shown. Nodes labeled $B(i,j)$ represent the units that feed data to $\text{PE}(i,j)$, which is the (i,j)th PE in the PE array. Nodes $B'(i,j)$ and $B''(i,j)$ are required by the configurable delay unit, which is part of the PE array design. Since the proposed 2D systolic array architecture consists of $M \times N$ PEs, number of "B" nodes in

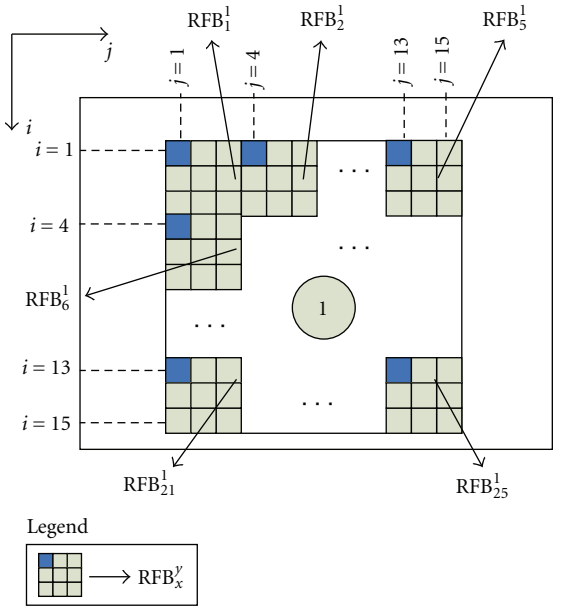

Figure 6: Illustration of reference frame blocks processed during a single control step for $p = q = 3$ and $M = N = 16$.

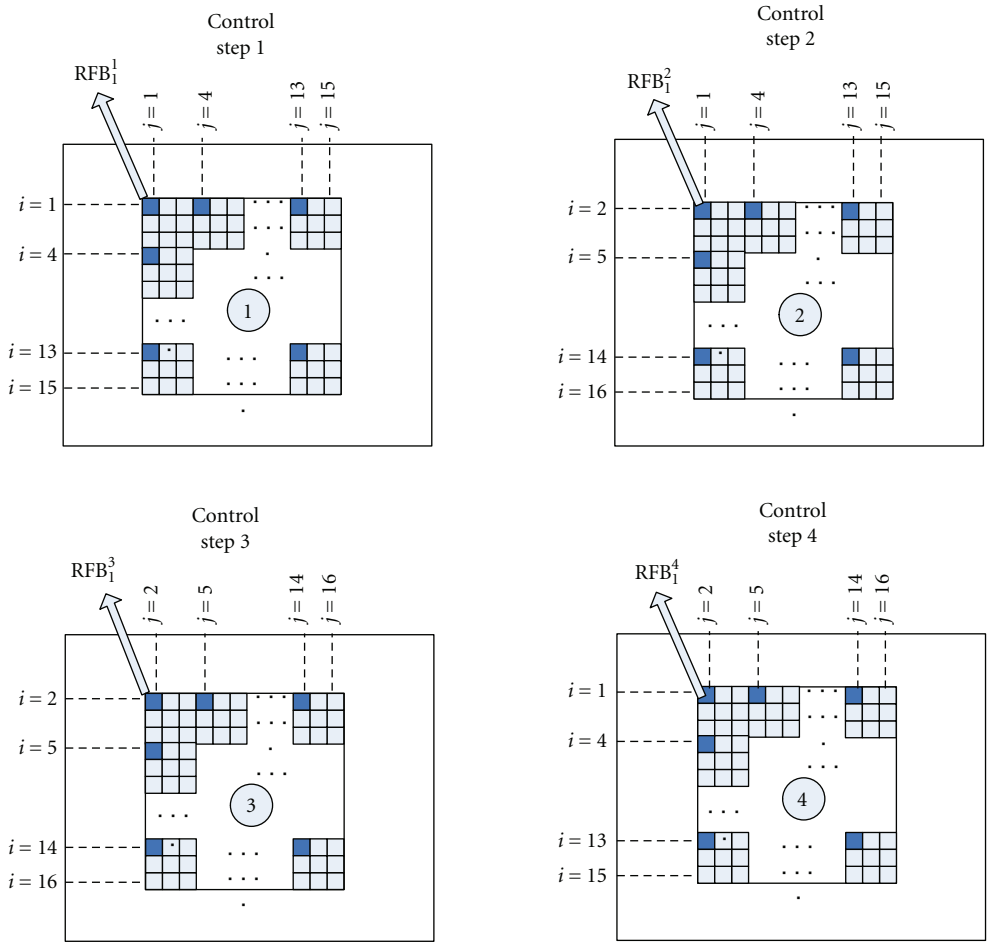

Figure 7: Illustration of candidate blocks and three types of data flow amongst candidate blocks over multiple control steps.

