Dynamically Parameterized Algorithms and Architectures to Exploit Signal Variations

Prashant Jain, Andrew Laffely, Wayne Burleson, Russell Tessier, and Dennis Goeckel
University of Massachusetts Amherst

309 Knowles Engineering Building
University of Massachusetts Amherst
Amherst, MA 01003

January 2002

Abstract: Signal processing algorithms and architectures can use dynamic reconfiguration to exploit variations in signal statistics with the objectives of improved performance and reduced power. Parameters provide a simple and formal way to characterize incremental changes to a computation and its computing mechanism. This paper develops a framework for dynamic parameterization and applies it to MPEG-4 motion estimation. A novel motion estimation architecture facilitates the dynamic variation of parameters to achieve power-compression tradeoffs. Our work shows that parameter variation in motion estimation helps achieve power reduction by an order of magnitude, trading off higher compression for lower power. The magnitude of the tradeoffs depends on the input signal variation. The monitoring of input and output signal statistics and subsequent variation of parameters is accomplished by a hardware controller. To provide the controller with a model of the parameter space and corresponding measures in terms of power and performance, a configuration sample space graph is developed. This graph identifies the parameters which present the best power-performance tradeoffs. The controller samples the operating environment to select the appropriate parameters. This reduces the average power consumption by 40% for 2% loss in compression. Four other signal dependent computations : 1) 2D Discrete Cosine Transform, 2) Lempel-Ziv lossless compression, 3) 3D graphics light rendering and 4) Viterbi decoding are briefly discussed to demonstrate the applicability of dynamic reconfiguration.

1 Introduction

Reconfigurable computing has been proposed for signal processing with various objectives, including high-performance, flexibility, specialization and most recently adaptability. Adaptive systems attempt to balance the use of limited resources, such as processing elements, time, and power, through the use of environment-driven dynamic reconfiguration. An adaptive system can be characterized by: 1) reconfiguration speed, 2) the amount of reconfiguration, and 3) the number of configurations used. Microprocessors are, in general, well suited for adaptive systems but lack the performance necessary for compute-intensive applications. Although FPGAs provide parallelism, they may not meet the performance and power needs of adaptive systems due to slow reconfiguration rates and high power dissipation. ASICs provide high performance at low power consumption, although they generally lack the flexibility needed for use in adaptive systems.

This work discusses the development of application-specific adaptable architectures for signal processing. These consist of a partially predefined configuration architecture style to achieve high performance, low power consumption and high application specific flexibility. Although adaptation can be used to manage any aspect of performance, the goal of the proposed architecture development is to reduce overall power consumption. It will be shown that by applying adaptability at the algorithm and architecture levels it is possible to reduce power consumption by several orders of magnitude. Specifically, power consumption can be reduced by: 1) exploiting variations in system signals and reducing the computations required to achieve a given level of quality, or 2) compromising the quality of the algorithm result. Both of these methods require an understanding of the system configuration space. The application specific architectures presented in this paper use parameters as a formal means for implementing dynamic reconfiguration. This process of *dynamic parameterization* varies architectural parameters at run time in response to input signal variations.

In this paper, five components of large heterogeneous systems for wireless multimedia are examined. Section 2 gives a brief description of some recent configurable architectures. Section 3 presents a discussion on *dynamic parameterization* and the methodology of selecting a parameter to be reconfigurable. A motion estimation architecture is described in detail, illustrating our parameterized architectural approach in section 4. For motion estimation, a test architecture is presented along with a complete parameter analysis. Based on this analysis, a simple adaptation controller has been developed and tested achieving 40% power savings with roughly 2% quality degradation. In Section 5, the results of multiple parameter analyses are shown to illustrate the applicability of *dynamic parameterization* across a family of multimedia applications.

2 Configurable Architectures

Recently, it has been recognized that heterogeneous, domain-specific reconfigurable architectures achieve higher performance with lower energy consumption then FPGA's for DSP operations[1]. Zhang [2] proposed a similar domain specific approach and demonstrated it with functional silicon. This system [2] focuses on reconfiguration for task and standards specialization using domain-specific processors. These processors feature varying degrees of configuration granularity and are connected by a reconfigurable mesh interconnect network. It realizes low-energy operation through the use of a combination of architectural and circuit-level optimizations. Kuhn [3] presents a high performance and a low power flexible architecture for motion estimation. The work suggests a programmable processor based implementation for an MPEG-4 encoder supporting various modes and video objects. Due to the disadvantages of limited flexibility and high power consumption, ASICs and microprocessors were not used. Instead, flexible VLSI architectures were developed to support several motion estimation algorithms.

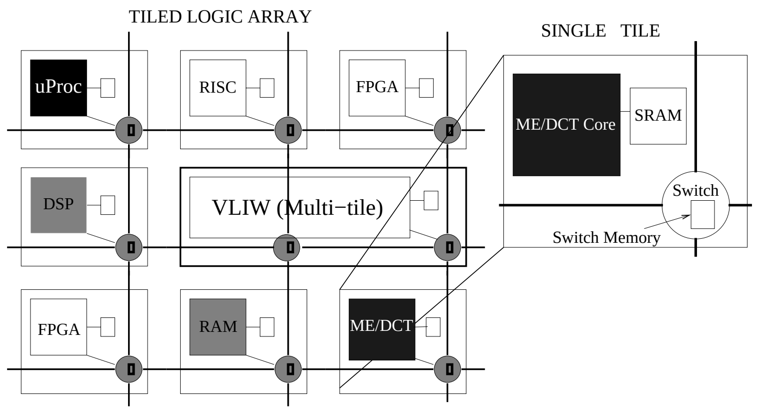

Figure 1 Tiled architecture for aSOC

Our work develops component subsystems for an adaptive system founded on a reconfigurable interconnect. This adaptive System on a Chip (aSoC) architecture [4] allows diverse computing modules to be implemented in a tiled structure as shown in figure 1. A statically scheduled interconnect fabric is used to connect the cores. These cores can consist of general-purpose systems such as RISCs, DSPs, RAMs and FPGAs, as well as application specific blocks like the ones discussed in this paper. In [5], aSoC is shown to support configurable cores with minimal power and throughput overhead.

Our current work focuses on the application-specific cores needed to achieve high performance levels for computations such as motion estimation (ME), discrete cosine transform (DCT) and Viterbi decoding. A rich set of efficient algorithms and architectures already exist for these signal processing problems [3, 6]. When implemented, these architectures usually trade flexibility for performance and efficiency. Recently the use of ASIC "point" systems to make a power aware composite system is discussed in [7]. In operation, a single point systems is activated based on the system environment. The goal is to organize the composite system such that it operates nearly as efficiently as the optimal point system in a given environment. This work applies similar concepts in developing special purpose tiles, which can be dynamically reconfigured to consume the lowest possible power for a given set of environment variables. In most cases this can be accomplished with little impact on performance. Our focuses here is evaluating the design space to find parameters for adaptation as well as the environment triggers, which will control the reconfiguration.

3 Dynamically Parameterized Algorithms and Architectures

Computational parameters provide a simple and formal way to characterize incremental changes in algorithms and their implementations. As such, these parameters can be classified as either functional or architectural. Functional parameters vary the output of a computation, and typically result in trading output accuracy for speed or power. Architectural parameters keep the output constant allowing tradeoffs between area, performance, power and reliability. Although these parameters can be bound at varying stages of the system's design cycle shown in figure 2, it is our goal to find and vary selected parameters automatically at runtime.

