1-1-2008

Scalable FPGA Hardware Acceleration for H.264 Motion Estimation

Theepan Moorthy
Ryerson University

SCALABLE FPGA HARDWARE ACCELERATION FOR H.264 MOTION ESTIMATION

by

Theepan Moorthy

B. Sc. in Computer Engineering, Queen's University, Kingston, ON, 2004

A thesis
presented to Ryerson University
in partial fulfillment of the
requirements for the degree of
Master of Applied Science (MASc)
in the Program of
Electrical and Computer Engineering

Toronto, Ontario, Canada, 2008

Author's Declaration

I hereby declare that I am the sole author of this thesis.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Signature

_____

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Signature

_____

Abstract

Scalable FPGA Hardware Acceleration

for H.264 Motion Estimation

Master of Applied Science (MASc)

Department of Electrical and Computer Engineering

Ryerson University

The H.264 video compression standard uses enhanced Motion Estimation (ME) features to improve both the compression ratio and the quality of compressed video. The two primary enhancements are the use of Variable Block Size Motion Estimation (VBSME) and multiple reference frames. These two additions greatly increase the computational complexity of the ME algorithm, to the point where a software based real-time (30 frames per second (fps)) implementation is not possible on present microprocessors. Thus hardware acceleration of the H.264 ME algorithm is necessary in order to achieve real-time performance for the implementation of the VBSME and multiple reference frames features. This thesis presents a scalable FPGA-based ME architecture that supports real-time H.264 ME for a wide range of video resolutions ─ from 640x480 VGA to 1920x1088 High Definition (HD). The architecture contains innovations in both the data-path design and memory organization to achieve scalability and real-time performance on FPGAs. At 37% FPGA device utilization, the architecture is able to achieve 31 fps performance for encoding full 1920x1088 progressive HDTV video.

Acknowledgments

I would like to thank Professor A. Anpalagan for initially having accepted me into the Master of Engineering program, and thus having granted me a wonderful opportunity to further my education. His support and guidance was invaluable to me at a time when I had needed it the most and will never be easily forgotten.

In every academic sense it was to my great fortune that Professor Andy Gean Ye had accepted to supervise the work within this thesis. His genuine love of FPGA research and development, combined with his good natured personality towards teaching others, has made the path to this thesis a very rewarding and enjoyable process. From him I have learned to not only formulate my thoughts in a more constructive manner but more importantly I have learned how to express them in a manner that is more easily accessible to others, for this I will always be grateful. My first rewarding steps as a nascent researcher have been deeply enriched by the privilege of having had Professor Ye as my supervisor.

Lastly I would like to thank Dr. S. Sudharsanan for having introduced me, amongst numerous other beneficial acts of kindness, to the wonderful world of hardware based video processing. Your support and encouragement of my graduate study dreams has not been forgotten.

சமர்ப்பணம்

மொழி, கலாச்சாரம், பண்பாடு, பொருளாதாரம் என பற்பல தடங்கல்களையும் கடந்து; என் தங்கையினதும், எனதும் கல்வி ஒன்றையேதான், **தன் வாழ்வின்** இலக்காக எண்ணி, எம்மை வழி நடத்திய **என் தாயாருக்கு** எனது முதுகலை ஆராய்ச்சி ஆவணத்தை சமர்ப்பிக்கின்றேன்.

Dedication

To my mother who has struggled, achieved, and prayed restlessly for the educational betterment of my sister and I. Her commitment and dedication shown through the many sacrifices made and obstacles overcome to fulfill for her children the rewards of an education, denied to her by the circumstances of her own youth, is deeply appreciated and inspiring. In the eyes of our mother – we saw the presence of Sarasvathi within our hearts.

Contents

List of Figures & Tables

Chapter 1

Introduction

Motion Estimation (ME) is the process of creating motion vectors to track the motion of objects within video footage. It is an essential part of many compression standards and is a crucial component of the H.264 video compression standard [1, 2]. In particular, ME can consist of over 40% of the total computation [3].

Due to this high computing demand, many hardware architectures have been proposed to accelerate the computation of motion vectors for H.264 [4]–[11]. Most of the architectures, however, have been implemented in Application Specific Integrated Circuit (ASIC) technology. Except for limited commercial implementations [12]–[14], little information exists on how these algorithms would perform on reconfigurable technologies such as Field-Programmable Gate Arrays (FPGAs). In particular, the FPGA implementation presented in [15] specifically targets portable multimedia devices with CIF-level resolution and cannot be easily scaled. The FPGA implementation presented in [16], on the other hand, only reaches VGA-level resolution and 27 fps performance. It too cannot be scaled. In this work, we propose a scalable hardware architecture based on the Propagate Partial SAD architecture [11] and measure its performance on FPGAs as the design scales.

1 Introduction

The use of FPGAs encourages design reuse and can greatly enhance the upgradability of digital systems. The programmability of FPGAs is particularly useful for highly flexible encoding systems that can accommodate a multitude of existing standards as well as the emergence of new standards. In particular, the scalable hardware architecture designed in this thesis can be incorporated into a single FPGA solution for targeting both low-resolution applications and high performance high-resolution applications as well.

The proposed architecture is based on one of the three widely used architectures for motion estimation — the Propagate Partial SAD [4] [11], SAD Tree [10], and the Parallel Sub-Tree [9]. The Propagate Partial SAD architecture was selected due to its unique blend of efficiency and scalability. While the SAD Tree architecture has the highest performance amongst the three [10], it requires the support of a complex array of shifting registers that must have the capability of shifting in both horizontal and vertical directions. This array, while efficient to implement in ASICs, consumes a large amount of FPGA resources. The Parallel Sub-Tree architecture, on the other hand, is the most compact design amongst the three. The architecture, however, inherently does not scale well for high performance applications [9].

As proposed in [4] and [11], the Propagate Partial SAD architecture processes a single group of 16 reference blocks at a time. Our design enhances the original design by allowing it to be scaled to process several groups of 16 reference blocks simultaneously. These groups intelligently share a large amount of their reference pixels. This sharing minimizes the increase in memory bandwidth as the design scales and makes high performance FPGA-based design feasible in our work.

1 Introduction

The following chapters will cover our design work in more detail. Chapter 2 provides the necessary background on digital video compression to understand the role that motion estimation plays in video compression. Chapter 3 explains motion estimation in detail, and covers some of the estimation features that are specific to the H.264 compression standard. Chapter 4 discusses our experimental setup, including the specifics of our FPGA device and our chosen base architecture. Chapter 5 moves on to explain the design of the Pixel Processing Unit (PPU), which is the basic building block of scalability in our work. Chapter 6 then covers the specific scalability challenges that were overcome to achieve scalability within practical resource limits. Chapter 7 discusses the aspects of memory design that are required to successfully support such a scalable system. Chapters 8 and 9 then conclude the work by providing the performance and hardware cost results for the design, along with the conclusions and future work that can be drawn from the results.

Chapter 2

Background & Motivation

In this chapter, we examine the video compression process. We first review how video is captured in analog form and converted into digital formats. We then examine the various digital formats and standard resolutions available for video. Finally several relevant video compression standards and compression techniques are discussed.

2.1 Analog Video

Raw digital video in its uncompressed form requires a considerable amount of storage. This storage requirement is a direct consequence of converting analog signals into digital formats, while preserving the greatest level of fidelity. All colour video cameras break down light into three primary colours of red, green, and blue (the RGB colour scheme). The choice of these three specific colours is based on the tri-chromatic theory of colour vision [3], which states that the human eye is biologically built with three categories of red, green, and blue photoreceptive cone cells. Hence the use of the RGB colour scheme reduces the amount of distortions that are detectable by a human viewer and enhances fidelity.

Capturing light in RGB is ideal for true fidelity. However, reproducing light in RGB was impractical in traditional television technologies. Thus in different parts of the world different colour coding systems were developed to convert the high-fidelity RGB signals into more easily reproducible signals based on 1 luminance (black & white component) and 2 chrominance (colour components of hue and saturation) values. Out of the various colour-coding systems, the three that are the most widely used include: the *Phase Alternation Line* (PAL) system, the *SEquential Couleur Avec Memoire* (SECAM) system, and the *National Television System Committee* (NTSC) system. The difference in these three systems lies within their own unique methods of transforming and manipulating the RGB signals to derive the luminance and chrominance values. Regardless of which colour coding system is used, converting from RGB to luminance and chrominance values does introduce a slight loss in fidelity. Thus it is important to be aware that the original RGB colour scheme is the preferred video input method in modern video systems.

2.2 Digital Video

Nonetheless, the luminance & chrominance format was selected as the format for the analog to digital conversion. This is due to the fact that luminance & chrominance representation is more convenient for digital video compression. This convenience revolves around the human visual system being more responsive to luminance (brightness) spatial details than to chrominance (colour) spatial details [3]. What this means is that when humans take in the view of an image as a whole, our eyes are more sensitive to changes in brightness over minute areas than to colour. This aspect of human visual perception can be exploited to sub-sample the two analog chromi-

nance signals when performing analog to digital video conversion. Thus the raw number of bytes required to capture the analog video can be reduced in the analog to digital conversion process itself.

Many digital formats exist ─ varying in their target resolutions, chrominance sub-sampling ratios, and analog signal sampling frequencies. The video resolutions used in this work are 640x480 Video Graphics Array (VGA), 800x608 Super VGA (SVGA), 1024x768 Extreme VGA (XVGA), and 1920x1088 High Definition (HD). The chrominance sub-sampling ratio that is most often used for these resolutions in high quality applications is 4:1:1 ─ for every four luminance samples only two independent chrominance values are used. All luminance and chrominance values are sampled at 8-bit quantization levels, thus each sample is represented by a single byte of data [17]. To process this data in real-time, encoders/decoders need to operate at the rate that the digital video is captured. Consequently a video encoder needs to operate at a frame rate of at least 30 fps in order to support both the real-time sampling frequencies of 24 fps for motion picture and PAL video and 30 fps for NTSC video.

The storage requirement of a video standard depends on its frame resolutions, the chrominance sub-sampling, the quantization level, and its frame rate. Uncompressed HD video, for example, requires 2,080,000 bytes of data to store just the luminance values in a single frame. At the 4:1:1 chrominance sub-sampling ratio 0.52 MB is required per chrominance values per frame. Thus the total amount of data per frame is 3.12 MB. Capturing the video at a real-time rate of 30 fps would require 93.6 MB of storage for every second of video.

Note that 93.6 MB/s is a staggering amount of data since a dual-layer double-sided Blu-ray disk (with a storage capacity of 100 Giga Bytes per disc) would be able to hold only 18 minutes worth of uncompressed HD video. The problem exacerbates as HD resolutions are increased in the future. Thus video compression is an essential component of digital video technology. Video compression has become even more important in today's networked environments, where videos are transmitted either through wires or wirelessly in real-time through networks often with extremely limited bandwidth.

2.3 Digital Video Compression

Video compression is achieved on two separate fronts by eliminating spatial redundancies and temporal redundancies from video signals. Removing spatial redundancies involves the task of removing video information that is consistently repeated within certain areas of a single frame. For example a frame shot of a blue sky will have a consistent shade of blue across the entire frame. This information can be compressed through the use of various discreet cosine transformations [2] that map a given image in terms of its light or colour intensities. This paves the way for spatial compression by only capturing the distinct intensities, instead of the spread of intensities over the entire frame. Since compression through removing spatial redundancies does not involve the use of motion estimation, this topic is not examined further. Interested readers can refer to [2].

Compression through the removal of temporal redundancies involves compressing information that is repeated over a given sequence of frames. For example the objects in the background

of a news anchor being filmed are not likely to change over the course of the footage. This redundancy can be taken advantage of to reduce the storage space required for the footage. When the background does happen to move, recording only the motion of objects over consecutive frames in the form of motion vectors can still achieve significant amounts of compression. Consequently, the motion estimation process is the process of deriving a suitable Motion Vector (MV) that best describes the spatial movement of objects from one frame to the next. Motion estimation is discussed in detail in the next chapter.

The spatial and temporal compression techniques discussed above have been widely implemented in former compression standards such as MPEG 2 (developed by the Motion Picture Experts Group committee). At present, the same two fundamental techniques have been enhanced and optimized to form the new standard used in H.264 video compression. The H.264 standard was jointly formed by the International Telecommunications Union – Telecommunications Standardization Sector (ITU – T) Video Coding Experts Group (VCEG) and the International Organization for Standardization (ISO) MPEG committee. The standard was formally finalized in March 2003. Where as MPEG 2 had compression ratios of between 20:1 and 30:1, the new H.264 standard can achieve compression ratios as high as 50:1 and 60:1 and achieve better video quality [2].

Among many other new features and enhancements, the most notable features of the H.264 standard for this work are its ability to achieve a finer granularity of motion estimation and its ability to capture periodic motion. These topics form the heart of the next sections, where they will be explained and discussed in detail.

2.4 Motion Estimation

The following sections discuss Motion Estimation (ME) in detail. We first examine the relationship between motion estimation and temporal video compression. We then define the Block Matching Motion Estimation (BME) algorithm used in this thesis. The concepts of search windows and Sum of Absolute Differences (SAD) values are also defined. Finally two H.264 specific enhancements to ME – Variable Block Size ME (VBSME) and Multiple Reference Frames (MRF) – are fully discussed.

Motion compensation is a key process in temporal video compression, which eliminates temporal redundancies found in video. Temporal redundancy occurs when an identical set of pixels exist across multiple video frames. This is often caused when an object appears in a set of video footages, which might correspond to several thousands of frames. From frame to frame the object might change its position due to motion resulting from camera pans or zooming (global motion), the active motion of the object itself (translational motion), or a combination of both global and translational motion.

Motion estimation is the technique of finding a suitable Motion Vector (MV) that best describes the movement of a set of pixels from its original position within one frame to its new positions in the subsequent frame. Encoding just the motion vector for the set of pixels requires significantly less bits than what is required to encode the entire set of pixels, while still retaining enough information to reproduce the original video sequence.

2.5 Block Matching Motion Estimation

Several different algorithms derived from various theories, including object-oriented tracking, exist to perform motion estimation [3]. Among them, one of the most popular algorithms is the Block Matching Motion Estimation (BME) algorithm. BME treats a frame as being composed of many individual sub-frame blocks, known as macroBlocks. Motion vectors are then used to encode the motion of the macroBlocks through frames of video via a frame by frame matching process.

When a frame is brought into the encoder for compression, it is referred to as the current frame. It is the goal of the BME unit to describe the motion of the macroBlocks within the current frame relative to a set of reference frames. The reference frames may be previous or future frames relative to the current frame. Each reference frame is also divided into a set of sub-frame blocks, which are equal to the size of the macroBlocks. These blocks are referred to as referenceBlocks. The BME algorithm will scan several candidate referenceBlocks within a reference frame to find the best match to a macroBlock. Once the best referenceBlock is found a motion vector is then calculated to record the spatial displacement of the macroBlock relative to the matching referenceBlock, as shown in Figure 2.5.

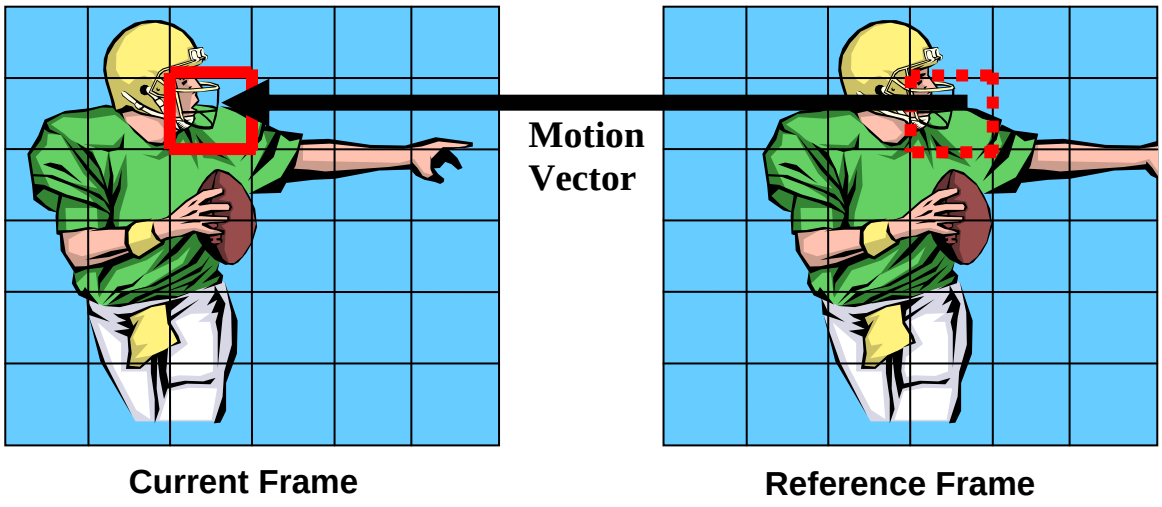


Figure 2.5 Block Matching Between Current & Reference Frames

Note that the BME algorithm can introduce a lot of errors and distortions, which need to be handled by other encoding processes external to the BME unit. In particular, BME assumes that all pixels in a macroBlock are moving with one uniform velocity. There, however, can be many instances within a video frame where this assumption simply is not true. Consider a macroBlock that captures the edge of a baseball bat coming into contact with the edge of a baseball. After the contact, the ball and the bat will have very different velocities. Thus no single motion vector for that macroBlock will accurately capture the two different velocities of the ball and bat simultaneously. Nevertheless, a BME unit will still provide a motion vector for that macroBlock that describes the best possible position match. It is the role of the encoder to determine if this "estimated" motion vector should be accepted or not. The encoder might also fill in the necessary corrections as appropriate.

Reducing the macroBlock sizes will increase the likelihood that all pixels within the macroBlock have a uniform velocity. Since a smaller macroBlock is less likely to contain several ob-

jects that move in different directions and speeds. But such an increase in accuracy comes at the cost of an increase in the number of macroBlocks per frame. Thereby increasing the total number of motion vectors that are required per frame. In turn this increases the amount of required computations. Through much analysis and experimentation a 16 pixel x 16 pixel 256-pixels-square macroBlock size has become the standard since it offers the best compromise between accuracy and computational complexity [18].

The H.264 standard utilizes an adaptive macroBlock size switching technique, which employs macroBlocks smaller than the standard 16x16 size, a when finer granularity of motion is detected within a video sequence. This enhancement is described in detail in section 3.5, where ME enhancements specific to the H.264 standard are discussed.

2.6 Search Windows

When searching a reference frame for possible macroBlock matches, the entire reference frame is not searched. Instead the search is restricted within a search window. Search windows in most H.264 implementations have a size of 48-pixel (rows) x 63-pixel (columns). In this thesis, we use the same 48x63 search window size. This window consists of a vertical search range of [-16, +16] and a horizontal search range of [-24, +23] pixels as illustrated in Figure 2.6.

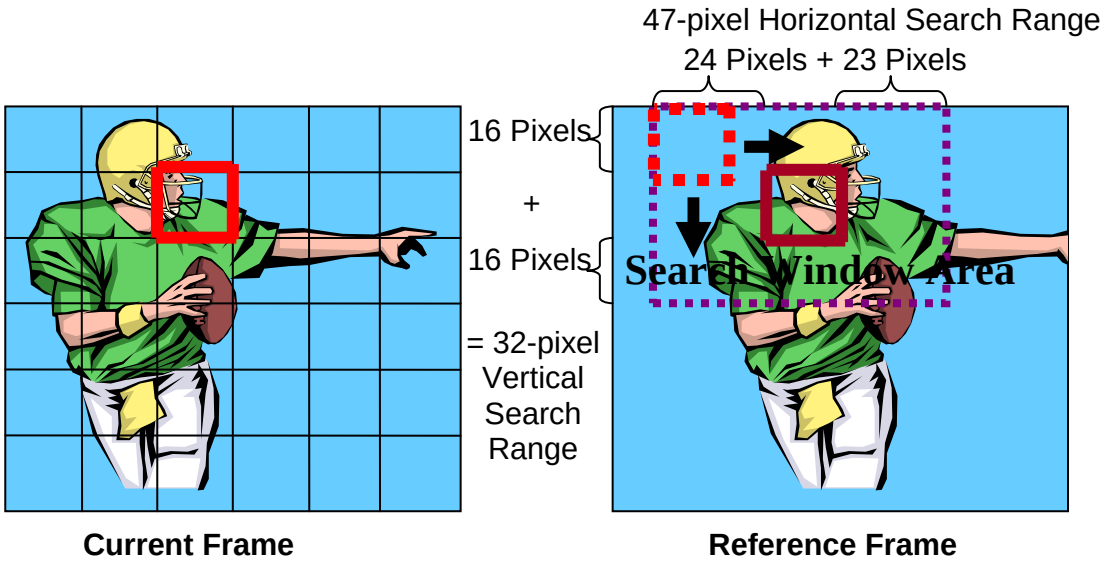


Figure 2.6 Search Window Size Definition

In the figure, the dashed large rectangle in the reference frame represents the 48x63 search window area. The dashed square in the top left corner of the search window represents the first of the 1584 possible candidate 16x16 referenceBlocks. Each subsequent referenceBlock is offset by either one pixel row or one pixel column from its predecessor while the entire search window area is covered by the overlapping candidate referenceBlocks. Note that the original 16x16 macroBlock is positioned at the centre of the search window. In order to compare it to every candidate referenceBlock within the search window, the macroBlock has a maximum displacement of 24 pixels to the left, 23 pixels to the right, 16 pixels up, and 16 pixels down from its original position – resulting in a horizontal search range of [-24, +23] and a vertical search range of [-16, +16].

Note that the horizontal search range is greater than the vertical search range. This is due to the fact that in real life video coverage greater velocities of motion often exist in the horizontal

plane than in the vertical plane. For example in sports coverage, greater motion more often exists horizontally along the playing field than it does vertically to the field. Video coverage of events with extremely fast motion may call for even wider search windows since the BME algorithm cannot detect objects that move outside of the search window boundaries. Encoders specifically designed to handle extremely fast-motion video typically use larger search windows and must also handle the increase in computational complexity.

Within a search window of 48x63 pixels there will be a total of 1584 (33x48) possible candidate referenceBlock matches to be made per macroBlock. An exhaustive motion estimation algorithm examines all 1584 matches for every macroBlock. Such an exhaustive search results in the best video quality and compression ratios [18]. Consequently an exhaustive search BME algorithm is used in this work.

Note that the referenceBlocks in Figure 2.6 are offset from each other by either one row or one column of pixels. This is commonly referred to as Integer Motion Estimation (IME). However, there exists another branch of motion estimation that deals with referenceBlocks offset from each other by displacements less than one pixel of movement. Called fractional or sub-pel motion estimation, it is most often performed by a second independent Fractional Motion Estimation (FME) unit. The FME unit is external to the IME unit and uses the output motion vectors of the IME to further refine the best matched referenceBlock to sub-pel accuracy levels within the search window. The design of FME units is beyond the scope of this thesis and is not discussed further.

2.7 SAD Matching Function

Given that a digital video signal is composed of a single luminance component and two chrominance (colour) components, motion estimation units utilize only the luminance data. This design choice stems from the fact that a lack of colour does not adversely affect one's ability to discern motion. The luminance components (and the chrominance components as well) are quantized to 8-bit data values. Each 8-bit luminance value represents a single byte of an unsigned integer value ranging from 0 to 255, where 0 corresponds to zero luminance (purely dark/black) and 255 to maximum luminance (purely bright/white).

The mathematical matching function employed by the BME algorithm considered in this thesis is widely known as the Sum of Absolute Differences (SAD) function. This function is used to assign a numerical SAD value to every single macroBlock-referenceBlock pair and is defined as follows:

$$SAD = \sum_{k=1}^{k=256} \left| Pk - Rk \right| \qquad (1)$$

In equation (1) Pk and Rk refer to the luminance of pixels within a macroBlock and a candidate referenceBlock, respectively. Explicitly stated, the SAD function, calculates the absolute luminance differences between all pixels in a macroBlock and their corresponding pixels in a referenceBlock. The function then accumulates these absolute difference values over all 256 macroBlock-pixel to referenceBlock-pixel pairs. Consequently to obtain a SAD value for a 16x16 macroBlock to referenceBlock comparison requires a total of 256 subtractions & additions.