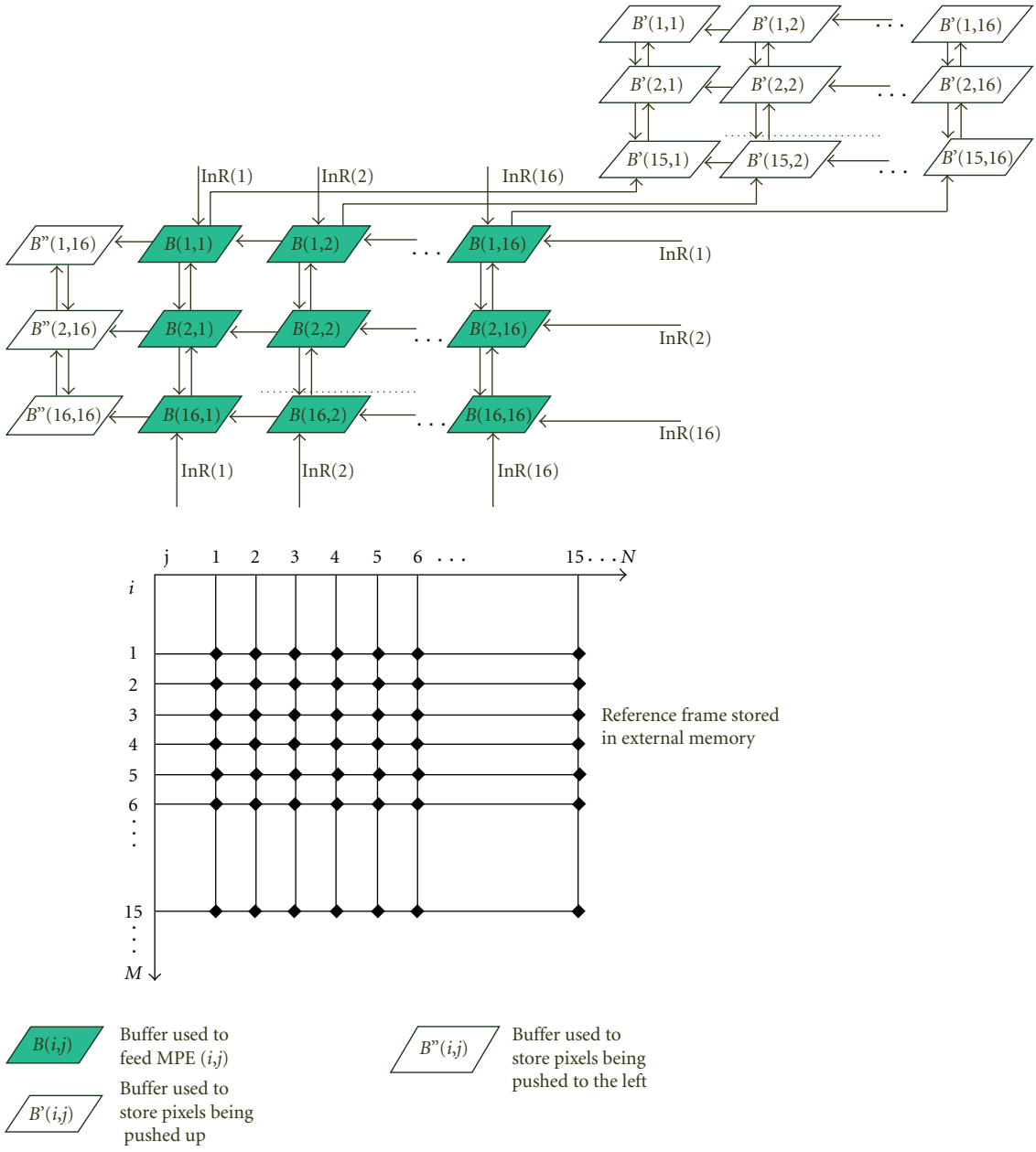

Figure 8: 2D Configurable register array.

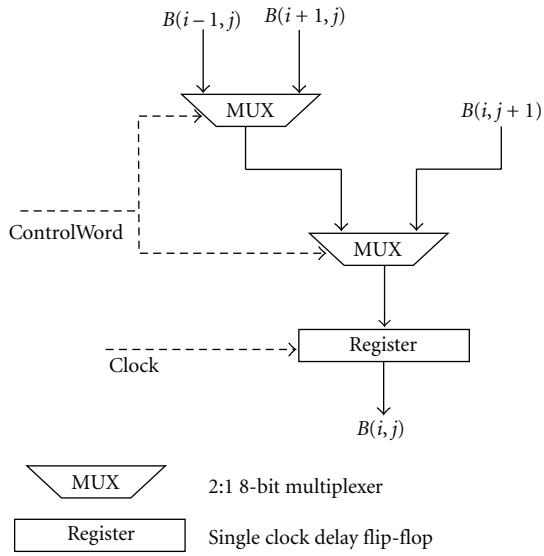

Figure 9: Inner design of $B(i, j)$ node in 2-D CRA. *ControlWord* is derived based on the data flow during a given control step.

CRA is also fixed to be $M \times N$. Based on block size $p \times q$, only a portion of CRA is required to provide $\lfloor M/p \rfloor \times p \times \lfloor N/q \rfloor \times q$ contiguous columns of pixels to the 2D systolic array during every control step. For a block size of 3×3 and 2-D array size of 16×16, 15×15 register nodes are utilized, and the remaining 31 nodes are wasted. This is one of the overheads to support arbitrary block sizes and formulated in (26):

$$\text{NodesWasted} = M \times N - \lfloor M/q \rfloor \times \lfloor N/q \rfloor. \quad (26)$$

Proposed 2D CRA exploits the property of data reuse based on how the candidate blocks are accessed from the reference frame. Table 2 shows the set of operations that modify the contents of CRA for each control step. Based on the data flow required during a control step, the operation varies. In these operations, all reference frame pixels are assumed to be stored in a memory system, and this memory system is capable of providing M pixels from a predetermined location in a single row or N pixels from a predetermined location in a single column in the reference frame. Such a set of M (or N) pixels is represented by the vector *InR*. Input bandwidth of the 2D CRA can thus be defined as $\max(M, N)$.

Operation of the 2D CRA is illustrated here for block size 3×3, 2D merged PE array of size 16×16 and the search area is defined with four candidate blocks (for sake of illustration). Figure 7 illustrates the access pattern within the reference frame across multiple control steps (for Full Search). During the first 16 control steps, a new row of pixels is fed into the 2D CRA (see Table 2). During control step x, vector *InR* is fetched from the reference frame as shown in (27):

$$\text{InR}(j) = \text{RFB}(x, j) \quad \forall\ 1 \le j \le 16. \quad (27)$$

After 16 control steps, CRA consists of the following pixels:

$$B(i, j) = \text{RFB}(i, j) \quad \forall\ 1 \le i \le 16; 1 \le j \le 16 \quad (28)$$

2D CRA now contains all the pixels corresponding to the first set of 25 candidate blocks: RFB_1^1, RFB_2^1, RFB_3^1,..., RFB_{25}^1. Proposed systolic array architecture can now start all 25 window-based operations in parallel. In control step 17, a new row is fetched as follows:

$$\text{InR}(j) = \text{RFB}(17, j) \quad \forall\ 1 \le j \le 16. \quad (29)$$

Data is now shifted upwards and 2D CRA now contains all the pixels corresponding to the second set of candidate blocks, namely, RFB_1^2, RFB_2^2, RFB_3^2, ... , RFB_{25}^2. In control step 18, a new row is fetched as follows:

$$\text{InR}(j) = \text{RFB}(i, j) \quad \forall\ 2 \le j \le 17. \quad (30)$$

Data is now shifted eastward and 2D CRA now contains all the pixels corresponding to the third set of candidate blocks. This process is continued till all candidate blocks have been fed to the 2D PE array. The last set of candidate blocks is presented to the systolic array after the $15 + n$Searches control steps.

It is observed that the proposed 2D CRA provides the required set of candidate blocks to PE array during each control step (after the initial latency). This observation is applicable only in the case of a Full Search based window-based operations. To support window-based operations involving other search patterns (diamond search, etc.), 2D CRA operates in a similar fashion but is unable to provide the required set of candidate blocks to PE array during each control step. This is a tradeoff to support multiple search patterns.

4.2.2. Resource Requirements and Performance Estimates. This subsection discusses an analytical model to estimate overall resource and input bandwidth requirements of the proposed 2D CRA for arbitrary block sizes ($p \times q$) and systolic array sizes ($M \times N$). Based on Figure 8, we compute

Table 2: 2-D CRA operation based on data flow during one control step.

