Novel Spiral Search Architecture for Motion Estimation

MS Project Report

Fall 2010

Vinod Reddy Nalamalapu

Under the Supervision of:

Katherine Compton, Michael J. Schulte

Abstract

Hardware accelerators for motion estimation has been an active area of research over recent years. Specialized memory organizations and hardware structures are proposed to address computation intensive full search motion estimation. With the video intensive multimedia applications being deployed in embedded systems like smart phones etc., power consumption and reduced complexity are gaining importance. Existing hardware accelerators for Full search motion estimation address the computation complexity well but are not power efficient. We propose a flexible full search architecture that can start the search from any predicted search center in a given search window which is beneficial for algorithms trying to leverage the neighboring motion vector information to reduce the SAD computations. The proposed architecture also allows support for early termination algorithms based on a threshold value. In the latest H.264, if the variable block size partitioning is decided earlier based on rate-distortion performance, Our architecture can utilize the mode decision to reduce the number of SAD computations. The proposed architecture is implemented in Verilog and synthesized at 500 MHz clock speed which can easily render VGA resolution (640x480) at 60 fps. The TSMC 64nm standard cell technology library is used for synthesis.

1 Introduction

Typically in a video sequence, multiple consecutive time frames are similar to each other. This redundancy in the temporal domain is exploited by video compression algorithms to achieve better compression. Motion estimation is the process of finding the correlation between two consecutive frames. After the first frame is transmitted, the next frame in the video sequence is only coded with the difference from the previous frame. In case of moving objects between two frames, motion estimation tries find the best match within a search window region. Figure 1.1 below shows the reference and the current frame.

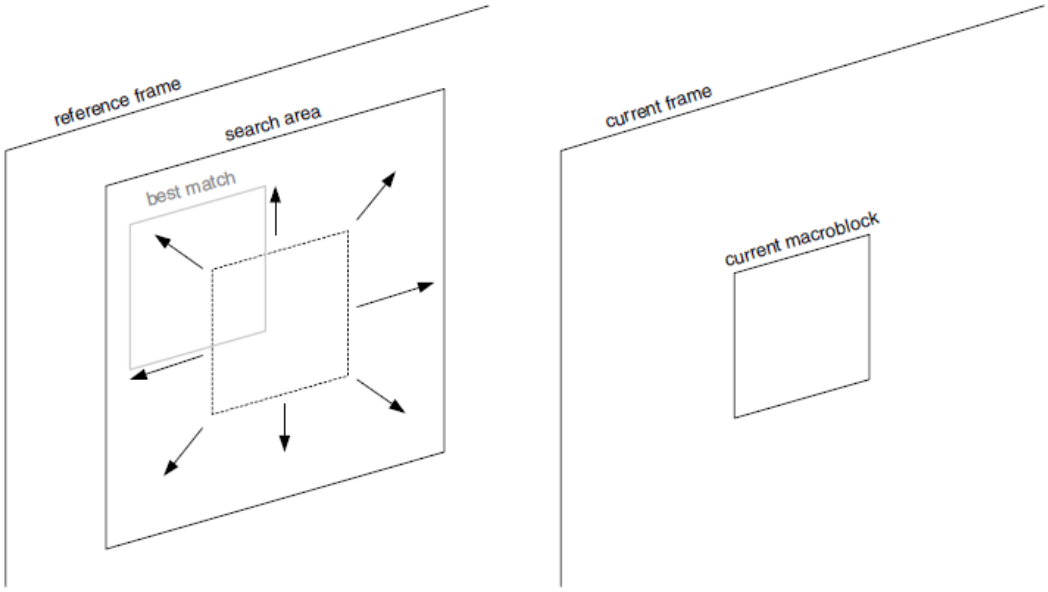


Figure 1.1. Motion Estimation from [1]

The current macroblock in the current frame is searched for the best match in the search area of the reference frame. For ease of computation, in earlier standards, like H.263, each frame is divided into multiple fixed-size 16x16 blocks. Each 16x16 block in the current frame is matched with the blocks in the search window region of the reference frame. Usually, the reference frame is an earlier frame in the time domain. The best matching block is subtracted from the current frame and the difference in the pixel values are transmitted to the decoder.

In this work, I propose a flexible full search architecture based on the spiral search algorithm [1]. It begins the full search from a predicted search center, and expands out in a spiral fashion until the boundaries of the search window are reached, or a sufficiently good match is found. This latter feature allows early termination of the search. We assume that a good match will usually be near the start location, and search those first. If we find a sufficiently good match, we can stop the search and move on to the next macroblock. If we do not, we can continue to search until we have examined the complete window, to find the closest match in that region. Early termination provides the benefit of completing the motion estimation phase of H.264 more quickly, improving performance. Furthermore, for real-time video encoding, we can translate that performance improvement into energy savings by cutting off power and having the circuitry "sleep" during the saved time.

2 Background

As discussed in the introduction, motion estimation attempts to find, for each block in the current frame, a closely-matching nearby block in the reference frame (often the previous frame). The

current block can then be encoded using fewer bits by transmitting only the vector of movement between the two frames, as well as the difference between the pixel values at those two locations. If the blocks are very closely matched, there will be relatively little difference in value that must be transmitted. This section describes several aspects of the motion estimation process. Finally, we discuss the spiral search algorithm for motion estimation, which is the approach used by my proposed architecture.

2.1 Motion Estimation Macroblock Matching

The best matching block is chosen based on the "energy difference" between the reference block in the search window and the current block, where blocks are well-matched if their energy difference is low (their pixel values in each location are similar). The equations below summarize the most widely used energy difference metrics, where N x N is the block size and C_{ij} and R_{ij} are the current and reference area pixels, respectively.

Mean Squared Error: $$MSE = \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \left(C_{ij} - R_{ij}\right)^2$$

Mean Absolute Error: $$MAE = \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \left|C_{ij} - R_{ij}\right|$$

Sum of Absolute Errors: $$SAE = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \left|C_{ij} - R_{ij}\right|$$

Sum of Absolute Errors (SAE) or Sum of Absolute Differences (SAD) is the most common due to the simplicity of its hardware or software implementation.