A lower SAD value between two macroBlocks indicates a better match. A perfect match will have a SAD value of zero since all of the pixel luminance values in the macroBlock will be identical to the pixel luminance values in the referenceBlock. The worst match compares a completely dark (black) macroBlock to a completely bright (white) referenceBlock. In this case, all of the pixel luminance values in the macroBlock will be equal to 0 and all of the pixel luminance values in the referenceBlock will be equal to 255. Consequently, the absolute difference between each pair of macroBlock and referenceBlock pixels will be 255. This difference is accumulated 256 times across all pixel pairs and results in a maximum SAD value of 65,280.

A noteworthy limitation to the SAD match criterion lies in the fact that by design it estimates purely 2 dimensional motion only. In other words, motion vectors are only created for objects (macroBlocks) that have 2D movement between frames, and in fact any 3D movement that may exist in a video will introduce severe errors in SAD-based motion estimation. Consider video footage that contains the sphere of a model globe spinning on its axis, a macroBlock that captures the entire globe will have no 2D spatial movement from frame to frame. But due to the rotation of the globe, pixel values within the macroBlock will be entirely different in the next frame. To the ME unit (through SAD measurements), the macroBlock has disappeared in the next frame.

Smaller macroBlocks will not improve motion estimation for the globe. Real life 3D motion introduces complex variations in light illumination and reflection. The mathematical SAD criterion used by BME to calculate a match, however, does not account for such 3D based pixel variations. Thus 3D motion is essentially ignored and treated as miss-matched SAD values by

the encoder. There are other mathematical match criteria that use various averaging schemes to better quantify matches, but they are not widely used in industry [18]. Consequently only the SAD criterion is investigated in this thesis.

2.8 H.264 – VBSME

H.264 introduces two new features to ME – the Variable Block Size Motion Estimation (VBSME) and the Multiple Reference Frame (MRF) searching. VBSME has slightly higher computational complexity than traditional ME, but significantly increases the output bandwidth of the ME unit. MRF searching, on the other hand, does not change the output bandwidth of an ME unit but does significantly increase the amount of required calculations. These points are examined in turn.

In previous video compression standards, such as MPEG 2, a fixed 16x16 macroBlock size was used. The H.264 standard introduces VBSME to provide the encoder with the capability of adapting to smaller macroBlocks during encoding, as shown in Figure 2.8.1.

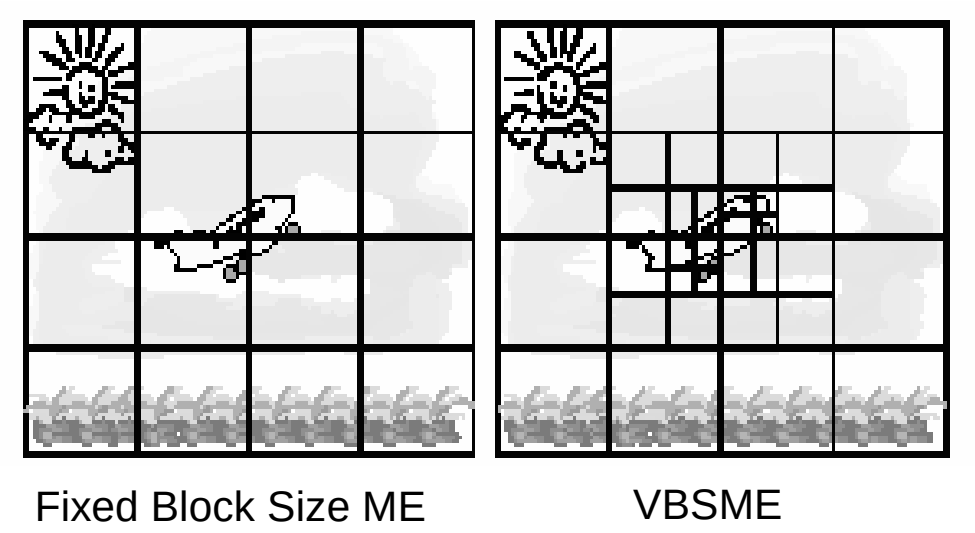


Figure 2.8.1 Comparisons between FBSME and VBSME [19]

As shown, the fixed 16x16 macroBlocks sizes are well suited for large areas of consistent motion, such as the clouds moving through the sky. However, when more fine grain motion occurs, such as the plane's landing gear retracting into the plane during takeoff, smaller macroBlocks will more accurately capture the minute details of the motion. Therefore the H.264 standard requires the ME unit to provide not only the SAD value for the large 16x16 macroBlock, but also 40 other SAD values for 40 subBlocks of the 16x16 block, as shown in Figure 2.8.2.

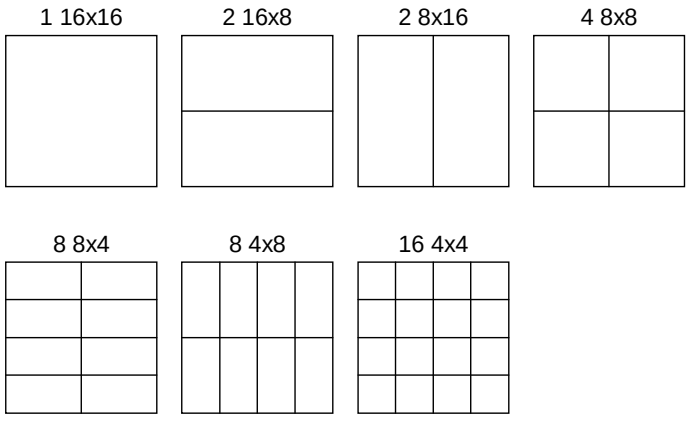


Figure 2.8.2 The 1 macroBlock and 40 subBlocks in VBSME

When all subBlocks are in uniform motion, all subBlock motion vectors will be the same as the motion vector for the entire macroBlock. However, if fine grain motion exists and subBlocks are moving in different directions, subBlock motion vectors can differ significantly from each other and from the motion vector of the macroBlock. Consequently the ME unit must be able to generate a separate motion vector for each of the subBlocks.

In addition to using the motion vectors to perform FME on subBlocks, the encoder also has to decide on whether or not the minimum SAD values being reported are indeed acceptable motion matches. To do this, the encoder has a set of user-set SAD threshold values (one for each of the 41 blocks), which it uses to determine whether the "match" being reported by the ME unit should be rejected or used. As an example, consider a video sequence that fades out to black in one scene and then opens up in the next scene with a frame of blinding white light. If the completely white frame is passed to the ME unit as the current frame and the completely black frame is used as the reference frame, all matches will be equally bad with a maximum SAD value of 65,280. The threshold values allow the encoder to discard all the SAD values and directly encode the current frame instead of using the motion vectors. Consequently for VBSME, the ME unit must deliver 41 minimum SAD values along with the positions at which these SADs are found for every macroBlock. These 41 SAD values and motion vectors are typically delivered in parallel to ease the design of the subsequent units that consume these values.

Although having the 41 separate SAD values and motion vectors increases the output bandwidth of the ME unit, the extra calculations required to produce the 40 extra SADs for the subBlocks are relatively insignificant. In particular as it will be shown in Chapter 4, 256 addi-

tions and subtractions will still be needed to generate the 16 SADs for the 16 4x4 subBlocks. After this point the individual SADs of the 16 subBlocks can be added in various combinations to form the SADs of the larger subBlocks and the macroBlock. In this manner only 25 extra accumulations are required to generate the remaining 25 SADs from the 16 smallest 4x4 subBlocks' SADs [11].

2.9 H.264 – Multiple Reference Frames

One of the most significant enhancements to ME in H.264 is the support of motion estimation across Multiple Reference Frames (MRF). This means that when searching for the potential match for a macroBlock, the macroBlock is compared against one search window in each of the multiple reference frames. Therefore using two reference frames doubles the total number of operations required to find a match. Similarly using 4 reference frames quadruples the amount of operations.

Most industry applications demand support for the use of at least 4 reference frames even for real-time ME, although the H.264 standard allows for up 16 reference frames. As such the ME design requirement in this thesis also supports 4 reference frames in real-time for each target video resolution. The advantages of multiple reference frames are best illustrated in Figure 2.9.

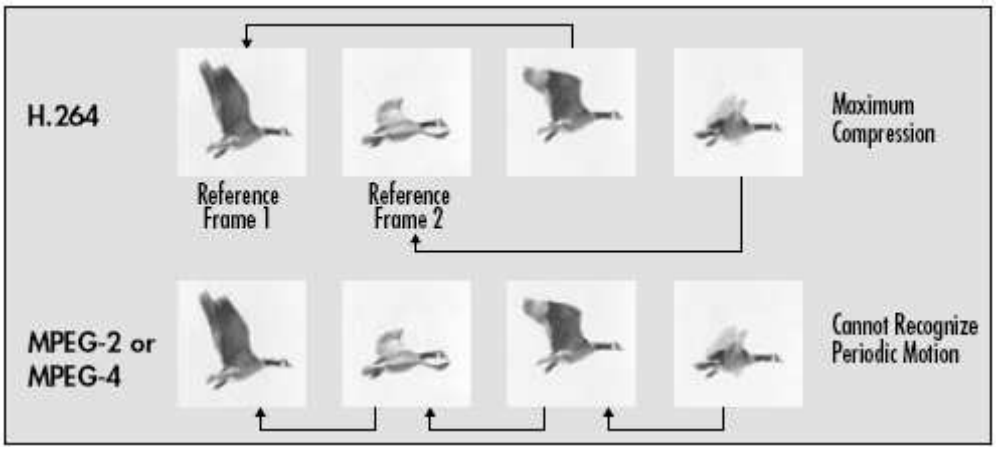


Figure 2.9 MRF in H.264 verses SRF in Previous Standards [20]

The figure shows that when using only a single reference frame, the ME unit can not recognize periodic motion. In the figure, better matches for the wing position of the bird can be detected if extra frames prior to the immediately previous frame are also considered and scanned. A higher number of acceptable matches not only increases the compression ratio, but also increases the quality of the video by allowing the motion to be more life-like [20]. However, these benefits do come at the cost of the ME unit having to scan multiple search windows before deciding upon the best matching referenceBlock.

2.10 Hardware Architectures for H.264 VBSME

The proposed architecture for this work is based on one of the three widely used VBSME architectures — the Propagate Partial SAD [4] [11], SAD Tree [10], and the Parallel Sub-Tree [9]. The Propagate Partial SAD architecture was selected due to its unique blend of efficiency and scalability. While the SAD Tree architecture has the highest performance amongst the three [10], it requires the support of a complex array of shifting registers that must have the capability of

shifting in both horizontal and vertical directions. This array, while efficient to implement in ASICs, consumes a large amount of FPGA resources. The Parallel Sub-Tree architecture, on the other hand, is the most compact design amongst the three. That architecture, however, inherently does not scale well for high performance applications [9]. These points are further explained in this section.

The SAD Tree architecture achieves the highest performance levels by using a purely combinational fast adder tree circuitry to compute all of the 41 SAD values of VBSME. It accomplishes this by initially using a 2-D grid of absolute-difference circuits, or Processing Elements (PEs), to derive the SAD values for the smaller 4x4 blocks. The outputs of these smaller SAD blocks are then funneled down into an upside down pyramid like adder tree structure to generate the SADs of the larger block comparisons [10].

The SAD Tree architecture uses multiple instantiations of its combinational adder tree to increase its parallelism and thereby its performance. These multiple instantiations can be efficiently routed within the ASIC domain. However, since each tree structure does not inherently have any register stages, routing multiple copies of such structures within an FPGA domain poses excessive delay problems. The lower clock speeds of FPGAs are generally countered with increased parallelism and/or deeper pipelining methods. Since massive parallelism is already being employed within the SAD Tree architecture itself, the only available option to mitigate slower FPGA clock speeds is to implement pipelined stages. Adding pipeline stages to the adder tree, however, voids one of the original advantages of the SAD tree architecture – its ability to use fast combinational adder trees.

Moreover, the SAD Tree architecture's ability to use fast adder trees does come at a cost of requiring the support of a highly flexible reference pixels register array to feed its multiple adder trees. As each SAD adder tree executes reference block comparisons vertically down a search window, the array must continually shift vertically down the reference pixels and feed the new data to an adder tree during its operations [10]. In addition, after the set of multiple adder trees have finished processing several columns of reference blocks vertically, a horizontal shift of reference pixels must be performed in order to feed in the next horizontally adjacent set of vertical reference blocks. This requirement of the register array to perform both vertical and horizontal shifts poses yet another problem in successfully implementing the SAD Tree architecture on an FPGA platform. Although shift register chains are available within modern FPGA fabrics, they are unilaterally one-directional and cannot be used to achieve shifts in both the vertical and horizontal directions. Building a custom shifting array on an FPGA, on the other hand, would require excessive FPGA resources and is unlikely to achieve high performance.

The Parallel Sub-Tree architecture offers the lowest performance levels out of the three categories of architectures introduced in this section. However, as a trade off for its low performance it gives the smallest area utilization. As its name suggests, it does not use a full blown adder tree to calculate all 41 SADs in one clock cycle. Rather, it uses a single row of sub reference data at a time and uses 16 cycles in total to process all 16 rows of a 16x16 referenceBlock. By this process, intermediate registers are required to store the partial SAD accumulations. This architecture can also use multiple instantiations of the sub-tree in parallel to scale its performance. However, since only a sub-tree portion is being used at each instantiation, the gains in performance are only marginal. Numerous copies cannot be efficiently created to compensate for the

marginal gains due to the proportional increase in the number of registers required to store the intermediate SAD values.

In light of the disadvantages to the above two categories of VBSME architectures, in regards to achieving scalability on an FPGA platform, the Propagate Partial SAD architecture was chosen as the most suitable architecture for this work. The detailed operations of the architecture and more importantly how it can be scaled on FPGAs are explained in the following two chapters.

Chapter 3

System Design for Scalability

This chapter presents the overall system design. The advantages of using a Field Programmable Gate Array (FPGA) based hardware platform over an Application Specific Integrated Circuit (ASIC) platform is presented and the specific FPGA device used for this thesis is discussed. General system operation of this thesis's Scalable FPGA ME System is explained along with the computational demands placed on the system for the application of real-time H.264 ME.

3.1 FPGA Device

The flexibility of Field-Programmable Gate Arrays (FPGAs) encourages design reuse and can greatly enhance the upgradability of digital systems. This flexibility is particularly useful in the design of highly flexible video ME units that need to handle varying resolutions of video while trying to maintain real-time performance requirements. Application Specific Integrated Circuits (ASICs), by the very nature of their construction do not have such flexibility.

The specific FPGA device selected for this work is the XC5VLX330 device from the Virtex 5 family of FPGAs [21]. This particular device was selected primarily due to its large In-

put/Output (IO) bandwidth (with 1,200 user accessible IO pins) and its relatively large logic capacity (with over 207 K Look-Up-Tables (LUTs)). The availability of 288 36 Kbit RAM Blocks on-chip, also makes the device attractive for its Search Window buffering capabilities.

3.2 System Operation

At its most basic level the ME system accesses macroBlock and referenceBlock data from storage registers or memory and calculates their SAD values. For every macroBlock to referenceBlock comparison, 512 bytes of data need to be access. 256 bytes are accessed for the 256 pixels of the 16x16 pixel macroBlock, and another 256 bytes for the candidate referenceBlock of equal pixel size. The number of macroBlock to referenceBlock comparisons needed for real-time ME of High Definition (HD) video is roughly equal to 1.5 Giga comparisons. Thus data/memory access issues, along with high performance data paths, play an important role in the design of efficient ME systems.

Most high-performance architectures tend to store the 256 pixels of a macroBlock locally on-chip since the macroBlock is repeatedly accessed each time a macroBlock-referenceBlock comparison is made. The Scalable FPGA ME unit of this thesis also adheres to this design principle. An overview of the system block diagram is presented in Figure 3.2.

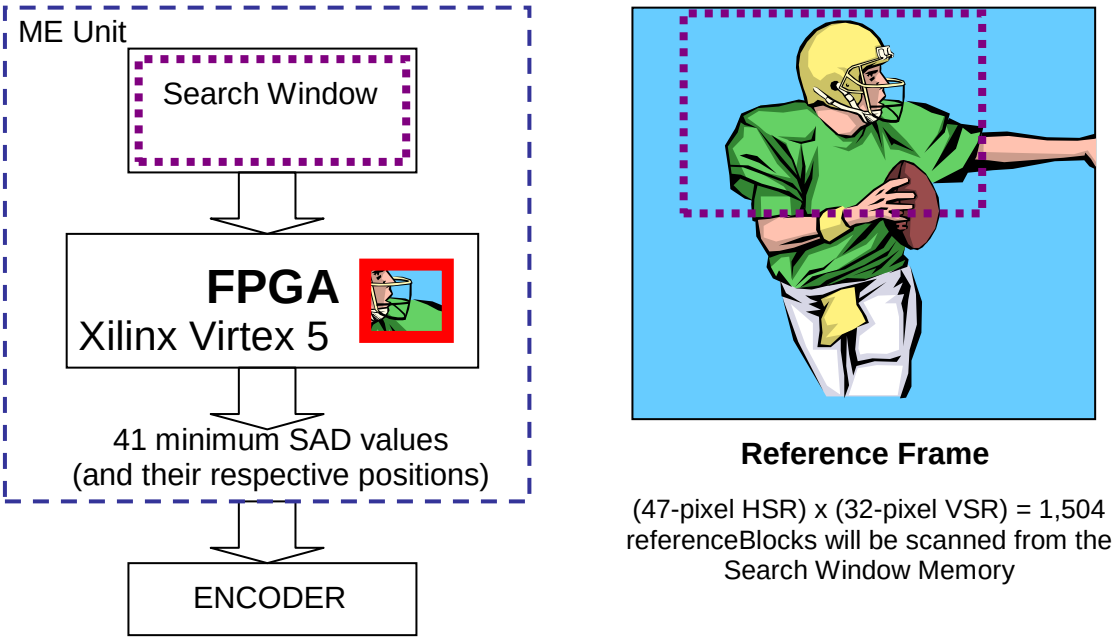


Figure 3.2 System operation of the Scalable FPGA ME unit

In Figure 3.2 the macroBlock for which a referenceBlock match is currently sought is the block that contains the football player's helmet-visor. Notice that it is stored locally on the Virtex 5 chip itself. The remaining data for all of the 1504 (a [-24, +23] horizontal x [-16, +16] vertical search range) candidate referenceBlocks is illustrated as being stored off-chip while they can also be implemented using the on-chip RAM Blocks. The 1504 comparisons are executed on the FPGA and the 41 minimum SAD values (i.e. the best matches) are reported to the encoder along with the referenceBlock positions at which each of the 41 SADs was found.

3.3 Core Architecture

The selection of a core architecture to use within the FPGA ME unit was based on the following two criteria. First the architecture had to have high performance. That is to say that the architecture would focus on delivering the SADs per comparison as rapidly as possible, without being

overly concerned as to the hardware cost. Second the architecture had to lend itself to be instantiated multiple times. In these two aspects, a literature survey revealed that the Propagate Partial SAD Architecture (PPSA) defined within Ikenaga et al.'s work [11] was the most suitable.

The Ikenaga architecture is a high-performance architecture. It utilizes 256 8-bit registers to store all of the pixels of the macroBlock locally and it is able to produce all of the 41 SAD values per macroBlock-referenceBlock comparison at every clock cycle. The original Ikenaga is implemented in 0.18µ ASIC technology and can be run at a maximum clock frequency of 227 MHz.

At 227 MHz, although a high performance design, the Ikenaga architecture falls short of real-time HD video performance and can only support real-time (30 fps) ME for VGA (640x480) video. We will first elaborate on the detailed design of the Ikenaga architecture. Then we will perform some detailed analysis on its performance. Figure 3.3 presents a semi black-box view of the Ikenaga architecture.

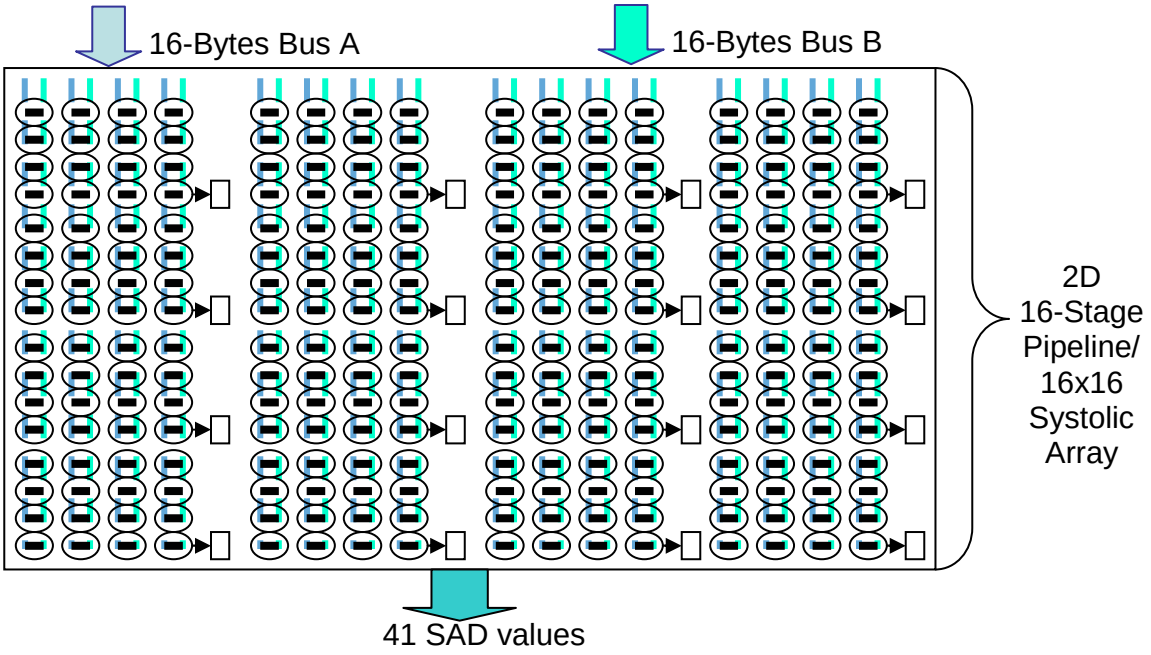


Figure 3.3 Propagate Partial SAD Architecture (PPSA) Black-Box View

The PPSA can be viewed as a 2 dimensional 16-stage pipeline, which consists of a 16x16 systolic array [4] of Processing Elements (PEs). To keep the pipeline fully functional and to achieve a 100% throughput, without any bubbles in the pipeline, the architecture requires a dual-bus input interface. Each input bus is 16 bytes wide. Since the luminance value of a single pixel is a 1-byte (8-bit) entity, each of the 16-byte buses will be read as a row of 16-pixels in every clock cycle. Buses A and B are not always simultaneously active for all clock cycles, but they must function simultaneously for cyclic portions of the clock cycles.

Note that neither of the two buses is used to load macroBlock data into the datapath. Both buses are used to deliver only referenceBlock data. The 256-pixels worth of macroBlock data is pre-loaded into the 256 PEs illustrated as the 16x16 grid of ovals in Figure 3.3. Each PE contains an 8-bit register to store one pixel of the current macroBlock. In addition to the register,

each PE also contains a fast absolute difference circuit. This circuit is used to calculate the difference between the macroBlock pixel stored in the 8-bit register and a referenceBlock pixel. The reference pixel is selectively chosen from one of the two input buses. To make this choice, each PE is also wired to a multiplexer at its input that allows it to read from either of the two buses.

In the early four-stages of the pipeline operation the smallest SAD values for the 4x4 sub-Blocks of VBSME are calculated. Then these 4x4 SAD values are propagated further down the pipeline and accumulated with other absolute differences to form the larger SAD values, all the way up the last 16x16 SAD value. The 16x16 SAD value along with the other 40 smaller SAD values form the set of 41 SAD values required for full VBSME support. The detailed design of Ikenaga's PPSA will be presented in the next chapter along with a description on how it can be incorporated into a scalable design.