Figure 3 shows a block diagram representation of a dynamically-parameterized system. It consists of a signal processing block and a controller. The processing block consists of the architecture and the algorithms being implemented. The controller sets the parameters of the processing block based on the following inputs:

1. System requirements and constraints (power, performance, etc.)

2. Signal statistics from the input signals

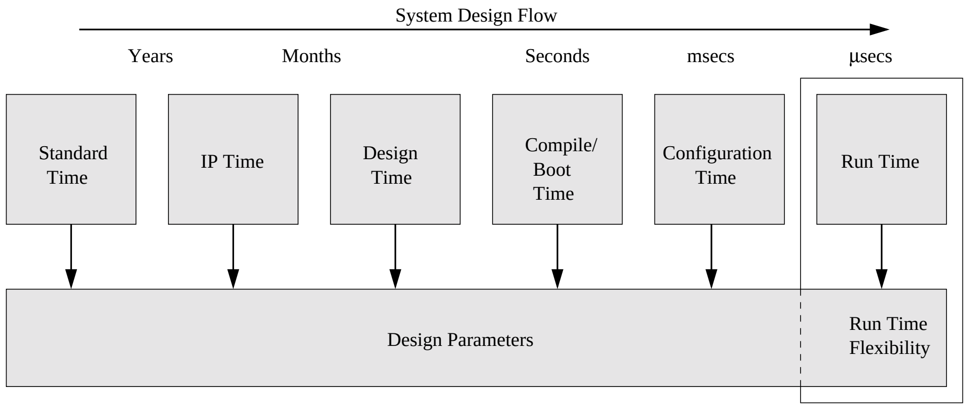

Figure 2 The spectrum of parameter binding times in a system design cycle

3. Algorithm statistics from post processing of the output signals (e.g. motion vectors fed back into the parameter controller)

For most applications there are endless possibilities for parameterization. The decision to allow runtime variation of a given parameter must be made by weighing the costs in system complexity against the potential benefit of adaptation. It is important to limit the extent of adaptation to only those parameters with the best possible performance trade-offs. A first step in parameter selection is the observation of a wide subset of parameter variations and their individual effects on the system's power consumption. Although, a software implementation can give initial insight into the parameter-power tradeoffs, only a fully-parameterized test architecture can accurately evaluate the design space. These observations can be used to not only to develop a controller but also to constrain the required flexibility of the processing block.

The controller represents additional overhead for a dynamically-parameterized system. As such, excessive controller calculations can outweigh the benefit of system adaptation. The controller should be simple and react to the most beneficial system statistics. The key to simplicity is limiting the number and complexity of its inputs, since the controller

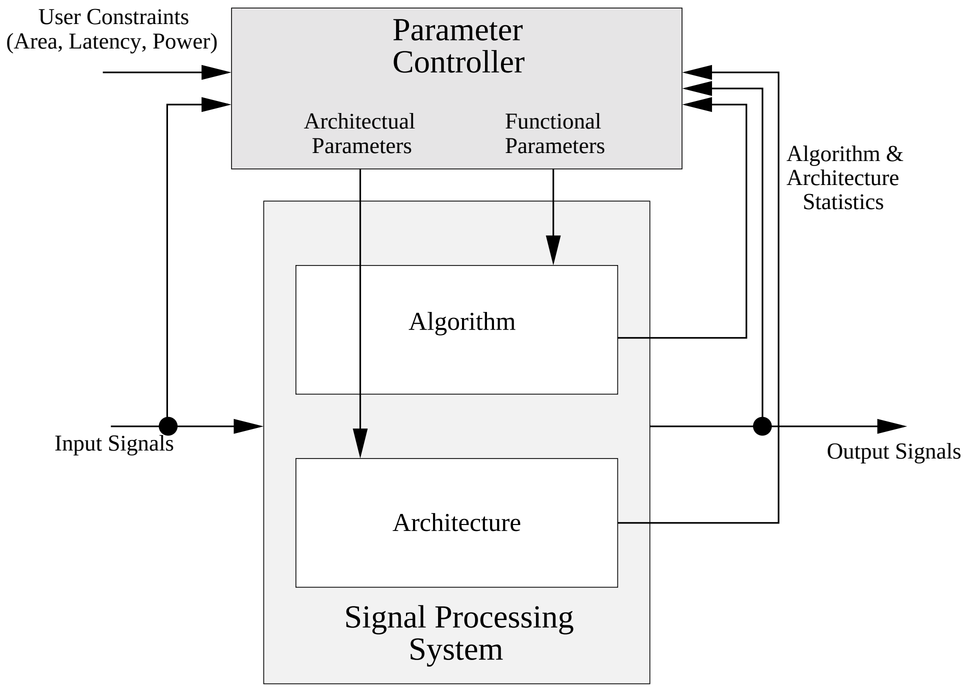

Figure 3 Dynamically parameterized system approach

will have to process all the inputs in search of a reconfiguration triggering event. The best input statistics are highly correlated indicators of specific power-performance tradeoffs. The controller must evaluate these statistics to set the parameters and exploit these tradeoffs. So, minimizing the overhead of this controller requires a thorough understanding of system input and output statistics as well as the user requirements.

4 Dynamically Parameterized Architecture Example: Motion Estimation

Most of the applications targeted by MPEG video standards use lossy coding techniques to meet specified storage and transmission requirements. An important compression technique reduces temporal redundancies by transmitting only the difference between consecutive frames. This technique can be enhanced if the images in the frames can be aligned to minimize the overall difference. In this case, the information is coded as a frame difference and a series of associated alignments, called motion vectors. These motion vectors represent the movement of image components from one frame to the next. A motion estimation process is used to find these motion vectors.

Motion estimation compares a current frame with a previous or sometimes future search frame. The current frame is divided into macro blocks of 16×16 pixels. Each macro block in the current frame is compared against a region in the search frame, referred to as a search window. The coordinates of the best matching block in the search frame form the motion vector for the block under consideration.

The compression aspect of motion estimation reduces the number of bits sent through the rest of the system. Fixed bandwidth systems, which use such a lossy compression scheme inherently trade quality for compression. Thus, compression ratio achieved during motion estimation directly impacts video quality in fixed bandwidth systems.

4.1 Parameterized Motion Estimation Architecture

To analyze the design space, parameter flexibility is designed into a pipelined motion estimation architecture, shown in figure 4. Although independently designed, this matrix based architecture is similar to the GA-2D systolic array designed in [3]. Both architectures use a matrix of processing elements to compute the absolute difference of pixels between the two blocks. Our base architecture contains a pipeline designed primarily for speed, and can evaluate a series of 352×240 frames at 30 frames a second in full search mode at 106MHz. In addition to the processing array, the architecture includes an address generator unit (AGU), which selects how to pull data from memory, and input FIFOs to arrange the data and setup the pipeline. The block in the current frame, the current block, is stored in the array one pixel per processing element and the search pixels are input every clock cycle. The differences calculated at each element are passed down the array and added to eventually compute the Sum of Absolute Differences (SAD) for all the pixels in a current block against a search block.