2.2 Fixed vs. Variable Blocks for Motion Estimation

Fixed-size block motion estimation (FSBME) does not perform well when each 16x16 block contains multiple smaller objects moving in different directions. To compensate for the disadvantages of FSBME, variable block-size motion estimation (VBSME) is adopted in advanced video coding standards. In the H.264/advanced video coding (AVC) standard, each 16x16 macroblock can be partitioned into 16x8 or 8x16 or 8x8 sub-blocks. An 8x8 sub-macroblock can be further partitioned into multiple 4x4 or 8x4 or 4x8 blocks. Based on the partition decision of the macroblock, each partition is searched for the best matching partition in the search window. Various partition sizes used in H.264 are shown in Figure 1.2.

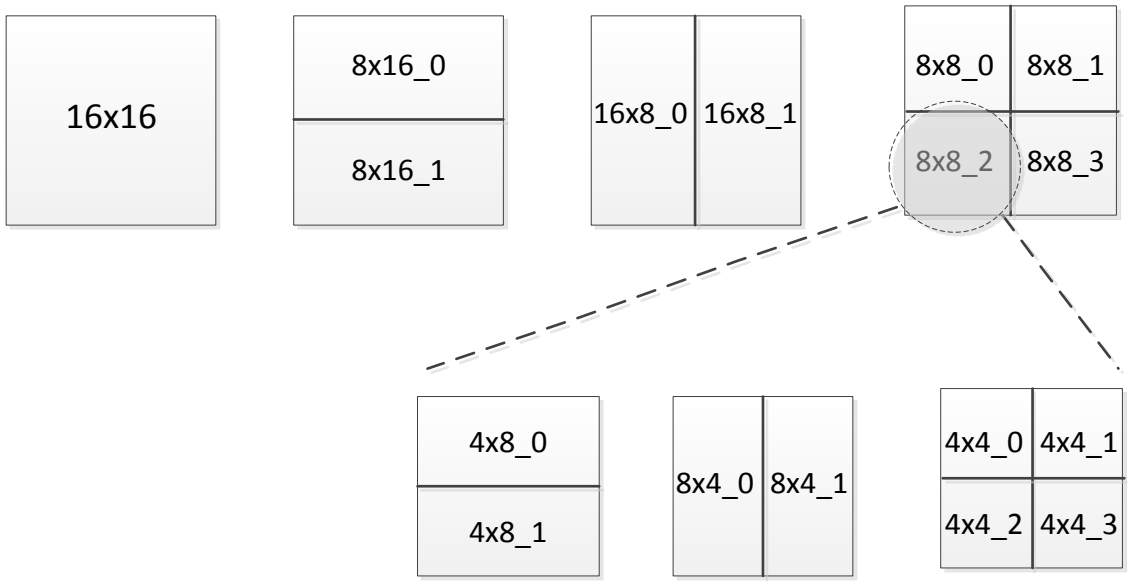


Figure 1.2. Variable Block Sizes from a 16x16 Block Size in H.264

Variable block size motion estimation tries to find the best match for any block size ranging from 4x4 to 16x16 in the reference search window. This further increases the computational complexity of VBSME compared to FBSME.

2.3 Previous Motion Estimation Search Algorithms

Full search tries to find the best match in the entire search window. Full search involves the computation of SAD at each location in the search window. For a search window of size +/- M pixels, the number of search locations is $(2M+1)^2$. For a search window of 32x32 and a block size of 16x16, a total of 289 locations are searched to find the best match with the minimum SAD value. This results in significant computational complexity. Many algorithms have been proposed to reduce the computational complexity of full search motion estimation. Some of the popular ones are the Three Step Search (TSS [2]), the New Three Step Search (NTSS [3]), the Four Step Search (4SS [4]), the Diamond Search (DS [5]) and the Adaptive Rood Pattern Search (ARPS [6]). These algorithms try to do small square (TSS, NTSS, 4SS) or diamond shaped (DS) search around a search center, and refine the search around the best matching block. The computational cost is also reduced by early termination techniques based on a SAD threshold value. Algorithms like ARPS employ sophisticated search center prediction as the start point. Though these algorithms address computational cost well, the performance in terms of PSNR is close to Full Search (FS) algorithms.

Full search still is attractive for high performance applications, but at the cost of increased computation cost and power consumption. Many hardware architectures are proposed for full search motion estimation. These architectures try to compute the SAD at all search locations in

the search window. A few of the popular hardware architectures are the Partial Propagate SAD architecture [7], the SAD Tree [8], the 1D Tree Architecture [9], and the 2D Architecture [10]. These hardware architectures access the blocks in a raster scan or snake scan to maximize pixel reuse. These architectures are highly optimized for full search schemes. The main disadvantage of these architectures is the inability to start the full search around a search center prediction or use early termination techniques based on a threshold SAD value. Early termination techniques are advantageous as they result in reduced computations, which in turn also results in significant power savings.

2.4 Spiral Search Motion Estimation Algorithm

Spiral search [1] starts with the pixel access for the macroblock location (X,Y) as shown in Figure 1.3. The next macroblock accessed is one position to the left of the initial search location (X,Y). This is followed by accessing a macroblock down and a macroblock to the right. Spiral search proceeds in this manner, extending in increasing rectangles from the initial search location, until all the macroblocks in the search window are accessed or an early termination condition is met. The initial search location can be anywhere in the search region. Figure 1.3 shows the order of macroblocks accessed to perform the spiral search. The inherent complexity for the spiral search is due to the pixel access pattern within the search window.