For this thesis the IO usage required to fully support the PPSA design and the level of performance that can be gained if the IO requirement is supported, as the architecture is scaled, are the two essential factors for real time performance on FPGAs.

As mentioned earlier, the PPSA supports a rapid output of all 41 SADs per clock cycle which means that a single macroBlock to referenceBlock comparison is being completed at very cycle. A modern High Definition (HD) resolution frame of video has a pixel area of 1920 (columns) x 1088 (rows). For this 1920x1088 area, 8160 16x16 macroBlocks are needed to encode the frame. The 48-pixel Vertical by 33-pixel Horizontal search range requires that 1584 candidate

referenceBlocks be compared against each macroBlock. H.264 based ME implementation further demands four reference frames to be used for multiple reference frames. This means that 6336 (4x1584) comparisons are needed for each macroBlock. Thus over 51 Mega comparisons are needed for every frame of HD video. At 30 fps for real-time video, this amounts to over 1.5 Giga comparisons per second.

The PPSA supports one macroBlock to referenceBlock comparison per clock cycle. Therefore it would need to operate at ~1.5 G Hz in order to perform the VBSME required for HD video at 30 fps. However, since its 0.18µ CMOS technology only supports a maximum clock rate of 227 MHz, at this speed it is only able to support VGA (640x480) resolutions at 30 fps. Given that FPGA speeds are nowhere near being capable of 1.5GHz clock frequencies, in order for an FPGA implementation to deliver real time HD video performance, it needs to achieve massive parallelism through the multiple instantiations of the PPSA. In this work a single instantiation of the Propagate Partial SAD Architecture will henceforth be referred to as a Pixel Processing Unit (PPU), as such multiple PPSA instantiations will be called PPUs. The following sections introduce and explain the challenges that were overcome to achieve a high level of parallelism on the FPGA platform through the multiple instantiation of PPUs.

3.4 Standard PPU Data Flow

Each PPU requires two 16-byte wide input buses and 41 13-to-17 bit wide output buses (one for each of the 41 SAD values). The following sections describe how this standard design can be scaled to obtain higher performances. The new scalable motion estimation unit design intelli-

gently shares input data and funnels the SAD outputs through a series of comparators to minimize the increase in its IO bandwidth as the design scales. We first discuss the data flow of a standard PPU. We then describe the scalable motion estimation unit and how it utilizes the data-flow of the PPUs to reduce its IO bandwidth.

As is discussed in section 3.2 and 3.3, the Propagate Partial SAD architecture speeds up VBSME algorithms by simultaneously calculating SAD values for 16 reference blocks at a time. In particular, the architecture takes advantage of the fact that, in a search window, every vertical group of 16 reference blocks share a common row of 16-pixels as is shown in Figure 3.4. In the Propagate Partial SAD architecture, this common row is then used to simultaneously calculate 16 absolute difference values for each of the 16 reference blocks. A specialized pipeline structure within a PPU is then used to accumulate these absolute difference values to produce the 41 SAD values per reference block at every clock cycle.

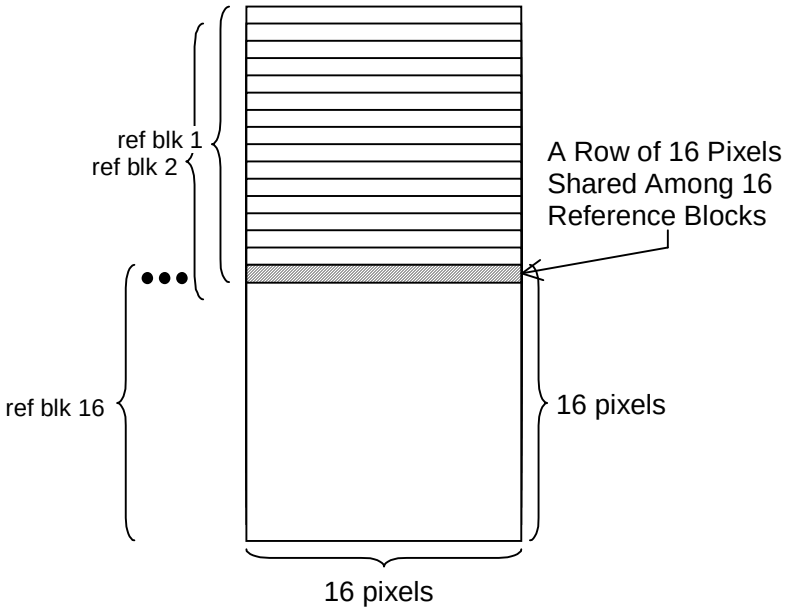


Figure 3.4 Pixel Sharing within Pipeline

To support the performance described above, the memory unit is required to feed a 16-pixel-row of search window data to the PPU at every clock cycle. Furthermore, as the column of 16-pixel rows nears its completion, an additional 16 pixels from the next column must be simultaneously fed as well. The reason for this is to keep a PPU's pipeline filled at all times without suffering additional latencies. This dataflow requirement dictates that a 32-pixel/byte bandwidth must be given to each PPU. The next section describes how this can be accomplished for *n* PPUs without the total input bandwidth being increased by *n* x 32 bytes, as the performance of the motion estimation unit is scaled.

3.5 IO Bandwidth Reduction

The overall structure of the scalable VBSME architecture is shown in Figure 3.5.1. It consists of a bank of memory that stores the search window, an input distribution unit, *n* Pixel Processing

Units (PPUs), and two sets of comparators. The memory storing the search window is divided into two partitions. Each partition contains an output of 15+ *n* pixels. These outputs are expanded into 2*n* buses by the input distribution unit, where each bus contains 16 pixels. The 2*n* buses are then fed into *n* PPUs, which have been initialized with a macroBlock's pixel values.

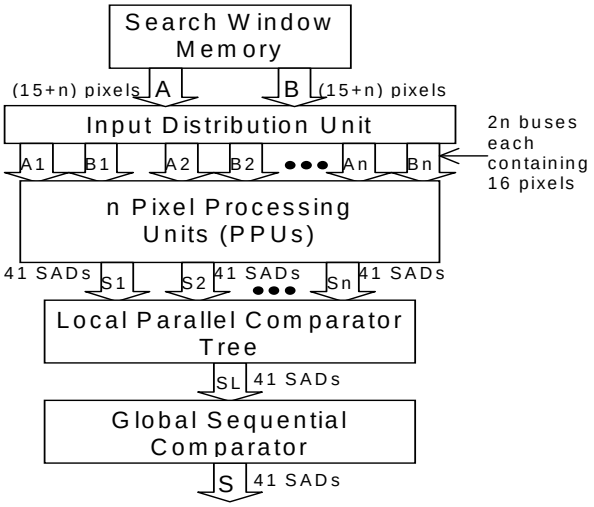


Figure 3.5.1 The Scalable ME Architecture

The PPUs are used to produce *n* x 41 SAD values at each clock cycle. These *n* x 41 SAD values are then used to compute the minimum SAD values of the search window in two steps. First, the *n* x 41 SADs are fed into the local parallel comparator tree. This tree computes 41 minimum SAD values from its n x 41 inputs. The local minimum SAD values are forwarded to the global sequential comparator, which determines the 41 minimum SAD values for the entire search window. Note that the global comparator is of a conventional less-than comparator design as shown in Figure 3.5.1 [11] and the scaling of the VBSME architecture does not affect its complexity.

The detailed design of the input distribution unit, the PPUs, and the local parallel comparator tree is shown in Figure 3.5.2. As shown, the core of the scalable VBSME architecture is the PPUs, which are based on the Propagate Partial SAD architecture. Each PPU produces 41 SAD values (corresponding to an entire set of SADs for a single reference block) at every clock cycle. The number of PPUs utilized in the scalable architecture, therefore, corresponds directly to the number of reference blocks that can be processed in a clock cycle and the overall performance of the system. However, as the number of PPUs increases, the output bandwidth required for the search window memory increases as well. In particular, in order to keep a PPU fully utilized during motion estimation, one would require two rows of 16-pixels to be forwarded from the search window memory to the PPU at every clock cycle (one row from each of the search window memory partitions) [11]. Typically, a byte is used to encode a pixel, therefore one needs to transport 32 bytes from the search window to a PPU in every clock cycle.

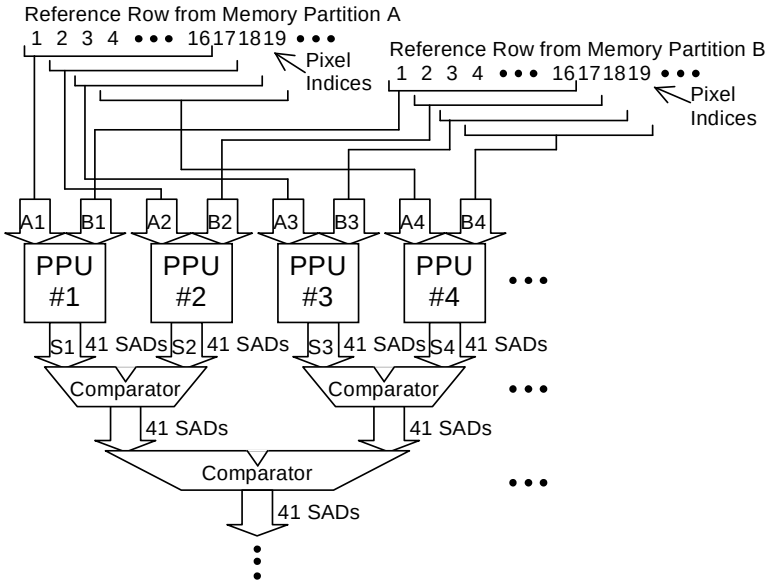


Figure 3.5.2 Input Distribution Unit, PPUs and Local Comparators

A naive approach would be to simply increase the output of the search window memory by 32 bytes for every additional PPU. However, this can quickly exhaust the internal memory bandwidth of an FPGA (if the search window is stored on the same chip as the PPUs) or the IO pin limit of even the largest modern FPGAs (if the search window is stored off chip). For example, the Xilinx XC5VLX330 is the largest device that Xilinx currently offers. It contains 1200 available IO pins. Assume that the search window is stored off chip. Implementing a single PPU on the XC5VLX330 would require 256 input pins. Implementing four PPU copies would require 1024 pins (over 85% of the available IOs on the XC5VLX330) – leaving an insufficient number of IOs for output and control signals.

More importantly, the above approach does not take into account the large number of pixels that are shared among the reference blocks. For example, Figure 3.5.3 shows 32 reference blocks in a search window. These blocks are divided into two groups where each group contains 16 reference blocks. Within a group, the reference blocks are organized to be processed by their own dedicated PPU, where all blocks are contained within a single 16-pixel wide column and one block is offset from the next by a single row of pixels (which is fed to the PPU cycle by cycle).

Since one group is offset from another by a single column of pixels, all 32 blocks in Figure 3.5.3 share 15 common pixels. To increase performance, these two groups can be simultaneously processed by two PPUs (shown as PPU x and PPU (x+1) in the figure). Since 15 pixels are shared between the groups, one would require 17 pixels (instead of 32) to be read from the search window (per single bus) at a time. In particular, if pixels (a, y), (a+1, y), …, (a+15, y) of

the search window are being processed by PPU x, pixels (a+1, y), (a+2, y), …, (a+16, y) should be simultaneously processed by PPU (x+1).

In general, to fully utilize n PPUs, one would require (15 + n) pixels to be read from each partition of the search window memory for every clock cycle. These signals should then be distributed using the topology shown in Figure 3.5.2.

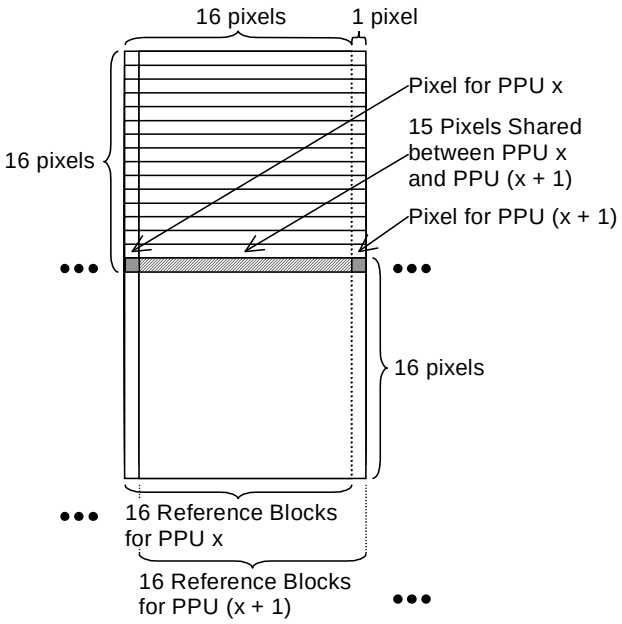


Figure 3.5.3 Sharing of Pixels among PPUs

At its output, each PPU shown in Figure 3.5.2 produces 41 SAD values at every clock cycle. These SAD values amount to 573 bits of data. To keep the output width constant as the number of PPUs increases, the local parallel comparator tree is implemented on the same FPGA as the PPUs. Note that the number of comparator tree stages is equal to $\lceil \log_2(n) \rceil$ where n is the number of PPUs that the architecture contains. We observe that by registering the values produced at each stage of the comparator tree one can ensure that the comparator tree does not become the

critical path of the system. Consequently, the overall system performance does not degrade significantly when an increasing number of PPUs are used. Any minor drops in clock speeds, by 1 to 2 MHz, as more PPUs are added to the system are more likely due to a slight increase in routing delay as the size of the comparator tree grows rather than any increase in logic delay.

Up till now, only the flow and organization of data after it has been outputted by the Search Window Memory has been presented. It has been assumed that the memory structure is capable of outputting the correct amount of pixels, row by row – with a maximum bandwidth of simultaneously outputting two 31-pixel rows (16-PPUs case) – at the correct time. Chapter 4 will discuss in detail the logical to physical memory mappings and memory organizations that are required to sustain this high output bandwidth from the Search Window Memory, as well as the detailed design of the PPU itself.

Chapter 4

PPU & Memory Design

The fundamental building block of this work is the Pixel Processing Unit (PPU). Its detailed design is covered in the first three sections of this chapter. A bottom to top approach will be used to cover all the sub-modules that have been used to build the PPU. The final sections discuss the detailed memory organization aspects of this work.

4.1 Absolute Difference Accumulation Circuitry

The absolute difference calculation, between the luminance value of a current pixel and a reference pixel, forms the very basis of a Motion Estimation unit. In the Propagate Partial SAD Architecture (PPSA) of Ikenaga's work [11], four simultaneous absolute difference operations and their accumulation form the basic building block of their architecture. In our work this basic building block is referred to as the Row_Adder module.

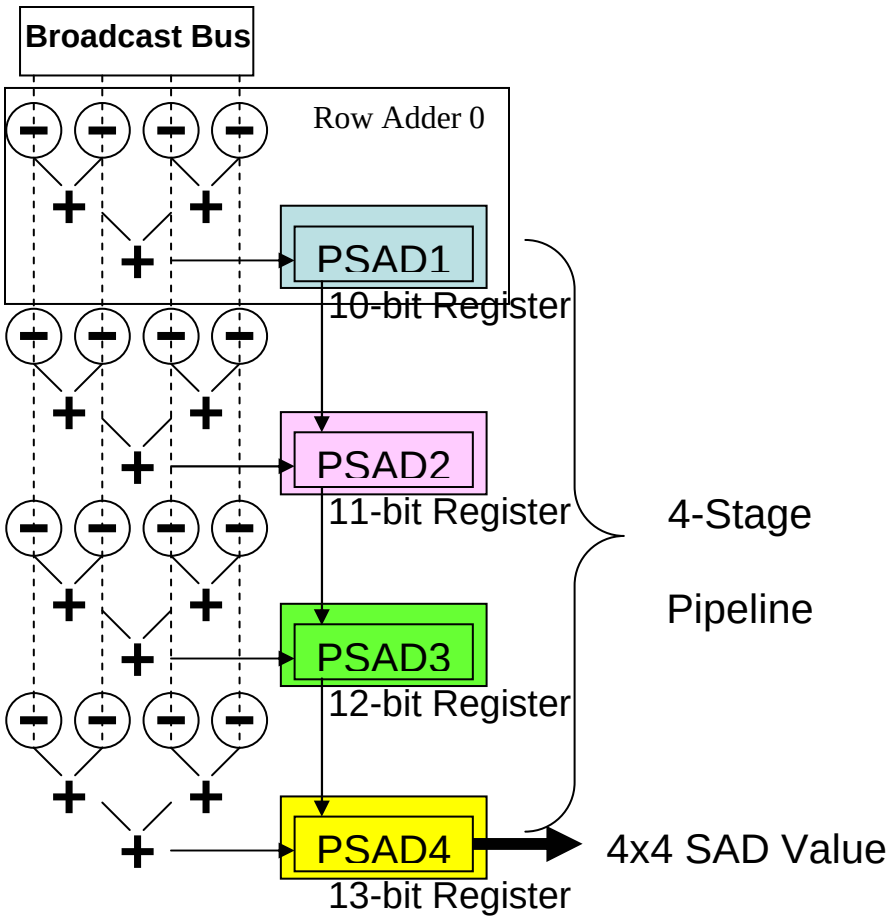


Figure 4.1 4 Row_Adder modules and the Four_by_Four_Block module

Four Row_Adder modules are connected together serially to calculate the absolute differences of 16 current and reference pixels, as is illustrated in Figure 4.1. In the figure, the subtraction signs enclosed by a circle represent the Processing Elements (PEs). Each PE contains the absolute difference calculation circuitry and an 8-bit register to store a pixel value from the current macroBlock. The three "+" symbols beneath each set of 4 PEs represent the adder-tree used to create the output of the Row_Adder module.

Each of the Row_Adder modules represents a single or partial stage of the four stage accumulation, thus its output is referred to as a Partial Sum of Absolute Differences (PSAD). With the exception of the very first Row_Adder module, the remaining 3 Row_Adder modules have to add their own four absolute difference values to the preceding PSAD value passed to them. Thus the adder tree and final storage register width of each Row_Adder stage grows in the subsequent stages.

The Row_Adder modules are the critical path of the whole PPU design. Thus Ikenaga et al. in their work have chosen to implement a Carry Save Adder (CSA) tree structure for the accumulation of the four absolute difference results at each of the 4 Row_Adder stages. The CSA trees are used in Ikenaga's ASIC domain to optimize these critical paths and thus minimize the gate delays to the lowest possible levels. On the FPGA platform, however, through experimental results, it was found that the CSA tree structure speed advantage only applies to the last 3 Row_Adder stages. The use of a CSA tree for the first stage actually slows down the performance by 0.042 nanoseconds (from 3.076ns obtained without CSA).

For the very first stage, accumulation of a previous PSAD is not required and thus its width is the narrowest at only 10-bits. For this narrow width, the built in carry-chain adders within the FPGA fabric designed for fast-arithmetic outperform Ikenaga's CSA implementation in ASIC. However, when the width of the Row_Adder modules gets larger in the latter 3 stages the CSA tree structure begins to outperform the carry-chain. Therefore in this work a straightforward adder tree is implemented in RTL Verilog for the first Row_Adder module, which the Xilinx Synthesis Tool (XST) then optimizes using the FPGA carry-chains. But for the latter 3 wider

Row_Adder modules Structural Verilog was used to implement the custom CSA trees. This variance in design ensured that the shortest critical path was achieved at each Row_Adder stage.

4.2 Datapath Construction

The four Row_Adder modules are grouped together into a Four_by_Four_Block module, as shown in Figure 4.1. The module contains a 4x4 array of PEs. It is capable of outputting a single 4x4 SAD value for the 16 pixels of current and the 16 pixels of reference values that it compares.

Within the Four_by_Four_Block, in addition to the 16 PEs and four adder-tree structures, there are also 16 8-bit multiplexers attached to each PE unit. These multiplexers enable each PE to read from either one of two bus lines that feed into the Four_by_Four_Block. This increase in hardware allows for the pipeline to remain operational without any pipeline bubbles.

The Four_by_Four_Block modules are instantiated 16 times to create the 16x16 systolic array architecture shown in Figure 3.3. The 4x4 PSAD values produced by a preceding row of Four_by_Four_Block modules are required for the calculation of larger SAD values at the latter stages of the pipeline, thus the architecture also requires delay lines. These delay lines serve the purpose of holding and delivering the smaller PSAD values, of the subBlocks within VBSME, at the exact time at which they are required.

For example, let us consider the derivation of the PSAD for an 8x8 subBlock. As shown in Figure 4.2, the 8x8 PSAD can be obtained through the accumulation of the PSADs within its four separate 4x4 subBlocks.

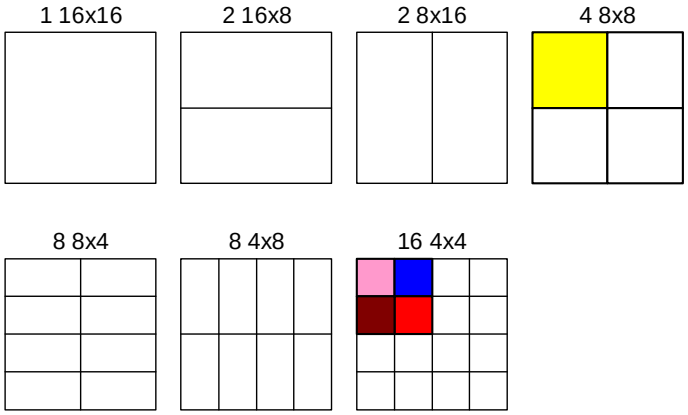


Figure 4.2 The 1 macroBlock and 40 subBlocks in VBSME

There are two 4x4 subBlocks within the upper half of the 8x8 subBlock and two 4x4 sub-Blocks within the lower half of the 8x8 subBlock. The two upper half 4x4 blocks would output their PSADs after the first four cycles of system operation. However, only after another 4 more cycles (thus 8 cycles total) would the lower 2 4x4 subBlocks produced their PSADs. Thus it is necessary for the upper 2 PSADs to be stored within a 4-stage delay line, so that they become available for use four clock cycles later. Using the exact analogy, an 8-stage delay line is required to compute a 16x16 SAD by accumulating the upper and lower halves of 8x8 subBlocks.

Note that the accumulation of PSADs from the smaller subBlocks to form the larger sub-Blocks is performed through a purely combinational-logic adder tree structure. Although these adder trees are wider than the internal Row_Adder trees found within the Four_by_Four_Block

module they lack the absolute difference circuits. Consequently these adders are not on the critical path and are implemented using the conventional FPGA adder carry-chains.

4.3 IO Requirements per PPU

The reference pixels broadcasted into the PPU is a significant input bandwidth issue. A single broadcast consists of inputting 16 pixels to the PPU per clock cycle. With each pixel representing an 8-bit luminance value this amounts to 128 bits of input per cycle. To avoid any pipeline bubbles dual broadcast lines are required.

The PPU processes reference pixels within a search window memory one 16-pixel-wide column at a time. A single column of 16 pixels is broadcasted into the 16-stage pipeline of the PPU one row at a time at each clock cycle, until the bottom of the column has been reached. Initially, when the PPU first becomes operational there will 4 cycles of latency before the first 4x4 subBlock's, 8 cycles of latency before the first 8x8 subBlock's, and 16 cycles of latency before the first full 16x16 macroBlock's SAD value is outputted. However, after these initial latencies, a complete set of 41 SAD values are outputted at each clock cycle. As the PPU works its way down the 16-pixel wide column, a single 16-pixel broadcast bus is sufficient to keep the pipeline filled. However, as the PPU begins to receive broadcasts of the last 15 rows of the first column the primary stages of this pipeline will begin to idle if they are not simultaneously fed with the next column's worth of 16-pixels. This is shown in Figure 4.3.

