Optical Engineering

SPIEDigitalLibrary.org/oe

Self-reconfigurable approach for computation-intensive motion estimation algorithm in H.264/AVC

Jooheung Lee
Chul Ryu
Soontae Kim

Self-reconfigurable approach for computation-intensive motion estimation algorithm in H.264/AVC

Jooheung Lee
Hongik University
Department of Electronic and Electrical Engineering
300 Shinan, Jochiwon, Yeongi
Chungnam 339-701, Republic of Korea
E-mail: joolee@hongik.ac.kr

Chul Ryu
Dongguk University-Seoul
Department of Information and Communications Engineering
26, 3 Pil-dong, Chung-gu
Seoul 100-715, Republic of Korea

Soontae Kim
Korea Advanced Institute of Science and Technology
Department of Computer Science
335 Gwahnagno, Yuseong-gu
Daejeon 305-701, Republic of Korea

Abstract. The authors propose a self-reconfigurable approach to perform H.264/AVC variable block size motion estimation computation on field-programmable gate arrays. We use dynamic partial reconfiguration to change the hardware architecture of motion estimation during run-time. Hardware adaptation to meet the real-time computing requirements for the given video resolutions and frame rates is performed through self-reconfiguration. An embedded processor is used to control the reconfiguration of partial bitstreams of motion estimation adaptively. The partial bitstreams for different motion estimation computation arrays are compressed using LZSS algorithm. On-chip BlockRAM is used as a cache to pre-store the partial bitstreams so that run-time reconfiguration can be fully utilized. We designed a hardware module to fetch the pre-stored partial bitstream from BlockRAM to an internal configuration access port. Comparison results show that our motion estimation architecture improves toward data reuse, and the memory bandwidth overhead is reduced. Using our self-reconfigurable platform, the reconfiguration overhead can be removed and 367 MB/sec reconfiguration rate can be achieved. The experimental results show that the external memory accesses are reduced by 62.4% and it can operate at a frequency of 91.7 MHz.
 [DOI: 10.1117/1.OE.51.4.047008]

Subject terms: bitstream compression; configuration speed; dynamic partial reconfiguration; field-programmable gate array; motion estimation.

Paper 111607 received Dec. 20, 2011; revised manuscript received Feb. 24, 2012; accepted for publication Feb. 28, 2012; published online Apr. 24, 2012.

1 Introduction

The H.264/AVC video compression standard developed by the ITU-T Video Coding Experts Group and the ISO/IEC Moving Picture Experts Group provides many advanced coding techniques, such as integer discrete cosine transform (DCT), intra prediction in the spatial domain, multiple reference pictures, variable block size motion estimation and compensation, context adaptive variable length coding, and context adaptive binary arithmetic coding to achieve higher coding efficiency.[1–3] Among these coding techniques, motion estimation (ME) plays a key role to reduce temporal redundancy in the video coding processes.[4] However, motion estimation engine demands extremely high computing capability especially in the H.264/AVC standard due to its support for a wide range of block sizes. H.264/AVC standard supports motion estimation for seven different block sizes, i.e., 16×16, 16×8, 8×16, 8×8, 8×4, 4×8, and 4×4. While the number of bits required for motion vector encoding can be reduced by selecting 16×16 macro block (MB) size, a smaller block size, such as 4×4, can increase the possibility of finding a better matching block in the previous frame so that its resultant residual errors to be encoded can be further reduced. Therefore, variable block size motion estimation can choose its block size dynamically to achieve higher coding performance in the video encoder. However, its computational complexity is increased significantly. Therefore, optimized hardware architecture that can support for variable block size motion estimation becomes essential for real-time video encoding applications. It is shown that a motion estimation module consumes more than 60% of time in the complete video codec system.[5]

The full search block matching motion estimation algorithm is often employed for the selection of the best motion vector with the use of sum of absolute differences (SAD) criterion. While computationally intensive, the full search method has been preferred in hardware implementations due to its high algorithmic performance, regular data flow, and scalable architecture. For variable block size motion estimation (VBSME) supported in H.264/AVC, the SADs of smaller 4×4 blocks can be computed first and then merged to form blocks of seven different sizes.[6,7] This paper proposes a self-reconfigurable full search variable block size ME architecture to compute SADs and corresponding motion vectors (MVs).

When it comes to real-time implementation of such techniques using a reconfigurable hardware platform on field-programmable gate arrays (FPGA), adaptability of computing systems to support various multimedia formats and equipment with different computational requirements is highly desirable. However, it significantly reduces the benefits to adopt pre-constructed intellectual property (IP) cores on the reconfigurable hardware platform since these traditional pre-constructed IP cores suitable for the standard application-specific integrated circuit (ASIC) design flow cannot change its own architecture efficiently to adapt to changing environments. In this paper, we design a modular architecture for ME, which can make use of reconfigurable