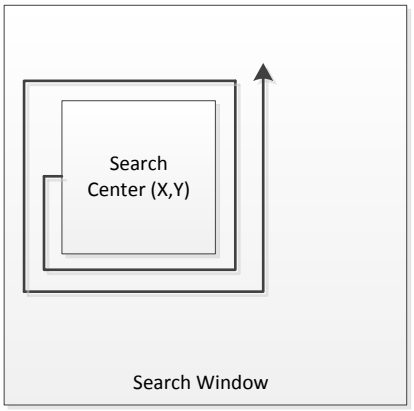


Figure 1.3. Spiral Access

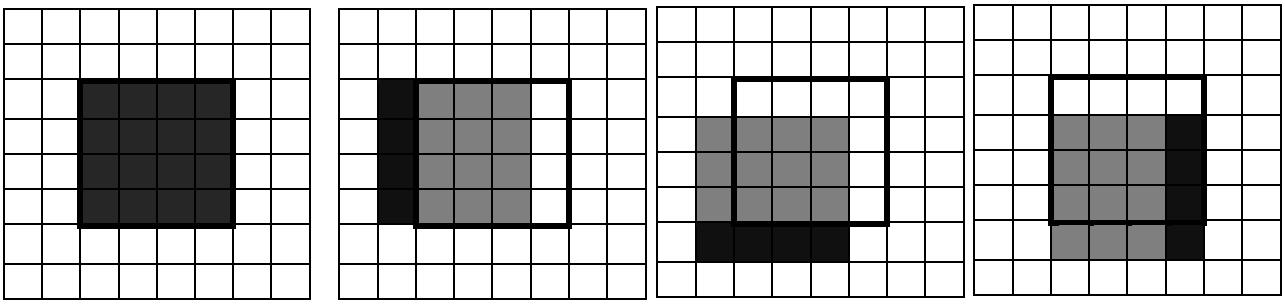
3 Proposed Spiral Search Architecture

The proposed architecture is composed of two main components: the memory that holds the reference frame pixel data for the complete search window, and the structure that performs compares the current macroblock to a chosen location within the search window. Although both features could theoretically be implemented in a single structure, the resulting hardware would be extremely large to find the SAD value for all possible macroblock locations within the

window. Thus, we move data from the search window storage to the macroblock comparator unit. Prior to a macroblock search, we first load the search window pixel data into the first structure, and the current macroblock pixel data (a much smaller array of pixel data) into the second. Section 3.2 describes the design of the search window to facilitate single-cycle data transfer of data for each new location examined, section 3.3 describes the address generator used to access the pixel data in the search window for movement to the macroblock comparator, and section 3.4 describes the macroblock comparator structure, which is also designed to handle variable block sizes. First, however, section 3.1 provides a general overview of the movement of data between the search window memory and the macroblock comparator.

3.1 General Operation

Figure 1.4 illustrates the order of pixel accesses for the first "ring" of a spiral search for a 4x4 macroblock. Once all the pixels of the initial search position (X,Y) are read from the search window in Step 1 as shown in Figure 1.4.a, the next block (X-1,Y) needs only an extra column of pixels highlighted by black in Step 2 of the Figure 1.4.b, since the pixel data from the previous step is reused. The next block in spiral access (X-1,Y-1) needs an extra row of pixels, as highlighted in Step 3 of the Figure 1.4.c. Step 4 follows. After Step 4, another column of pixels (to the right of the just-loaded column) would be loaded from the search memory. Every step in the spiral search reuses many of the pixels from the previous step, along with an additional row or column which is read from the search memory. Reusing the pixels from the previous step helps to reduce accesses to the search memory and increases the throughput. Each cycle, only the extra Row or Column of pixels shown in black in Figure 1.4 are read from the search window.



(a). Step 1 (b). Step 2 (c). Step 3 (d). Step 4

Figure 1.4. Order of pixel access in a spiral search. The dark outline is the initial position. Gray pixels are reused from the previous step; black pixels are newly loaded from the search memory.

3.2 Search Window Memory Organization

The maximum search window for our proposed architecture is 32x32 pixels in size. The design could, however, be extended for larger (or smaller) search windows. To perform the spiral access efficiently, one should be able to read a single row or column from the search window memory

in a single cycle. To achieve a single cycle column or row access the pixels are reorganized in the search memory. One possible scheme is to rotate the pixels [11] to the right when writing to the search memory. This results in the column of pixels scattered into different banks of the search window memory. The diagram below shows the original pixels in the search memory and the reorganized search memory.

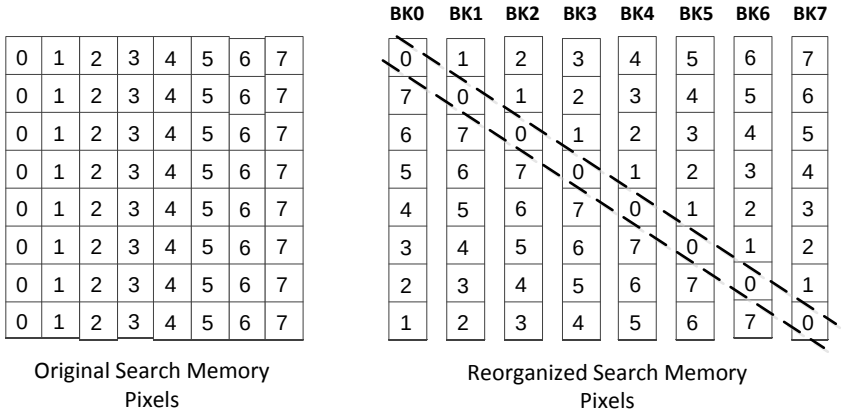


Figure 2.1. Search Memory Organization

The 8x8 search memory is divided into 8 banks, each of which is a single pixel wide. Each column of pixels in the original memory can be accessed in single cycle from the reorganized memory as the pixels of a single column are scattered into different banks. Row access also takes a single cycle. During reads/writes, the row or column pixels read from the reorganized memory have to be rearranged to be in the same order as in original memory to preserve pixel locations.

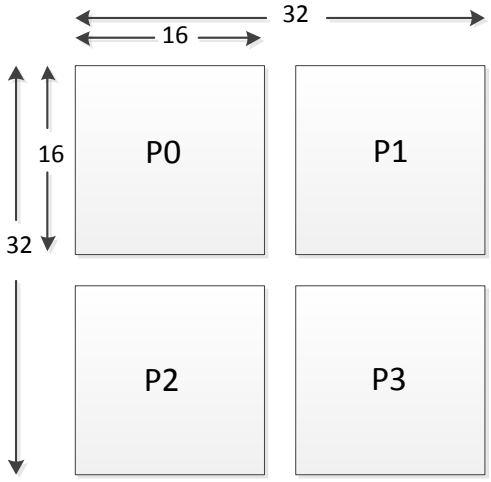


Figure 2.2. Search Memory Partitions

Our 32x32 search window memory is divided into four partitions, P0, P1, P2, P3, each of size 16x16 as shown in Fig 2.2. Partitioning the memory reduces the area required for the rotating multiplexers used to move pixels to the correct memory locations during memory writes. Each

partition now requires a 16-to-1 multiplexer as opposed to a bigger 32-to-1 multiplexer for the entire search memory. The multiplexer are reused across the all the partitions to save area. Pixel row 0 is written unaltered to the search memory, Pixel row 1 is rotated to the right by one pixel, Pixel row 2 is rotated to the right by two pixels, and so forth. The rotated pixels are written to partition P0 or P1 in an alternating fashion. Once P0 and P1 are full, pixels are written to the P2 and P3 partitions.