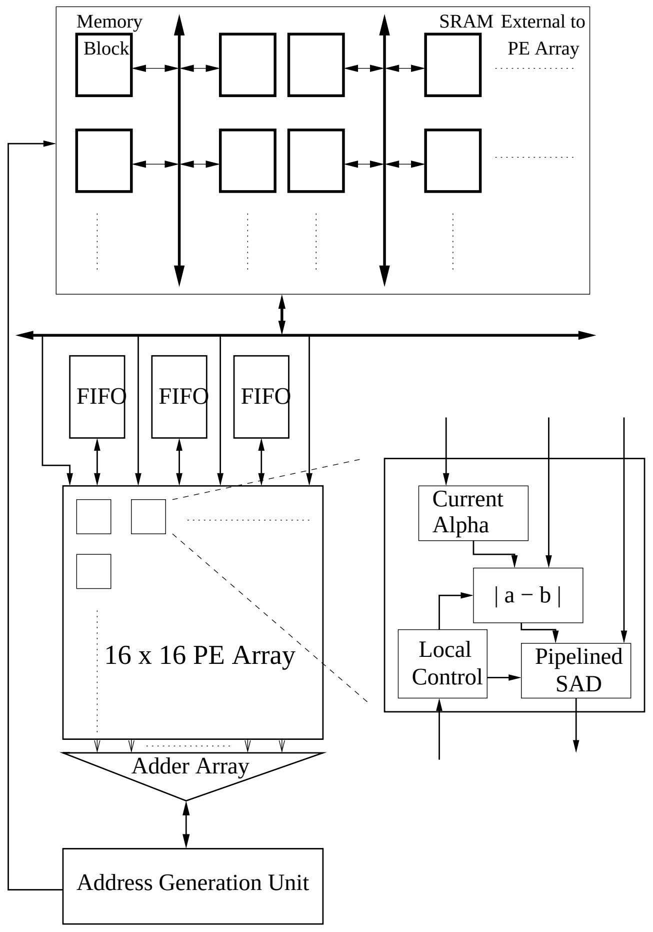

Figure 4 Parameterized motion estimation architecture

This architecture allows variation of the following parameters:

- *Algorithms* - Full search (FSA), 3-step search (TSS) and Spiral search Algorithm[3]

- *Search Window Size in FSA*
- *SAD Threshold for Spiral Search*
- *Pel Subsampling in FSA and TSS*
- *Pixel Width in FSA and TSS*

All search algorithms use the processing element array and differ only in the addresses of the search blocks fetched. The AGU implements a state machine to generate the appropriate memory access patterns for the different search methods and sizes. The FIFOs store and share these fetched pixels to reduce external memory accesses when possible. The SAD thresholding is implemented at the end of the pipeline and signals the AGU to test the next block. Both Pel Subsampling and Pixel Width variation are implemented in the processing array.

All circuit components except the memory, are synthesized with a TSMC .18u standard cell library and evaluated using Synopsys RTL Power Estimator. This tool takes the input design, the TSMC technology file and our video stimulus to calculate system switching activity and the associated power. Power for the unimplemented memory is approximated by counting accesses generated by the AGU. Memory power ranges from 14% of the total power during full search, when the FIFOs can most effectively share data, to 43% in three step search, where the data for each search block must come directly from memory. The input/output overheads of the bus were not evaluated. The motion estimation architecture can be used in both MPEG-2 and MPEG-4 standard encoders. The present implementation, however, does not include an MPEG-4 alpha-plane and no parameters were tested that affect this portion of the MPEG-4 standard.

At this stage it is important to recall that the test architecture has been developed to evaluate the design space, not as a final low power implementation. As such, it incorporates more flexibility then will be needed or even effective in a final system. In spite of this limitation, the test architecture, including reconfiguration overhead and the inefficiencies of standard cell implementation, has comparable performance specifications to recent implementations by Toshiba[8]and Matsushita[9] . Table 1 compares the three implementations.

Parameters	Matsushita	Toshiba	ME 2001
Power Consumption	90mW *	240 mW *	30 mW - 1W
Frame Size	176x144	176x144	352x240
Frame Rate	15 fps	15 fps	30 fps
Clock frequency	54 MHz	60MHz	110 MHz
Algorithms	n.a.	n.a.	FSA, TSS, Spiral
Process	1.8u	2.5u	0.18u
Vdd	1.8V	2.5V	1.8V

Table 1 Architecture comparison with commercially available devices * Power numbers include components not associated with motion estimation including audio processing. Motion estimation numbers and search algorithm are not available, n.a.

It is hoped that the structures and results of this parameter space analysis can be used to develop array based MPEG motion estimation architectures. Clearly, parameter variations, which impact the size of the search space, will alter the power consumption for any implementation. The relative impact of these parameters against those of Pel Subsampling and Pixel Width, however, are only reasonable in similar array-based architectures. In addition, the results are valid for technology scaling as long as leakage power does not dominate.

The following sections show the detailed results for variations selected parameters. Two blocks with extreme cases of motion are pulled from the 'table tennis' sequence shown in figure 5. A low motion block is represented by a block from the background and a high motion block is represented by a part of the ball in motion. The SAD value is used to represent the reciprocal of compression ratio. In these experiments both processing speed and voltage supply are fixed to allow the worst case parameter set to complete 352×240 pixel frames at 30 frames per second. The architecture is in an idle state for parts of the runtime when processing under non-worst-case parameters. The graphs in each of the following sections compares SAD value to dynamic power predicted by Power Estimator. The graphs represent power in mW, and show the relative effects of varying selected parameters.

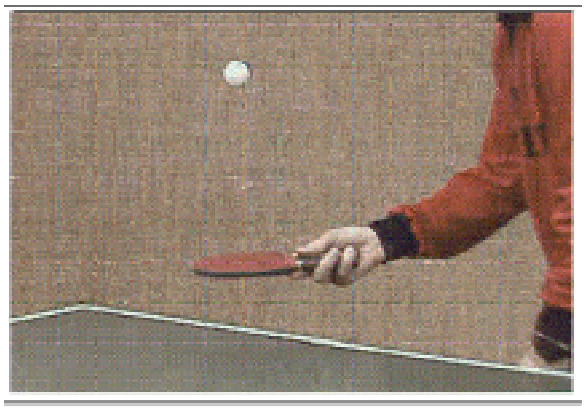

Figure 5 A frame from the table tennis sequence

4.2 Algorithm Selection

The architecture uses one of three different search algorithms for motion estimation: Full, 3-Step, and Spiral [3]. The most basic, the full search algorithm (FSA), compares the current block with every possible block within a specified subset, or search window, of the search frame. The number of blocks checked in FSA depends on the size of the search window.

3-step search (TSS) [3] starts at the coordinates of the current block. A search is performed by comparing the current block to nine evenly spaced search blocks surrounding the starting location. The search block with the best match is the center for the next search stage. In this stage, a similar search is performed around this new center. The only difference is that the radius of the search, called the step size, is now smaller. This two step process continues until the step size is reduced to one pixel. For completion in 3 steps, the third stage separation must be one pixel and the resulting number of comparisons is 27.

The third algorithm used is the spiral search [3]. This algorithm chooses search coordinates in a spiral pattern starting at a prespecified location. The search stops when the selected search block produces a SAD below a specified threshold. As a result, the number of block matching operations required is not fixed.

Figure 6 shows power consumption versus the SAD value for our table tennis example blocks and the different algorithms. A full search window size of 64×32 pixels is used requiring more than 8000 block matching operations. In addition, the spiral SAD completion threshold is fixed at 2762.