0091-3286/2012/$25.00 © 2012 SPIE

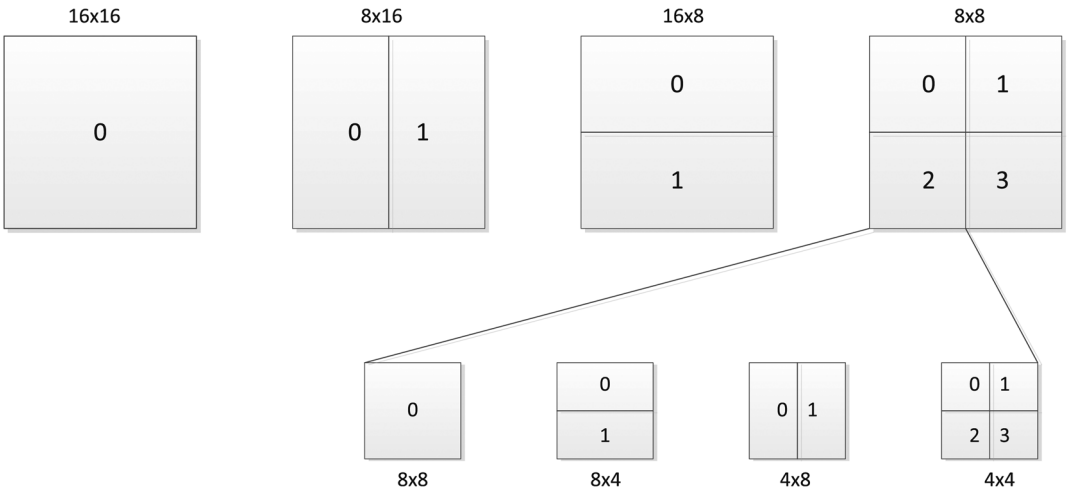

Fig. 1 Different block sizes for motion estimation in H.264/AVC standard.

hardware in FPGA efficiently to support different video resolutions and frame rates and enhance data reuse and memory bandwidth overhead by autonomous configurations of processing element (PE) arrays during run-time.

Dynamic partial reconfiguration is to change the configuration of parts of the FPGA while the rest is still working.[8–10] Self-reconfiguration usually involves an embedded processor to control the reconfiguration process so that the FPGA system can be reconfigured automatically. We develop a self-reconfigurable platform using FPGA to implement our variable block size ME architecture. As a result, our scalable ME design can change the architecture adaptively according to the given computing requirements, such as video resolutions and frame rates. In addition, bitstreams' sizes and their reconfiguration time are also reduced by dynamic partial reconfiguration.

For the experiments, we use internal configuration access port (ICAP) as a configuration interface and PowerPC as an embedded processor to control the autonomous reconfiguration during run-time in the proposed self-reconfigurable platform. We use LZSS algorithm to compress the partial bitstreams first, and compressed partial bitstreams are stored on the compact flash card in order to reduce overall external memory accesses. Then, we use embedded processor to decompress the partial bitstream of ME and store it into the on-chip BlockRAM to reduce the reconfiguration overhead. In addition, a hardware core designed to fetch the partial bitstream from on-chip BlockRAM to ICAP for reconfiguration is used to minimize the reconfiguration overhead from PowerPC.

The main contributions of our work are: first, a self-reconfigurable design for variable block size ME computation using PowerPC and reconfigurable fabrics on FPGA; second, a scalable full search ME architecture with regular data flow, low memory addressing complexity, and high computational capability; third, a configuration manager for choosing different ME configuration modes during run-time; fourth, a self-reconfiguration approach using compressed ME bitstreams to reduce external memory accesses and storage sizes; fifth, using BlockRAM to prefetch the partial bitstream of ME for fast reconfiguration speed and a customized hardware module to reduce the reconfiguration overhead from PowerPC.

The rest of this paper is organized as follows. In Sec. 2, we present our proposed self-reconfigurable platform for motion estimation in H.264/AVC. In Sec. 3, we show the experimental results and our analysis. In Sec. 4, we briefly conclude our work.

2 Proposed Aproach for Fast Reconfiguration of Motion Estimation in H.264/AVC

2.1 *Reconfigurable Motion Estimation Architecture*

For a block-based ME, its basic function is to find out the best matching block by calculating the distortion between the current image block and all candidate blocks in the search

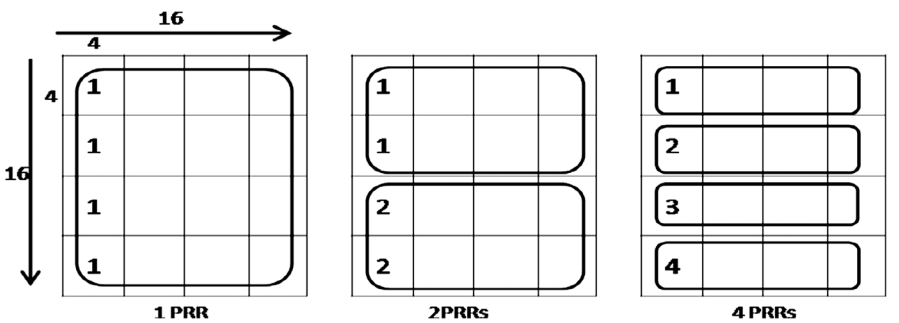

Fig. 2 Number of blocks covered when the number of PRRs in active mode is one, two, and four. More divisions imply higher degree of parallelism in concurrent processing.