Three multiplexing stages are required to read the needed pixel data from the search memory, which is done at each step of the spiral search (Figure 2.3). For lower read addresses, pixels are either read from {P0, P1} for row accesses or from {P0, P2} for column accesses. For higher read addresses, pixels are read from {P2, P3} for row accesses and from {P1, P3} for column accesses. The next two stages of multiplexing put the pixels into the needed order to form a complete 32-pixel column or 32-pixel row in the order matching the original image (rather than the translated ordering of the search window memory). The final set of multiplexers in Figure 2.3 helps select any 16 pixels from 32 row or column pixels based on the address offset.

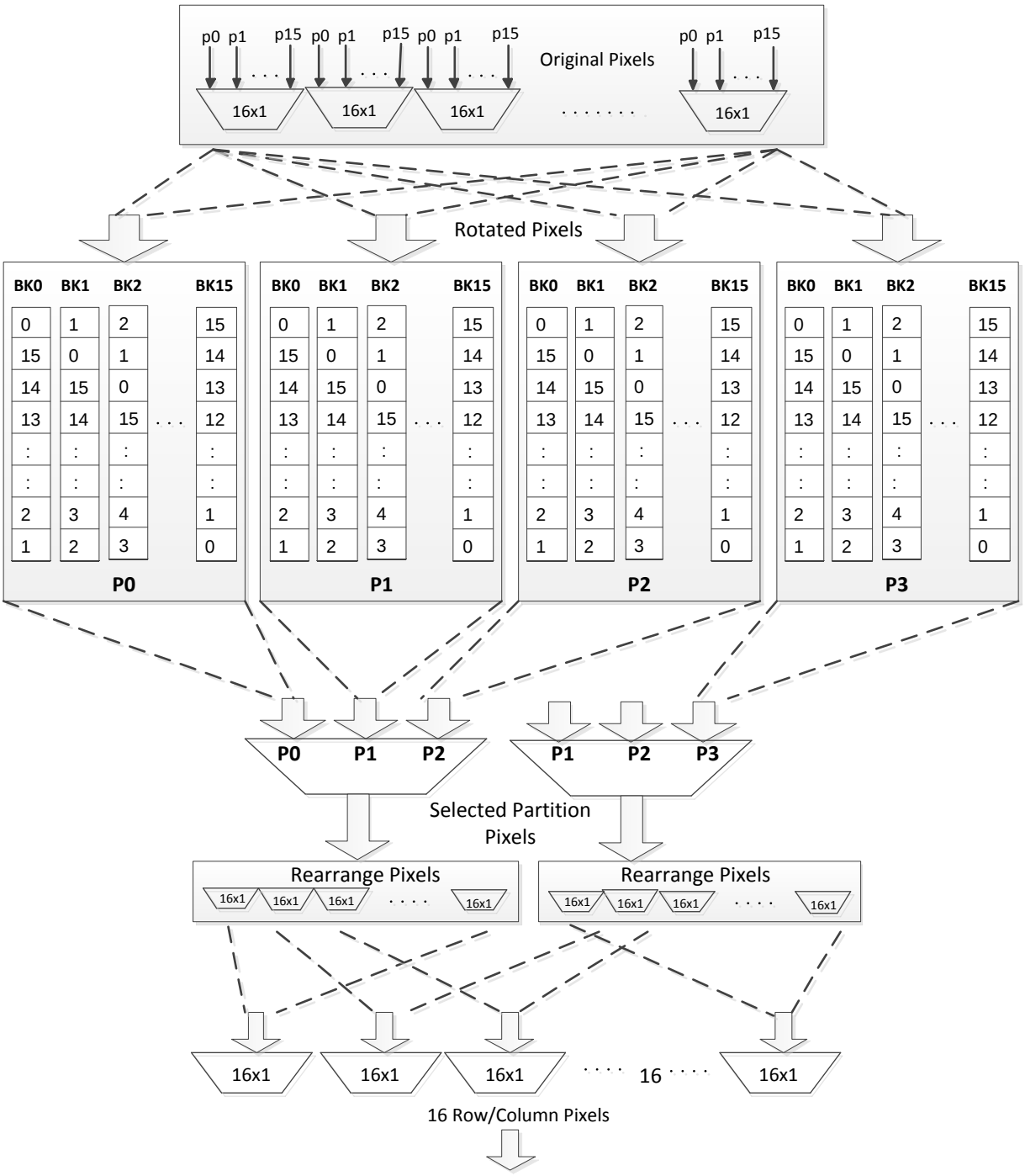


Figure 2.3. 32x32 search window memory organization

Since the size of the macroblock is 16x16, each spiral access requires an extra 16-pixel row or column from the search memory. This search memory organization delivers a 16-pixel row/column with any offset in a 32-pixel row/column search memory in a single cycle.

3.3 Spiral Address Generator

The Spiral Address generator generates the row or column read address for the search window memory. The control hardware required for spiral address generation is summarized in the flow chart (Figure 2.4) Once a valid spiral center location is issued, the LEFTMAX, RIGHTMAX, UPPERMAX, and DOWNMAX boundaries are calculated for a search ring of radius *1* around the spiral center. For a block size of 16x16 and a spiral center of (X,Y) = (3,4), (LEFTCOL_MAX(X-1), RIGHTCOL_MAX(X+MB_X), UPPERROW_MAX(Y-1), DOWNROW_MAX(Y+MB_Y)) are calculated to be (2,19,3,20), respectively. MB_X and MB_Y are the macroblock horizontal and vertical size. Once the boundaries are calculated, the control starts with accessing the left columns until the left max boundary is reached. Then, the control generates addresses to access the rows below followed by columns to the right and columns to the left as shown in Figure 2.4.