The best possible case is to minimize power consumption while reducing the SAD value to increase compression. As expected, full search power consumption is the same for the two macro blocks. As can be seen in figure 6, the full search algorithm always finds the minimum SAD value. It, unfortunately, results in the largest power consumption of the three approaches. TSS power consumption is constant since it always performs 27 SAD calculations. This approach results in reduced power consumption when compared to FSA. Unfortunately, TSS often misses the best match macro block for high motion images. As a result overall compression may be much worse than that achieved using FSA. Spiral search appears to achieve the same compression (SAD) as FSA, but the power consumed varies dramatically between the high and low-motion block types.

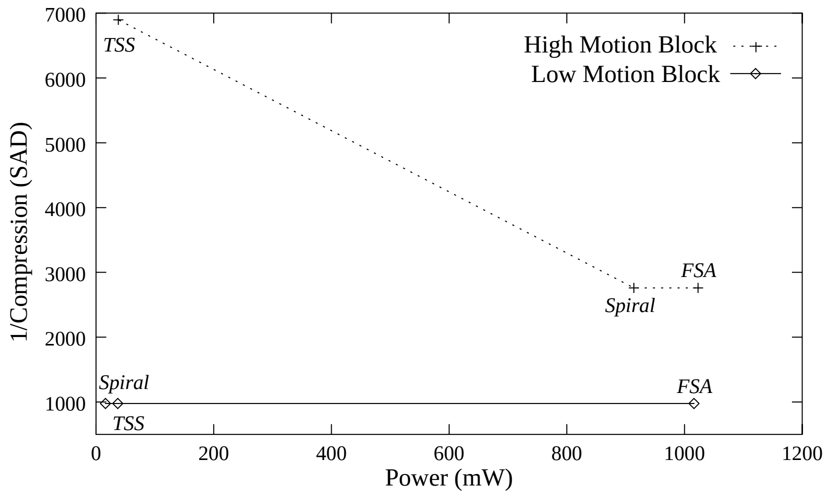

Figure 6 SAD versus power consumption for different algorithms

While figure 6 shows initial observations the results for both FSA and Spiral methods can depend on other parameters. The next subsections explore other parameters to clarify the advantages of each algorithm.

4.3 Search Window Size

A search window is a range of pixels containing the candidate blocks in a search frame. Although a large window searches many blocks, a smaller window size performs fewer search operations and memory accesses, reducing overall power consumption. Our architecture allows a variable search window size in the FSA, provided the size fits within the frame dimensions.

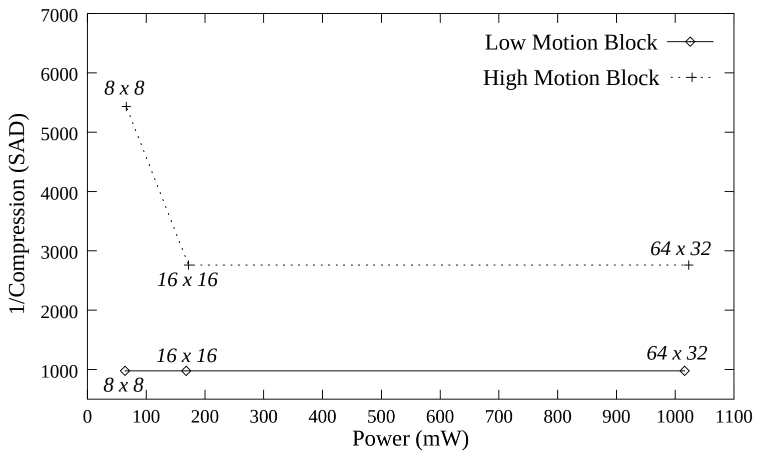

Figure 7 SAD versus power consumption for different window sizes

Figure 7 shows power consumption versus the inverse of compression for three different window sizes. For the block with low motion, all three window sizes provide the same compression, with the 8 × 8 pixel window size consuming an order of magnitude lower power than the 64 × 32 pixel window size. For small or no motion, the best match will be located close to the corresponding current block and can be found using a smaller window size.

For the block with high motion, the range of motion is more than eight pixels. Hence, the 8 × 8 pixel window size is insufficient and unable to capture the best matching block. This is shown in figure 7 by the higher SAD value achieved. The 16 × 16 pixel window size achieves the same compression as the 64 × 32 pixel window size, with a reduction in power consumption by a factor of six.

Figure 7 shows that variation of the search window size results in large changes in power consumption. For example, a change from the size of 64 × 32 pixels to a size of 16 × 16 pixels, reduces power by nearly 84%, while a change from 16 × 16 pixels to 8 × 8 pixels reduces the power an additional 62% for both curves. The optimal window size depends on the amount of motion observed in the input video block. Input signal statistics can be used to dynamically reconfigure to the optimal search window size and achieve the power and compression values desired.

4.4 SAD Threshold for Spiral Algorithm

As stated in section 4.2, spiral search performs a search around a starting location in a spiral fashion until the calculated SAD value is below the set threshold. The number of operations performed depends on the value set for the SAD threshold. A high threshold finds a match early in the block matching process, consuming less power, but achieving low compression. Figure 8 shows a graph for different threshold values for the low and high motion blocks. The lowest power consumption is achieved if the threshold is met during the first block comparison. A lower threshold value increases the required number of matching operations and power consumption, but produces better compression (lower SAD values).

The optimal SAD threshold value for one block may not be appropriate for other blocks that have different ranges of motion. Hence, power and compression may be traded by dynamically varying the threshold for every block, depending on the amount of motion expected in the block.

4.5 Pixel Element (Pel) Subsampling

This architecture supports pel subsampling at two levels. Pel subsampling 2:1 uses every alternate pixel in a block for block matching, while 4:1 uses one for every four pixels. This reduces the number of required operations and saves power. Figure 9 plots power versus the SAD value for the two motion blocks using a full search algorithm with a 64×32

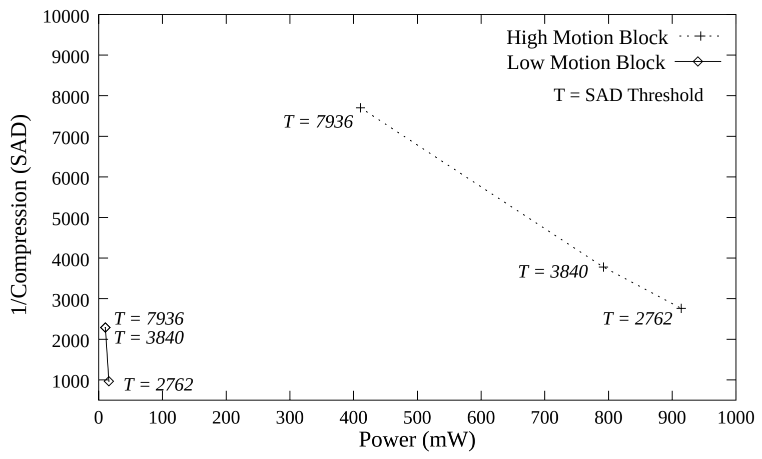

Figure 8 SAD versus power consumption for different SAD thresholds

pixel search window size. Pel subsampling reduces power consumption by up to 29% when compared to full search without subsampling. Pel subsampling 2:1 and 4:1 give negligible compression degradation with respect to full search for both experimental blocks. This implies that Pel subsampling 4:1 and 2:1 can always be used, removing it as a reconfiguration parameter.