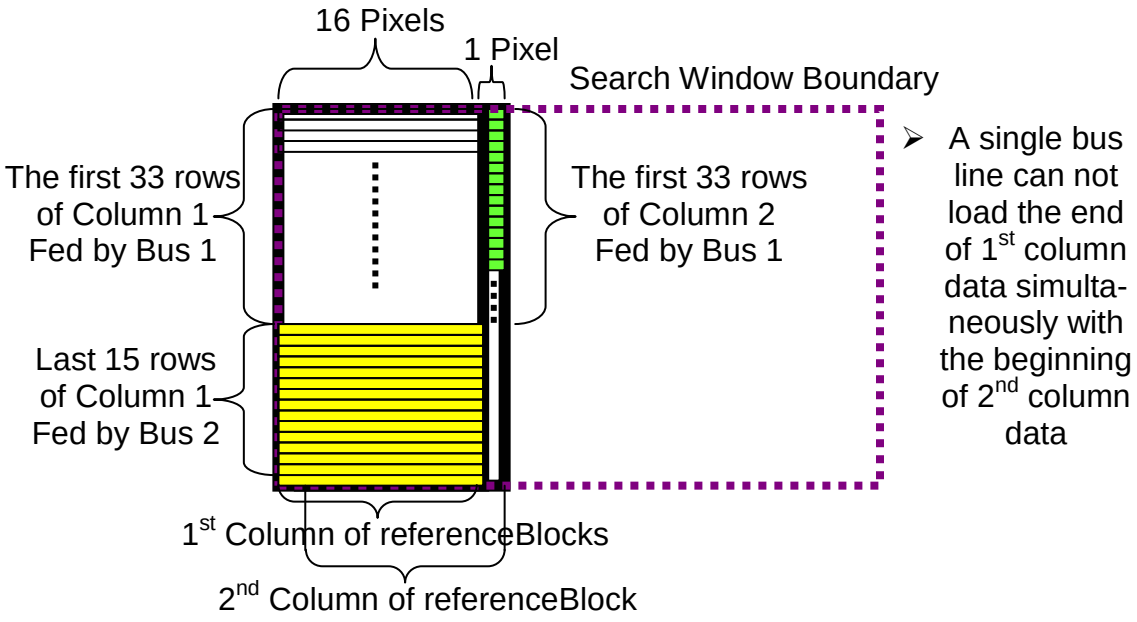


Figure 4.3 Processing Two referenceBlock Columns Back-to-Back

This necessitates filling the initial pipeline stages of the PPU, with the 2nd column's data, from a 2nd broadcast bus line. The 2nd broadcast bus line continues to deliver the last 15 rows of the 1st 16-pixel column of reference pixels, thus freeing the 1st broadcast bus line to begin to broadcast the next set of 16-pixels from the next column over. Therefore during a column's last 15 row's worth of SAD calculations both bus lines are simultaneously utilized. This requires a maximum bandwidth of 32 pixels per clock cycle. Given that each pixel is a byte representation of a luminance value, 32 bytes per cycles of input bandwidth is required. This translates to 256 (i.e. 256 bits) input pins required per PPU, in addition to the control signals that are required to keep the PPU functioning.

The output bandwidth for a PPU is simply dictated by the bit-widths that are required to represent each of the subBlock PSAD values and the final macroBlock SAD value without any

overflow. For example the PSAD value for a 4x4 subBlock only requires a 13-bit quantization to represent the maximum current to reference subBlock distortion, whereas a full 16x16 macroBlock SAD value needs a 17-bit quantization to represent its maximum distortion levels. The accumulation of the bit-quantization number over all 41 SAD values gives the 573 bits that are required to represent all 41 SAD values. Therefore 573 output pins in total are required to deliver the 41 SAD values in parallel at each clock cycle.

Note that all 41 SAD values are required by the encoder since the decision to switch to smaller block size encoding is made by the encoder itself, when it continually detects across a couple of frames that the smaller subBlocks are producing much better matching PSAD values than the larger macroBlock SAD values to which they belong. This type of decision making is not a memory-less type process since it requires the analysis of several sets of 41 SADs across consecutive frames. Consequently, the encoder requires all 41 SADs to dynamically interpret the form of motion that is being produced by the video. The total 829 pin (256 input + 573 output) IO requirement for a single PPU poses severe constraints in terms of scalability when incorporating multiple PPUs into a motion estimation unit design.

Chapter 3 covered the design methods by which these high IO bandwidth requirements were mitigated, to make scalability feasible. The following sections will now examine the physical organization of the search window memory. In particular, how this physical organization facilitates the scalability of the motion estimation unit will be explained. The benefits of double-buffering search window data will also be examined.

4.4 Logical to Physical Mapping

Figure 4.4.1 shows the logical partitions of the search window memory for a 4 PPU design of the scalable motion estimation unit. As shown, the search window memory is partitioned into two logical partitions with memory partition A containing the top 33 rows of the search window and memory partition B containing the bottom 15 rows. Each partition is then divided into 16 sub-partitions with each sub-partition containing 4 columns of pixels.

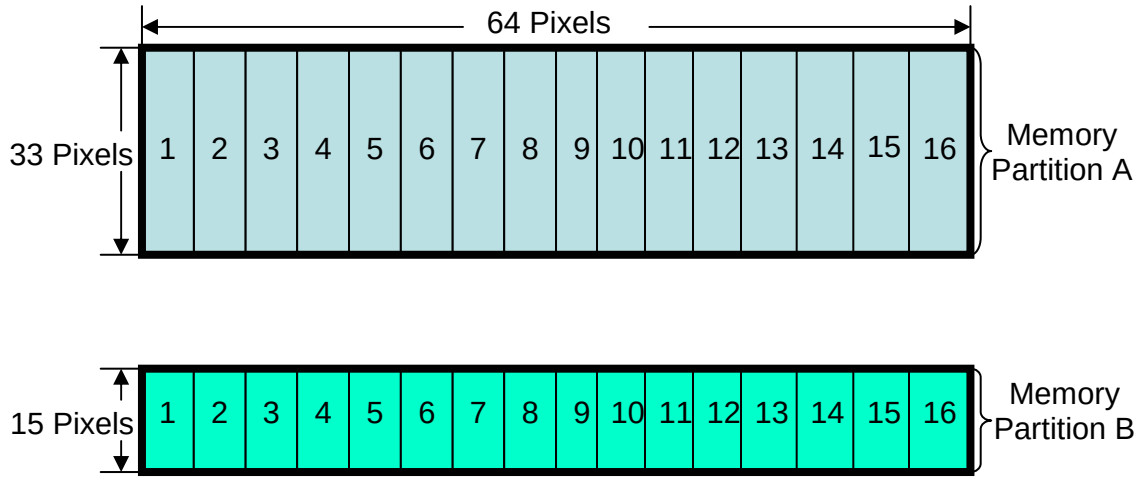


Figure 4.4.1 Logical Memory Partitioning of Search Window Pixel Rows (4 PPUs)

The number of columns that each sub-partition contains is determined by the number of PPUs in the motion estimation unit. In particular, for an n PPU design, there should be $\lceil 64/n \rceil$ sub-partitions per logic partition. $\lceil 64/n \rceil$ - 1 of the sub-partitions should contain n pixels and the remaining columns should be stored in the last sub-partition.

Figure 4.4.2 shows the memory output distribution of pixels, for the same 4-PPU system, during the initial stage of operation for the motion estimation unit. As shown, in this stage pixel-

columns 1 to 19 are accessed. These pixel-columns correspond to sub-partitions 1 to 5 in Figure 4.4.1. Note that the 20th pixel-column (located within sub-partition 5) is not used by the 4 PPUs, and thus is discarded from the output for both logic partitions A and B. Once the pixel-columns 1-19 are fully processed by the PPUs, the output from the logic partitions are advanced by 4 pixel-columns (one sub-partition).

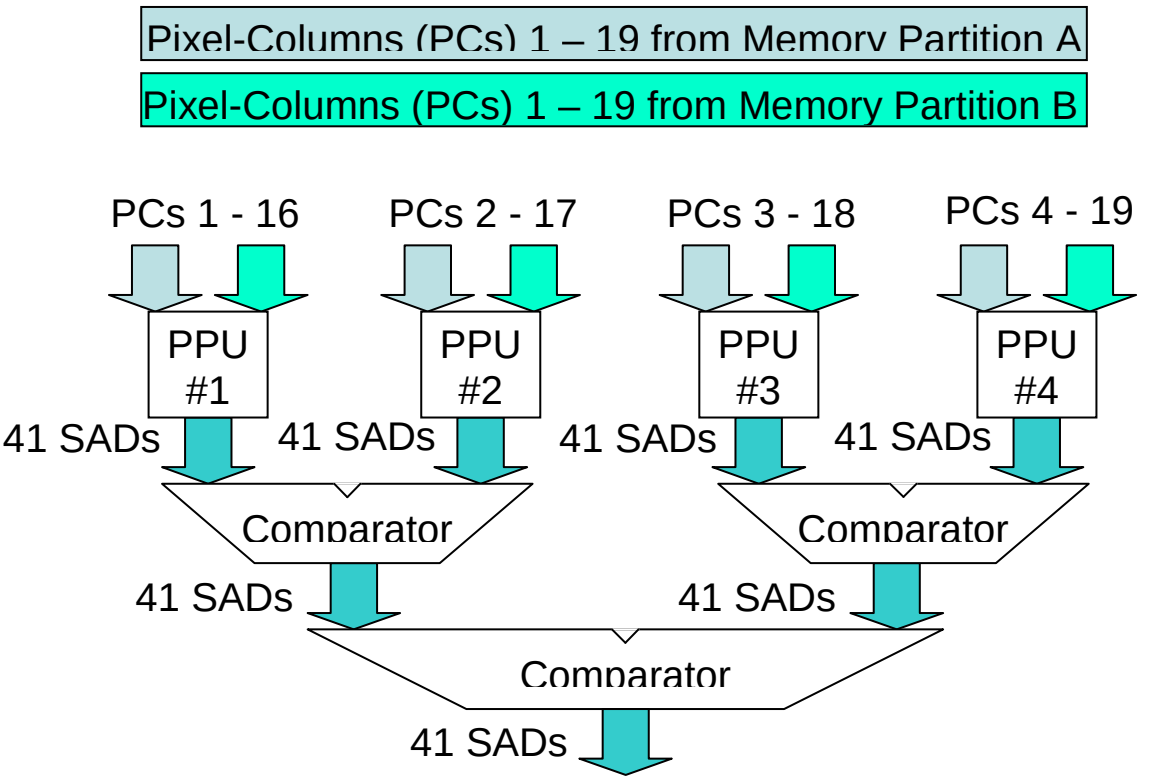


Figure 4.4.2 Memory Output Distribution for a 4-PPU System

Consequently, pixel-columns 5 (from sub-partition 2) to 23 (from sub-partition 6) is accessed in the second stage with pixel-columns 5-20 distributed to PPU #1, 6-21 distributed to PPU #2, 7-22 distributed to PPU #3, and 8-23 distributed to PPU #4. Similarly, the logic partition output is advanced by 4 pixel-columns in each of the subsequent stages of computation.

In Figure 4.4.3, five physical memory blocks are used to store all of the 16 sub-partitions for both the A and B logic partitions. The sub-partitions are stacked into the physical memory blocks in a left-to-right and then bottom-to-top manner.

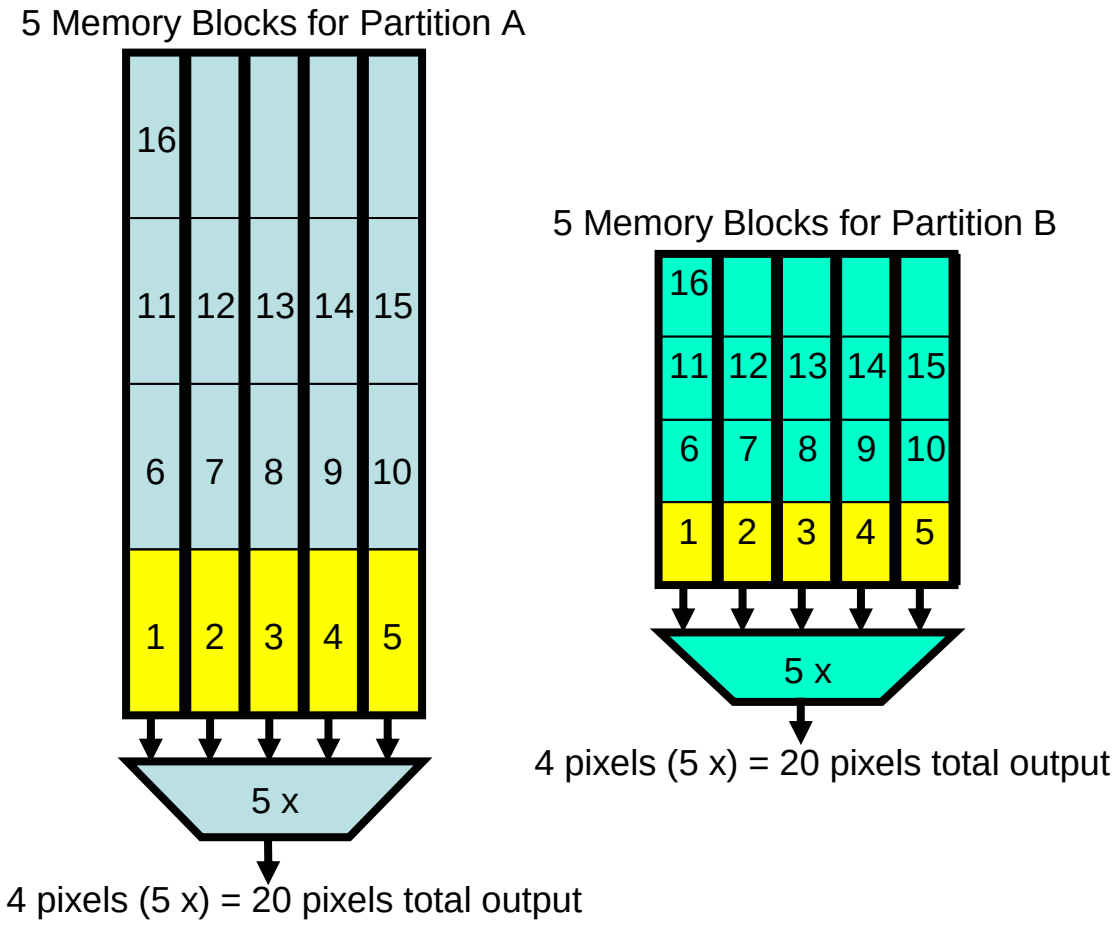


Figure 4.4.3 Physical Memory Organization (4 PPUs)

The 5 physical memory blocks are multiplexed by 5 individual multiplexers. Each multiplexer outputs 1 of the 5 available physical memory blocks, to generate the 19 pixels shown in Figure 4.4.2 at each stage of computation. Note that in this configuration, each physical memory

block, must have its own independent address lines so that the depth of memory addressing can be varied to access the sub-partitions stored within each block independently.

In general, for *n* PPUs, this memory mapping algorithm requires $p = \left\lceil \frac{(16+n)}{n} \right\rceil$ physical (*p*) memory blocks for each of the A and B logic partitions. The width of each physical memory block must be equal to the width of each logic sub-partition, which is also equal to *n*. Consequently as the number of PPUs increases, the physical memory blocks become fewer and wider. In particular, a 16-PPU system would require only 2 memory blocks with each block being 16 pixels wide. In general, systems with a greater number of PPUs will access an increasing number of pixel-columns per cycle, and advance by a larger number of pixel-columns after each stage of computation.

4.5 On-Chip Double Buffering

When targeting an FPGA with a moderate number of user-available IO pins, the use of off-chip memory may still become an IO bottleneck. Consider the case of a system scaled to 16 PPUs using external off-chip memory. Such a system requires that 31 pixels be inputted on each of its dual bus lines (16 pixels for the initial PPU, followed by 1 extra pixel for each of the 15 additional PPUs). With each pixel being encoded as a byte of data, this drives the total input data bandwidth across the dual buses to 62 Bytes. The output will consist of 41 SAD values (independent of the number of PPUs used) and would consume 72 bytes of IO. When control signals are considered, the total IO requirement for a 16-PPU motion estimation unit using off-chip memory becomes at least 135 bytes or 1080 IO pins/bits.

On devices where such a number of IO pins is not available, the on-chip RAM blocks available on most modern FPGAs can be utilized to buffer two search windows. Having search window data buffered on-chip eliminates the need for wide bandwidth data access to external off-chip memory. In general, the required input bandwidth to an on-chip double buffered memory structure will decrease as the number of PPUs is reduced – a motion estimation unit with a large number of PPUs will process more data at a faster rate. Thus the memory structure supporting it should be continuously updated, with a higher input bandwidth. A motion estimation unit with fewer number of PPUs will be slower. Thus it memory structure can also afford to be slower with a lower input bandwidth.

Using double buffering, while the 1st search window is being processed, the 2nd buffer can be used to load in the next search window data at a slower rate. This is possible since PPUs must scan each search window multiple times – whereas new search windows need to be written in only once. The time that a search window is processed by the PPUs poses a limit on the time that can be used to load the 2nd search window. This limit corresponds to a lower bound on the input-bandwidth of an on-chip double buffered system.

The lower bound is governed by the rate at which *n* PPUs can process a 64x48 pixel search window. Recall that the 48 rows of the search window are separated into two logical partitions as shown in Figure 4.4.1. This separation facilitates the simultaneous access of partition B with partition A. Each column of the search window can be accessed in 33 cycles, which corresponds to the number of rows that partition A contains.

The number of cycles that a motion estimation unit with *n* PPUs requires to process a 64x48 search window is defined as *cycles* = 33 x $\left\lfloor \frac{64-15}{n} \right\rfloor$. This equation shows that as n increases, the number of cycles that are required to process a search window decreases and the time that can be used to load the next search window decreases as well.

For the 16-PPU system, the processing time per search window is 99 cycles. Thus 3072 (64x48) bytes of new search window data must be loaded into the 2nd buffer in the same amount of time. This requires an input bandwidth of 32 bytes ($\left\lceil \frac{3072}{99} \right\rceil$). Recall that this 16-PPU system configured for off-chip external memory access requires a total input bandwidth of 62 bytes without double buffering. Thus 30 bytes (240 IO bit pins) can be saved through the use of on-chip double buffering. Figure 4.5 shows the overall on-chip double buffered structure for the 16-PPU system.

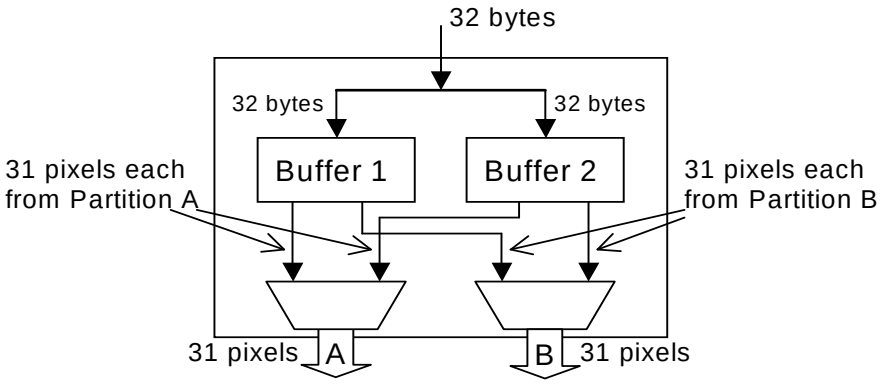


Figure 4.5 Double Buffered On-Chip Memory Structure (16 PPUs)

4.5 On-Chip Double Buffering

The 16 x 16 pixel macroBlock stored locally within the PPUs themselves can also be double-buffered. In the case of macroBlock data, however, each buffer would only have to be updated after every four search windows have been processed (each window being from a unique reference frame, to make up the Multiple Reference Frames component of H.264).

Chapter 5

Performance and Hardware Costs

In this chapter, we first present the reduction in input bandwidth that can be achieved through on-chip double buffering. The performance and hardware resource costs of implementing the motion estimation unit are then presented as a function of the number of PPUs. Finally, the testing and verification methods used in this work are presented.

5.1 Design Results

Table 8.1.1 summarizes the input bandwidth required for implementing motion estimation systems containing 1, 2, 4, 8, and 16 PPUs. Column 1 of the table indicates the number of PPUs being used. Columns 2 and 3 give the required input bandwidth if external off-chip memory is used. Columns 4 and 5 present the lower bounds on the input bandwidth that can be achieved through on-chip double buffering. Finally column 6 shows the lower bounds on input pins needed to update the macroBlock buffers.

Table 5.1.1: Lower Bound on Input Bandwidth

# of PPUs	Search Window				macroBlock
	Without On-Chip Memory		With On-Chip Memory		
	Bytes	Input Pins (bits)	Bytes	Input Pins (bits)	Input Pins (bits)
1	32	256	1.94	15.6	0.33
2	34	272	3.88	31.1	0.65
4	38	304	7.76	62.1	1.30
8	46	368	15.6	124.2	2.59
16	62	496	31.1	248.3	5.18

From the table it can be observed that from roughly 2 (16 input pins) to 31 Bytes (248 input pins) can be saved by implementing a double buffer on-chip.

To evaluate the performance and area efficiency of the scalable VBSME architecture, five variations of the design shown in Figure 5.1 were implemented on a Xilinx Virtex 5 XC5VLX330 FPGA. Each design contains 1, 2, 4, 8, or 16 PPUs. As the design scales, the target resolution scales as well from VGA (640x480) to High-Definition (HD) Video (1920x1088).

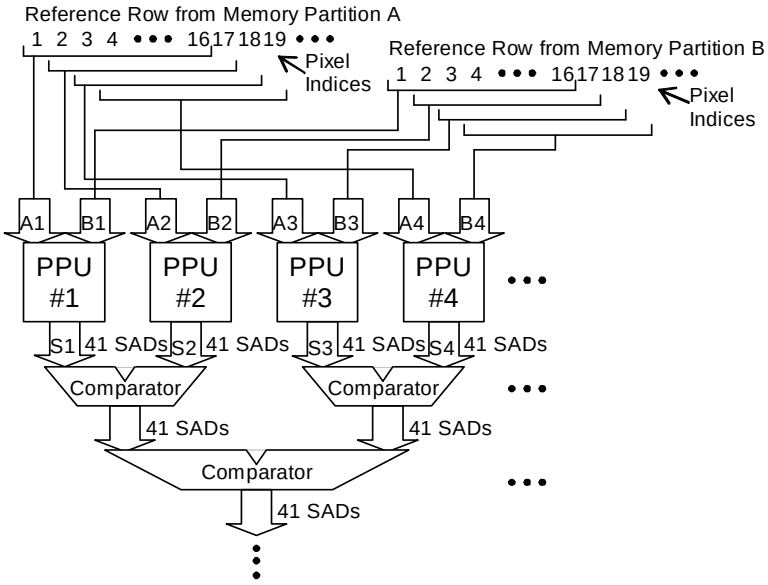


Figure 5.1 Input Distribution Unit, PPUs and Local Comparators

These designs were implemented in Verilog and synthesized using the Xilinx Synthesis Tool (XST) in the Xilinx Integrated Software Environment (ISE). The synthesis constraints are set to maximize speed. All designs met the IO constraints of the XC5VLX330 with 70%, 71%, 74%, 79%, and 90% IO utilization, respectively. The performance and area of each implementation is summarized in Table 5.1.2.

Table 5.1.2: Area and Performance Results

# of PPUs	Area*				Performance		
	LUTs		DFFs		Target Resolution	Freq. (MHz)	fps
	# (K)	%	# (K)	%			
1	8.71	4.20	3.42	1.65	640x480 (VGA)	200.6	28
2	18.5	8.92	5.49	2.65	800x608 (SVGA)	199.0	34
4	37.8	18.2	9.64	4.65	1024x768 (XVGA)	198.3	42
8	76.4	36.8	18.0	8.68	1920x1088 (HD Video)	198.3	31
16	154	74.3	34.6	16.7	1920x1088 (HD Video)	198.3	62

* Xilinx's Vertex 5 devices use 4 DFFs & 4 6-input LUTs per Slice

Column 1 of the table lists the number of PPUs in the design. Columns 2 and 3 lists the number of LUTs required for the design and the number of LUTs required as a percentage of the total number of LUTs in the FPGA, respectively. The same values are summarized in column 4 and 5 for DFFs. Finally column 6 lists the target resolution of each design. The maximum operating frequencies of the circuits are shown in column 7 and their corresponding frame-per-second performances are shown in column 8.