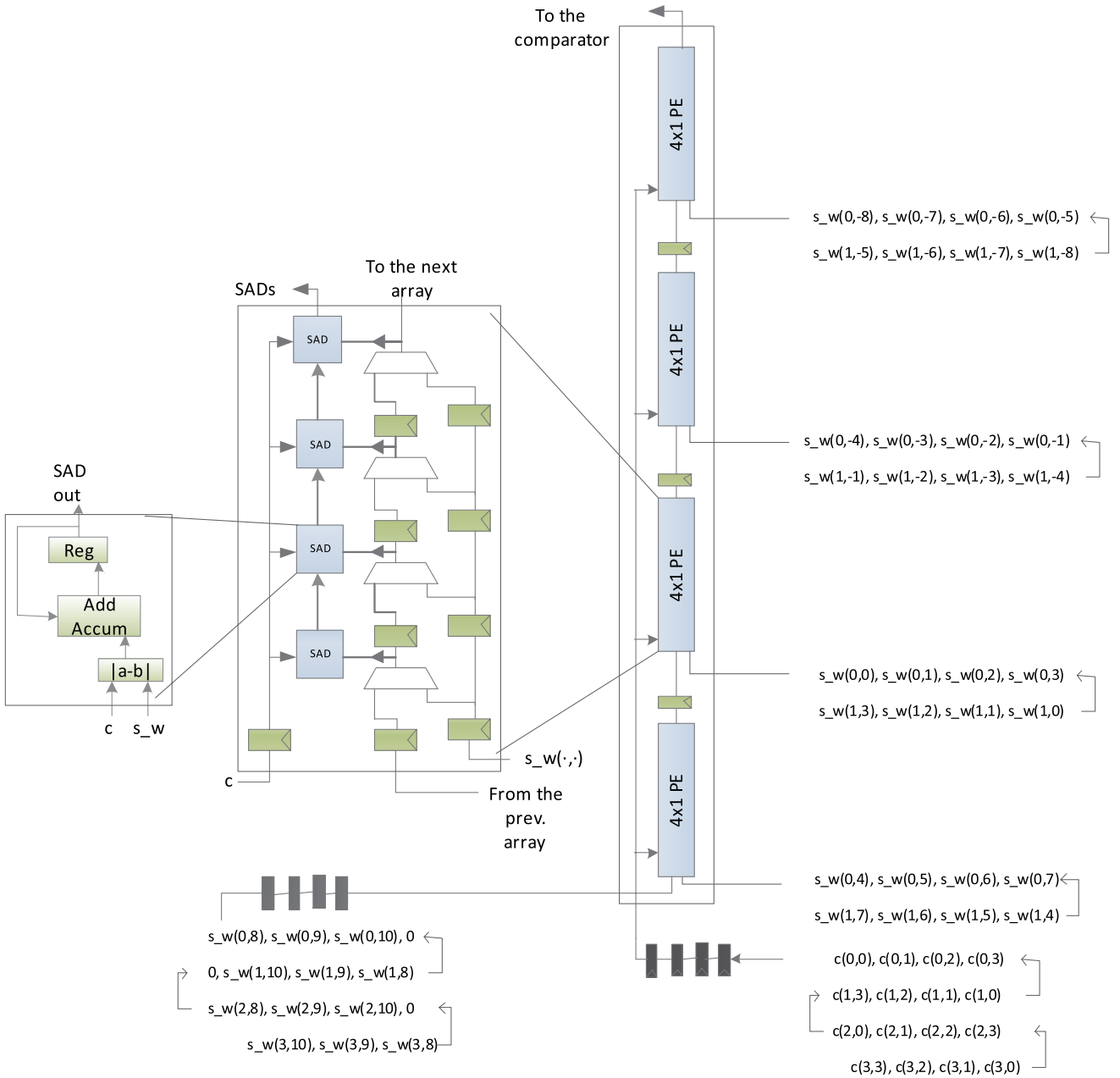

Fig. 3 16×1 PEs for SAD computation in each PRR.

range of the reference frame. Let the block size be $N \times N$ and location of each block in the current frame C is represented by (i,j). This block must be matched with a block within the search window (h,v) in the reference frame. SAD of such searching candidate block is given by

$$\mathrm{SAD}_{(i,j)}(h,v) = \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} |c_{(i,j)}(x,y) - sw_{(i,j)}(x+h, y+v)|, \tag{1}$$

where $c(x,y)$ and $sw(x,y)$ represent pixel values in the current block and the search candidate block in the search window, respectively.

In H.264/AVC, seven different block sizes are specified for prediction process as shown in Fig. 1. Therefore, an ideal encoder has to examine all possible 259 combinations of MB modes to select the best among them. Hence, VBSME improves motion tracking over a fixed block size algorithm especially by giving attention to highly active sub-blocks. A 4×4 block can be considered as a basis block. Using SAD of this sub-block, SADs of larger blocks can be computed by simply adding the corresponding sub-blocks' SADs. Since 16×16 macro block can be divided into seven different kinds of sub-blocks, including 16×16, 16×8, 8×16, 8×8, 8×4, 4×8, and 4×4 as in Fig. 1, a total of 41 sub-blocks $(1 + 2 + 2 + 4 + 8 + 8 + 16)$ have to be evaluated for the motion vector selection.

Partial reconfigurable regions (PRRs) defined on FPGA are very useful to implement multiple functions by using partial bitstreams, which can time-share the same FPGA resources. Here, the full search block matching motion

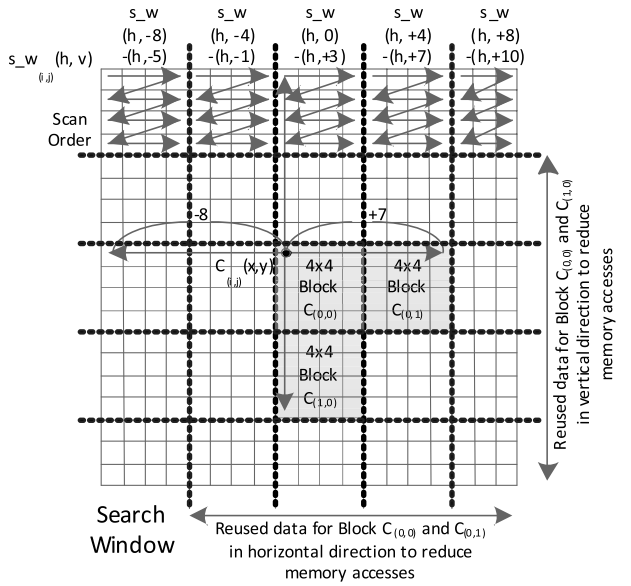

Fig. 4 Scanning of pixel information in the search window $[-8, +7]$.

estimation algorithm is separated into different PRRs. Each PRR is used to implement motion estimation PE arrays. They generate minimum SAD and motion vector for smaller 4×4 blocks, starting from the top left block vertically downwards. For example, if only one PRR is configured, the SADs of all the 4×4 blocks are computed by the FPGA resources in this PRR only. If two PRRs are configured, the first PRR computes SADs of the top eight blocks and the second one does SADs of the bottom eight blocks in parallel, and so on as shown in Fig. 2.