4.6 Pixel Width

A pixel typically consists of 8 bits. Least significant bit reduction reduces macro block matching accuracy while achieving a reduction in power. Figure 10 plots power versus SAD value for the two motion blocks using a full search algorithm with a 64×32 pixel search window. Three pixel bit widths (8-bit, 4-bit and 1-bit) are plotted. While the 1-bit configuration shows a power reduction of 32% and higher SAD values, the 4-bit configuration reduces power by 17% with a negligible increase in SAD values over the 8 bit configuration.

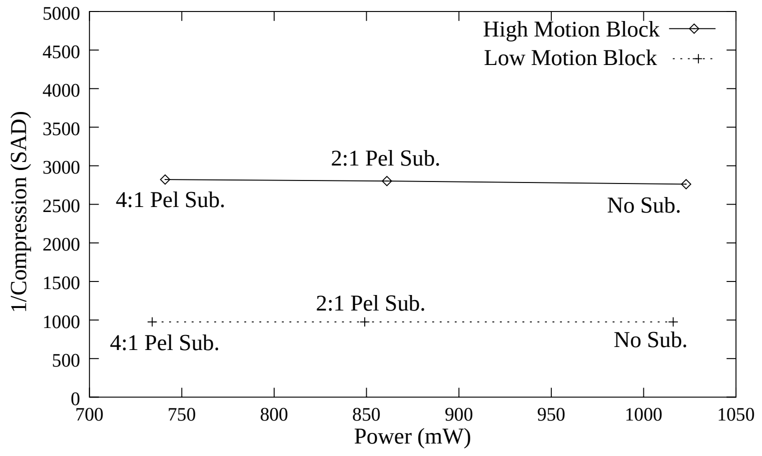

Figure 9 SAD versus power consumption for 2:1 and 4:1 Pel subsampling

Using the four most significant bits of a pixel will give good compression and low power consumption (compared to 8 bits) for *most* blocks. Using less than four bits of a pixel reduces power consumption further at the cost of significant compression degradation. As in Pel subsampling, the effectiveness of bit width reduction shows very low dependence on input signal statistics.

4.7 Configuration Sample Space

Figure 11 combines the results for the various parameters tested. In general, the best operating conditions are represented by points close to the origin with both low power consumption and low SAD values. The configuration points within the solid oval represent the best operating conditions for high motion blocks and are characterized by a full search algorithm with a search window size of 16×16 pixels. Similarly, the dashed oval contains configuration points, which provide low power and high compression for low motion blocks. The points are characterized by a full search algorithm with window sizes of 8×8 and

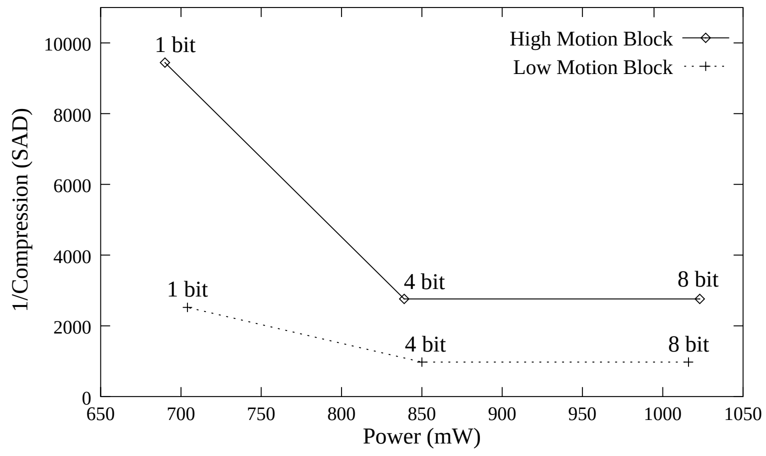

Figure 10 SAD versus power consumption for different bit widths

16×16 pixels. The 3-step and the spiral search techniques also work well with low motion blocks and are included in the dashed oval.

4.8 Controller Variables and Adaptation Triggering

The previous parameter analysis indicates that search window size most clearly delineates the operating conditions best suited for analyzing blocks with high and low motion. This section attempts to find a control stimulus, which can be used to select one of two possible search window sizes; large, 16×16, or small, 8×8. The first step involves finding a way to identify the amount of motion expected in each block so the controller can select, in the second step, the most power efficient window size for processing. In order to perform this analysis, a simple signal statistic must be found to accurately predict the amount of motion without introducing significant controller overhead. Three candidate motion predictor methods were tested for correlation with motion vector magnitude:

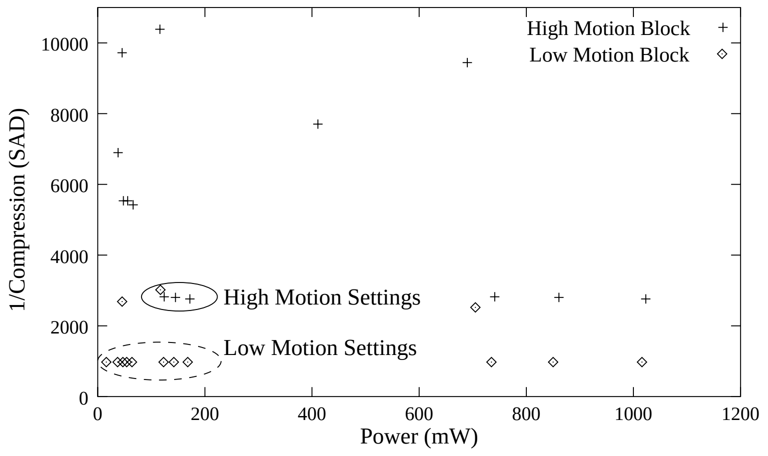

Figure 11 Parameter summary versus SAD value

1. The SAD value of current block and search block with same coordinates: For this correlation method the controller computes the SAD value for collocated blocks in the current and search frames. If this value is larger than a specified threshold it is assumed that the motion vector will also be large. This method introduces an overhead of 330 block SAD calculations per 352×240 pixel frame. This analysis must be performed prior to motion estimation using the existing array of processing elements, leading to reduced pipeline throughput. Additional memory space and dada accesses are needed to accommodate these SAD values.

2. Pixel contrast in the current block (peak to peak difference): The contrast in a block is represented by the difference between its highest and lowest luminance values. When the contrast value is high the block is likely to have high frequency components, which may make the matching process more difficult. As a result, a larger search window may be more appropriate. The controller must calculate the contrast of the current block before attempting to find the motion vectors. The controller would have

to contain this contrast hardware. Pixel contrast can be performed in parallel with motion estimation and will incur no memory overhead.

3. Motion vectors from the previous frame: Motion vectors from the previous frame are used by the controller to predict the motion in the current frame. A simple threshold determines processing with a large or small window size. The only other overhead is additional memory space and data accesses.

To test the correlation of each of the possible predictor methods four video sequences were processed: table tennis, football, flower garden, and mobile. The motion vectors of each frame were found using the immediate predecessor as the reference. Figure 12 shows the correlation between the candidate predictor magnitude and current motion vectors. To make this comparison the pixel contrast and SAD predictor magnitudes were scaled to match the range of magnitudes found in the motion vectors. A candidate predictor is correlated if its value can be used to predict the magnitude of the current motion vectors within 5 pixels. Clearly, the previous motion vectors, with a total of 90% correlation, provide the best prediction method with the least overhead.