Data flow step	Boundary node operation	Other node operation
New row from bottom	$B(M, j) = \text{InR}(j)$ for all $1 \le j \le N$	$B(i, j) = B(i-1, j)$ for all $2 \le i \le M; 1 \le j \le N$
New row from top	$B(1, j) = \text{InR}(j)$ for all $1 \le j \le N$	$B(i, j) = B(i+1, j)$ for all $1 \le i \le M-1; 1 \le j \le N$
New column from right	$B(i, N) = \text{InR}(i)$ for all $1 \le i \le M$	$B(i, j) = B(i, j+1)$ for all $1 \le i \le M; 1 \le j \le N-1$

the total number of register nodes here. In this discussion, $n(a)$ represents number of duplicate units of "a" that need to be implemented:

$$\begin{aligned} n(\text{BufNodes}) &= n(B) + n(B') + n(B'') \\ &= M \times N + M + (M-1) \times N \\ &= 2 \times M \times N + M - N. \end{aligned} \tag{31}$$

Each register node is shown in Figure 9 and consists of two 2 : 1 8-bit multiplexers and one 8-bit register. Multiplexers and registers can be realized using Look-Up Tables (LUTs) and Flip-Flops (FFs), which are the primitive resources in Xilinx FPGAs. Total resource requirements for the proposed 2D CRA are shown in (32) and (33):

$$\begin{aligned} n\text{LUT}(\text{BufNodes}) &= (2 \times M \times N + M - N) \\ &\quad \times \left(2 \times n\text{LUT}(\text{Mux}) + n\text{LUT}(\text{Re}\,g)\right), \end{aligned} \tag{32}$$

$$\begin{aligned} n\text{FF}(\text{BufNodes}) &= (2 \times M \times N + M - N) \\ &\quad \times \left(2 \times n\text{FF}(\text{Mux}) + n\text{FF}(\text{Re}\,g)\right). \end{aligned} \tag{33}$$

Here, nLUT(x) denotes number of LUTs used to realize the FPGA design of "x", and nFF(x) denotes number of FFs used to realize the FPGA design of "x". Also, the CRA requires an input bandwidth of $\max(M, N)$ to be available from the external memory. Performance is estimated in terms of (a) Bandwidth provided by CRA, (b) initial latency and (c) total latency due to idle cycles caused by hops in the search pattern.

Effective bandwidth is computed in (38)using these three factors that are listed in(34), (35), (36):

$$\text{Band width}(\text{BufNodes}) = M \times N, \tag{34}$$

$$\text{Initial Latency}(\text{BufNodes}) = M, \tag{35}$$

$$\begin{aligned} &\text{Total LatencyPerSP}(\text{BufNodes}) \\ &= \sum_{i=1}^{n\text{Searches}-1} \left(\text{xhop}_i + \text{yhop}_i - 1\right). \end{aligned} \tag{36}$$

Proposed 2D CRA provides $M \times N$ pixels to the 2D PE array during each clock cycle. Number of clock cycles required to send a new set of RFBs (corresponding to the search pattern) is nSearches. Total number of clock cycles spent in sending all the required sets of RFBs is computed in (37).

Table 3: Effective bandwidth (in terms of number of pixels per clock cycle) provided by the proposed 2D CRA.

Search pattern	Number of searches	$\text{bandwidth}_{\text{eff}}$
Full search [5]	16	132
Diamond search [15]	8	68
Hexagon search [13]	6	53
Big cross search [15]	12	61

$$\begin{aligned} &\text{Total Latency}(\text{BufNodes}) \\ &\quad = \text{Initial Latency}(\text{BufNodes}) \\ &\qquad + \text{Total Latency Per SP}(\text{BufNodes}) \\ &\qquad + n\text{Searches}. \end{aligned} \tag{37}$$

Effective bandwidth is a measure of the amount of useful data that the 2D CRA is able to feed the 2D PE array for a single clock cycle. The greater the effective bandwidth is , the better is the performance of 2D CRA. It is computed as shown in (38):

$$\begin{aligned} &\text{Band width}_{\text{eff}}\,(\text{BufNodes}) \\ &= \frac{\text{Total Pixels}}{\text{Total Latency}(\text{BufNodes})} \\ &= \frac{\lfloor M/p \rfloor \times p \times \lfloor N/q \rfloor \times q \times n\text{Searches}}{M + \sum_{i=1}^{n\text{Searches}-1} \left(x\text{hop}_i + y\text{hop}_i\right)}. \end{aligned} \tag{38}$$

Table 3 lists the effective bandwidth for $M = N = 16$ and $p = q = 4$ for various search patterns. An ideal 2D CRA with the best performance should be able to provide $M \times N$ (256) pixels per clock cycle. Reduced performance of proposed CRA is a tradeoff to support multiple search patterns with a limited input bandwidth.

4.3. Merged PE Array. This section discusses the merged PE systolic array (MPESA) architecture required to compute multiple window-based operations for arbitrary block sizes. During every control step (after initial latency), proposed 2D CRA is responsible for providing the array architecture with a new set of reference frame blocks that are candidates for a given set of blocks from the current frame. For search patterns other than Full Search, some idle cycles are introduced as the CRA needs to prepare the next set of candidate blocks. During these cycles, the PE array continues to read the data from 2D CRA and process them, but the

results are ignored. Following subsections discuss the PE array architecture in detail.

4.3.1. Architecture Derivation. The merged PE array needs to support three different modules.

(i) *Configurable delay unit.* This unit is responsible for delaying the reference frame pixels according to the block size, 2D array size, and location of PE in the 2D array.

(ii) *PEs for per pixel operation.* This unit is responsible for computing the absolute difference or square error between a pair of pixels from the current frame and reference frame.

(iii) *PEs for reduction operation.* This unit is responsible for computing the summation (or weighted summation) of the results of the PE array for per pixel operation.

Proposed methodology to derive MPESA architecture begins at the level of a base architecture to support a single window-based operation involving a pair of reference and current frame blocks. This operation is represented in the form of a dependence graph, as shown in Figure 2, and is well suited to be processed using a systolic array architecture. Figure 2 displays the dependence graph for the computation of SAD operation (block size = $p \times q$).

To map the dependence graph onto systolic array architecture, we need to determine the projection vector, and scheduling vector [12]. It should be noted that there is no requirement for a projection vector for mapping a 2-dimensional dependence graph to a 2-dimensional systolic array. After evaluating multiple options, it is observed that $(1, 0)$ is the best option for a scheduling vector and this obeys all the rules required for a systolic array design. Hence a systolic array based on this schedule vector is chosen as a base architecture that will be used to perform block computations. This systolic array is illustrated in Figure 10. While the pixels from the current frame are stored in place in each processing element (PE), $p \times q$ pixels from sets of reference frame blocks (X, Z, A) are fed into different PEs of the systolic array from the 2D CRA. Arrows marked "D" indicate a single clock delay associated with data transfer between PE and arrows marked (P)D indicate a delay of P clock cycles. Such delays are realized using a series of edge-triggered register units. In this figure, X_{ij} corresponds to $\mathrm{RFB}_1^1(i, j)$, and $\mathrm{RFB}_1^1(1, 1)$ represents the top and leftmost pixel in the RFB_1^1 block. Similarly Z_{ij} corresponds to $\mathrm{RFB}_1^2(i, j)$, and A_{ij} corresponds to $\mathrm{RFB}_1^3(i, j)$.

Two types of PEs are required in this systolic array. "Dark" PE performs the computation corresponding to "dark" node shown in Figures 2 and 3. "Light" PE performs computations corresponding to "light" node shown in Figures 2 and 3. $(p, j)_1$ performs the computation corresponding to (7) and feeds the result after a variable delay to $(p, j)_2$ which, in turn, performs the computation corresponding to (8). To support the computation of SADs for pairs of multiple contiguous blocks (as illustrated in Section 3), proposed systolic array is replicated, and an $M \times N$ PE Systolic Array (PESA) is realized. Based on block size, multiple configurations of PESA are required. Figure 11 shows the configuration of PESA for two possible block sizes for a given number of PEs. M and N are set to be 6 for the purpose of illustration:

(1) PESA1: $(p = q = 3)$,

(2) PESA2: $(p = q = 2)$.