The 16-PPU system was also implemented with on-chip double buffering. The implementation consumes 32 36Kbit block rams, 155K LUTs and 35.2K DFFs. The system performance is lowered to 191.6 MHz due to an increase in routing delay resulting when on-chip memory is added. This corresponds to a frame-per-second performance of 60 fps.

The decrease in clock frequencies from 200.6, 199.0, and to the stable 198.3 MHz as the design scales from 1, 2, and 4 PPUs respectively is due to the initial increases in routing based net delays, and not any increases in logic gate delays. However, the fact that the circuit performance remains consistently near 200 MHz as the design scales from 1 to 16 PPUs offers much promise for FPGA-based H.264 motion estimation especially as future resolutions are scaled beyond HD Video. Table 2 shows that real time motion estimation performances can be achieved with 1, 2, 4, and 8 PPUs for the resolutions of VGA, SVGA, XVGA, and HD Video, respectively. It also shows that with 16 PPUs and beyond one can achieve real time motion estimation performance for resolutions that are beyond HD Video.

5.2 Verification Methods

Clock cycle accurate C models were used to verify the correctness of all Verilog design files. Each Verilog module/file had its own corresponding C model, represented as a C function. A main system program written in C was then used to integrate the entire C sub-functions according to the hierarchy in which the Verilog modules were instantiated to build the above motion estimation unit cases. Through this procedure, all Verilog modules and complete motion estimation units could be crosschecked with C program results. The correctness of the C based ME calculations themselves were trusted to be true due to the use of high-level functions that are built into the C libraries, such as C's abs (absolute difference) function. A few cases of C based output was also manually verified to ensure correctness. All of the C models used for verification, along with the Verilog modules, are available for reference in Appendix A of this work.

5.2 Verification Methods

Initial testing and verification of the Register Transfer Level (RTL) Verilog code was performed through wave signal analysis using the ModelSim RTL systems simulation CAD tool. For independent module based testing, custom Verilog test-benches were written to test all possible input combinations to the RTL circuits. These test-benches were then run on the ModelSim Simulator, and their output was visually verified. For the more complicated testing of the overall motion estimation unit, ModelSim was again used to convert the binary output vectors of the RTL ME system into decimal number ASCII files. These ASCII files were then passed to the C based ME program to be crosschecked. Similarly, input test case vectors generated by the C program were also parsed into the ModelSim RTL simulator, to verify that the RTL ME unit does indeed produce the expected output SAD vectors – and thus matching the C vectors as well.

The on-chip double buffer memory units were independently verified in ModelSim. The dual buses that interface the memory system to the motion estimation unit's datapath were selected as the data verification point for functional correctness of the memory. The C models were used to generate both random and known input values to these buses. A Verilog memory test-bench was then created to load the double buffers with these values. The double buffers were then run along with their matching 16-PPU system on ModelSim and their bus output was both manually verified and crosschecked with C based data. In this manner output from the memory system was verified to be providing the correct pixel vectors at the correct expected time points.

A screenshot example of the ModelSim based wave form analysis of the dual-buffer memory system for accuracy is shown in Figure 5.2. It shows the signal transition of all the input control signals and the input test-data vectors to the dual-buffered memory system. Through ModelSim, for this particular example as well as for all other modules, the output response (signal transitions) of the unit according to the input test vectors given was analysed to test and ensure the correct functionality for each of our design units.

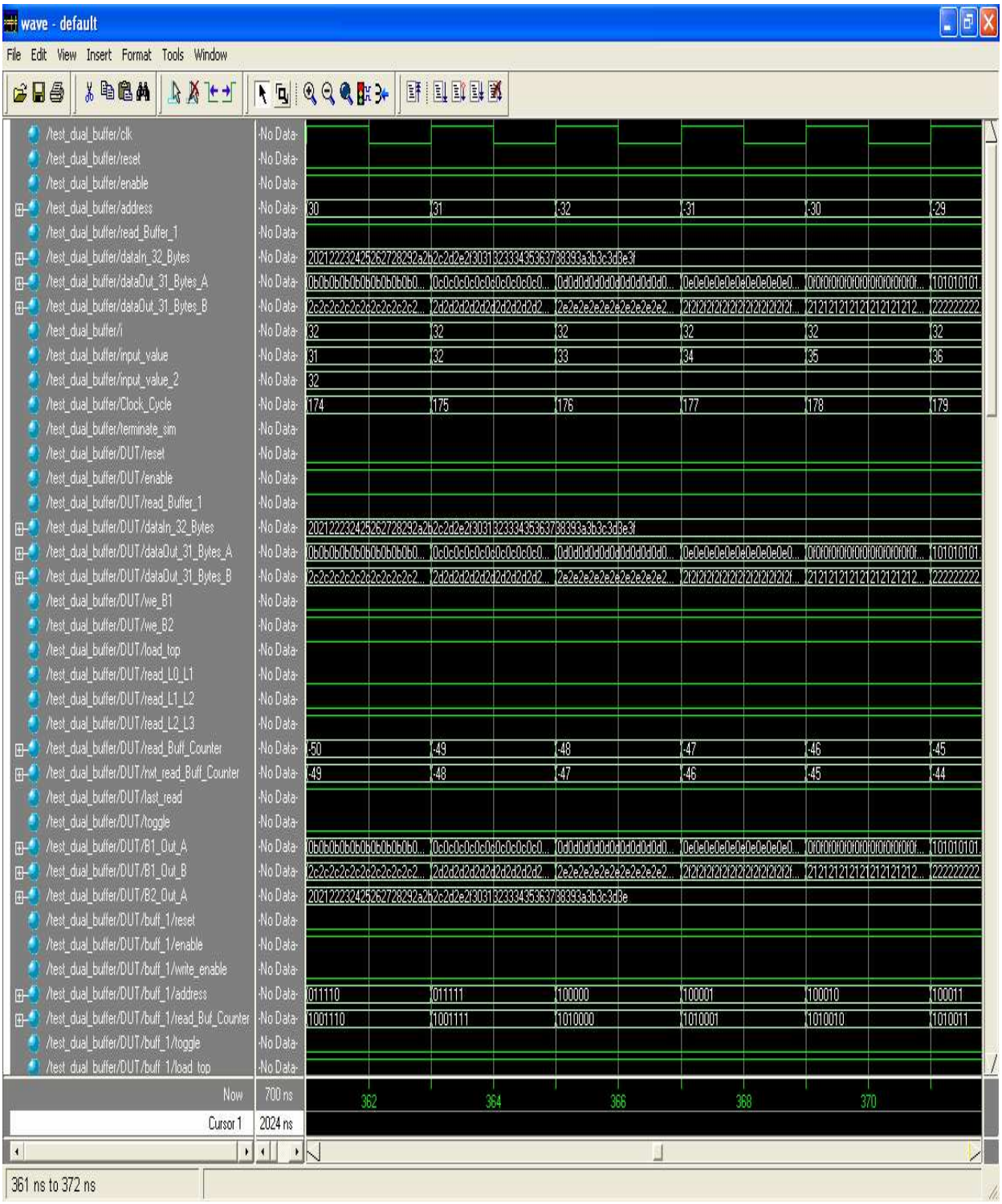


Figure 5.2 ModelSim based Wave Form Analysis

5.3 RTL Synthesis Schematics

This section presents a selection of a few Xilinx ISE 9.2i CAD based screen shots of various PPU design synthesis stages, to illustrate the manner in which the RTL design was implemented onto the XC5VLX330 FPGA device by the CAD tool.

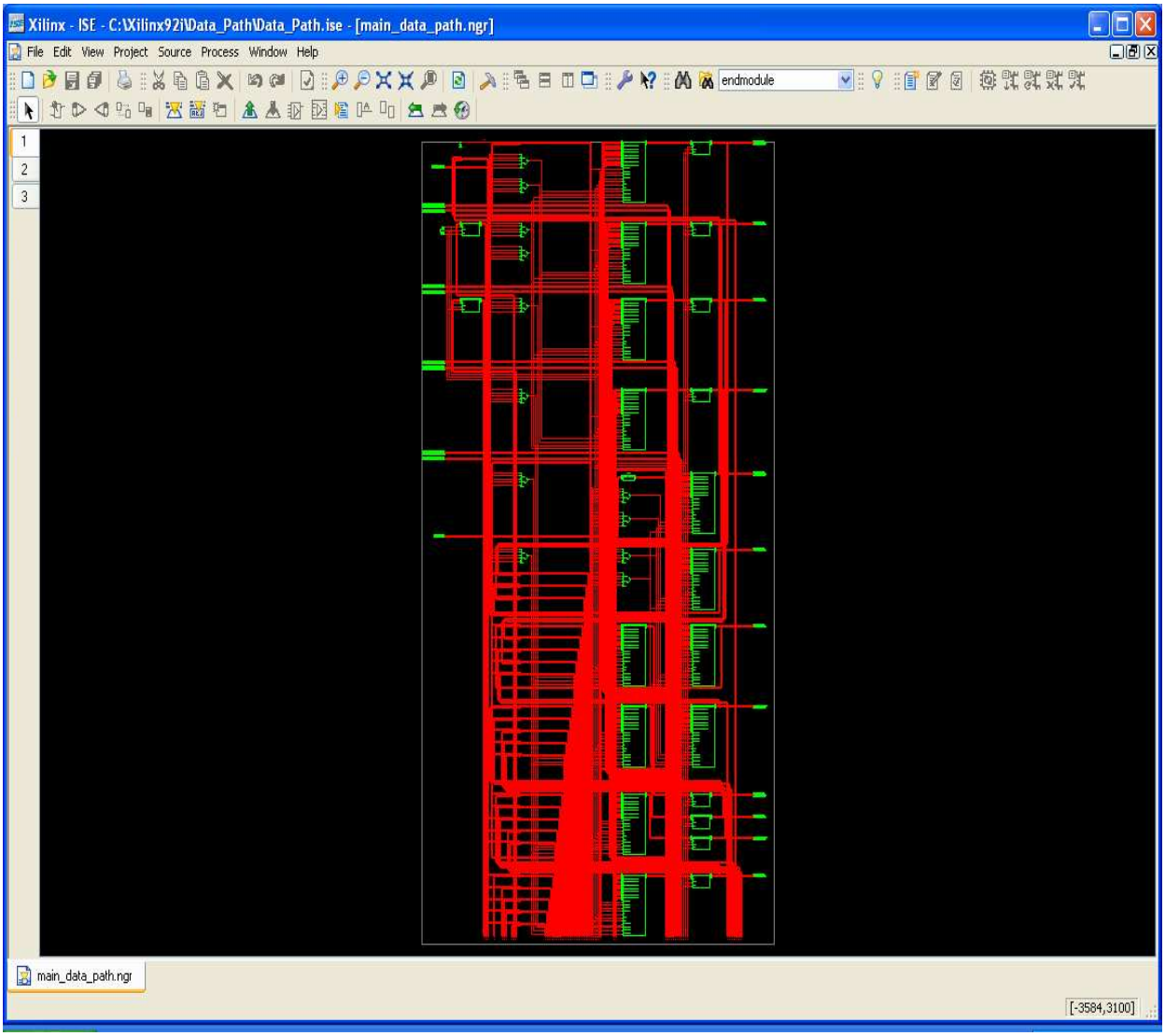


Figure 5.3.1 RTL Schematic view of PPU main Data-Path

Figure 5.3.1 gives a visual indication of the complex data-path routing required for even the single instantiation of a PPU. It shows primarily how the two 16-Byte dual-bus lines are wired to the Four_by_Four_Block modules that make up the 16x16 pipelined array of the PPU. From the routing complexity observed for the single PPU in the figure, we can infer that the routing for a system scaled to 16-PPUs is a non-trivial requirement.

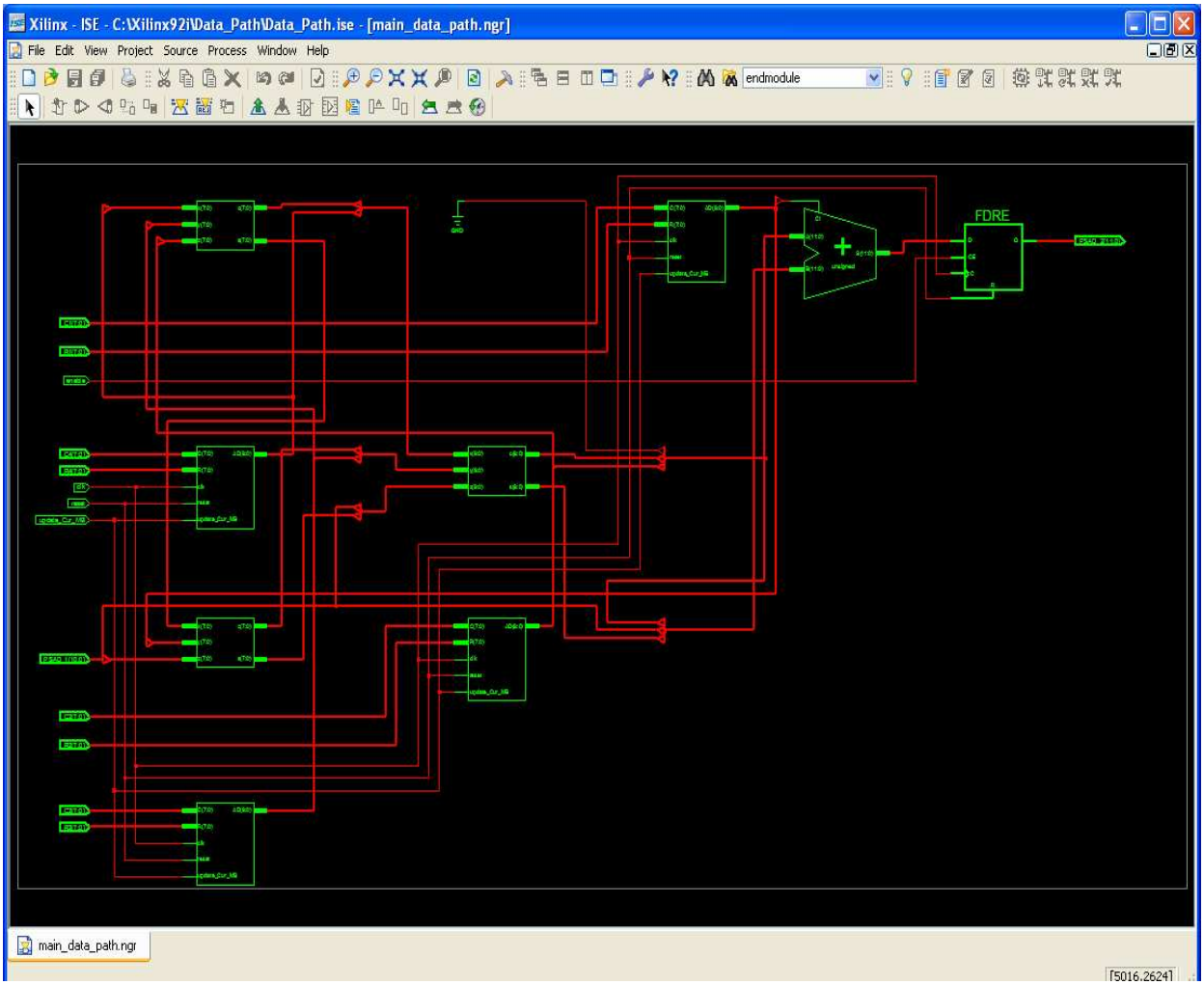


Figure 5.3.2 RTL Schematic view of a Four_by_Four_Block module

Figure 5.3.2 shows the post-synthesis internal logic organization for the Four_by_Four_Block module. Each of the 4 individual Row_Adder modules are depicted by the green rectangular boxes, the first three PSAD registers 0 to 2 can also be recognized as the three smaller square green boxes. All though the final 4th PSAD register #3 was coded in RTL Verilog in the exact manner in which the first three PSAD registers had been coded, it is interesting to note that synthesis has deviated from the RTL code and has placed its own Adder BLE (Basic Logic Element) between the last Row_Adder module and the final PSAD register. This is one of the many ob-

servations through out the schematic analysis of all the other RTL modules, in which independent optimizations via synthesis can be seen.

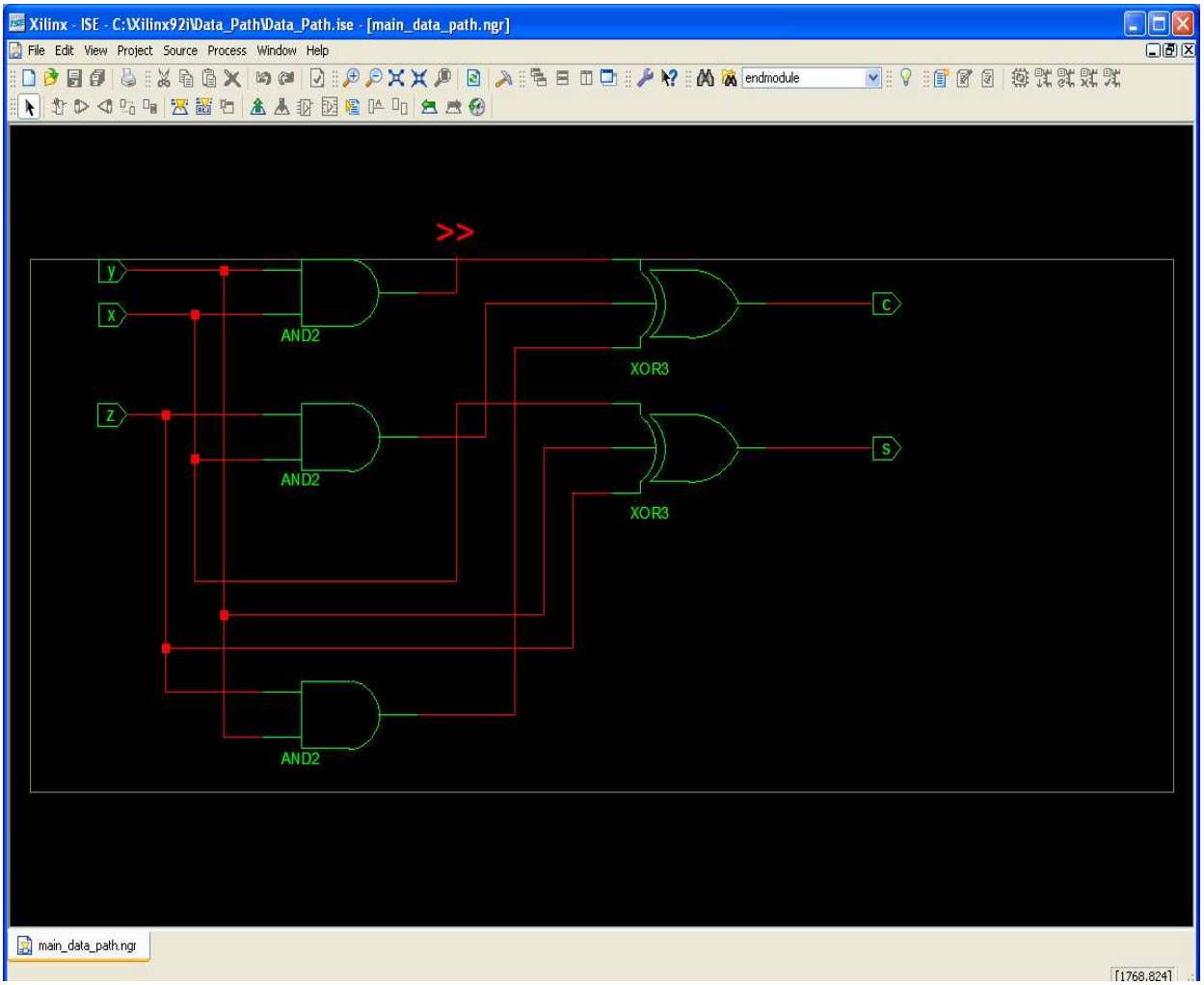


Figure 5.3.3 Carry-Save Full-Adder Circuit

Figure 5.3.3 shows the internal circuitry for the CSA (Carry-Save) full-adder. The CSA modules were the only modules that were coded in structural Verilog as opposed to Functional or RTL Verilog. This was done to ensure that the application specific advantages to using CSA adders in this part of the design, as was discussed in section 4.1, would not be over-rode by the independent synthesis optimizations to use the available FPGA carry-chain fast adders (which in this case

would not actually be faster than the CSA adders). Thus in Figure 5.3.3, it can be visually verified, that the structural coding has been followed by the XST (Xilinx Synthesis Tool) to implement the gate-level circuit that was desired, as opposed to XST following through with its own optimized RTL version.

Detailed synthesis reports for all 6 designs used within this work:

1. Design with 1 PPU
2. Design with 2 PPUs
3. Design with 4 PPUs
4. Design with 8 PPUs
5. Design with 16 PPUs
6. 16-PPU System Design with On-Chip Double-Buffer

are available within the Appendix, and cover the synthesis results for each and every individual Verilog module within those designs as well.

Chapter 6

Conclusions and Future Work

This chapter first presents a concluding summary on the research and development goals that were accomplished within this work. We then compare the results of this work to several existing state of the art designs. Finally, a discussion of future work concludes this chapter and this thesis.

6.1 Concluding Summary

Based on a survey of the present FPGA-based H.264 VBSME architectures ([15]-[16], [22]), the proposed architecture is the first to reach HD-level real time performances. We found that the architecture is able to perform real time (31 fps) H.264 Motion Estimation on 1920x1088 progressive HD video and is capable of being scaled for future higher resolutions. The performance is measured with four reference frames and a search window size of 63 x 48 pixels. When scaled for HD-level performance, the architecture utilizes 77 K LUTs and 18 K DFFs (with 8 processing units), and has a maximum clock frequency of 198 MHz when implemented on a Xilinx XC5VLX330 (Virtex-5) FPGA. Furthermore, the scalability of the architecture makes it suitable

for FPGA-based applications where the upgradability and flexibility of the video encoder are essential requirements.

6.2 Comparative Study

For comparative purposes, the Intellectual Property (IP) core for H.264 Motion Estimation developed by Xilinx [22] is examined in further detail. To the best of the author's knowledge this is the only FPGA implementation that claims to support HD resolution (1920x1088) H.264 ME in real-time at 30 fps. Altera Inc. (partnered with Ateme Inc.) also offers H.264 solutions but since they only develop whole encoders, as opposed to isolated Motion Estimation units, their product is not compared within this work.

Upon initial review, the only drawback to Xilinx's ME IP core relative to our ME unit appears to be that it can not be scaled for lower resolutions. The Xilinx IP offers only 1080 interlaced resolution at 60 fps or 1080 progressive resolution at 30 fps. With the exception of this drawback, it first appears to be a superior design since it runs at a faster clock frequency of 275 MHz compared to our 200 MHz, and it can also be implemented on less expensive FPGA devices such as the ones within the Virtex 4 and Spartan families.

However, upon closer examination it is shown that Xilinx has made algorithmic-level simplifications to reduce the computational complexity demanded from their hardware. The first of which is the use of a non-exhaustive search when comparing candidate referenceBlocks to macroBlocks. In software, predictive methods are often employed to implement fast-searches (i.e.

non-exhaustive) in order to reduce the complexity burden and make real-time ME feasible on non dedicated general processors. Xilinx in their IP core, implements such a fast-search algorithm to only make a 120 comparisons per macroBlock. 120-comparisons relative to the 1536-comparisons required in an exhaustive search is a great reduction in computational complexity and can result in a significant loss in compression quality [18].

Furthermore, although the H.264 standard recommends Multiple Reference Frames to be used during ME, it does not enforce it when granting H.264 labelling rights. Thus Xilinx has chosen to use 1 reference frame only in their ME design. This brings the complexity of their macroBlock-to-referenceBlock comparisons down to 120 (1x120), rather than the 6336 (4x1584) comparisons that are performed within our design. The fact that reproduced video quality is much improved when encoded using multiple reference frames combined with exhaustive searches within each frame is well documented in literature [18]. The VBSME aspect of H.264 is also slightly degraded in the Xilinx implementation since it only supports subBlocks down to 8x4 pixels rather than down to the smallest 4x4 subBlocks supported by the standard. This reduction also contributes to achieving more cost savings in hardware.