The number of macroblocks accessed is tracked using a counter. The number of spiral search accesses in a ring of radius *R* around the search center is given by *8*R.* To understand why this is the case, consider that the upper-left pixel of a block is its origin point. To examine each possible macroblock location one pixel away from this point, we can think of a 3x3 grid of pixels, with our initial position as the center location. The eight pixels that surround this center location represent the eight different possible macroblock origin points that we will test with a search radius of 1.

For example, for the first ring, which has a radius of 1, the spiral ring is expanded by one after the counter reaches 8 (8*1, where 1 is the radius *R*), and the boundaries are updated to (1, 20, 2, 21). The process repeats to access the macroblocks in the next spiral ring (*R* = 2) until the count reaches 16. Spiral addresses are generated in this manner until the boundaries reach the search window size limits.

The control is summarized in Figure 2.4 below. After a valid spiral center is available the boundaries are updated. The updated boundaries are checked against the search memory window limits. If the boundaries exceed the search window limits the spiral search is terminated. Otherwise, the spiral access proceeds to access the left columns from the search memory. Left column are accessed until the left max boundary is reached. This is repeated for Right Columns and Up/Down Rows. The number of accesses are tracked with a counter Once the counter reaches *8*R*, *R* is incremented and boundaries are updated to a larger spiral ring. The spiral access of Rows/Columns is repeated for the larger spiral ring until the boundaries exceed the search memory window size. (PosX, PosY) in the Figure 2.4 is the starting search location, #Pos is the number of Rows/Columns accessed so far.

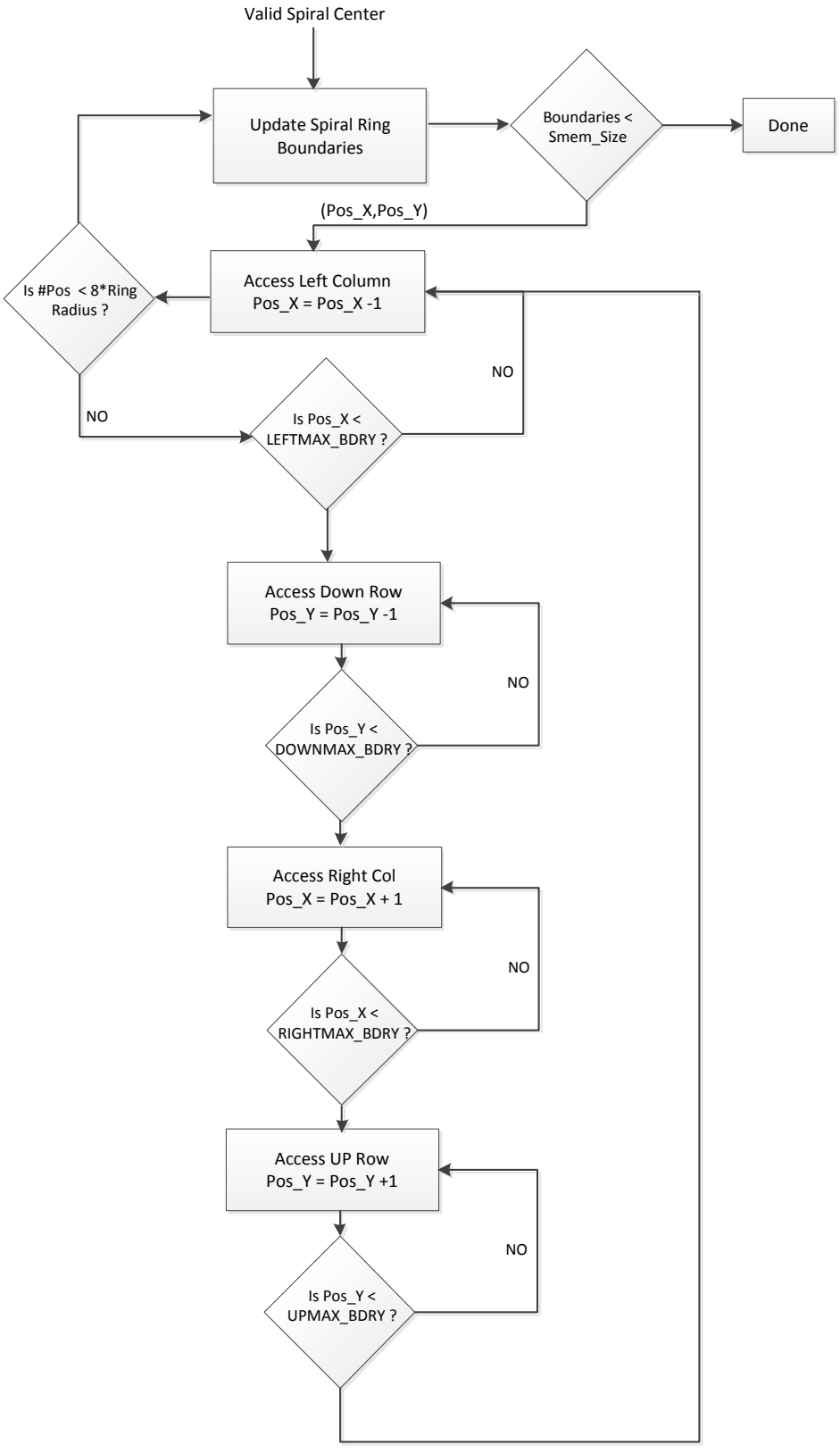


Fig 2.4 Spiral Address Generator

3.4 Pixel Reuse Array (Macroblock Comparator)

This section describes the structure that stores the current macroblock and compares it to a macroblock in a given location in the reference search window. The spiral address generator produces the read addresses to access only the extra row or column needed for the next macroblock in the spiral ring. All the other pixels from the previous blocks are reused efficiently inside a structure called the Pixel Reuse Array (PRA). This is a special memory structure equal in size to a macroblock, and it holds a macroblock's worth of pixel data. Once the PRA is loaded with all the search center pixels, a new left column access from the search window memory is shifted into the PRA from the left column input. All the other pixels in the array are shifted to the right. A right column read from the search memory results in shifting the pixels to the left. Similarly on a row access from the search memory, the pixels in PRA are shifted up/down based on the new row location relative to the current pixels in the PRA.