Figure 3 shows the internal structure of a 16×1 PE array implemented. This unit consists of 16 PEs responsible for simultaneously calculating SADs of all the search locations of one row $[-8, +7]$ (i.e., 16 pixels) in the search window. The four PRRs are responsible for calculating SADs of four different blocks simultaneously. Among these, one, two, or four PRRs can be configured for computation. Now, if only one PE array is used for a specific block, then parallelism exists row-wise. By increasing the number of such PRRs and implementing synchronized data flow for maximizing data reuse, the SADs and MVs of neighboring 4×4 blocks

Table 1 Dataflow when one PRR is used for motion estimation with search window $[-8, +7]$.

$sw(h, v)$	$C_{(i,j)}(x, y)$	SAD locations (h, v)	$SAD_{(i,j)}$, i.e., SADs of corresponding 4×4 Block $C_{(i,j)}$ (# of PRR = 1)
$sw(-8,-8) - (-8,10)$			
$sw(-7,-8) - (-7,10)$	$C_{(0,0)}(0,0) - C_{(0,0)}(0,3)$		
$sw(-6,-8) - (-6,10)$	$C_{(0,0)}(1,0) - C_{(0,0)}(1,3)$		
$sw(-5,-8) - (-5,10)$	$C_{(0,0)}(2,0) - C_{(0,0)}(2,3)$		
$sw(-4,-8) - (-4,10)$	$C_{(0,0)}(3,0) - C_{(0,0)}(3,3)$		
$sw(-3,-8) - (-3,10)$	$C_{(0,0)}(0,0) - C_{(0,0)}(0,3)$	$(-8,-8) - (-8,7)$	While calculating SADs in the given search window, Min. $SAD_{(0,0)}$ will be selected at the end of the cycles.
$sw(-2,-8) - (-2,10)$	$C_{(0,0)}(1,0) - C_{(0,0)}(1,3)$	$(-7,-8) - (-7,7)$	
$sw(-1,-8) - (-1,10)$	$C_{(0,0)}(2,0) - C_{(0,0)}(2,3)$	$(-6,-8) - (-6,7)$	
$sw(0,-8) - (0,10)$	$C_{(0,0)}(3,0) - C_{(0,0)}(3,3)$	$(-5,-8) - (-5,7)$	
$sw(1,-8) - (1,10)$	$C_{(0,0)}(0,0) - C_{(0,0)}(0,3)$	$(-4,-8) - (-4,7)$	
$sw(2,-8) - (2,10)$	$C_{(0,0)}(1,0) - C_{(0,0)}(1,3)$	$(-3,-8) - (-3,7)$	
$sw(3,-8) - (3,10)$	$C_{(0,0)}(2,0) - C_{(0,0)}(2,3)$	$(-2,-8) - (-2,7)$	
$sw(4,-8) - (4,10)$	$C_{(0,0)}(3,0) - C_{(0,0)}(3,3)$	$(-1,-8) - (-1,7)$	
$sw(5,-8) - (5,10)$	$C_{(0,0)}(0,0) - C_{(0,0)}(0,3)$	$(0,-8) - (0,7)$	
$sw(6,-8) - (6,10)$	$C_{(0,0)}(1,0) - C_{(0,0)}(1,3)$	$(1,-8) - (1,7)$	
$sw(7,-8) - (7,10)$	$C_{(0,0)}(2,0) - C_{(0,0)}(2,3)$	$(2,-8) - (2,7)$	
. . .	$C_{(0,0)}(3,0) - C_{(0,0)}(3,3)$	$(3,-8) - (3,7)$	
. . .	. . .	$(4,-8) - (4,7)$	
. . .	. . .	$(5,-8) - (5,7)$	
. . .	. . .	$(6,-8) - (6,7)$	
. . .	. . .	$(7,-8) - (7,7)$	