To complete this analysis, the predictive motion vector magnitude at which the system would switch from a large to a small search window size was determined. A simple threshold technique is used and our adaptive system is tested with the four sets of video data. When a predictive vector is lager then the threshold the 16×16 pixel window is used. Those predictive vectors smaller then the threshold trigger use the 8×8 pixel window.

Figure 13 shows two sets of data. First, it shows the percentage of blocks, which find the minimum SAD value for a given trigger threshold value. The actual minimum was found by searching the entire frame. Second, it shows the percentage of blocks, which use the smaller search window size for the different trigger thresholds. As this trigger point

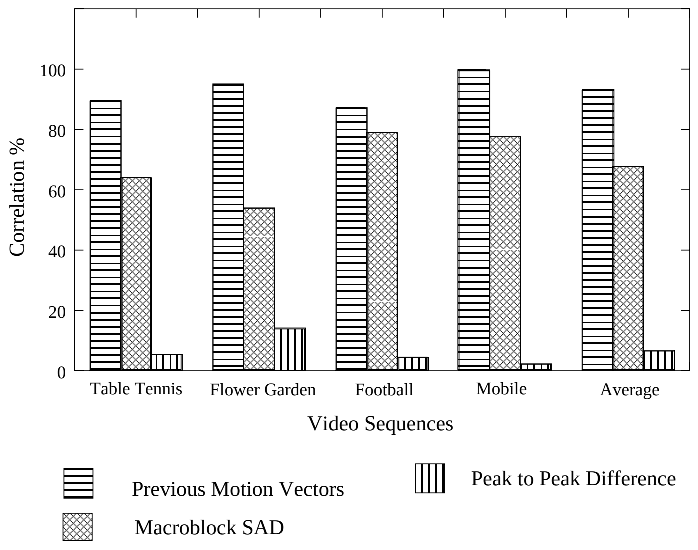

Figure 12 Predictor correlation with current motion vectors

approaches zero more blocks use the large window size. As a result of this larger window, more blocks find the minimum SAD. When the trigger point gets larger more blocks fail to find the minimum SAD. Our data indicates that when all motion vectors are calculated using the large window, 10% of the blocks fail to find the minimum SAD. This indicates that an even lager window is require for 10% of the blocks. Figure 13 shows that setting the threshold at 3 reduces the percentage of minimum SAD found by less than a 1%. At the same time this threshold enables the motion estimation architecture to calculate more then 70% of the vectors using the smaller search window. Using the 62% power savings of the 8×8 pixel window as shown in section 4.3, this system saves over 40% of the power of a static 16×16 pixel search window system. In this approximation, the predicted vector magnitude and trigger point comparison operations are assumed to consume little power. They are calculated only once per block in contrast to the 65k additions required to find the motion vectors in an 8×8 search window.

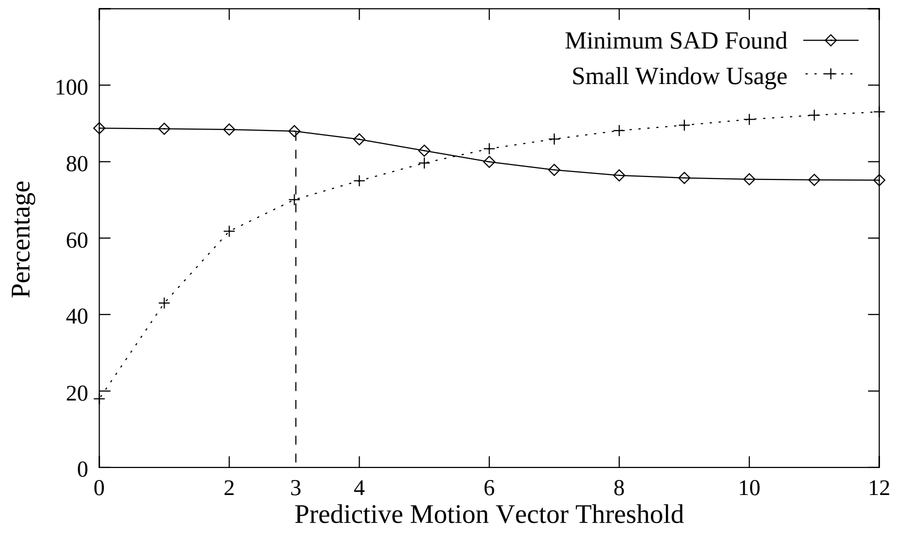

Figure 13 Search window selection and resulting performance for variations in predictive motion vector threshold

5 Other Computations

The design methodology applied to motion estimation was also applied to four other applications. A configurable architecture for each application was implemented in RTL. Experimentation on these systems provide information on system behavior due to parameter variation. Standard low power optimizations are performed for each implementation. All algorithm implementations, including motion estimation, were designed to occupy an Adaptive System on a Chip (aSoC).

5.1 Discrete Cosine Transform

The two dimensional discrete cosine transform (DCT) [10], is an integral part of many image and video compression systems. The DCT design discribed in[11] uses dynamic parameterization in the Row-Column Classification (RCC) power saving feature. RCC dynamically adjusts the number of arithmetic computations per calculation based on signal properties measured at an early stage in the pipeline. This adaptive technique shows a 35-40% power savings for a full custom implementation. A soft core DCT design [12] recently has been implemented to allow further design space exploration. Power for this design was determined with the Synopsys Power Estimator. Table 2 shows that a power benefit exists for RCC in soft core implementations.

Test Bench (Std. Dev.)	Power (mW)		Power Savings
(8x8 Pel Matrix)	NO RCC	With RCC	
Football Block587 (10.3)	743.430	633.532	14.78%
Football Block3 (61.9)	823.253	648.354	21.26%
Football Block1048 (97.8)	828.546	651.234	21.41%
Mobile Block496 (1052.1)	843.054	660.010	21.71%
Garden Block745 (2458.1)	843.736	660.675	21.69%
Tennis Block236 (2762.4)	826.153	655.954	20.60%
Mobile Block3 (7184.2)	801.535	654.584	18.33%
Football Block1297 (8602.2)	818.183	661.096	19.19%

Table 2 RCC power savings impact for a set of natural images

5.2 Lempel-Ziv Compression

Lempel-Ziv compression is a lossless compression technique that is used in a wide variety of communication and storage applications. The algorithm used to implement Lempel-Ziv compression represents a large class of computations, which rely on variable length matching sequences (e.g. bio-sequence matching, data mining). The parameters for the LZ algorithm can be set depending on input data statistics and system power and compression constraints.

The fine-grain parallelism of LZ compression has been exploited in a variety of recent systolic array and CAM implementations [13]. The LZ algorithm has two main parameters, which can be dynamically configured: 1) the longest matching length, and 2) the dictionary or sliding window length. Longest matching length can easily be tracked and used to modify the matching length parameter in the compression hardware. The size of the dictionary (sliding window length) can also be modified dynamically by tracking the LZ pointers to determine how frequently remote sections of the dictionary result in matches. Each of these parameters affect the compression ratio and speed. As a result, applications, which can withstand a varying compression ratio, could save power. Figure 14 shows how a small network is affected by load and compression ratio[14]. It is clear that the required compression ratio depends on the network load.