A 4x4 PRA is shown in Figure 2.5. Each cell in the PRA contains a multiplexer to load a pixel from left, right, up or down position into a register. The output of the register is a single reference pixel. The other input to the cell is the corresponding pixel from the current macroblock memory. The cell, as highlighted in the Figure 2.5, also calculates the absolute difference between the reference pixel from the search window memory and the pixel from the current macroblock memory. Each 4x4 PRA can shift in data from the left, right, up or down position based on whether a row or a column is being accessed from the search window memory. Outputs from 4x4 pixel array are 16 absolute values of the differences between the reference pixels and current macroblock pixels.

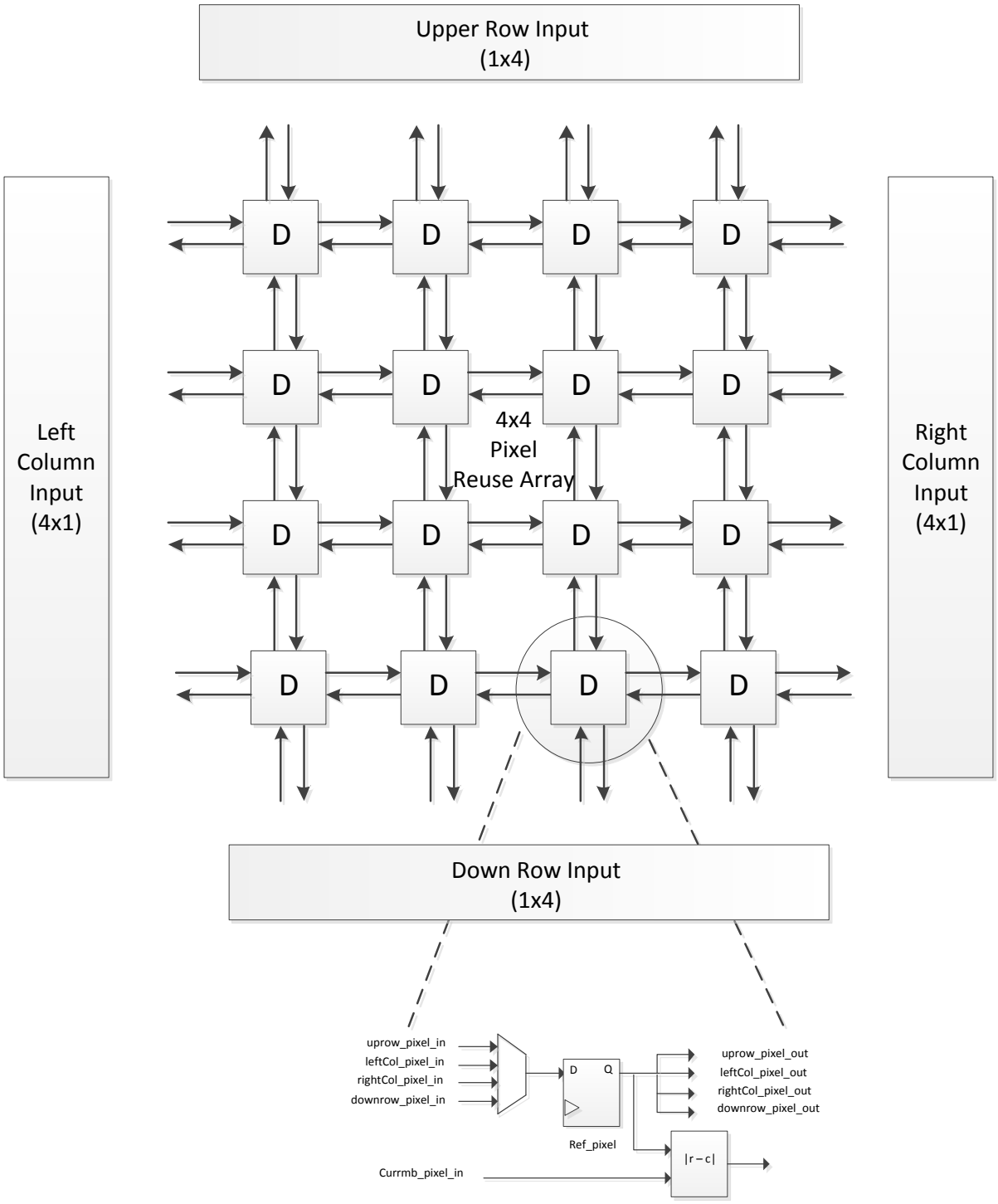


Figure 2.5. 4x4 Pixel Reuse Array

A bigger PRA for a macroblock of size 16x16 is implemented by chaining together 16 of the 4x4 PRAs (as sub-PRAs). A few multiplexers are added to the 16x16 pixel reuse array to support variable block size motion estimation, as show in Figure 2.6. For sub-blocks, these multiplexers bypass the inactive sub-PRAs and make the external right column/down row inputs available

internally to the active sub-PRAs. The sub-PRAs can be made inactive by not clocking them to save dynamic power. For example, for computing the SADs of an 8x8 block, only four of the sub-PRAs need to be active and clocked. The remaining 12 sub-PRAs (shown bounded by a dotted box in Figure 2.6) can be inactive and turned off.