In each PESA, each PE is represented using its coordinate (i, j). Following requirements guide us towards the design of a merged PESA to support arbitrary block sizes:

(i) Type of PE ("light" or "dark") depends on row index of the PE and block size (e.g., PE(3,3) is "light" PE in PESA1 and "dark" PE in PESA2).

(ii) Based on row index of a specific PE(i, j) and block size, either a zero or the intermediate result from the PE$(i - 1, j)$ is routed into this specific PE. (in PESA1, a zero is routed to PE$(4, 1)$. in PESA2, result from PE$(3, 1)$ is routed).

(iii) Based on column index of a specific PE(i, j) and block size, either a zero or the intermediate result from PE$(i, j - 1)$ is routed into this specific PE. (in PESA1, a zero is routed to PE(6,4). In PESA2, result from PE(5,4) is routed).

(iv) Input fed into each PE(i, j) is delayed by specific number of clock cycles. This depends on row index of the PE (i) and block size. (in PESA1, input to PE$(6, 1)$ is delayed by 2 clock cycles. In PESA2, input is delayed by 1 clock cycle).

(v) Data transfer within "light" PE is delayed by specific number of clock cycles. This depends on column index of the PE (j) and block size. (in PESA1, data within PE$(6, 3)$ is delayed by 3 clock cycles. In PESA2, data is delayed by 1 clock cycle).

Remainder of this section discusses the design of a merged PE Systolic Array (MPESA) that can be configured to support arbitrary block sizes. Each PE in MPESA can possibly be a "dark" PE or a "light" PE, as discussed above. By analyzing the computations performed by each type of PE, it is observed that the computation performed by "dark" PE is also performed by "light" PE. Thus, every PE in MPESA is realized as a "light" PE. Each PE is termed as a merged processing element (MPE) and can support (a) both MSE and SAD computations and (b) arbitrary block sizes. Each MPE is represented by its location in the MPESA and is referred to as MPE(i, j). Figure 12(a) shows the design of MPE(i, j). Figure 12(b) shows the internal design of the per-pixel operator which shows that the MPE can support both MSE and SAD computations. A pair of configurable delay units is included in this design to support arbitrary values of p and q. $(i\mathrm{MPE}_{\mathrm{Inter}} - 1, j)$ represents the intermediate results of computation within a single column. This corresponds to SAD(i, j), as shown in Figure 3. $\mathrm{MPE}_{\mathrm{Final}}(i, j)$ corresponds to SAD(p, j), as shown in Figure 3. Other terms used in Figure 12 are explained later.

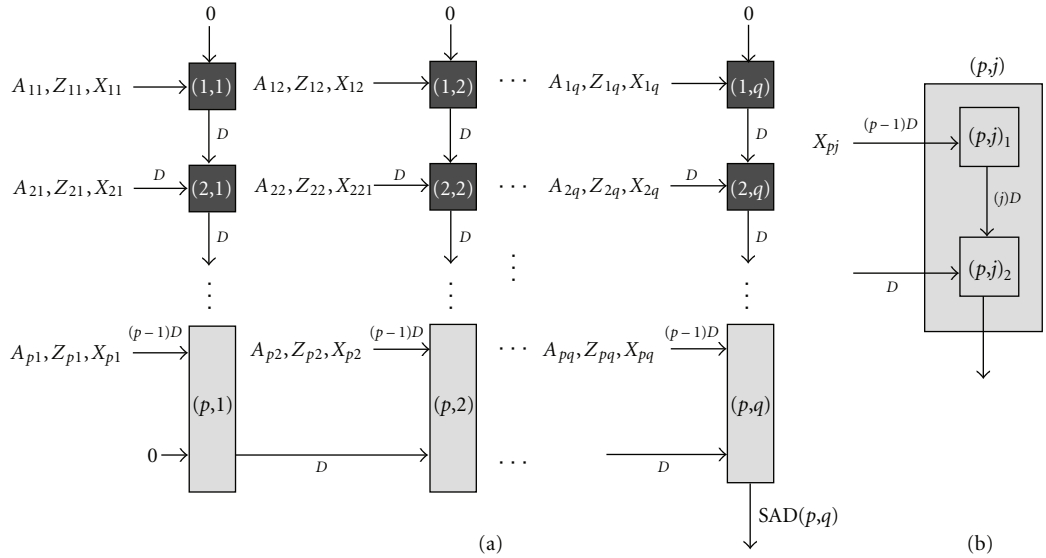

FIGURE 10: (a) Systolic array architecture for block computation ($p \times q$). Current pixels are stored in-place and reference pixels from blocks (X, Z, A, etc.) are fed into the array. D represents a single clock cycle delay. (b) Design of "light" PE. It is shown as two modules: $(p, j)_1$ and $(p, j)_2$.

Salient features of the proposed MPESA (and MPE) architecture and its derivation are explained with an example. A computation is assumed to start at control step CS = 16. At this instant, first set of blocks from the reference frame and current frame are available to the appropriate PEs from the 2D CRA (CFB(i, j) and RFB(i, j) are provided to MPE(i, j)). RFB(1, 1) corresponds to the top and leftmost pixel in the set of RFBs. RFB(i, j) is delayed before it is fed to the first compute unit (absolute difference or square error unit). For a given MPE(i, j), this delay depends on the row index "i" and is computed as follows:

$$\text{NumDelayRFB} = [(i-1) \pmod{p}]. \tag{39}$$

This value can be anywhere between 0 and $N - 1$. A configurable delay unit for reference pixels is derived to generate appropriate delays. We observed that delayed copies of the reference frame pixels are available in the 2D CRA. Thus, the configurable delay unit can be realized by reusing the registers present in the 2D CRA. Resource overhead for a single MPE is reduced to a single $N : 1$ multiplexer. This design is discussed in the next subsection. A single clock delay register is used to hold the intermediate result of ADDER number 1 before sending it to the next PE (MPE($i+1, j$)) or to the accumulator unit (ADDER number 2). It is observed that the MPE that is configured to act as "dark" PE sends the intermediate result of ADDER number 1 to MPE($i + 1, j$) after a single clock cycle and the result of ADDER number 2 is ignored by MPE($i, j + 1$). MPE that is configured to act as "light" PE sends the intermediate result of ADDER number 1 to ADDER number 2 after a specific number of clock cycles based on the location of the MPE (i, j) and the block size (as shown in Figure 10). This number depends on column index of the MPE(i, j) in the 2D MPESA and block size and is computed as follows:

$$\text{NumDelayInterSAD} = [(j-1) \pmod{q}]. \tag{40}$$

This value can be anywhere between 0 and $M - 1$. Such a configurable delay unit is realized using a circuitry involving multiple registers (single cycle delay) and one M:1 multiplexer. This is illustrated in Figure 13 (for $M = 16$). Based on the location of the MPE in a 2D MPESA and the value of p and q, inputs to select lines of the 2:1 multiplexers shown in Figure 12 that control the selection between zero and an intermediate value are modified. This is shown in (41)and (42):

$$\text{IsFirstPEI}n\text{Col} = [(i-1) \pmod{p}] == 0, \tag{41}$$

$$\text{IsFirstPEI}n\text{Row} = [(j-1) \pmod{q}] == 0, \tag{42}$$

This merged PE (MPE) is now used as the building block of a 16×16 MPESA (similar to the one illustrated in Figure 12). A possible list of block sizes and number of such block computations that can be performed during a single control step using the MPESA, for $M = N = 16$, is shown as follows:

(i) one 16×16 block,
(ii) Four 8×8 blocks,
(iii) Sixteen 4×4 blocks,
(iv) Nine 5×5 blocks,
(v) Fifteen 5×3 blocks,
(vi) Twenty-five 3×3 blocks.