6.3 Future Work

The plus side to the Xilinx ME IP core is its impressive minimal area usage. Their design only consumes 3K LUTs compared to the 76K LUTs used for our 8-PPU design. Xilinx is able to achieve such a small LUT count by taking advantage of their on-chip hard DSP blocks. In fact they clearly state that their IP core uses 3K LUTs in conjunction with 26 of their DSP48 Blocks.

The use of available DSP blocks to reduce the LUT count in our designs could be a very practical form of future investigation. Whether the DSP blocks can be efficiently integrated to support the 16x16 systolic-array structure that the Propagate Partial SAD architecture demands is a very complex yet interesting question that should be addressed in the future. Furthermore, deep-pipelining within PPU stages could also lead to higher or equivalent performance with less area usage. Such reduced area solutions would make FPGA platforms for H.264 Motion Estimation even more attractive. It would also give rise to the possibility of having whole encoder solutions implemented on FPGAs, without actually having to compromise on genuine H.264 capabilities.

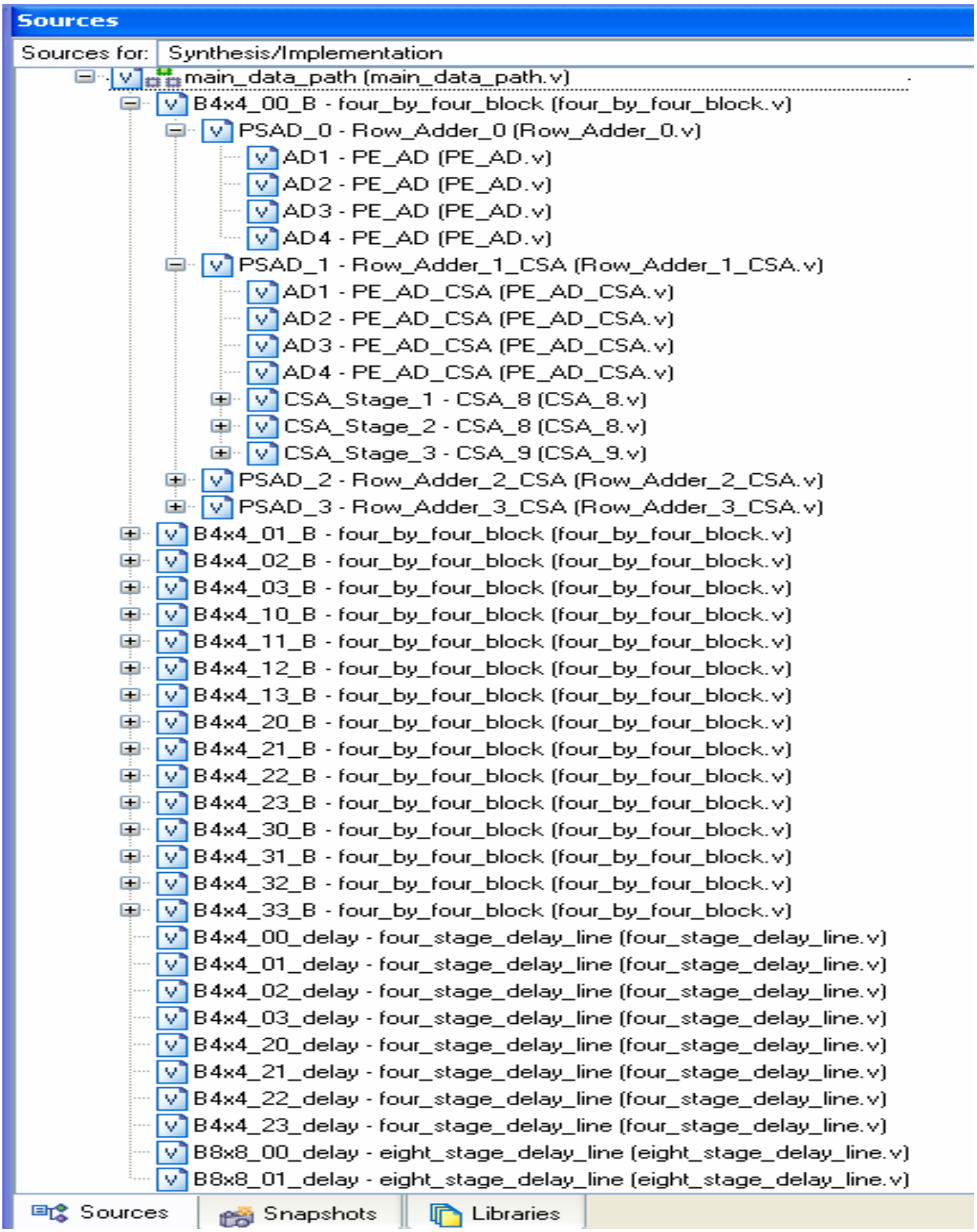
APPENDIX

This Appendix provides all of the Verilog and C code, within their respective files, which were used to compile and synthesise all of the design and test cases that were used within this work. Comments embedded within the code should serve to make the code self-explanatory. Where applicable, CAD based screen-snapshots have been provided to illustrate the design hierarchy of both the Verilog Modules and the C Functions. During the digital design process of Verilog RTL code development the following resources were used for guidance and assistance – [23], [24] to [27].

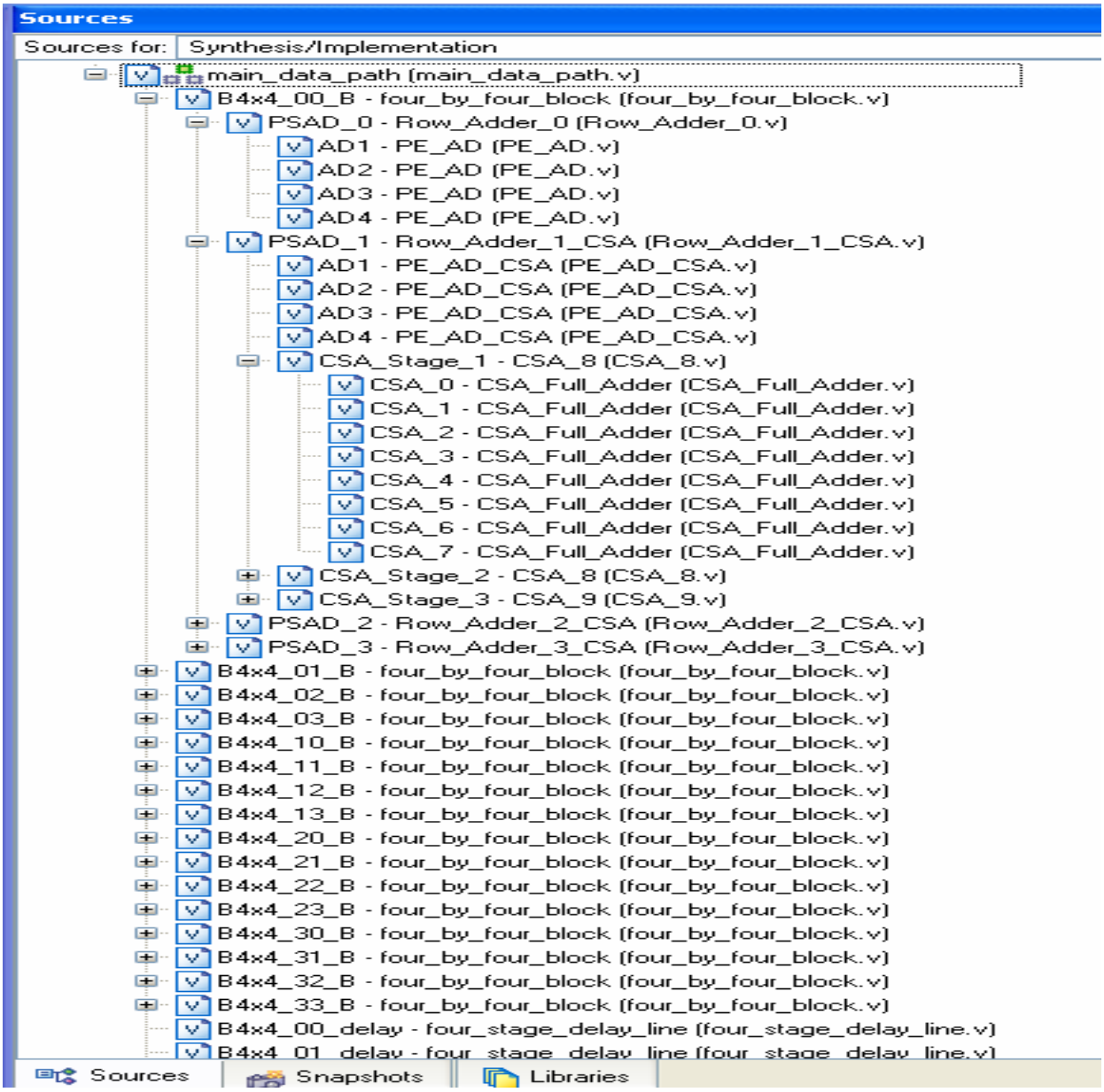
7.1 Verilog Code Files

The following screenshots of the Xilinx ISE Verilog source code tree shows the file/module hierarchy for the main_data_path file which instantiates a single PPU.

7.1 Verilog Code Files

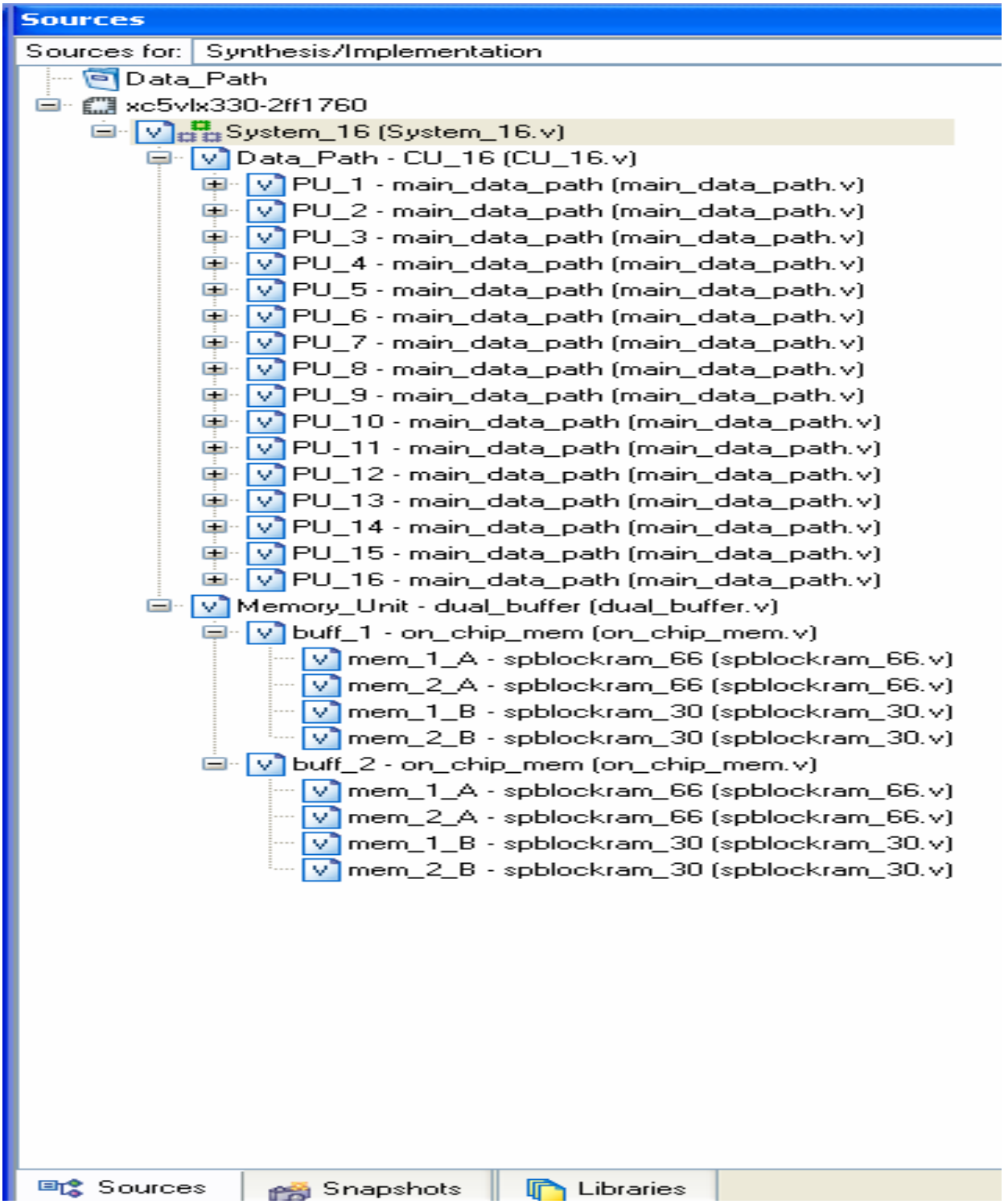


The main_data_path file hierarchy shown again below, with the CSA_Stage_1 module fully expanded.



7.1 Verilog Code Files

The Verilog file hierarchy for a 16-PPU Motion Estimation Unit implemented with on-chip double buffered memory.