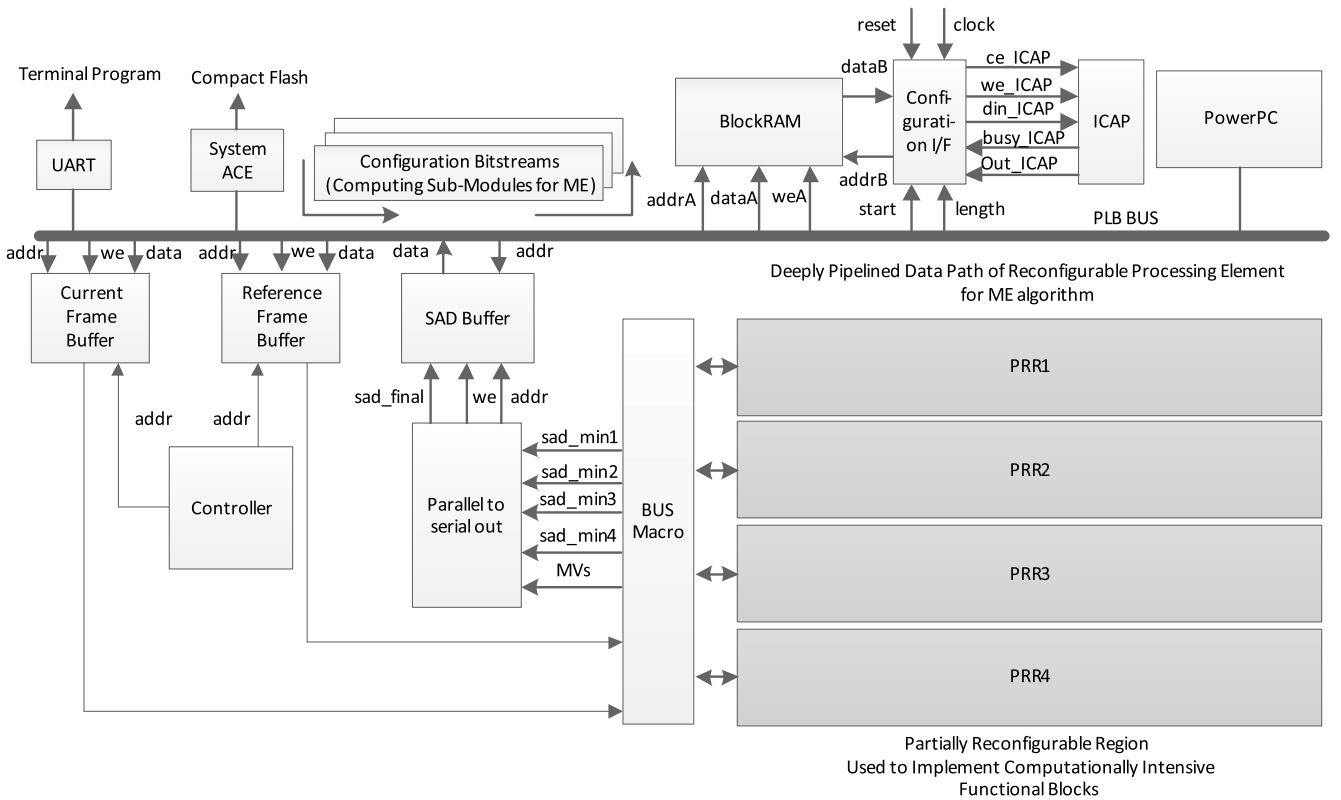

Fig. 5 Self-reconfigurable platform for scalable ME computation.

are computed in parallel, hence improving performance per MB. Partial bitstreams of motion estimation PE arrays are used as libraries by the embedded processor to provide real-time high throughput computing capabilities with reconfigurable hardware fabrics on FPGA.

In the proposed architecture, the search window is divided into five columns, and the pixel data are scanned simultaneously from these columns as shown in Fig. 4. During the initial four clock cycles, the pixel information of the first row in the search window is stored in the latches, and in the next clock cycle, the current pixel information ($c(0,0)$) becomes available to each PE when it starts SAD computations. During this cycle, the next row pixel information of the search window is continuously read and pipelined into the latches as shown in Fig. 3. These partial SADs are stored in each PE, and after 16 clock cycles, SADs are propagated into the comparator unit for motion vector selection. According to the number of PRRs operating, the controller unit can efficiently have the data flow pipelined among the PRRs to increase data reuse. Table 1 shows data flow when one PRR is used to implement a 16×1 PE array for SAD computations.

2.2 *Self-Reconfigurable Platform for ME*

In our previous work,[11] modular-based architecture for ME has been proposed to demonstrate its scalable computing capability to support different video resolutions and frame rates by dynamic partial reconfiguration of PE arrays on FPGA during run-time. In this paper, we extend our previous work to improve reconfiguration speed of partial bitstreams to implement the proposed ME algorithm in H.264/AVC by employing bitstream compression through the LZSS algorithm and On-Chip BlockRAM as bitstreams' internal cache.

Since reconfiguration time is very critical for many applications, such as real-time multimedia processing, many approaches have been proposed to reduce the reconfiguration time. In Ref. 12, a waveform-like architecture based on data graph is used to reduce reconfiguration overhead. The partial bitstreams stored on the external SRAM memory are loaded using MicroBlaze processor serially. In Ref. 13, an area efficient ICAP controller was designed and connected to the processor local bus (PLB). The required bitstreams are preloaded from a CF card into DDR SDRAM during initialization phase. Their ICAP controller can access the bitstreams from the SDRAM. Due to the partial bitstreams stored on the external memory for Refs. 12 and 13, the overall reconfiguration overhead to perform dynamic partial reconfiguration is increased. In Ref. 14, the compressed partial bitstreams are stored on the BlockRAM, and a hardware core was implemented to load the partial bitstreams to ICAP directly for reconfiguration. However, their approach requires decompression operations during the reconfiguration process, which increases real-time reconfiguration overhead. In addition, their approach will be inefficient when the number of partial bitstreams is increasing since they store all the partial bitstreams in the BlockRAM.

In our design, while ME computation is performed, the next compressed partial bitstream to be used for increasing or decreasing the throughput of ME computation is loaded and decompressed through PowerPC, and the decompressed partial bitstream is pre-stored into the BlockRAM. Therefore, the proposed approach can provide two benefits.