4.3.2. Resource Requirement and Performance Estimates. This subsection discusses an analytical model to estimate overall

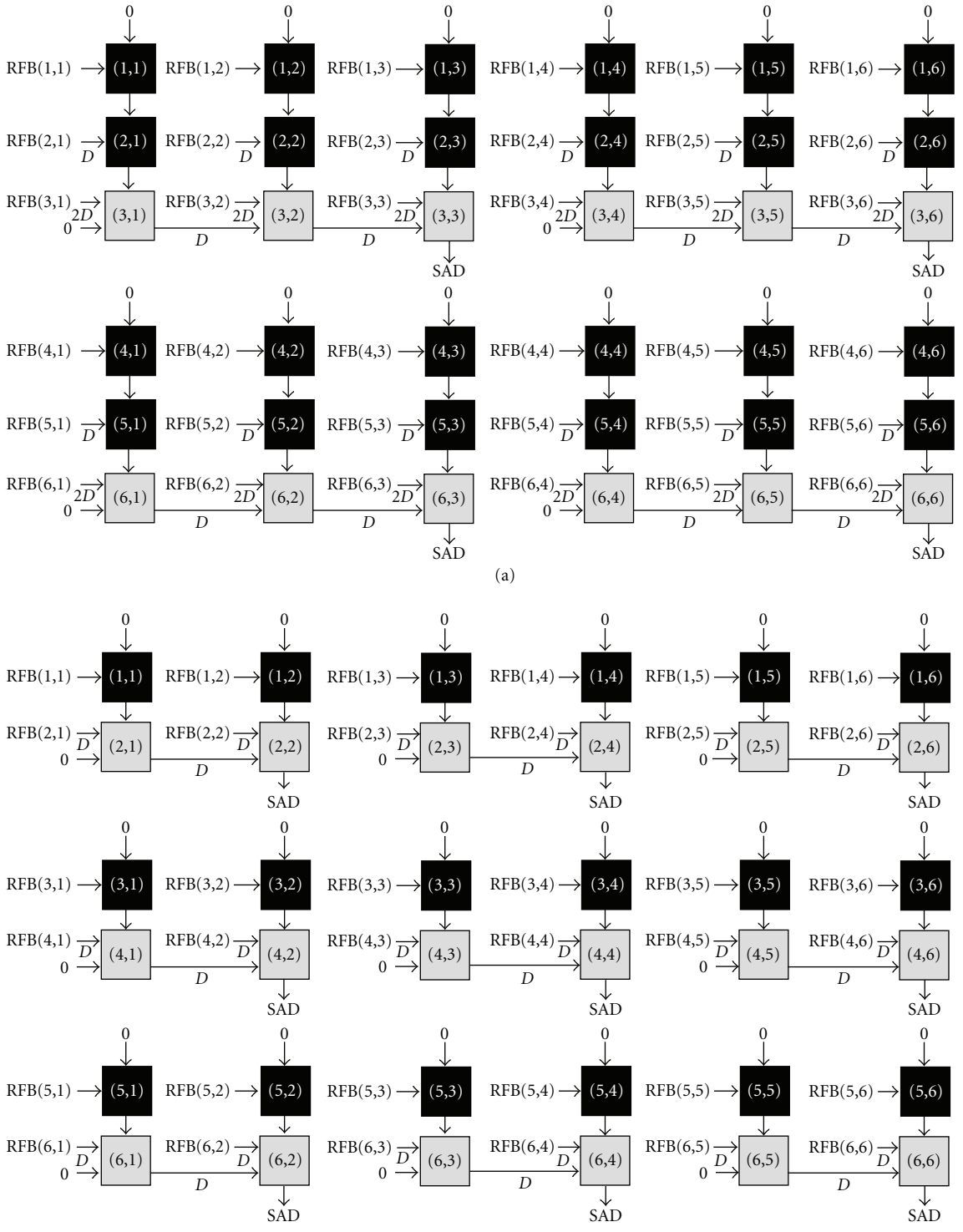

Figure 11: PESA configuration for block sizes (a) PESA1: $p = q = 3$ and (b) PESA2: $p = q = 2$ for $M = N = 6$.