Verilog Files

The general order of the Verilog files/modules given below is presented in the bottom to top module manner in which they were integrated during implementation. There are 17 Verilog design files and 7 Verilog Test-Bench files in total.

```
/************************
 * Theepan Moorthy          *
 * Ryerson University       *
 * Copyright July 2007      *
 * PE_AD.v                  *
 ************************/

// This Module takes two unsigned 8-bit operands
// and produces their 8-bit unsigned Absolute Difference

module PE_AD (  // Absolute Difference Processing Element (PE_AD)

clk,                    // Input clock signal
reset,                  // Input reset signal
update_Cur_MB,  // Input enable signal to load new Current MB data
C,                      // Input PEL from the Current Frame
R,                      // Input PEL from the Reference Frame

AD                      // Output Absolute Difference
);

//------------- Input Ports -------------------------//

input clk;
input reset;
input update_Cur_MB;
input [7:0] C;
input [7:0] R;

//------------- Output Ports ------------------------//

output [7:0] AD;
//------------- Internal Variables ------------------------//

reg  [7:0] C_Registered;
wire [7:0] C_NOT;
wire [8:0] Nine_Bit_Sum;
wire [7:0] XOR_Output;

//------------- Implementation ---------------------//
always @(posedge clk) begin
      if     (reset)        C_Registered <= 8'b0 ;
      else if (update_Cur_MB) C_Registered <= C;
```

```
end

assign C_NOT = ~C_Registered;

assign Nine_Bit_Sum = {1'b0,C_NOT} + {1'b0,R};

assign XOR_Output = Nine_Bit_Sum[7:0] ^ {8{~Nine_Bit_Sum[8]}};

assign AD = XOR_Output + { {7{1'b0}} , Nine_Bit_Sum[8]};

endmodule
```

```
/************************
 * Theepan Moorthy          *
 * Ryerson University       *
 * Copyright July 2007      *
 * PE_AD_CSA.v              *
 ************************/

// This Module takes two unsigned 8-bit operands
// and produces their 8-bit unsigned Absolute Difference

module PE_AD_CSA ( // Absolute Difference Processing Element (PE_AD)

clk,                    // Input clock signal
reset,                  // Input reset signal
update_Cur_MB,  // Input enable signal to load new Current MB data
C,                      // Input PEL from the Current Frame
R,                      // Input PEL from the Reference Frame

AD                     // Output Absolute Difference
);

//------------- Input Ports -------------------------//

input clk;
input reset;
input update_Cur_MB;
input [7:0] C;
input [7:0] R;


//------------- Output Ports ------------------------//

output [8:0] AD;


//------------- Internal Variables ------------------------//

reg  [7:0] C_Registered;
wire [7:0] C_NOT;
wire [8:0] Nine_Bit_Sum;
wire [7:0] XOR_Output;


//------------- Implementation ---------------------//

always @(posedge clk) begin
      if     (reset)        C_Registered <= 8'b0 ;
      else if (update_Cur_MB) C_Registered <= C;
end

assign C_NOT = ~C_Registered;

assign Nine_Bit_Sum = {1'b0,C_NOT} + {1'b0,R};
```

```
assign XOR_Output = Nine_Bit_Sum[7:0] ^ {8{~Nine_Bit_Sum[8]}};

assign AD = {Nine_Bit_Sum[8], XOR_Output};

endmodule
```

```
/************************
 * Theepan Moorthy      *
 * Ryerson University   *
 * Copyright July 2007  *
 * CSA_Full_Adder.v     *
 ************************/

// This Module is a 3-bit Full Adder Circuit
// it will be instantiated 8 times to implement
// a full CSA block, that will add 3 8-bit vectors

module CSA_Full_Adder (

x, // Input, 1st Operand Bit
y, // Input, 2nd Operand Bit
z, // Input, 3rd Operand Bit
// (in a traditional Full Adder Circuit this 3rd z bit
//  is regarded as the Carry In bit)

s, // Output, Sum Bit
c  // Output, Carry Bit
// (in a traditional Full Adder Circuit this Carry bit
//  is regarded as the Carry Out bit)
);


//------------- Input Ports -------------------------//

input x, y, z;


//------------- Output Ports ------------------------//

output s, c;


//------------- Implementation ----------------------//

assign s = x ^ y ^ z; // XOR x, y, and z

assign c = (x&y) ^ (x&z) ^ (y&z);

endmodule
```

```
/************************
 * Theepan Moorthy      *
 * Ryerson University   *
 * Copyright July 2007  *
 * CSA_8.v              *
 ************************/

// This Module is an 8-bit Carry Save Adder Block

module CSA_8 (

x, // Input, 1st 8-bit Operand
y, // Input, 2nd 8-bit Operand
z, // Input, 3rd 8-bit Operand

s, // Output, 8-bit Sum Vector
c  // Output, 8-bit Carry Vector
);


//------------- Input Ports -------------------------//

input [7:0] x;
input [7:0] y;
input [7:0] z;


//------------- Output Ports ------------------------//

output [7:0] s;
output [7:0] c;


//------------- Instantiations ----------------------//

CSA_Full_Adder CSA_0 (.x(x[0]),
                      .y(y[0]),
                      .z(z[0]),
                      .s(s[0]),
                      .c(c[0])
                      );

CSA_Full_Adder CSA_1 (.x(x[1]),
                      .y(y[1]),
                      .z(z[1]),
                      .s(s[1]),
                      .c(c[1])
                      );

CSA_Full_Adder CSA_2 (.x(x[2]),
                      .y(y[2]),
                      .z(z[2]),
                      .s(s[2]),
                      .c(c[2])
```

```
        );

CSA_Full_Adder CSA_3 (.x(x[3]),
        .y(y[3]),
        .z(z[3]),
        .s(s[3]),
        .c(c[3])
        );

CSA_Full_Adder CSA_4 (.x(x[4]),
        .y(y[4]),
        .z(z[4]),
        .s(s[4]),
        .c(c[4])
        );

CSA_Full_Adder CSA_5 (.x(x[5]),
        .y(y[5]),
        .z(z[5]),
        .s(s[5]),
        .c(c[5])
        );

CSA_Full_Adder CSA_6 (.x(x[6]),
        .y(y[6]),
        .z(z[6]),
        .s(s[6]),
        .c(c[6])
        );
CSA_Full_Adder CSA_7 (.x(x[7]),
        .y(y[7]),
        .z(z[7]),
        .s(s[7]),
        .c(c[7])
        );
endmodule
```

```
/************************
 * Theepan Moorthy      *
 * Ryerson University   *
 * Copyright July 2007  *
 * CSA_9.v              *
 ************************/

// This Module is an 9-bit Carry Save Adder Block

module CSA_9 (

x, // Input, 1st 9-bit Operand
y, // Input, 2nd 9-bit Operand
z, // Input, 3rd 9-bit Operand

s, // Output, 9-bit Sum Vector
c  // Output, 9-bit Carry Vector
);


//------------- Input Ports -------------------------//

input [8:0] x;
input [8:0] y;
input [8:0] z;


//------------- Output Ports ------------------------//

output [8:0] s;
output [8:0] c;


//------------- Instantiations ----------------------//

CSA_Full_Adder CSA_0 (.x(x[0]),
                      .y(y[0]),
                      .z(z[0]),
                      .s(s[0]),
                      .c(c[0])
                      );

CSA_Full_Adder CSA_1 (.x(x[1]),
                      .y(y[1]),
                      .z(z[1]),
                      .s(s[1]),
                      .c(c[1])
                      );

CSA_Full_Adder CSA_2 (.x(x[2]),
                      .y(y[2]),
                      .z(z[2]),
                      .s(s[2]),
                      .c(c[2])
```

```
        );

CSA_Full_Adder CSA_3 (.x(x[3]),
        .y(y[3]),
        .z(z[3]),
        .s(s[3]),
        .c(c[3])
        );

CSA_Full_Adder CSA_4 (.x(x[4]),
        .y(y[4]),
        .z(z[4]),
        .s(s[4]),
        .c(c[4])
        );

CSA_Full_Adder CSA_5 (.x(x[5]),
        .y(y[5]),
        .z(z[5]),
        .s(s[5]),
        .c(c[5])
        );

CSA_Full_Adder CSA_6 (.x(x[6]),
        .y(y[6]),
        .z(z[6]),
        .s(s[6]),
        .c(c[6])
        );

CSA_Full_Adder CSA_7 (.x(x[7]),
        .y(y[7]),
        .z(z[7]),
        .s(s[7]),
        .c(c[7])
        );

CSA_Full_Adder CSA_8 (.x(x[8]),
        .y(y[8]),
        .z(z[8]),
        .s(s[8]),
        .c(c[8])
        );

endmodule
```

```
/************************
 * Theepan Moorthy      *
 * Ryerson University   *
 * Copyright July 2007  *
 * Row_Adder_0.v        *
 ************************/

// This Module takes the Absolute Differences of 4 Processing Elements
// and produces their SUM to be used as a partial SAD in Row 0 (within
// the complete architecture)

module Row_Adder_0 (

clk,                  // Input clock signal
reset,                // Input reset signal
enable,               // Input enable signal
update_Cur_MB,  // Input enable signal to load new Current MB data
C1,       // Input PEL1 from the Current Frame
C2,       // Input PEL2 from the Current Frame
C3,       // Input PEL3 from the Current Frame
C4,       // Input PEL4 from the Current Frame
R1,       // Input PEL1 from the Reference Frame
R2,       // Input PEL2 from the Reference Frame
R3,       // Input PEL3 from the Reference Frame
R4,       // Input PEL4 from the Reference Frame

PSAD_0          // Output 4 PE based Partial Sum of Absolute Differences
);


//------------- Input Ports ------------------------//

input clk;
input reset;
input enable;
input update_Cur_MB;
input [7:0] C1;
input [7:0] C2;
input [7:0] C3;
input [7:0] C4;
input [7:0] R1;
input [7:0] R2;
input [7:0] R3;
input [7:0] R4;


//------------- Output Ports ------------------------//

output reg [9:0] PSAD_0;


//------------- Internal Variables ------------------------//

wire [7:0] AD1_Out;
```

```
wire [7:0] AD2_Out;
wire [7:0] AD3_Out;
wire [7:0] AD4_Out;
wire [8:0] Nine_Bit_Sum_1;
wire [8:0] Nine_Bit_Sum_2;


//------------- Instantiations ------------------------//

PE_AD AD1 (.clk(clk),
        .reset(reset),
        .update_Cur_MB(update_Cur_MB),
        .C(C1),
        .R(R1),
        .AD(AD1_Out)
        );

PE_AD AD2 (.clk(clk),
        .reset(reset),
        .update_Cur_MB(update_Cur_MB),
        .C(C2),
        .R(R2),
        .AD(AD2_Out)
        );

PE_AD AD3 (.clk(clk),
        .reset(reset),
        .update_Cur_MB(update_Cur_MB),
        .C(C3),
        .R(R3),
        .AD(AD3_Out)
        );

PE_AD AD4 (.clk(clk),
        .reset(reset),
        .update_Cur_MB(update_Cur_MB),
        .C(C4),
        .R(R4),
        .AD(AD4_Out)
        );


//------------- Implementation ----------------------//

assign Nine_Bit_Sum_1 = {1'b0,AD1_Out} + {1'b0,AD2_Out};

assign Nine_Bit_Sum_2 = {1'b0,AD3_Out} + {1'b0,AD4_Out};

always @(posedge clk) begin
     if     (reset)  PSAD_0 <= 10'b0 ;
     else if (enable) PSAD_0 <= {1'b0,Nine_Bit_Sum_1} + {1'b0,Nine_Bit_Sum_2};
end

endmodule
```

```
/************************
 * Theepan Moorthy      *
 * Ryerson University   *
 * Copyright July 2007  *
 * Row_Adder_1_CSA.v    *
 ************************/

// This Module takes the Absolute Differences of 4 Processing Elements
// and produces their SUM to be used as a partial SAD in Row 1 (within
// the complete architecture)

module Row_Adder_1_CSA (

clk,            // Input clock signal
reset,          // Input reset signal
enable,         // Input enable signal
update_Cur_MB, // Input enable signal to load new Current MB data
C1,       // Input PEL1 from the Current Frame
C2,       // Input PEL2 from the Current Frame
C3,       // Input PEL3 from the Current Frame
C4,       // Input PEL4 from the Current Frame
R1,       // Input PEL1 from the Reference Frame
R2,       // Input PEL2 from the Reference Frame
R3,       // Input PEL3 from the Reference Frame
R4,       // Input PEL4 from the Reference Frame
PSAD_0,          // Input Partial SAD from Row 0

PSAD_1          // Output Partial SAD Row 1
);


//------------- Input Ports -------------------------//

input clk;
input reset;
input enable;
input update_Cur_MB;
input [7:0] C1;
input [7:0] C2;
input [7:0] C3;
input [7:0] C4;
input [7:0] R1;
input [7:0] R2;
input [7:0] R3;
input [7:0] R4;
input [9:0] PSAD_0;


//------------- Output Ports -------------------------//

output reg [10:0] PSAD_1;
```

```
//------------- Internal Variables ------------------------//

wire [8:0] AD1_Out;
wire [8:0] AD2_Out;
wire [8:0] AD3_Out;
wire [8:0] AD4_Out;
wire [7:0] CSA_Stage_1_S;
wire [7:0] CSA_Stage_1_C;
wire [7:0] CSA_Stage_2_S;
wire [7:0] CSA_Stage_2_C;
wire [8:0] CSA_Stage_3_S;
wire [8:0] CSA_Stage_3_C;


//------------- Instantiations ------------------------//

PE_AD_CSA AD1 (.clk(clk),
            .reset(reset),
            .update_Cur_MB(update_Cur_MB),
            .C(C1),
            .R(R1),
            .AD(AD1_Out)
            );

PE_AD_CSA AD2 (.clk(clk),
            .reset(reset),
            .update_Cur_MB(update_Cur_MB),
            .C(C2),
            .R(R2),
            .AD(AD2_Out)
            );

PE_AD_CSA AD3 (.clk(clk),
            .reset(reset),
            .update_Cur_MB(update_Cur_MB),
            .C(C3),
            .R(R3),
            .AD(AD3_Out)
            );

PE_AD_CSA AD4 (.clk(clk),
            .reset(reset),
            .update_Cur_MB(update_Cur_MB),
            .C(C4),
            .R(R4),
            .AD(AD4_Out)
            );

CSA_8 CSA_Stage_1 (.x(AD4_Out[7:0]),
              .y(AD3_Out[7:0]),
              .z(AD2_Out[7:0]),
              .s(CSA_Stage_1_S),
              .c(CSA_Stage_1_C)
              );
```

```
CSA_8 CSA_Stage_2 (.x(CSA_Stage_1_S),
                .y(AD1_Out[7:0]),
                .z(PSAD_0[7:0]),
                .s(CSA_Stage_2_S),
                .c(CSA_Stage_2_C)
                );

CSA_9 CSA_Stage_3 (.x({CSA_Stage_1_C,AD4_Out[8]}),
                .y({CSA_Stage_2_C,AD3_Out[8]}),
                .z({PSAD_0[8],CSA_Stage_2_S}),
                .s(CSA_Stage_3_S),
                .c(CSA_Stage_3_C)
                );


//------------- Implementation ----------------------//

always @(posedge clk) begin
      if      (reset)  PSAD_1 <= 11'b0 ;
      else if (enable) PSAD_1 <= ( {1'b0,{CSA_Stage_3_C,AD2_Out[8]}} + {1'b0,{PSAD_0[9],CSA_Stage_3_S}}
)
                              + AD1_Out[8];
end

endmodule
```

```
/*************************
 * Theepan Moorthy       *
 * Ryerson University    *
 * Copyright August 2007 *
 * Row_Adder_2_CSA.v     *
 *************************/

// This Module takes the Absolute Differences of 4 Processing Elements
// and produces their SUM to be used as a partial SAD in Row 2 (within
// the complete architecture)

module Row_Adder_2_CSA (

clk,            // Input clock signal
reset,          // Input reset signal
enable,         // Input enable signal
update_Cur_MB, // Input enable signal to load new Current MB data
C1,       // Input PEL1 from the Current Frame
C2,       // Input PEL2 from the Current Frame
C3,       // Input PEL3 from the Current Frame
C4,       // Input PEL4 from the Current Frame
R1,       // Input PEL1 from the Reference Frame
R2,       // Input PEL2 from the Reference Frame
R3,       // Input PEL3 from the Reference Frame
R4,       // Input PEL4 from the Reference Frame
PSAD_1,          // Input Partial SAD from Row 0

PSAD_2          // Output Partial SAD Row 1
);


//------------- Input Ports ------------------------//

input clk;
input reset;
input enable;
input update_Cur_MB;
input [7:0]  C1;
input [7:0]  C2;
input [7:0]  C3;
input [7:0]  C4;
input [7:0]  R1;
input [7:0]  R2;
input [7:0]  R3;
input [7:0]  R4;
input [10:0] PSAD_1;


//------------- Output Ports ------------------------//

output reg [11:0] PSAD_2;
```

```
//------------- Internal Variables ------------------------//

wire [8:0] AD1_Out;
wire [8:0] AD2_Out;
wire [8:0] AD3_Out;
wire [8:0] AD4_Out;

wire [7:0] CSA_Stage_1_S;
wire [7:0] CSA_Stage_1_C;
wire [7:0] CSA_Stage_2_S;
wire [7:0] CSA_Stage_2_C;
wire [8:0] CSA_Stage_3_S;
wire [8:0] CSA_Stage_3_C;


//------------- Instantiations ------------------------//

PE_AD_CSA AD1 (.clk(clk),
        .reset(reset),
        .update_Cur_MB(update_Cur_MB),
        .C(C1),
        .R(R1),
        .AD(AD1_Out)
        );

PE_AD_CSA AD2 (.clk(clk),
        .reset(reset),
        .update_Cur_MB(update_Cur_MB),
        .C(C2),
        .R(R2),
        .AD(AD2_Out)
        );

PE_AD_CSA AD3 (.clk(clk),
        .reset(reset),
        .update_Cur_MB(update_Cur_MB),
        .C(C3),
        .R(R3),
        .AD(AD3_Out)
        );

PE_AD_CSA AD4 (.clk(clk),
        .reset(reset),
        .update_Cur_MB(update_Cur_MB),
        .C(C4),
        .R(R4),
        .AD(AD4_Out)
        );

CSA_8 CSA_Stage_1 (.x(AD4_Out[7:0]),
          .y(AD3_Out[7:0]),
          .z(AD2_Out[7:0]),
          .s(CSA_Stage_1_S),
          .c(CSA_Stage_1_C)
```

```
            );

CSA_8 CSA_Stage_2 (.x(CSA_Stage_1_S),
            .y(AD1_Out[7:0]),
            .z(PSAD_1[7:0]),
            .s(CSA_Stage_2_S),
            .c(CSA_Stage_2_C)
            );

CSA_9 CSA_Stage_3 (.x({CSA_Stage_1_C,AD4_Out[8]}),
            .y({CSA_Stage_2_C,AD3_Out[8]}),
            .z({PSAD_1[8],CSA_Stage_2_S}),
            .s(CSA_Stage_3_S),
            .c(CSA_Stage_3_C)
            );


//---------- Implementation ----------------------//

always @(posedge clk) begin
    if     (reset)  PSAD_2 <= 0 ;
    else if (enable) PSAD_2 <= ( {2'b00,{CSA_Stage_3_C,AD2_Out[8]}} +
{1'b0,{PSAD_1[10:9],CSA_Stage_3_S}} )
                        + AD1_Out[8];
end

endmodule
```

```
/*************************
 * Theepan Moorthy       *
 * Ryerson University    *
 * Copyright August 2007 *
 * Row_Adder_3_CSA.v     *
 *************************/

// This Module takes the Absolute Differences of 4 Processing Elements
// and produces their SUM to be used as a partial SAD in Row 3 (within
// the complete architecture)

module Row_Adder_3_CSA (

clk,        // Input clock signal
reset,      // Input reset signal
enable,     // Input enable signal
update_Cur_MB, // Input enable signal to load new Current MB data
C1,         // Input PEL1 from the Current Frame
C2,         // Input PEL2 from the Current Frame
C3,         // Input PEL3 from the Current Frame
C4,         // Input PEL4 from the Current Frame
R1,         // Input PEL1 from the Reference Frame
R2,         // Input PEL2 from the Refference Frame
R3,         // Input PEL3 from the Reference Frame
R4,         // Input PEL4 from the Reference Frame
PSAD_2,     // Input Partial SAD from Row 0

PSAD_3      // Output Partial SAD Row 1
);


//------------- Input Ports ------------------------//

input clk;
input reset;
input enable;
input update_Cur_MB;
input [7:0]  C1;
input [7:0]  C2;
input [7:0]  C3;
input [7:0]  C4;
input [7:0]  R1;
input [7:0]  R2;
input [7:0]  R3;
input [7:0]  R4;
input [11:0] PSAD_2;


//------------- Output Ports ------------------------//

output reg [12:0] PSAD_3;
```

```
//------------- Internal Variables ------------------------//

wire [8:0] AD1_Out;
wire [8:0] AD2_Out;
wire [8:0] AD3_Out;
wire [8:0] AD4_Out;

wire [7:0] CSA_Stage_1_S;
wire [7:0] CSA_Stage_1_C;
wire [7:0] CSA_Stage_2_S;
wire [7:0] CSA_Stage_2_C;
wire [8:0] CSA_Stage_3_S;
wire [8:0] CSA_Stage_3_C;


//------------- Instantiations ------------------------//

PE_AD_CSA AD1 (.clk(clk),
        .reset(reset),
        .update_Cur_MB(update_Cur_MB),
        .C(C1),
        .R(R1),
        .AD(AD1_Out)
        );

PE_AD_CSA AD2 (.clk(clk),
        .reset(reset),
        .update_Cur_MB(update_Cur_MB),
        .C(C2),
        .R(R2),
        .AD(AD2_Out)
        );

PE_AD_CSA AD3 (.clk(clk),
        .reset(reset),
        .update_Cur_MB(update_Cur_MB),
        .C(C3),
        .R(R3),
        .AD(AD3_Out)
        );

PE_AD_CSA AD4 (.clk(clk),
        .reset(reset),
        .update_Cur_MB(update_Cur_MB),
        .C(C4),
        .R(R4),
        .AD(AD4_Out)
        );

CSA_8 CSA_Stage_1 (.x(AD4_Out[7:0]),
          .y(AD3_Out[7:0]),
          .z(AD2_Out[7:0]),
          .s(CSA_Stage_1_S),
          .c(CSA_Stage_1_C)
```

```
            );

CSA_8 CSA_Stage_2 (.x(CSA_Stage_1_S),
            .y(AD1_Out[7:0]),
            .z(PSAD_2[7:0]),
            .s(CSA_Stage_2_S),
            .c(CSA_Stage_2_C)
            );

CSA_9 CSA_Stage_3 (.x({CSA_Stage_1_C,AD4_Out[8]}),
            .y({CSA_Stage_2_C,AD3_Out[8]}),
            .z({PSAD_2[8],CSA_Stage_2_S}),
            .s(CSA_Stage_3_S),
            .c(CSA_Stage_3_C)
            );


//------------- Implementation ----------------------//

always @(posedge clk) begin
    if     (reset)  PSAD_3 <= 0 ;
    else if (enable) PSAD_3 <= ( {3'b000,{CSA_Stage_3_C,AD2_Out[8]}} +
{1'b0,{PSAD_2[11:9],CSA_Stage_3_S}} )
                        + AD1_Out[8];
end

endmodule
```

```
/**************************
 * Theepan Moorthy        *
 * Ryerson University     *
 * Copyright October 2007 *
 * four_by_four_block.v   *
 **************************/

// This Module is a
// 4 Processing Elements x 4 Processing Elements
// PSAD block with 16 PEs in total

module four_by_four_block (

clk,                    // Input clock signal
reset,                  // Input reset signal
enable,             // Input enable signal

update_Row_0,               // Input enable signal to load new Current MB data
update_Row_1,               // Input enable signal to load new Current MB data
update_Row_2,               // Input enable signal to load new Current MB data
update_Row_3,               // Input enable signal to load new Current MB data

bus_Select_0,               // Input Select to Control each Row's Selection
bus_Select_1,               // from the following Dual Bus Lines:
bus_Select_2,
bus_Select_3,

broadcast_column_1_8bits_A, // Input 8-bits for broadcast bus #1_A
broadcast_column_2_8bits_A, // Input 8-bits for broadcast bus #2_A
broadcast_column_3_8bits_A, // Input 8-bits for broadcast bus #3_A
broadcast_column_4_8bits_A, // Input 8-bits for broadcast bus #4_A

broadcast_column_1_8bits_B, // Input 8-bits for broadcast bus #1_B
broadcast_column_2_8bits_B, // Input 8-bits for broadcast bus #2_B
broadcast_column_3_8bits_B, // Input 8-bits for broadcast bus #3_B
broadcast_column_4_8bits_B, // Input 8-bits for broadcast bus #4_B

PSAD_Out  // Output Partial SAD

);


//------------- Input Ports -------------------------//

input clk;
input reset;
input enable;
input update_Row_0;
input update_Row_1;
input update_Row_2;
input update_Row_3;
input bus_Select_0;
input bus_Select_1;
input bus_Select_2;
```

```
input bus_Select_3;
input [7:0] broadcast_column_1_8bits_A;
input [7:0] broadcast_column_2_8bits_A;
input [7:0] broadcast_column_3_8bits_A;
input [7:0] broadcast_column_4_8bits_A;
input [7:0] broadcast_column_1_8bits_B;
input [7:0] broadcast_column_2_8bits_B;
input [7:0] broadcast_column_3_8bits_B;
input [7:0] broadcast_column_4_8bits_B;

//------------- Output Ports ------------------------//

output [12:0] PSAD_Out;


//------------- Internal Wires ------------------------//

wire [9:0]  Row_0_Out;
wire [10:0] Row_1_Out;
wire [11:0] Row_2_Out;

wire [7:0] broadcast_column_1_0, broadcast_column_2_0, broadcast_column_3_0, broadcast_column_4_0;
wire [7:0] broadcast_column_1_1, broadcast_column_2_1, broadcast_column_3_1, broadcast_column_4_1;
wire [7:0] broadcast_column_1_2, broadcast_column_2_2, broadcast_column_3_2, broadcast_column_4_2;
wire [7:0] broadcast_column_1_3, broadcast_column_2_3, broadcast_column_3_3, broadcast_column_4_3;

//------------- Implementation ------------------------//

// controled selection of the Dual Bus Lines
// during operation for each of the Rows
assign broadcast_column_1_0 = (bus_Select_0)? broadcast_column_1_8bits_B : broadcast_column_1_8bits_A;
assign broadcast_column_2_0 = (bus_Select_0)? broadcast_column_2_8bits_B : broadcast_column_2_8bits_A;
assign broadcast_column_3_0 = (bus_Select_0)? broadcast_column_3_8bits_B : broadcast_column_3_8bits_A;
assign broadcast_column_4_0 = (bus_Select_0)? broadcast_column_4_8bits_B : broadcast_column_4_8bits_A;
assign broadcast_column_1_1 = (bus_Select_1)? broadcast_column_1_8bits_B : broadcast_column_1_8bits_A;
assign broadcast_column_2_1 = (bus_Select_1)? broadcast_column_2_8bits_B : broadcast_column_2_8bits_A;
assign broadcast_column_3_1 = (bus_Select_1)? broadcast_column_3_8bits_B : broadcast_column_3_8bits_A;
assign broadcast_column_4_1 = (bus_Select_1)? broadcast_column_4_8bits_B : broadcast_column_4_8bits_A;
assign broadcast_column_1_2 = (bus_Select_2)? broadcast_column_1_8bits_B : broadcast_column_1_8bits_A;
assign broadcast_column_2_2 = (bus_Select_2)? broadcast_column_2_8bits_B : broadcast_column_2_8bits_A;
assign broadcast_column_3_2 = (bus_Select_2)? broadcast_column_3_8bits_B : broadcast_column_3_8bits_A;
assign broadcast_column_4_2 = (bus_Select_2)? broadcast_column_4_8bits_B : broadcast_column_4_8bits_A;
assign broadcast_column_1_3 = (bus_Select_3)? broadcast_column_1_8bits_B : broadcast_column_1_8bits_A;
assign broadcast_column_2_3 = (bus_Select_3)? broadcast_column_2_8bits_B : broadcast_column_2_8bits_A;
assign broadcast_column_3_3 = (bus_Select_3)? broadcast_column_3_8bits_B : broadcast_column_3_8bits_A;