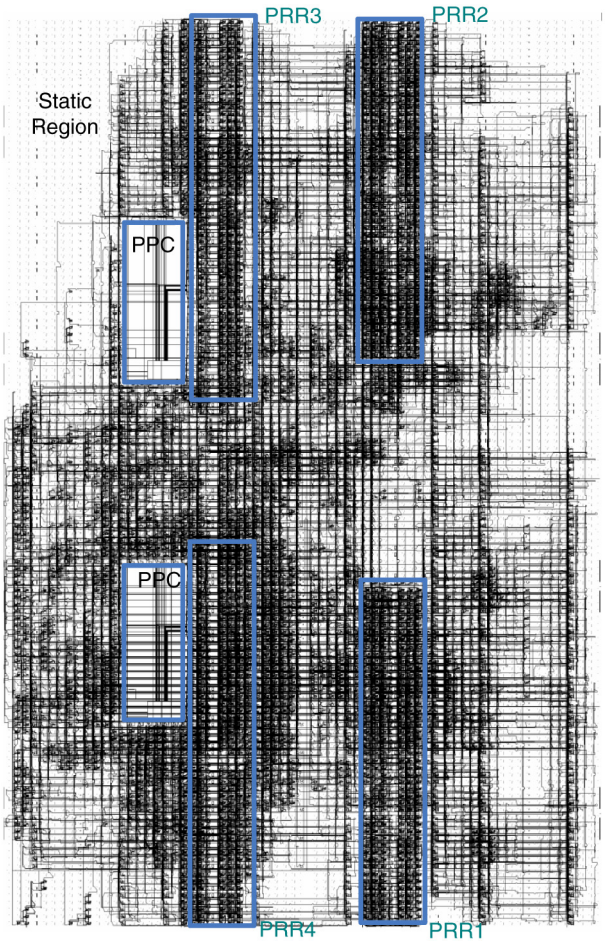

Fig. 6 Layout of our design.

First, latency overhead for loading and decompressing the partial bitstream of ME from the off-chip memory can be hidden since these operations are performed during the normal ME computation. Second, BlockRAM size can be reduced since only one partial bitstream to adjust ME throughput is pre-stored into the BlockRAM. Therefore, the proposed approach can enhance real-time reconfiguration rate and be more adaptable according to the required computing capability, which can vary based on the input resolution and frame rate of video sequences.

The self-reconfigurable platform for our scalable ME design is shown in Fig. 5. The PowerPC is used to implement the configuration manager that provides self-reconfiguration capability. It also prefetches the compressed partial bitstream, decompresses, and stores it in the BlockRAM on the FPGA. LZSS, i.e., a dictionary-based lossless data compression algorithm is used for bitstreams' compression.[15] It attempts to replace a string of symbols with a reference to a dictionary location of the same string. Its decoding steps are shown as follows:[16]

1. Initialize the dictionary to a known value.
2. Read the encoded/not encoded flag.
3. If the flag indicates an encoded string:
   3(a). Read the encoded length and offset, then copy the specified number of symbols from the dictionary to the decoded output.
   3(b). Otherwise, read the next character and write it to the decoded output.
4. Shift a copy of the symbols written to the decoded output into the dictionary.
5. Repeat from 2 until the entire input has been decoded.

The BlockRAM in Fig. 5 is a dual-port RAM on FPGA. "dataA" and "dataB" are two data ports of the RAM with 32-bit width. "addrA" and "addrB" are two address ports of the RAM with 14-bit width. "weA" is the write enable signal used for write or read operations of the RAM. The depth of the BlockRAM is made large enough to fit the largest partial bitstream of ME. The initial configuration file and all the partial bitstream files for ME are stored in the compact flash, and System ACE is used to connect the compact flash to the PLB Bus. Universal asynchronous receiver transmitter (UART) is used as a user interface through HyperTerminal. The Configuration Interface module developed is the customized hardware module, which controls the run-time

Table 2 Comparison of original bitstreams and compressed bitstreams.

	Original bitstream (Bytes)	Compressed bitstream (Bytes)	# of accesses—Original	# of accesses—Compressed	Accesses saving	Reconfig. time (μsec)
PRR1	71676	41042	140	81	42.1%	206.6
Blank1	36432	6676	72	14	80.6%	105
PRR2	41431	41431	141	81	42.6%	207.4
Blank2	7576	7576	81	15	81.5%	119
PRR3	45418	45418	182	89	51.1%	268
Blank3	11632	11632	122	23	81.1%	179.3
PRR4	49907	49907	180	98	45.6%	264.3
Blank4	16490	16490	132	33	75%	194.4

Table 3 Hardware resources.

	Slice Flip Flops	LUTs	RAM16
Configuration I/F & BlockRAM modules	58	37	58

reconfiguration, i.e., sending the partial bitstream of ME from BlockRAM to ICAP directly to reduce the loads of the PowerPC. The control flow of the configuration interface module is shown as below:

1. Assert "we_ICAP," which is the write enable pin of the ICAP interface.
2. After at least one clock cycle, assert "ce_ICAP," which is the chip enable pin of the ICAP interface.
3. Send 32 bits configuration data on din_ICAP while checking "busy_ICAP" signal from ICAP. It is the handshaking signal, indicating whether ICAP is ready to accept new configuration data or not.
4. Increase address counter.
5. If it is final address, go to 6, else go to 3.
6. Wait at least eight clock cycles.
7. Deassert "ce_ICAP."
8. Deassert "we_ICAP" after at least one clock cycle.

The "length" signal provides the length of the bitstream to the Configuration Interface module, and the "start" signal is used to initiate the reconfiguration process. These signals with "addrA," "dataA," and "weA" are all generated by PowerPC via PLB Bus to interact with ICAP.

The ME module is divided into a static region and a reconfigurable region. The static region of the ME module consists of the current frame buffer, reference frame buffer, controller, parallel to serial out, and SAD buffer. These modules remain unchanged after initial configuration. The controller generates the address to fetch the data from the current frame buffer and the reference frame buffer. The final results of 12 bits SAD value and 8 bits motion vector information are stored in the SAD buffer. In this work, four modules, i.e., from PE Array1 to PE Array4, are included and tested to implement scalable ME computation. Each module is defined as a partial reconfigurable region. Bus macros (BMs) are used to connect signals between the static region and each partial reconfigurable region.