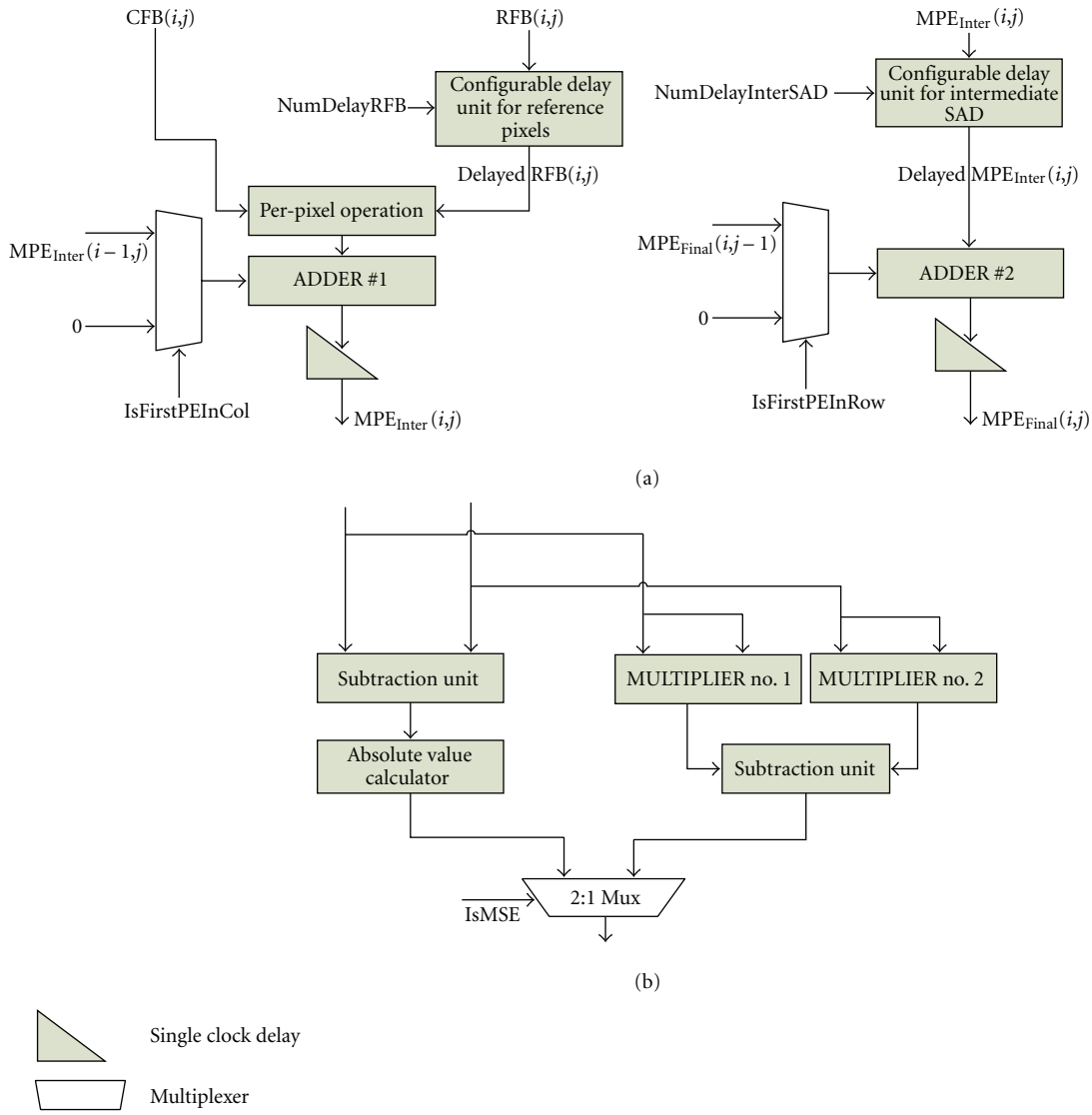

FIGURE 12: (a) Proposed MPE architecture. $\text{MPE}_{\text{Inter}}(i, j)$ and $\text{MPE}_{\text{Final}}(i, j)$ are two results generated. $\text{MPE}_{\text{Inter}}(i, j)$ is also reused inside the current MPE. (b) Per-pixel operation elaborated.

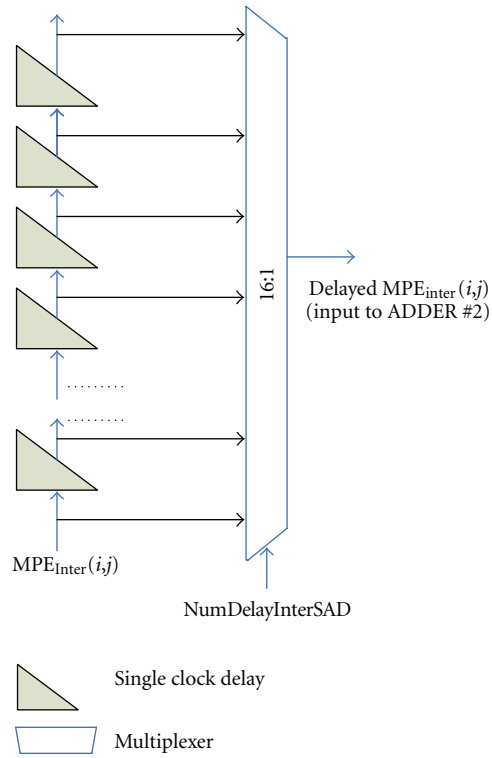

Figure 13: Resource overhead required to route output of ADDER number 1 into ADDER number 2 (refer Figure 12) after appropriate delays.

resource requirement for MPESA design. MPESA consists of $M \times N$ MPEs arranged in 2D systolic array architecture. Resource requirement for each MPE is calculated as shown in (43):

$$\begin{aligned} n\text{LUT(MPE)} = {} & n\text{LUT}(\text{CDU}_{\text{RP}}) + n\text{LUT(ABSDIFF)} \\ & + n\text{LUT(MULTIPLIER)} \\ & + 2 \times n\text{LUT(ADDER)} + n\text{LUT}(CDU_{IS}) \\ & + 2 \times n\text{LUT}(REG) + 2 \times n\text{LUT(MUX)}, \end{aligned} \tag{43}$$

CDU_{RP} is the configurable delay unit for reference pixels,

ABSDIFF is the absolute difference unit (for SAD calculations),

MULTIPLIER is the multiplier unit (for MSE calculations),

CDU_{IS} is the configurable delay unit for intermediate results.

$n\text{LUT}(\text{CDU}_{\text{RP}})$ and $n\text{LUT}(\text{CDU}_{\text{IS}})$ are computed as shown in (44) and (45), respectively. nLUT(MPESA) is computed in (46). nFF(MPESA) is computed in a similar fashion:

$$n\text{LUT}(\text{CDU}_{\text{RP}}) = (M-1) \times n\text{LUT(MUX)}, \tag{44}$$

$$\begin{aligned} n\text{LUT}(\text{CDU}_{\text{IS}}) = {} & (N-1) \times n\text{LUT(MUX)} \\ & + N \times n\text{LUT(REG)}, \end{aligned} \tag{45}$$

$$n\text{LUT(MPESA)} = M \times N \times n\text{LUT(MPE)}. \tag{46}$$

Performance of MPESA is estimated in terms of (a) Latency and (b) Throughput. Latency is number of clock cycles required by MPESA to compute the first set of valid SAD outputs. It is a function of block size and is computed as shown in (47):

$$\text{LatMPESA} = p + q. \tag{47}$$

Throughput (number of pixels processed per clock cycle) is computed as shown in (48). Throughput is affected by the following factors:

(i) latency of MPESA,

(ii) total number of clock cycles for CRA to send all blocks,

(iii) total number of pixels:

$$T\text{put MPESA} = \frac{\text{Total Pixels}}{\text{Lat MPESA} + \text{Total Latency(BufNodes)}}. \tag{48}$$

Table 4: MPESA throughput (in terms of number of pixels per clock cycle).

Search pattern	Number of searches	TotalLatency(BufNodes)	TputMPESA
Full search	16	31	107.8
Diamond search	8	30	55.35
Hexagon search	6	29	42.67
Big cross search	12	50	53.9

Table 5: Illustration of the MPESA operation for block size (3×3) on the first set of candidate subblocks.

Control step (y_{curr})	OpMPE(y_{curr},16) ∪	Operation of one MPE(i, j)	Num DelayY	NumDelay InterSAD	IsFirstPEInCol	IsFirstPEInRow
16	{MPE(i, j) | $i\epsilon\{1, 4, 7, 10, 13\}$; $1 \le j \le 16$}	$\text{MPE}_{\text{Inter}}(1,1) = 0 + \lvert\text{CFB}(1,1) - \text{RFB}(1,1)\rvert$	0	na	1	na
17	{MPE(i, j) | $i\epsilon\{2, 5, 8, 11, 14\}$; $1 \le j \le 16$}	$\text{MPE}_{\text{Inter}}(2,1) = \text{MPE}_{\text{Inter}}(1,1) + \lvert\text{CFB}(2,1) - \text{RFB}(2,1)\rvert$	1	na	0	na
18	{MPE(i, j) | $i\epsilon\{3, 6, 9, 12, 15\}$; $1 \le j \le 16$}	$\text{MPE}_{\text{Inter}}(3,1) = \text{MPE}_{\text{Inter}}(2,1) + \lvert CFB(3,1) - RFB(3,1)\rvert$	2	na	0	na
19	{MPE(i, j) | $i\epsilon\{3, 6, 9, 12, 15\}$; $j\epsilon\{1, 4, 7, 10, 13\}$}	$\text{MPE}_{\text{Final}}(3,1 = 0 + \text{MPE}_{\text{Inter}}(3,1)$	na	0	na	1
20	{MPE(i, j) | $i\epsilon\{3, 6, 9, 12, 15\}$; $j\epsilon\{2, 5, 8, 11, 14\}$}	$\text{MPE}_{\text{Final}}(3,2) = \text{MPE}_{\text{Final}}(3,1) + \text{MPE}_{\text{Inter}}(3,2)$	na	1	na	0
21	{MPE(i, j) | $i\epsilon\{3, 6, 9, 12, 15\}$; $j\epsilon\{3, 6, 9, 12, 15\}$}	$\text{MPE}_{\text{Final}}(3,3) = \text{MPE}_{\text{Final}}(3,2) + \text{MPE}_{\text{Inter}}(3,3)$	na	2	na	0

Table 4 illustrates the throughput computed for $M = N = 16$ and $p = q = 4$ for various search patterns. An ideal architecture with the best performance should be able to compute $M \times N$ (256) pixels per clock cycle. Reduced performance of proposed MPESA is a tradeoff to support multiple search patterns and reuse intermediate results.