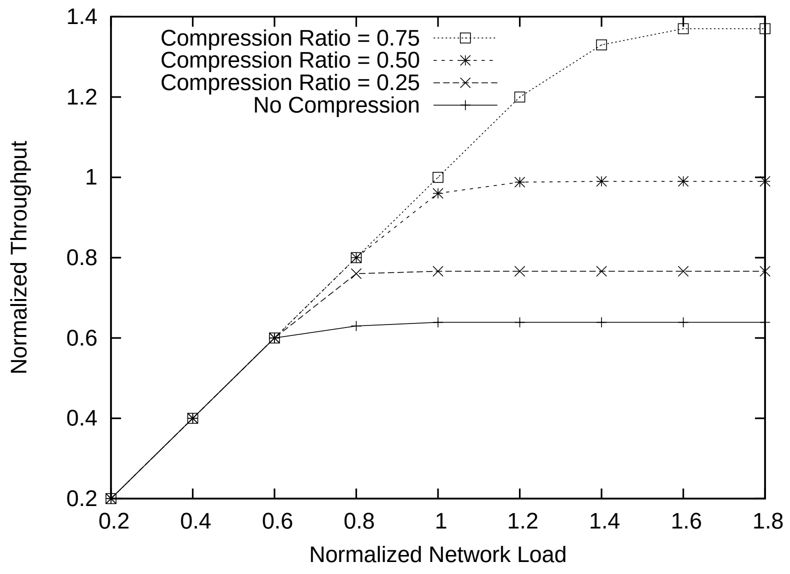

Figure 14 The effect of the mean compression ratio on a network of 10 nodes with probability of bit error = 1.0e-5

5.3 3D Graphics Light Rendering

Real-time 3D graphics will be a major contributor to power consumption in future portable embedded systems. Fortunately, we can exploit content variation and human visual perception to significantly reduce the power consumption of many aspects of 3D graphics rendering. In [15] we study the impact of novel adaptive Gouraud and Phong shading algorithms on power consumption. The adaptive algorithms exploit graphics content (e.g. motion, scene change) and human visual perception to achieve low power operation without noticeable quality degradation. Novel dynamically configurable architectures are proposed to efficiently implement the adaptive algorithms in power-aware systems with gracefully degradable quality.

There are two variable parameters considered in the 3D graphics architecture : Shading algorithms and Specular computation. Exploiting visual sensitivity to motion, Gouraud or Phong shading algorithm is selected, depending on the speed of an object and its distance from the camera. The same selection criteria is also used to select the type of specular computation to be used.

Figure 15 shows that power consumption between Phong and Gouraud shading varies by a factor of 20 for large triangles. In situations where human visual perception permits, using the lower quality shading algorithm (Gouraud) can save significant power. Results based on simulations using short but realistic rendering sequences indicate power savings up to 85%.

5.4 Adaptive Viterbi Decoding

Convolutional codes, which allow for efficient soft-decision decoding are widely employed in wireless communication systems. As convolutional codes become more powerful,

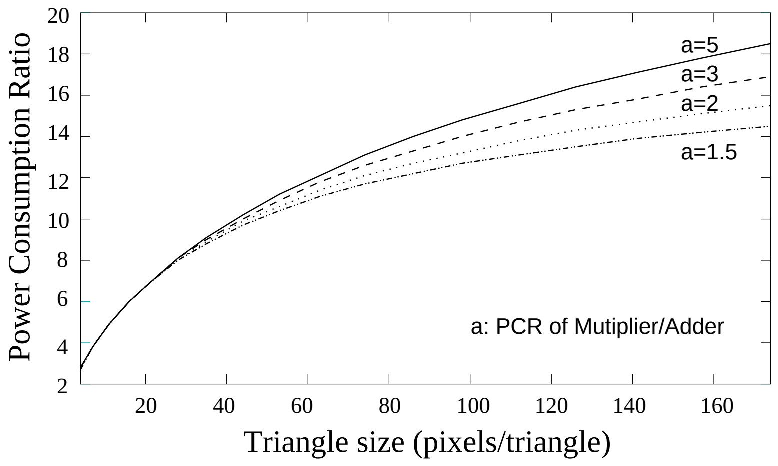

Figure 15 Power consumption ratio of phong shading and gouraud shading: one triangle shading

the complexity of the corresponding decoders generally increases. The Viterbi algorithm (VA) [16, 17], which is the most extensively employed decoding algorithm for convolutional codes, works well for codes with short constraint length K. For more powerful codes with large constraint lengths the Adaptive Viterbi algorithm (AVA) [18, 19] is used. It reduces the average number of computations per decoded information bit. Our work looks at using AVA to achieve reduced power consumption.

There are two dynamic parameters used in the architecture built for this work : constraint length and truncation length. The constraint length indicates the number of times each input bit has an effect on producing output bits [20]. A trellis diagram is used to determine the most likely transmitted data bits. The number of time steps used to identify the most likely transmitted symbol sequence is called the truncation length.

These parameters vary depending on the noise levels in a channel and the bit error rate (BER) requirement of the system. Table 3 shows a comparison of constraint lengths for various ranges of channel SNR. For a large amount of channel noise, the constraint length must be large to achieve a low BER, but under low noise conditions, it can be kept small. Table 3 demonstrates the speed advantages of this tradeoff, but power savings could potentially also be achieved for constant decode rates.

K	FPGA decode (Kbps)	Decode rate w/PCI overhead (Kbps)	Max. FPGA clock (MHz)	SNR range (dB)
4	333.7	186.0	40.5	6.3-6.5
5	164.2	117.7	20.1	6.1-6.3
6	162.3	116.3	19.9	5.5-6.1
7	160.8	114.2	19.7	3.9-5.5
8	143.6	109.4	17.6	3.7-3.9
9	141.1	107.8	17.3	3.1-3.7
10	101.5	NA	25.5	3.0-3.1
12	94.8	NA	24.7	2.8-3.0
14	82.3	NA	23.0	2.5-2.8

Table 3 Decode rate versus K for XC4036XL-08 (K = 4 to 9) and XCV1000-04[21] (K = 10 to 14)

6 Conclusions

In this paper, we have described algorithmic and architectural aspects of a low power multimedia project. Dynamic parameterization of compute intensive applications has been proposed to improve performance and output quality. Motion estimation was used as a demonstration application to illustrate the salient features of the proposed approach. Using a similar design methodology, other applications were shown to be good candidates for dynamic parameterization. A significant contribution of this paper is the outline of a design and experimentation methodology for dynamic parameterization that can adapt computing systems to varying computational environments.

The need for dynamic reconfiguration arises from non-uniform system input signal variation. This issue is especially important for multimedia. Table 4 presents a summary of varied computational parameters and corresponding observed tradeoffs. The tradeoffs provide valuable information that can be used to develop a hardware controller block. The controller block monitors input and output signal trends to subsequently vary the dynamically configurable parameters. This approach satisfies the system resource constraints in terms of power, speed and area. A preliminary prediction scheme using output motion vectors to predict future vectors was developed for motion estimation.

Future Work

Power estimates for the motion estimation unit were derived from a very small subset of the input sample space. To improve the confidence in these numbers, either more data must be collected using the RTL level modeling or a method for approximating the power for a high level model must be found. In addition, compression and SAD may not be inversely proportional in all cases. Our system must be run to determine the actual compression and, possibly, a measure of quality. The controller for the motion estimation system is simplistic and a higher level of sophistication is required in the design of this block. Currently, we are investigating a highly reduced search window size, using a directed search and our current set of predictive motion vectors.