Internal buffers are used to store pixel information of the reference frame and the current frame. In our implementation, we use quarter common intermediate format (QCIF) images (176×144 resolution) and 16×16 MB to calculate 16 minimum SADs of 4×4 blocks. The search range used for our implementation purpose is $[-8, +7]$. The control unit is responsible for the data flow for the reference frame and the current frame into the PE arrays. This module takes control of the full search process and synchronizes data flow of all the components. Data is broadcasted and shifted with the clock signal into the PEs from internal buffers. SADs and MVs coming from different PRRs are stored into the SAD buffer.

3 Experimental Results

In this section, we discuss the performance of our proposed ME architecture for different operating PRRs with internal fixed PE array (16×1). Our self-reconfigurable design is implemented in Xilinx Virtex-4 ML410 development system. Xilinx EDK is used to create the embedded processor system on FPGA. ISE is used for the synthesis process, and Planahead is used for floorplanning, placement, and routing. The compressed partial reconfigurable bitstreams for ME and system.ace files are stored in the compact flash card. The initial system.ace file size is 2.72 MB. Once the FPGA is powered on, it is configured with the initial configuration bitstream. If we use non-partial reconfiguration approach, for example, three ace files consuming 8.16 MB are needed to implement three types of ME modules. Therefore, the non-partial reconfiguration approach to implement the proposed ME computation becomes quite inefficient due to the increasing size of the bitstreams and longer latency to load them from the external memory, which limits real-time adaptive computing capabilities of ME modules according to time-varying characteristics of incoming video sequences.

To fully utilize the dynamic partial reconfigurable capabilities of the FPGA device, we first generated the initial configuration bitstream with empty BlockRAM. Then, partial bitstreams for different ME modules are generated. After storing all of the initial configuration bitstream and the partial bitstreams on the compact flash, the FPGA is ready to be powered on. First, the device is configured with the initial bitstream, and the compressed partial bitstream of ME is prefetched from the CF card and decompressed by the PowerPC while the ME computation is being performed. Then, the decompressed partial bitstream is stored in the BlockRAM within the FPGA so that the Configuration Interface module is able to initiate the required dynamic partial reconfiguration by sending the configuration data from BlockRAM to the

Table 4 Comparison of reconfiguration rates.

	Ref. 12	Ref. 13	Ref. 14	Proposed
Reconfiguration Rate	5.1 MB/sec	94.88 MB/sec	50 MB/sec	367 MB/sec
Device	Virtex 2-2000	Virtex-II Pro	Spartan-3S200	Virtex-4 FX60
ICAP Frequency	66 MHz	100 MHz	50 MHz	100 MHz

Table 5 Comparison of different hardware architectures for VBSME Algorithm.

Architectures	Bit width for ref. frame (bits/cycle)	Bit width for current frame (bits/cycle)	Bandwidth for ref. Frame (Kbits/MB)
[17]	136	—	48
1 PRR	40	8	184
2 PRRs	40	16	92
4 PRRs	40	32	46

ICAP interface directly. The layout of our design in Fig. 6 shows PowerPCs and four partial reconfiguration regions. The rest of the area is used as a static region.

Thirty-two-bit wide 100 MHz ICAP interface is available for fast reconfiguration of Virtex-4 devices. Our reconfiguration rate is 367 MB/sec, and the maximum frequency of our design is 91.7 MHz. The overall compressed bitstream size for four modules implemented in this work is 215 KB, including four blank partial bitstreams and four functional partial bitstreams of ME, as shown in Table 2. The original partial bitstreams' size is 523 KB. Therefore, the overall bitstream size is reduced by 58.9% through the compression of the original partial bitstreams of ME. We use blank partial bitstream to configure the PRR so that it has no switching activities, resulting in reduction of dynamic power consumption.

Comparison of external memory accesses to load original partial bitstreams and compressed partial bitstreams is also included in Table 2. Each time PowerPC accesses the CF card, it fetches 512 bytes of configuration data. The number of external memory accesses is reduced by 62.4% on average due to the compression of the partial bitstreams. The hardware resources required to implement the Configuration Interface module and the BlockRAM module are shown in Table 3.

As shown in Table 4 for comparison with previous works,[12–14] the proposed method achieves the reconfiguration rate of 367 MB/sec. The main reason for this improvement is that we use the 100 MHz 32-bit ICAP mode for the fast reconfiguration. Furthermore, using on-chip BlockRAM as a configuration cache to pre-store the partial bitstream of ME can remove all the latency overhead caused by the PLB bus operations and the time for accessing external memory. Therefore, our approach can achieve the maximum reconfiguration speed of Virtex-4 FPGA while performing computation-intensive ME operations using reconfigurable fabric on FPGA.

Table 5 shows bit widths for the reference frame and current frame, and bandwidth for the reference frame, summarizing the level of parallelism and data reuse. Here, the memory bit width is defined as the number of bits that the algorithm accesses from the memory in each cycle. It is one of the factors to determine efficient memory management and degree of parallelism. Memory bandwidth is defined as the number of memory accesses to complete ME for each MB. It is the one to determine the efficient data reuse capability. Table 5 shows that the proposed approach is highly scalable, and data reuse is significantly enhanced by increasing the number of PRRs when compared to Ref. 17. While only one PRR can be configured in applications where hardware resource is a critical factor, the proposed reconfigurable ME engine can speed up with the increase in PRRs.