4.4. Illustration. To illustrate the working of proposed architecture, an example of computing 3x3 SAD operations for Full Search is discussed here. Sequence of ($x\text{hop}_i$, $y\text{hop}_i$) is set to be {(0,1), (0,1), (0,1), (1,0), (0,−1), (0,−1), (0,−1), (1,0), (0,1), (0,1), (0,1), (1,0), (0,−1), (0,−1), (0,−1)}. A 16×16 MPESA, supported by a 16×16 Configurable Register Array (CRA), can be used to compute a maximum of twenty-five 3×3 SADs. Computation is assumed to start at control step $y = 0$. At this instant, the first row of pixels is fed into the CRA. At $y = 16$, first set of blocks from the reference frame and current frame are available to the appropriate PEs from the CRA (RFB(i, j) is provided to MPE(i, j)). At every subsequent control step, y, in this example, a set of twenty-five blocks of RFB_x^y ($1 \le x \le 25$) from the reference frame are fed into MPESA. During subsequent control steps, y is varied from 16 to 16 + nSearches, and appropriate blocks from the reference frame are fed into MPESA via CRA. During each control step, only a subset of MPEs from the 2D MPESA compute useful intermediate results. This set is represented as $Op\text{MPE}(y_{\text{curr}}, y_{\text{start}})$. y_{curr} represents current control step and y_{start} represents the control step during which a particular set of blocks was fed into MPE. For instance, the first set of blocks is fed into MPE at $y = 16$, thus $y_{\text{start}} = 16$. Table 5 illustrates the activity of MPESA during each control step.

The set of operations ($y_{\text{curr}} = 16$ to $y_{\text{curr}} = 21$) can be pipelined across multiple sets of candidate blocks, as different sets of resources are used during different control steps (i.e):

$$Op\text{MPE}\left(y_{\text{curr}(i)}, y_{\text{start}(j)}\right) \cap Op\text{MPE}\left(y_{\text{curr}(i)}, y_{\text{start}(k)}\right) = \varnothing \quad \forall\ i, j, k. \tag{49}$$

5. Results and Analysis

Merged PE systolic array (MPESA) architecture for performing arbitrary sized SAD computations is implemented in Verilog RTL and synthesized using Xilinx ISE design tools [19]. Xilinx Virtex-4 (XC4VSX35) FPGA [20] is used for the sake of prototyping, and designs can be ported onto custom technology libraries. For benchmarking performance, the proposed architecture was used to compute 16 possible 4×4 SAD values between pixels in current frame and the pixels in the search area of reference frame. This operation is repeated for all 256 candidate blocks in the search area (nSearches = 256).

Table 6: Evaluation of proposed architecture against all related motion estimation architectures (time taken represents number of clock cycles for computing 16 4×4 SADs).

	Time taken (clocks)	Latency (clocks)	Memory bandwidth (bits/clock)	Local I/O (bits)	Global I/O (bits)	Fanout (out/in)	LUT	FF
Ou et al. [5]	256	8	2048	352	32	16	12544	8192
Chen et al. (Type I) [6]	272	16	256	1920	0	4	3584	768
Chen et al. (Type II) [6]	272	17	128	6144	0	0	3968	2048
Tuang, Chang, Jen [7]	256	256	16	3840	256	0	53240	4096
Vos, Stegherr [3]	289	32	24	1792	256	2	53368	14592
Roma, Sousa [8]	289	32	24	1792	256	2	53368	13568
Komarek, Pirsch (AB1) [4]	8192	31	256	288	0	2	136	152
Komarek, Pirsch (AB2) [4]	512	32	256	4224	0	2	2176	2320
Komarek, Pirsch (AS1) [4]	8192	33	16	768	0	2	256	640
Komarek, Pirsch (AS2) [4]	256	33	4096	9792	0	2	3232	6280
Yeo, Hu [9]	289	256	24	4096	120	2	4224	4096
Yang, Sun, Wu (Type I) [2]	4096	256	24	512	0	16	384	640
Yang, Sun, Wu (Type II) [2]	4096	256	24	640	0	16	256	768
Lai, Chen [10]	256	256	24	8192	0	256	36864	6144
Kittitornkun, Hu [11]	289	289	32	4096	120	2	4096	8192
Proposed	271	7	128	7144	0	16	67584	36864

Following metrics were used to evaluate the proposed architecture:

(i) overall resource utilization,

(ii) initial latency,

(iii) time taken to determine 16 SAD values (1/Throughput),

(iv) memory bandwidth,

(v) local and global I/O,

(vi) Fanout.

Local I/O, Global I/O and Fanout were used to represent the network congestion of a given architecture. FPGAs are known to have limited routing resources, and hence an architecture with a lower network congestion will be a better fit to be implemented on an FPGA. Local I/O indicates the number of wires in the architecture that have a single source and single destination. Global I/O indicates the number of wires in the architecture that have multiple sources *and*/*or* multiple destinations. Table VI evaluates the proposed architecture against all the architectures listed in literature review section. In this comparison table, resource utilization is an estimated value, as different architectures have been implemented using different technology libraries (some papers have limited implementation details). It was observed that the proposed architecture had the lowest latency and one of the highest throughputs among all architectures. Memory bandwidth was found to be 128 bits. Network congestion of the proposed architecture was found

Table 7: Evaluation of proposed architecture against the fixed size SAD architecture. Resource count is expressed in terms of Flip-Flops (FF) and Look-Up Tables (LUT).

	F-SAD	MPESA
Resources (*FF*/LUT)	7576/8054	41496/83078
Latency	7	7
Time taken for best match	256	271
Memory Bandwidth	2048	128
Local I/O	2048	1744
Global I/O	0	0
Fanout	16	16

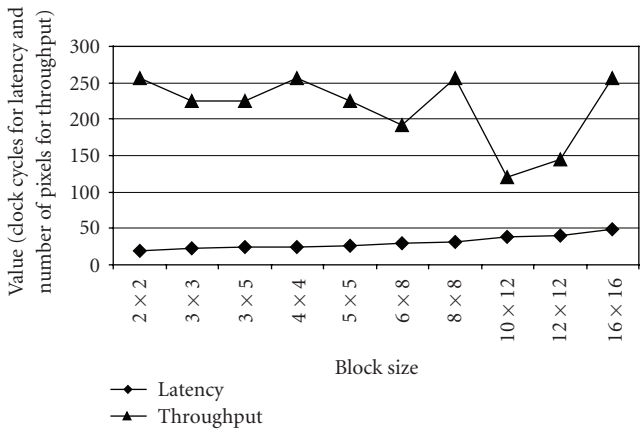

Figure 14: Latency and throughput of proposed architecture for various block sizes.

to be larger as we support multiple data access patterns. For comparing the resources used, each architecture is modeled as a set of adders, absolute difference units, multiplexers, and registers. Number of units is estimated from literature and overall resource utilization is estimated by summing up all the individual resource counts. It was observed that the proposed architecture resulted in the highest resource utilization, which is the penalty paid for context-adaptability. Among the other architectures, only Chen et al. [6] and Komarek et al. [4] have low latency, high throughput, and low resource count.