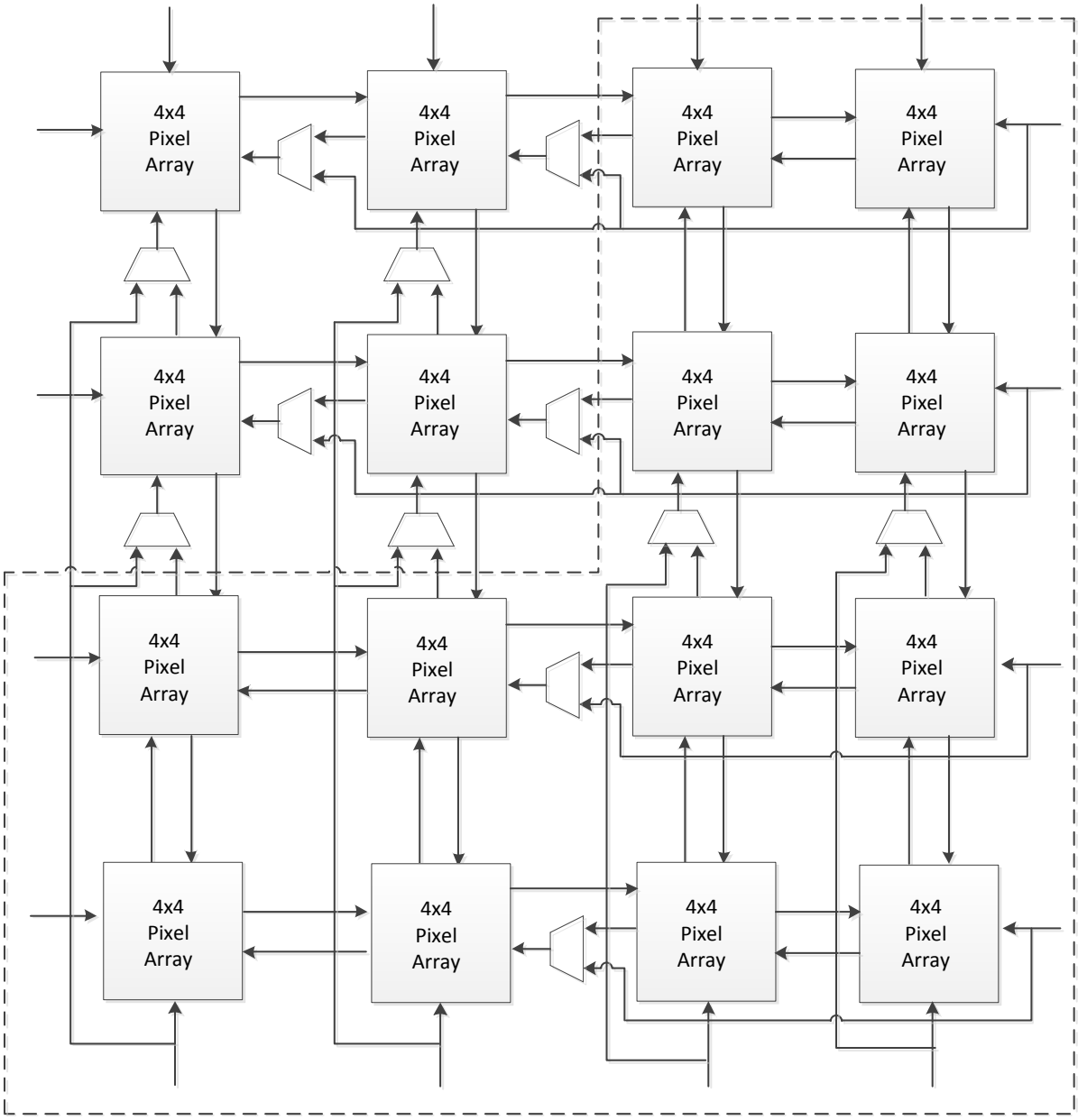


Fig 2.6 16x16 Pixel Reuse Array

3.5 Sum of Absolute Differences

Absolute differences from the 4x4 sub-PRAs are summed using adder trees. The critical path of the adder trees is reduced by adding pipeline stages. The first pipeline stage sums up the absolute values from the 4x4 sub-PRAs. The second pipeline stage sums up those results to

generate 8x8, 4x8 and 8x4 SAD values for different block sizes and locations. The final pipeline stage generates 16x16, 8x16 and 16x8 SAD values. This method of building the summation tree allows us to generate all of the required sub-block SAD values simultaneously [8].

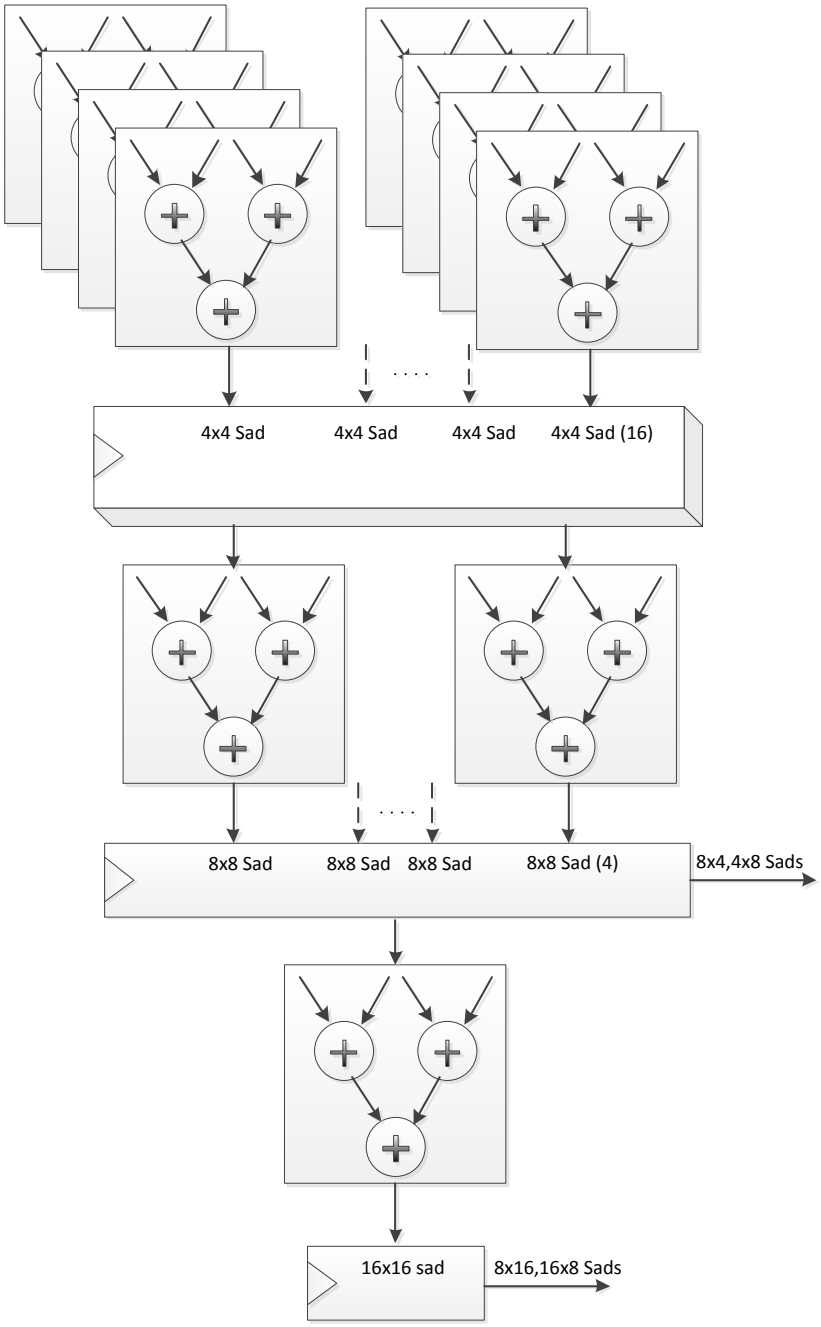


Fig 2.7 Sum of absolute differences hardware

4 Results and Conclusions

Table 3.1 summarizes the area, clock rate, and power results of several existing hardware architectures for H.264.