Computation	Parameters	Range	Tradeoffs			
			Performance		Cost	
			Latency	Compression or Quality	Area	Power
Motion Estimation	Algorithms	Full Search	Degrades	Improves	-	Degrades
		TSS	Improves	Degrades	-	Improves
		Spiral	Good for Small Searches	SAD Threshold Dependent	-	Good for Small Searches
	Search Window Size	1×1 Pixels to Complete Frame	Degrades	Improves	-	Degrades
	Pel Subsampling	Psub 2:1	Degrades	Slight Degradation	-	Improves
		Psub 4:1	Degrades	Slight Degradation	-	Improves
	Bit-Width	1-8 Bits	-	Improves	-	Degrades
	SAD Threshold	0 to 65280	Improves	Degrades	-	Improves
DCT	MSBR	Bit Variation	Degrades	Improves	-	Degrades
	RCC	1 to 4 Clk Cycles	Degrades	Improves	-	Degrades
	RAC	1 or 2 Units	Improves	-	Degrades	Improves Vdd scaling
Lempel-Ziv	Matching Length	1-2048	Degrades	Improves	Degrades	Degrades
	Sliding Window Length	256, 512, 1024, 2048	-	Improves	Degrades	Degrades
3D Graphics	Shading Algorithms	Phong	Degrades	Improves	Degrades	Improves
		Gouraud	Improves	Degrades	Improves	Improves
	Specular Computation	Exponential	Degrades	Improves	-	Degrades
		Iterative Multiplication	Improves	Degrades	-	Improves
Adaptive Viterbi	Constraint Length	3 to 14	Degrades	Improves	Degrades	Degrades
	Truncation Length	9, 18, 27, 36, 45	Degrades	Improves	Degrades	Degrades

Table 4 Dynamically reconfigurable parameters and tradeoffs

Acknowledgements

We would like to acknowledge the following people for their contribution and support throughout this project.

J. Euh

S. Swaminathan

S. Venkatraman

V. Thyagarajan

M. Sinha

This work has been partially supported by NSF 9988238.

REFERENCES

[1] J. Rabaey,"Reconfigurable Processing: The Solution to Low-Power Programmable DSP", in *Proceedings, International Conference on Acoustics, Speech and Signal Processing*, Munich, Germany, Apr. 1997, pp. 275–278.

[2] H. Zhang, V. Prabhu, V. George, M. Wan, M. Benes, A. Abnous, and J. Rabaey, "A 1V Heterogeneous Reconfigurable Processor IC for Baseband Wireless Applications", in *Proceedings, International Solid State Circuit Conference*, San Francisco CA, Feb. 2000, pp. 68-69.

[3] P. Kuhn, *Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation,* Netherlands: Kluwer Academic Publications, 1999.

[4] J. Liang, S. Swaminathan, and R. Tessier, "aSOC: A Scalable, Single-Chip Communications Architecture", in *Proceedings, IEEE International Conference on Parallel Architectures and Compilation Techniques*, Philadelphia PA, Oct. 2000.

[5] A. Laffely, J. Liang, P. Jain, N. Weng, W. Burleson, and R. Tessier, "Adaptive System on a Chip (aSoC) for Low-Power Signal Processing", in *Proceedings, Thirty-Fifth Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove CA, Nov. 2001.

[6] P. Pirsch, N. Demassieux and W. Gehrke, "VLSI Architectures for Video Compression - A Survey", *Proceedings of the IEEE,* Vol 83, No.2, Feb. 1995, pp. 220-248.

[7] M. Bhardwaj, R. Min and A. Chandrakasan, "Quantifying and Enhancing Power-Awareness of VLSI Systems", *IEEE Transactions on VLSI Systems*, 2001.

[8] T. Nishikawa, M. Takahashi, M. Hamada, T. Takayanagi, H. Arakida, N. Machida, H. Yamamoto, T. Fujiyoshi, Y. Matsumoto, O. Yamagishi, T. Samata, A. Asano, T. Terazawa, K. Ohmori, J. Shirakura, Y. Watanabe, H. Nakamura, S. Minami, T. Kuroda, and T. Furuyama "A 60MHz 240mW MPEG-4 Video-Phone LSI with 16Mb Embedded DRAMToshiba", in *Proceedings, International Solid State Circuit Conference*, San Francisco CA, Feb. 2000.

[9] T. Hashimoto, S. Kuromaru, M. Matsuo, H. Nakajima, Y. Kohashi, K. Ishida, T. Mori-iwa, M. Ohashi, K. Hashimoto, T. Yonezawa, M. Hamada, T. Nakamura, M. Toujima, Y. Sugisawa, T. Kondo, H. Otsuki, M. Arita, H. Fujimoto, H. Toida, and H. Ito, "A 90mW MPEG4 Video Codec LSI with the Capability for Core Profile", in *Proceedings, International Solid State Circuit Conference*, San Francisco CA, Feb. 2001.

[10] V. Bhaskaran and K. Konstantinides, *Image and Video Compression Standards - Algorithms and Architectures*, Norwell: Kluwer Academic Publishers, Second Edition, 1997.

[11] T. Xanthopoulos, and A. P. Chandrakasan, "A Low-Power DCT Core Using Adaptive Bitwidth and Arithmetic Activity Exploiting Signal Correlations and Quantization", *IEEE Journal of Solid-State Circuits*, Vol 35, No.5, May 2000.

[12] S. Venkatraman “A Power-Aware Synthesizable Core for the Descrete Cosine Transfrom”, *Thesis, M.S. E.C.E.*, University of Massachusetts Amherst, Sep. 2001.

[13] B. Jung and W. Burleson, “VLSI Algorithm, Architecture and Implementation for High-Speed Lempel-Ziv Data Compression”, *IEEE Transactions on VLSI Systems*, Sep. 1998.

[14] B. Jung, “VLSI Arrays for Source Coding in Wireless Local Area Networks”, *Dissertation, Ph.D. E.C.E*, University of Massachusetts Amherst, Feb 1997.

[15] J. Euh and W. Burleson, “Exploiting Content Variation and Perception in Power-Aware 3D Graphics Rendering”, in *Proceedings, Power-Aware Computing Symposium*, Cambridge MA, 2000.

[16] A. Viterbi, “Error bound for convolutional codes and an asymptotically optimum decoding algorithm,” *IEEE Transactions on Information Theory,* Vol. 13, Apr. 1967, pp. 260-269.

[17] J. Omura, “On the Viterbi decoding algorithm,” *IEEE Transactions on Information Theory,* Vol. 15, Apr. 1969, pp. 177-179.

[18] F. Chan and D. Haccoun, “Adaptive Viterbi decoding of convolutional codes over memoryless channels,” *IEEE Transactions on Communications*, Vol. 45, Nov. 1997, pp. 1389-1400.

[19] S. Simmons, “Breadth-first trellis decoding with adaptive effort,” *IEEE Transactions on Communications*, Vol. 38, Cambridge, MA, 2000.

[20] S. Swaminathan, R. Tessier, D. Goeckel, and W. Burleson, “A Dynamically Reconfigurable Adaptive Viterbi Decoder”, in *Proceedings, 10th International ACM/SIGDA Symposium on Field Programmable Gate Arrays*, Monterey, California, Feb. 2002.

[21] Xilinx Corporation. Virtex II data sheet, 2001. *http://www.xilinx.com*