For evaluating the resource overhead of the proposed architecture over a fixed size SAD architecture, a reference design (F-SAD) was implemented using the same technology. F-SAD implementation is the stripped down version of the proposed implementation. All the logic required for context-adapting the design during run-time based on the value of $p \times q$ are removed. p, q, and nSearches are set at 4, 4, and 256, respectively. Table 7 shows the comparative results between the two architectures. It is observed that the proposed implementation requires $10.3x$ more LUTs and $5.5x$ times more Flip-Flops. This overhead is dedicated toward supporting all possible block shapes and sizes. All the other numbers in Table 7 are comparable.

Remainder of this section provides results to analyze the performance of the proposed architecture. 720P HDTV video sequences consisting of 30 frames per second (fps) are considered to be the test sequence. Proposed architecture can be used to accelerate multiple window-based algorithms. In this paper, we provide results for Full Search Motion Estimation (FSME) with block size set to be 4×4. Search pattern is defined using the sequence of $(xhop_i, yhop_i)$ that is set to be (0,1), (0,1), (0,1), (1,0), (0,−1), (0,−1), (0,−1), (1,0), (0,1), (0,1), (0,1), (1,0), (0,−1), (0,−1), and (0,−1). Proposed architecture has been tested to run at a frequency of 100 MHz. It is observed that 720P frames has 57600 4×4 blocks in each frame. Each block in a current frame has 16 candidate blocks in the reference frame, and 16 current blocks can be executed in parallel using the proposed architecture. By using the formulas presented in Section 4 for estimating performance, it is determined that 16 current blocks can be processed in 40 clock cycles. To process one frame, 144 000 clock cycles are required. To process 30 frames, 432 0000 clock cycles (0.0432 seconds at a clock frequency of 100 MHz) are required. Thus the proposed architecture is shown to support 30 fps (real time performance). Similar analysis can be performed for other applications as well (deinterlacing, fast search, etc.).

Performance of the proposed architecture is evaluated for multiple block sizes. Figure 14 shows the latency and throughput for multiple block sizes. In this figure, throughput is computed as the number of pixels processed by the MPESA per clock cycle. Overall latency is computed as sum of latencies of 2D CRA and MPESA.

6. Conclusion

This paper proposes a context-adaptable architecture to accelerate compute-intensive video processing kernels comprising of arbitrary sized block SAD/MSE operations and a variable search pattern. Results are presented by comparing the performance and resource utilization of the proposed architecture against a reference architecture and other architectures found in literature. Compared to fixed architecture (F-SAD), proposed architecture provides context-adaptability (p, q varying anywhere between 1 and 16) at an increased cost of $5.5x$ for FFs and $10.3x$ for LUTs, while maintaining a similar performance (latency and throughput).

References

[1] N. S. Kim, T. Austin, D. Blaauw, et al., "Leakage current: Moore's law meets static power," *Computer*, vol. 36, no. 12, pp. 68–75, 2003.

[2] K.-M. Yang, M.-T. Sun, and L. Wu, "A family of VLSI designs for the motion compensation block-matching algorithm," *IEEE Transactions on Circuits and Systems*, vol. 36, no. 10, pp. 1317–1325, 1989.

[3] L. de Vos and M. Stegherr, "Parameterizable VLSI architectures for the full-search block-matching algorithm," *IEEE Transactions on Circuits and Systems*, vol. 36, no. 10, pp. 1309–1316, 1989.

[4] T. Komarek and P. Pirsch, "Array architectures for block matching algorithms," *IEEE Transactions on Circuits and Systems*, vol. 36, no. 10, pp. 1301–1308, 1989.

[5] C.-M. Ou, C.-F. Le, and W.-J. Hwang, "An efficient VLSI architecture for H.264 variable block size motion estimation,"

IEEE Transactions on Consumer Electronics, vol. 51, no. 4, pp. 1291–1299, 2005.

[6] C.-Y. Chen, S.-Y. Chien, Y.-W. Huang, T.-C. Chen, T.-C. Wang, and L.-G. Chen, "Analysis and architecture design of variable block-size motion estimation for H.264/AVC," *IEEE Transactions on Circuits and Systems*, vol. 53, no. 3, pp. 578–593, 2006.

[7] J.-C. Tuan, T.-S. Chang, and C.-W. Jen, "On the data reuse and memory bandwidth analysis for full-search block-matching VLSI architecture," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 1, pp. 61–72, 2002.

[8] N. Roma and L. Sousa, "Efficient and configurable full-search block-matching processors," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 12, pp. 1160–1167, 2002.

[9] H. Yee and Y. H. Hu, "A novel modular systolic array architecture for full-search block matching motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 5, no. 5, pp. 407–416, 1995.

[10] Y.-K. Lai and L.-G. Chen, "A data-interlacing architecture with two-dimensional data-reuse for full-search block-matching algorithm," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, no. 2, pp. 124–127, 1998.

[11] S. Kittitornkun and Y. H. Hu, "Frame-level pipelined motion estimation array processor," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 2, pp. 248–251, 2001.

[12] H. Kung and C. Leiserson, in *Introduction to VLSI Systems*, C. Mead and L. Conway, Eds., Addison-Wesley, Reading, Mass, USA.

[13] R. Verma and A. Akoglu, "A coarse grained reconfigurable architecture for variable block size motion estimation," in *Proceedings of the International Conference on Field Programmable Technology (ICFPT '07)*, pp. 81–88, December 2007.

[14] M. Porto, L. Agostini, S. Bampi, and A. Susin, "A high throughput and low cost Diamond Search architecture for HDTV motion estimation," in *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME '08)*, pp. 1033–1036, April 2008.

[15] L. Zhang and W. Gao, "Reusable architecture and complexity-controllable algorithm for the integer/fractional motion estimation of H.264," *IEEE Transactions on Consumer Electronics*, vol. 53, no. 2, pp. 749–756, 2007.

[16] M. Byun, M. K. Park, and M. G. Kang, "EDI-based deinterlacing using edge patterns," in *Proceedings of the International Conference on Image Processing (ICIP '05)*, vol. 2, pp. 1018–1021, September 2005.

[17] E. Trucco and A. Verri, *Introductory Techniques for 3-D Computer Vision*, Prentice-Hall, Englewood Cliffs, NJ, USA, 1998.

[18] "Overview of the MPEG-4 Standard," 2002, http://www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.htm.

[19] "Xilinx ISE 10.1 manual," 2007, http://www.xilinx.com.

[20] "Virtex-4 family overview," 2007, http://www.xilinx.com/support/documentation/datasheets/ds112.pdf.