	1-D [9]	2-D [10]	SAD Tree [8]	Partial Propagate SAD [8]	Efficient Partial Propagate SAD [7]
PE Number	16	256	256	256	256
Technology	0.13u	0.18u	0.18u	0.18u	0.18u
Clock (MHz)	294	100	110	110	227
Area (um^2)	61k	154k	-	-	94.3k (only combinational portions reported)
Power(mW)	573.4	-	-	-	461

Table 3.1 Area/Performance Comparison of Existing H.264 Architectures

The efficient Partial Propagate architecture [7] has the best area and performance figures among compared existing architectures. Hence we choose partial propagate as our baseline architecture to closely compare against our proposed spiral search architecture. Also the implementation in [7] only has the area results for combinational and control logic.

All the modules for the baseline and the proposed architecture are designed in Verilog and synthesized using Synopsys Design Compiler. The technology library used for standard cell synthesis is TSMC 65nm. Synthesis results are present in the Table 3.2.

	Baseline [7]	Proposed	Baseline [7]	Proposed	Baseline [7]	Proposed
Technology	65nm	65nm	65nm	65nm	65nm	65nm
Clock (MHz)	500MHz	500MHz	666MHz	666MHz	800MHz	800MHz
Area (um^2) Combinational Sequential & control	210k 125k 84k	245k 127k 117k	230k 146k 84k	263k 143k 119k	246k 162k 84k	282k 162k 119k
Power (mW)	68	59	91	80	112	97

Table 3.2. Area/Clock Rate/Power Comparison Baseline vs. Proposed

The Baseline columns show the results of the implementation of work proposed in [7] and the proposed columns show the results of our work for several different target clock rates: 500 MHz, 666MHz, and 800MHz. Our architecture has more sequential and control area overhead compared to the baseline implementation. Sequential and control overhead comes from the spiral search address generation and pipeline stages of the SAD tree.

Power numbers reported are from Design Compiler with Operating Voltage at 1.0V. The proposed architecture has lower power consumption than the baseline. Our architecture can have more high-level energy savings as it supports early termination algorithms and spiral search around any predicted motion vector. If the macroblock partition mode is decided in advance, more power savings can be achieved by turning off the inactive pixel reuse arrays. In many of the advanced motion estimation algorithms, the partition mode is decided based on Rate Distortion performance. Hence this architecture can exploit the early mode decision and save power by reducing the computations. Spiral search around a predicted motion vector and early termination techniques can drastically reduce power consumption compared to the Exhaustive Full Search in the search window. For example 4SS requires SAD to be computed only for 4x8 = 32 positions as opposed to (2 x 8 + 1) x (2 x 8 + 1) = 289 pixels for a search window of size +/- 8. Thus, our spiral search implementation is a flexible and power efficient architecture.

5 References

[1]Iain E. G. Richardson, Video Codec Design, West Sussex: John Wiley & Sons Ltd., 2002, Ch. 4, 5, & 6.

[2] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro, "Motion- compensated interframe coding for video conferencing*,"Iin Proc. NTC 81*, pp. C9.6.1-9.6.5, New Orleans, LA, Nov./Dec. 1981

[3] Renxiang Li, Bing Zeng, and Ming L. Liou, "A New Three-Step Search Algorithm for Block Motion Estimation", *IEEE Trans. Circuits And Systems For Video Technology*, vol 4., no. 4, pp. 438-442, August 1994.

[4] Lai-Man Po, and Wing-Chung Ma, "A Novel Four-Step Search Algorithm for Fast Block Motion Estimation", *IEEE Trans. Circuits And Systems For Video Technology*, vol 6, no. 3, pp. 313-317, June 1996.

[5] Shan Zhu, and Kai-Kuang Ma, " A New Diamond Search Algorithm for Fast Block-Matching Motion Estimation", *IEEE Trans. Image Processing*, vol 9, no. 2, pp. 287-290, February 2000.

[6] Yao Nie, and Kai-Kuang Ma, "Adaptive Rood Pattern Search for Fast Block-Matching Motion Estimation", *IEEE Trans. Image Processing*, vol 11, no. 12, pp. 1442-1448, December 2002

[7] Zhenyu Liu, Yiqing Huang, Yang Song, Satoshi Goto, Takeshi Ikenaga, "Hardware-Efficient Propagate Partial SAD Architecture for Variable Block Size Motion Estimation in H.264/AVC," *Proceedings of the 17th Great Lakes Symposium on VLSI*, pp. 160-163, 2007

[8] Chen Ching-Yeh ; Chien Shao-Yi ; Huang Yu-Wen ; Chen Tung-Chien ; Wang Tu-Chih ; Chen Liang-Gee, "Analysis and Architecture Design of Variable Block Size Motion Estimation for H.264/AVC," IEEE Transactions on Circuits and Systems I, Volume PP, Issue 99, 2005

[9] S. Y. Yap, J. V. McCanny, "A VLSI architecture for variable block size video motion estimation", *IEEE Trans. On CAS-II* Vol.51, No. 7, pp. 384-389, 2004.

[10] M. Kim, I. Hwang, and S. I. Chae, "A fast vlsi architecture for full-search variable block size motion estimation in mpeg-4 avc/h.264" In *Proceedings of ASP-DAC 2005*, volume 1, pages 631–634, January 2005.

[11] S Seo, M Woh, S Mahlke, T Mudge, S Vijay, "Customizing Wide-SIMD Architectures for H. 264" *International symposium on on Systems, Architectures, Modeling and Simulation 2009.*