4 Conclusion

We present an FPGA design for the proposed variable block size scalable ME algorithm using dynamic partial reconfiguration. Its modular design and regular data flow structure make it attractive for the implementation of full search variable block size algorithm. Simulation results show that our design can increase data reuse significantly and thereby reduce the memory bandwidth overhead. Compressed bitstreams are used to reduce external memory accesses and storage sizes. BlockRAM is used as a cache to reduce the reconfiguration overhead. The whole design is implemented in Virtex-4 ML410 evaluation board. The PowerPC is included to control the reconfiguration of scalable ME architecture to exploit trade-offs among different requirements set by the system. It also controls the prefetching and decompression of the partial bitstreams. Experimental results show that our approach can reduce the external memory accesses by 62.4%, and achieve 367 $MB/$ sec reconfiguration rate.

Acknowledgments

This work was supported by 2012 Hongik University Research Fund.

References

1. T. Wiegand and G. J. Sullivan, "The H.264/AVC video coding standard," *IEEE Signal Process. Mag.* **24**(2), 148–153 (2007).
2. I. E. G. Richardson, *H.264 and MPEG-4 Video Compression*, Wiley, Hoboken, NJ (2003).
3. H.264/AVC Reference Software, http://iphome.hhi.de/suehring/tml/.
4. H. Yin et al., "A hardware-efficient multi-resolution block matching algorithm and its VLSI architecture for high definition MPEG-like video encoders," *IEEE Trans. Circ. Syst. Video Tech.* **20**(9), 1242–1254 (2010).
5. W. C. Chung, "Implementing the H.264/AVC video coding standard on FPGAs," *Xilinx Xcell J.* (51), 40–43 (2004).
6. M. Sayed, W. Badawy, and G. Jullien, "Towards an H.264/AVC HW/SW integrated solution: an efficient VBSME architecture," *IEEE Trans. Circ. Syst. II, Exp. Briefs* **55**(9), 912–916 (2008).
7. I. Amer et al., "An efficient variable block size selection scheme for the H.264 motion estimation," *International Workshop on System-on-Chip for Real-Time Applications*, Cairo, Egypt, IEEE, Washington DC, USA, 5–9 (2006).
8. P. Lysaght et al., "Invited paper: enhanced architectures, design methodologies and CAD tools for dynamic reconfiguration of Xilinx FPGAs," *International Conference on Field Programmable Logic and Applications*, Madrid, Spain, IEEE, Washington DC, USA, **30**, 1–6 (2006).
9. Xilinx Inc., PlanAhead User guide, http://www.xilinx.com/support/documentation/sw_manuals/PlanAhead_UserGuide.pdf.
10. Xilinx Inc., Partial Reconfiguration Design Example with PlanAhead, http://www.xilinx.com/support/prealounge/protected/archive_91.htm.
11. S. Kodipyaka and J. Lee, "A scalable H.264/AVC variable block size motion estimation engine using partial reconfiguration," *International Conference on Engineering of Reconfigurable Systems and Algorithms*, Las Vegas, Nevada, USA, CSREA, San Diego, CA, USA, 219–225 (2009).
12. L. Braun et al., "Data path driven waveform-like reconfiguration," *International Conference on Field Programmable Logic and Applications*, Heidelberg, Germany, IEEE, Washington DC, USA, 607–610 (2008).
13. C. Claus et al., "A new framework to accelerate Virtex-II Pro dynamic partial self-reconfiguration," *IEEE International Parallel and Distributed Processing Symposium*, Long Beach, CA, USA, IEEE, Washington DC, USA, 1–7 (2007).
14. S. Bayar and A. Yurdakul, "Self-reconfiguration on Spartan-III FPGAs with compressed partial bitstreams via a parallel configuration access

port (cPCAP) core," in *Proc. Ph.D. Res in Microelectron. Electron.*, Istanbul, Turkey, IEEE, Washington DC, USA, 137–140 (2008).
15. J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inform. Theor.* **IT-23**(3) 337–343 (1977).
16. M. Dipperstein, *LZSS (LZ77) discussion and implementation*, http://michael.dipperstein.com/lzss/.
17. C. Y. Chen et al., "Analysis and architecture design of variable block-size motion estimation for H.264/AVC," *IEEE Trans. Circ. Syst.* **53**(3), 578–593 (2006).

Jooheung Lee has been working on various topics in the areas of multimedia signal processing algorithms and low power VLSI systems design. His research interests include image and video coding algorithms, multimedia systems, power aware and reliable VLSI systems design, and reconfigurable computing for signal processing applications. Previously, he worked at the Wireless Multimedia Communications Laboratory at the R&D Complex of LG Electronics in 1998, where he worked on low power video codec ASIC design for mobile applications. After completing his PhD at Pennsylvania State University in 2006, he joined the Department of Electrical Engineering and Computer Science at the University of Central Florida, Orlando, Florida, where he was a full-time faculty member. Currently, he is an assistant professor of the Department of Electronic and Electrical Engineering at Hongik University, Republic of Korea.

Chul Ryu received his MS and PhD degrees in electrical engineering in 1991 and 1997, respectively, both from the Polytechnic Institute of New York University. From 1998 to 1999, he served as a research engineer of advanced wireless technologies lab at LG Electronics. He has been a professor of the Department of Information and Communication Engineering, Dongguk University since 1999. His research area includes visual communication, super-resolution reconstruction, and real-time hardware design for video codec.

Soontae Kim has been with Department of Computer Science as an assistant professor since June 2007. He was an assistant professor in the Department of Computer Science and Engineering at University of South Florida at Tampa from fall 2004 to spring 2007. He earned the PhD degree in computer science and engineering from the Pennsylvania State University in December 2003. His research interests include embedded system/software, computer architecture, low-power, and reliable and real-time computing.