assign broadcast_column_4_3 = (bus_Select_3)? broadcast_column_4_8bits_B : broadcast_column_4_8bits_A;


//------------- Instantiations ------------------------//

Row_Adder_0 PSAD_0 (
                .clk(clk),
                 .reset(reset),
                 .enable(enable),
```

```
                .update_Cur_MB(update_Row_0),
                .C1(broadcast_column_1_8bits_A),
                .C2(broadcast_column_2_8bits_A),
                .C3(broadcast_column_3_8bits_A),
                .C4(broadcast_column_4_8bits_A),
                .R1(broadcast_column_1_0),
                .R2(broadcast_column_2_0),
                .R3(broadcast_column_3_0),
                .R4(broadcast_column_4_0),
                .PSAD_0(Row_0_Out)
                );

Row_Adder_1_CSA PSAD_1 (
                .clk(clk),
                 .reset(reset),
                 .enable(enable),
                 .update_Cur_MB(update_Row_1),
                .C1(broadcast_column_1_8bits_A),
                .C2(broadcast_column_2_8bits_A),
                .C3(broadcast_column_3_8bits_A),
                .C4(broadcast_column_4_8bits_A),
                .R1(broadcast_column_1_1),
                .R2(broadcast_column_2_1),
                .R3(broadcast_column_3_1),
                .R4(broadcast_column_4_1),
                .PSAD_0(Row_0_Out),
                .PSAD_1(Row_1_Out)
                 );

Row_Adder_2_CSA PSAD_2 (
                .clk(clk),
                 .reset(reset),
                 .enable(enable),
                 .update_Cur_MB(update_Row_2),
                .C1(broadcast_column_1_8bits_A),
                .C2(broadcast_column_2_8bits_A),
                .C3(broadcast_column_3_8bits_A),
                .C4(broadcast_column_4_8bits_A),
                .R1(broadcast_column_1_2),
                .R2(broadcast_column_2_2),
                .R3(broadcast_column_3_2),
                .R4(broadcast_column_4_2),
                .PSAD_1(Row_1_Out),
                .PSAD_2(Row_2_Out)
                 );

Row_Adder_3_CSA PSAD_3 (
                .clk(clk),
                 .reset(reset),
                 .enable(enable),
                 .update_Cur_MB(update_Row_3),
                .C1(broadcast_column_1_8bits_A),
                .C2(broadcast_column_2_8bits_A),
                .C3(broadcast_column_3_8bits_A),
```

```
        .C4(broadcast_column_4_8bits_A),
        .R1(broadcast_column_1_3),
        .R2(broadcast_column_2_3),
        .R3(broadcast_column_3_3),
        .R4(broadcast_column_4_3),
        .PSAD_2(Row_2_Out),
        .PSAD_3(PSAD_Out)
        );

endmodule
```

```
/*****************************
 * Theepan Moorthy           *
 * Ryerson University        *
 * Copyright August 2007     *
 * four_stage_delay_line.v   *
 *****************************/

// This Module uses four 13-bit registers
// connected serially to create the 4-stage delay line
// used to propagate half of the 4x4 PSADs as required

module four_stage_delay_line (

clk,              // Input clock signal
reset,            // Input reset signal
enable,       // Input enable signal
input_13_bits, // Input 13-bits that are to be delayed
output_13_bits // Outputs the initial input 13-bits by four clock cycles

);


//------------- Input Ports ------------------------//

input clk;
input reset;
input enable;
input [12:0] input_13_bits;


//------------- Output Ports ------------------------//

output reg [12:0] output_13_bits;


//------------- Internal Registers ---------------------//

reg [12:0] Reg_1;
reg [12:0] Reg_2;
reg [12:0] Reg_3;


//------------- Implementation ---------------------//

always @(posedge clk) begin
    if    (reset)  Reg_1 <= 0;
    else if (enable) Reg_1 <= input_13_bits;
end

always @(posedge clk) begin
    if (enable) Reg_2 <= Reg_1;
end
```

```
always @(posedge clk) begin
    if (enable) Reg_3 <= Reg_2;
end

always @(posedge clk) begin
    if (enable) output_13_bits <= Reg_3;
end

endmodule
```

```
/*****************************
 * Theepan Moorthy           *
 * Ryerson University        *
 * Copyright August 2007     *
 * eight_stage_delay_line.v  *
 *****************************/

// This Module uses eight 15-bit registers
// connected serially to create the 8-stage delay line
// used to propagate half of the 8x8 PSADs as required

module eight_stage_delay_line (

clk,                  // Input clock signal
reset,                // Input reset signal
enable,         // Input enable signal
input_15_bits,  // Input 15-bits that are to be delayed
output_15_bits // Outputs the initial input 15-bits by four clock cycles

);


//------------- Input Ports -------------------------//

input clk;
input reset;
input enable;
input [14:0] input_15_bits;


//------------- Output Ports ------------------------//

output reg [14:0] output_15_bits;


//------------- Internal Registers ---------------------//

reg [14:0] Reg_1;
reg [14:0] Reg_2;
reg [14:0] Reg_3;
reg [14:0] Reg_4;
reg [14:0] Reg_5;
reg [14:0] Reg_6;
reg [14:0] Reg_7;


//------------- Implementation ---------------------//

always @(posedge clk) begin
      if     (reset)  Reg_1 <= 0;
      else if (enable) Reg_1 <= input_15_bits;
end
```

```
always @(posedge clk) begin
    if (enable) Reg_2 <= Reg_1;
end

always @(posedge clk) begin
    if (enable) Reg_3 <= Reg_2;
end

always @(posedge clk) begin
    if (enable) Reg_4 <= Reg_3;
end

always @(posedge clk) begin
    if (enable) Reg_5 <= Reg_4;
end

always @(posedge clk) begin
    if (enable) Reg_6 <= Reg_5;
end

always @(posedge clk) begin
    if (enable) Reg_7 <= Reg_6;
end

always @(posedge clk) begin
    if (enable) output_15_bits <= Reg_7;
end

endmodule
```

```
/*****************************
 * Theepan Moorthy           *
 * Ryerson University        *
 * Copyright September 2007  *
 * main_data_path.v          *
 *****************************/

// This Module is the top level data path module
// which integrates and assembles the entire data path architecture

module main_data_path (

clk,                    // Input clock signal
reset,                  // Input reset signal
enable,             // Input enable signal
update_Cur_MB,      // Input control signal to load new Current MB data
bus_Select,         // Input control signal for Bus Line Selection

broadcast_column_1_32bits_A, // Input 32-bits from memory broadcast bus #1
broadcast_column_2_32bits_A, // Input 32-bits from memory broadcast bus #2
broadcast_column_3_32bits_A, // Input 32-bits from memory broadcast bus #3
broadcast_column_4_32bits_A, // Input 32-bits from memory broadcast bus #4

broadcast_column_1_32bits_B, // Input 32-bits from memory broadcast bus #1
broadcast_column_2_32bits_B, // Input 32-bits from memory broadcast bus #2
broadcast_column_3_32bits_B, // Input 32-bits from memory broadcast bus #3
broadcast_column_4_32bits_B, // Input 32-bits from memory broadcast bus #4

// The 16 4x4 SAD output registers, labeled by row/column
B4x4_00_out, B4x4_01_out, B4x4_02_out, B4x4_03_out,
B4x4_10_out, B4x4_11_out, B4x4_12_out, B4x4_13_out,
B4x4_20_out, B4x4_21_out, B4x4_22_out, B4x4_23_out,
B4x4_30_out, B4x4_31_out, B4x4_32_out, B4x4_33_out,

// The 8 4x8 SAD output registers, labeled by row/column
B4x8_00_out, B4x8_01_out, B4x8_02_out, B4x8_03_out,
B4x8_10_out, B4x8_11_out, B4x8_12_out, B4x8_13_out,

// The 8 8x4 SAD output registers, labeled by row/column
B8x4_00_out, B8x4_01_out,
B8x4_10_out, B8x4_11_out,
B8x4_20_out, B8x4_21_out,
B8x4_30_out, B8x4_31_out,

// The 4 8x8 SAD output registers, labeled by row/column
B8x8_00_out, B8x8_01_out,
B8x8_10_out, B8x8_11_out,

// The 2 8x16 SAD output registers, labeled by column
B8x16_0_out, B8x16_1_out,

// The 2 16x8 SAD output registers, labeled by row
B16x8_0_out, B16x8_1_out,
```

```
// The 1 16x16 SAD output register
B16x16_out

);

//------------- Input Ports -------------------------//

input clk;
input reset;
input enable;
input [4:0] update_Cur_MB;
input [3:0] bus_Select;

input [31:0] broadcast_column_1_32bits_A;
input [31:0] broadcast_column_2_32bits_A;
input [31:0] broadcast_column_3_32bits_A;
input [31:0] broadcast_column_4_32bits_A;

input [31:0] broadcast_column_1_32bits_B;
input [31:0] broadcast_column_2_32bits_B;
input [31:0] broadcast_column_3_32bits_B;
input [31:0] broadcast_column_4_32bits_B;

//------------------- Output Ports ------------------------//

// The 16 4x4 SAD output registers, labeled by row/column
output [12:0] B4x4_00_out, B4x4_01_out, B4x4_02_out, B4x4_03_out;
output [12:0] B4x4_10_out, B4x4_11_out, B4x4_12_out, B4x4_13_out;
output [12:0] B4x4_20_out, B4x4_21_out, B4x4_22_out, B4x4_23_out;
output [12:0] B4x4_30_out, B4x4_31_out, B4x4_32_out, B4x4_33_out;

// The 8 4x8 SAD output registers, labeled by row/column
output reg [13:0]
B4x8_00_out, B4x8_01_out, B4x8_02_out, B4x8_03_out,
B4x8_10_out, B4x8_11_out, B4x8_12_out, B4x8_13_out;

// The 8 8x4 SAD output registers, labeled by row/column
output reg [13:0]
B8x4_00_out, B8x4_01_out,
B8x4_10_out, B8x4_11_out,
B8x4_20_out, B8x4_21_out,
B8x4_30_out, B8x4_31_out;

// The 4 8x8 SAD output registers, labeled by row/column
output reg [14:0]
B8x8_00_out, B8x8_01_out,
B8x8_10_out, B8x8_11_out;

// The 2 8x16 SAD output registers, labeled by column
output reg [15:0] B8x16_0_out, B8x16_1_out;

// The 2 16x8 SAD output registers, labeled by row
output reg [15:0] B16x8_0_out, B16x8_1_out;
```

```
// The 1 16x16 SAD output register
output reg [16:0] B16x16_out;


//------------- Internal Wires ------------------------//

reg update_Block_Row_0;
reg update_Block_Row_1;
reg update_Block_Row_2;
reg update_Block_Row_3;
reg update_Block_Row_4;
reg update_Block_Row_5;
reg update_Block_Row_6;
reg update_Block_Row_7;
reg update_Block_Row_8;
reg update_Block_Row_9;
reg update_Block_Row_10;
reg update_Block_Row_11;
reg update_Block_Row_12;
reg update_Block_Row_13;
reg update_Block_Row_14;
reg update_Block_Row_15;

reg bus_Select_0;
reg bus_Select_1;
reg bus_Select_2;
reg bus_Select_3;
reg bus_Select_4;
reg bus_Select_5;
reg bus_Select_6;
reg bus_Select_7;
reg bus_Select_8;
reg bus_Select_9;
reg bus_Select_10;
reg bus_Select_11;
reg bus_Select_12;
reg bus_Select_13;
reg bus_Select_14;
reg bus_Select_15;

wire [12:0]
B4x4_00, B4x4_01, B4x4_02, B4x4_03,
B4x4_20, B4x4_21, B4x4_22, B4x4_23;

wire [14:0]
B8x8_00_delayed, B8x8_01_delayed;

//------------- Implementation ------------------------//

// Control of updating Current MB row selection...
always @(*)
begin
      case (update_Cur_MB)
```

```
0 : begin
   update_Block_Row_0  = 1;
   update_Block_Row_1  = 0;
   update_Block_Row_2  = 0;
   update_Block_Row_3  = 0;
   update_Block_Row_4  = 0;
   update_Block_Row_5  = 0;
   update_Block_Row_6  = 0;
   update_Block_Row_7  = 0;
   update_Block_Row_8  = 0;
   update_Block_Row_9  = 0;
   update_Block_Row_10 = 0;
   update_Block_Row_11 = 0;
   update_Block_Row_12 = 0;
   update_Block_Row_13 = 0;
   update_Block_Row_14 = 0;
   update_Block_Row_15 = 0;
   end
1 : begin
   update_Block_Row_0  = 0;
   update_Block_Row_1  = 1;
   update_Block_Row_2  = 0;
   update_Block_Row_3  = 0;
   update_Block_Row_4  = 0;
   update_Block_Row_5  = 0;
   update_Block_Row_6  = 0;
   update_Block_Row_7  = 0;
   update_Block_Row_8  = 0;
   update_Block_Row_9  = 0;
   update_Block_Row_10 = 0;
   update_Block_Row_11 = 0;
   update_Block_Row_12 = 0;
   update_Block_Row_13 = 0;
   update_Block_Row_14 = 0;
   update_Block_Row_15 = 0;
   end
2 : begin
   update_Block_Row_0  = 0;
   update_Block_Row_1  = 0;
   update_Block_Row_2  = 1;
   update_Block_Row_3  = 0;
   update_Block_Row_4  = 0;
   update_Block_Row_5  = 0;
   update_Block_Row_6  = 0;
   update_Block_Row_7  = 0;
   update_Block_Row_8  = 0;
   update_Block_Row_9  = 0;
   update_Block_Row_10 = 0;
   update_Block_Row_11 = 0;
   update_Block_Row_12 = 0;
   update_Block_Row_13 = 0;
   update_Block_Row_14 = 0;
   update_Block_Row_15 = 0;
   end
```

```
3 : begin
   update_Block_Row_0  = 0;
   update_Block_Row_1  = 0;
   update_Block_Row_2  = 0;
   update_Block_Row_3  = 1;
   update_Block_Row_4  = 0;
   update_Block_Row_5  = 0;
   update_Block_Row_6  = 0;
   update_Block_Row_7  = 0;
   update_Block_Row_8  = 0;
   update_Block_Row_9  = 0;
   update_Block_Row_10 = 0;
   update_Block_Row_11 = 0;
   update_Block_Row_12 = 0;
   update_Block_Row_13 = 0;
   update_Block_Row_14 = 0;
   update_Block_Row_15 = 0;
   end

4 : begin
   update_Block_Row_0  = 0;
   update_Block_Row_1  = 0;
   update_Block_Row_2  = 0;
   update_Block_Row_3  = 0;
   update_Block_Row_4  = 1;
   update_Block_Row_5  = 0;
   update_Block_Row_6  = 0;
   update_Block_Row_7  = 0;
   update_Block_Row_8  = 0;
   update_Block_Row_9  = 0;
   update_Block_Row_10 = 0;
   update_Block_Row_11 = 0;
   update_Block_Row_12 = 0;
   update_Block_Row_13 = 0;
   update_Block_Row_14 = 0;
   update_Block_Row_15 = 0;
   end
5 : begin
   update_Block_Row_0  = 0;
   update_Block_Row_1  = 0;
   update_Block_Row_2  = 0;
   update_Block_Row_3  = 0;
   update_Block_Row_4  = 0;
   update_Block_Row_5  = 1;
   update_Block_Row_6  = 0;
   update_Block_Row_7  = 0;
   update_Block_Row_8  = 0;
   update_Block_Row_9  = 0;
   update_Block_Row_10 = 0;
   update_Block_Row_11 = 0;
   update_Block_Row_12 = 0;
   update_Block_Row_13 = 0;
   update_Block_Row_14 = 0;
   update_Block_Row_15 = 0;
```

```
    end
  6 : begin
    update_Block_Row_0  = 0;
    update_Block_Row_1  = 0;
    update_Block_Row_2  = 0;
    update_Block_Row_3  = 0;
    update_Block_Row_4  = 0;
    update_Block_Row_5  = 0;
    update_Block_Row_6  = 1;
    update_Block_Row_7  = 0;
    update_Block_Row_8  = 0;
    update_Block_Row_9  = 0;
    update_Block_Row_10 = 0;
    update_Block_Row_11 = 0;
    update_Block_Row_12 = 0;
    update_Block_Row_13 = 0;
    update_Block_Row_14 = 0;
    update_Block_Row_15 = 0;
    end
  7 : begin
    update_Block_Row_0  = 0;
    update_Block_Row_1  = 0;
    update_Block_Row_2  = 0;
    update_Block_Row_3  = 0;
    update_Block_Row_4  = 0;
    update_Block_Row_5  = 0;
    update_Block_Row_6  = 0;
    update_Block_Row_7  = 1;
    update_Block_Row_8  = 0;
    update_Block_Row_9  = 0;
    update_Block_Row_10 = 0;
    update_Block_Row_11 = 0;
    update_Block_Row_12 = 0;
    update_Block_Row_13 = 0;
    update_Block_Row_14 = 0;
    update_Block_Row_15 = 0;
    end
  8 : begin
    update_Block_Row_0  = 0;
    update_Block_Row_1  = 0;
    update_Block_Row_2  = 0;
    update_Block_Row_3  = 0;
    update_Block_Row_4  = 0;
    update_Block_Row_5  = 0;
    update_Block_Row_6  = 0;
    update_Block_Row_7  = 0;
    update_Block_Row_8  = 1;
    update_Block_Row_9  = 0;
    update_Block_Row_10 = 0;
    update_Block_Row_11 = 0;
    update_Block_Row_12 = 0;
    update_Block_Row_13 = 0;
    update_Block_Row_14 = 0;
    update_Block_Row_15 = 0;
```

```
    end
  9 : begin
    update_Block_Row_0  = 0;
    update_Block_Row_1  = 0;
    update_Block_Row_2  = 0;
    update_Block_Row_3  = 0;
    update_Block_Row_4  = 0;
    update_Block_Row_5  = 0;
    update_Block_Row_6  = 0;
    update_Block_Row_7  = 0;
    update_Block_Row_8  = 0;
    update_Block_Row_9  = 1;
    update_Block_Row_10 = 0;
    update_Block_Row_11 = 0;
    update_Block_Row_12 = 0;
    update_Block_Row_13 = 0;
    update_Block_Row_14 = 0;
    update_Block_Row_15 = 0;
    end
  10 : begin
    update_Block_Row_0  = 0;
    update_Block_Row_1  = 0;
    update_Block_Row_2  = 0;
    update_Block_Row_3  = 0;
    update_Block_Row_4  = 0;
    update_Block_Row_5  = 0;
    update_Block_Row_6  = 0;
    update_Block_Row_7  = 0;
    update_Block_Row_8  = 0;
    update_Block_Row_9  = 0;
    update_Block_Row_10 = 1;
    update_Block_Row_11 = 0;
    update_Block_Row_12 = 0;
    update_Block_Row_13 = 0;
    update_Block_Row_14 = 0;
    update_Block_Row_15 = 0;
    end
  11 : begin
    update_Block_Row_0  = 0;
    update_Block_Row_1  = 0;
    update_Block_Row_2  = 0;
    update_Block_Row_3  = 0;
    update_Block_Row_4  = 0;
    update_Block_Row_5  = 0;
    update_Block_Row_6  = 0;
    update_Block_Row_7  = 0;
    update_Block_Row_8  = 0;
    update_Block_Row_9  = 0;
    update_Block_Row_10 = 0;
    update_Block_Row_11 = 1;
    update_Block_Row_12 = 0;
    update_Block_Row_13 = 0;
    update_Block_Row_14 = 0;
    update_Block_Row_15 = 0;
```

```
    end
  12 : begin
    update_Block_Row_0  = 0;
    update_Block_Row_1  = 0;
    update_Block_Row_2  = 0;
    update_Block_Row_3  = 0;
    update_Block_Row_4  = 0;
    update_Block_Row_5  = 0;
    update_Block_Row_6  = 0;
    update_Block_Row_7  = 0;
    update_Block_Row_8  = 0;
    update_Block_Row_9  = 0;
    update_Block_Row_10 = 0;
    update_Block_Row_11 = 0;
    update_Block_Row_12 = 1;
    update_Block_Row_13 = 0;
    update_Block_Row_14 = 0;
    update_Block_Row_15 = 0;
    end
  13 : begin
    update_Block_Row_0  = 0;
    update_Block_Row_1  = 0;
    update_Block_Row_2  = 0;
    update_Block_Row_3  = 0;
    update_Block_Row_4  = 0;
    update_Block_Row_5  = 0;
    update_Block_Row_6  = 0;
    update_Block_Row_7  = 0;
    update_Block_Row_8  = 0;
    update_Block_Row_9  = 0;
    update_Block_Row_10 = 0;
    update_Block_Row_11 = 0;
    update_Block_Row_12 = 0;
    update_Block_Row_13 = 1;
    update_Block_Row_14 = 0;
    update_Block_Row_15 = 0;
    end
  14 : begin
    update_Block_Row_0  = 0;
    update_Block_Row_1  = 0;
    update_Block_Row_2  = 0;
    update_Block_Row_3  = 0;
    update_Block_Row_4  = 0;
    update_Block_Row_5  = 0;
    update_Block_Row_6  = 0;
    update_Block_Row_7  = 0;
    update_Block_Row_8  = 0;
    update_Block_Row_9  = 0;
    update_Block_Row_10 = 0;
    update_Block_Row_11 = 0;
    update_Block_Row_12 = 0;
    update_Block_Row_13 = 0;
    update_Block_Row_14 = 1;
    update_Block_Row_15 = 0;
```

```
        end
    15 : begin
        update_Block_Row_0  = 0;
        update_Block_Row_1  = 0;
        update_Block_Row_2  = 0;
        update_Block_Row_3  = 0;
        update_Block_Row_4  = 0;
        update_Block_Row_5  = 0;
        update_Block_Row_6  = 0;
        update_Block_Row_7  = 0;
        update_Block_Row_8  = 0;
        update_Block_Row_9  = 0;
        update_Block_Row_10 = 0;
        update_Block_Row_11 = 0;
        update_Block_Row_12 = 0;
        update_Block_Row_13 = 0;
        update_Block_Row_14 = 0;
        update_Block_Row_15 = 1;
        end
    16 : begin
        update_Block_Row_0  = 0;
        update_Block_Row_1  = 0;
        update_Block_Row_2  = 0;
        update_Block_Row_3  = 0;
        update_Block_Row_4  = 0;
        update_Block_Row_5  = 0;
        update_Block_Row_6  = 0;
        update_Block_Row_7  = 0;
        update_Block_Row_8  = 0;
        update_Block_Row_9  = 0;
        update_Block_Row_10 = 0;
        update_Block_Row_11 = 0;
        update_Block_Row_12 = 0;
        update_Block_Row_13 = 0;
        update_Block_Row_14 = 0;
        update_Block_Row_15 = 0;
        end
    default : begin
        update_Block_Row_0  = 0;
        update_Block_Row_1  = 0;
        update_Block_Row_2  = 0;
        update_Block_Row_3  = 0;
        update_Block_Row_4  = 0;
        update_Block_Row_5  = 0;
        update_Block_Row_6  = 0;
        update_Block_Row_7  = 0;
        update_Block_Row_8  = 0;
        update_Block_Row_9  = 0;
        update_Block_Row_10 = 0;
        update_Block_Row_11 = 0;
        update_Block_Row_12 = 0;
        update_Block_Row_13 = 0;
        update_Block_Row_14 = 0;
        update_Block_Row_15 = 0;
```

```
            end
        endcase
end

// Control of the Dual Bus Line selection...
always @(*)
begin
      case (bus_Select)
        0 : begin
            bus_Select_0  = 0;
            bus_Select_1  = 0;
            bus_Select_2  = 0;
            bus_Select_3  = 0;
            bus_Select_4  = 0;
            bus_Select_5  = 0;
            bus_Select_6  = 0;
            bus_Select_7  = 0;
            bus_Select_8  = 0;
            bus_Select_9  = 0;
            bus_Select_10 = 0;
            bus_Select_11 = 0;
            bus_Select_12 = 0;
            bus_Select_13 = 0;
            bus_Select_14 = 0;
            bus_Select_15 = 0;
            end
        1 : begin
            bus_Select_0  = 0;
            bus_Select_1  = 1;
            bus_Select_2  = 1;
            bus_Select_3  = 1;
            bus_Select_4  = 1;
            bus_Select_5  = 1;
            bus_Select_6  = 1;
            bus_Select_7  = 1;
            bus_Select_8  = 1;
            bus_Select_9  = 1;
            bus_Select_10 = 1;
            bus_Select_11 = 1;
            bus_Select_12 = 1;
            bus_Select_13 = 1;
            bus_Select_14 = 1;
            bus_Select_15 = 1;
            end
        2 : begin
            bus_Select_0  = 0;
            bus_Select_1  = 0;
            bus_Select_2  = 1;
            bus_Select_3  = 1;
            bus_Select_4  = 1;
            bus_Select_5  = 1;
            bus_Select_6  = 1;
            bus_Select_7  = 1;
            bus_Select_8  = 1;
```

```
    bus_Select_9  = 1;
    bus_Select_10 = 1;
    bus_Select_11 = 1;
    bus_Select_12 = 1;
    bus_Select_13 = 1;
    bus_Select_14 = 1;
    bus_Select_15 = 1;
    end
  3 : begin
    bus_Select_0  = 0;
    bus_Select_1  = 0;
    bus_Select_2  = 0;
    bus_Select_3  = 1;
    bus_Select_4  = 1;
    bus_Select_5  = 1;
    bus_Select_6  = 1;
    bus_Select_7  = 1;
    bus_Select_8  = 1;
    bus_Select_9  = 1;
    bus_Select_10 = 1;
    bus_Select_11 = 1;
    bus_Select_12 = 1;
    bus_Select_13 = 1;
    bus_Select_14 = 1;
    bus_Select_15 = 1;
    end

  4 : begin
    bus_Select_0  = 0;
    bus_Select_1  = 0;
    bus_Select_2  = 0;
    bus_Select_3  = 0;
    bus_Select_4  = 1;
    bus_Select_5  = 1;
    bus_Select_6  = 1;
    bus_Select_7  = 1;
    bus_Select_8  = 1;
    bus_Select_9  = 1;
    bus_Select_10 = 1;
    bus_Select_11 = 1;
    bus_Select_12 = 1;
    bus_Select_13 = 1;
    bus_Select_14 = 1;
    bus_Select_15 = 1;
    end
  5 : begin
    bus_Select_0  = 0;
    bus_Select_1  = 0;
    bus_Select_2  = 0;
    bus_Select_3  = 0;
    bus_Select_4  = 0;
    bus_Select_5  = 1;
    bus_Select_6  = 1;
    bus_Select_7  = 1;
```

```
    bus_Select_8  = 1;
    bus_Select_9  = 1;
    bus_Select_10 = 1;
    bus_Select_11 = 1;
    bus_Select_12 = 1;
    bus_Select_13 = 1;
    bus_Select_14 = 1;
    bus_Select_15 = 1;
    end
  6 : begin
    bus_Select_0  = 0;
    bus_Select_1  = 0;
    bus_Select_2  = 0;
    bus_Select_3  = 0;
    bus_Select_4  = 0;
    bus_Select_5  = 0;
    bus_Select_6  = 1;
    bus_Select_7  = 1;
    bus_Select_8  = 1;
    bus_Select_9  = 1;
    bus_Select_10 = 1;
    bus_Select_11 = 1;
    bus_Select_12 = 1;
    bus_Select_13 = 1;
    bus_Select_14 = 1;
    bus_Select_15 = 1;
    end
  7 : begin
    bus_Select_0  = 0;
    bus_Select_1  = 0;
    bus_Select_2  = 0;
    bus_Select_3  = 0;
    bus_Select_4  = 0;
    bus_Select_5  = 0;
    bus_Select_6  = 0;
    bus_Select_7  = 1;
    bus_Select_8  = 1;
    bus_Select_9  = 1;
    bus_Select_10 = 1;
    bus_Select_11 = 1;
    bus_Select_12 = 1;
    bus_Select_13 = 1;
    bus_Select_14 = 1;
    bus_Select_15 = 1;
    end

  8 : begin
    bus_Select_0  = 0;
    bus_Select_1  = 0;
    bus_Select_2  = 0;
    bus_Select_3  = 0;
    bus_Select_4  = 0;
    bus_Select_5  = 0;
    bus_Select_6  = 0;
```

```
    bus_Select_7  = 0;
    bus_Select_8  = 1;
    bus_Select_9  = 1;
    bus_Select_10 = 1;
    bus_Select_11 = 1;
    bus_Select_12 = 1;
    bus_Select_13 = 1;
    bus_Select_14 = 1;
    bus_Select_15 = 1;
    end
  9 : begin
    bus_Select_0  = 0;
    bus_Select_1  = 0;
    bus_Select_2  = 0;
    bus_Select_3  = 0;
    bus_Select_4  = 0;
    bus_Select_5  = 0;
    bus_Select_6  = 0;
    bus_Select_7  = 0;
    bus_Select_8  = 0;
    bus_Select_9  = 1;
    bus_Select_10 = 1;
    bus_Select_11 = 1;
    bus_Select_12 = 1;
    bus_Select_13 = 1;
    bus_Select_14 = 1;
    bus_Select_15 = 1;
    end
  10 : begin
    bus_Select_0  = 0;
    bus_Select_1  = 0;
    bus_Select_2  = 0;
    bus_Select_3  = 0;
    bus_Select_4  = 0;
    bus_Select_5  = 0;
    bus_Select_6  = 0;
    bus_Select_7  = 0;
    bus_Select_8  = 0;
    bus_Select_9  = 0;
    bus_Select_10 = 1;
    bus_Select_11 = 1;
    bus_Select_12 = 1;
    bus_Select_13 = 1;
    bus_Select_14 = 1;
    bus_Select_15 = 1;
    end
  11 : begin
    bus_Select_0  = 0;
    bus_Select_1  = 0;
    bus_Select_2  = 0;
    bus_Select_3  = 0;
    bus_Select_4  = 0;
    bus_Select_5  = 0;
    bus_Select_6  = 0;
```

```
    bus_Select_7  = 0;
    bus_Select_8  = 0;
    bus_Select_9  = 0;
    bus_Select_10 = 0;
    bus_Select_11 = 1;
    bus_Select_12 = 1;
    bus_Select_13 = 1;
    bus_Select_14 = 1;
    bus_Select_15 = 1;
    end
  12 : begin
    bus_Select_0  = 0;
    bus_Select_1  = 0;
    bus_Select_2  = 0;
    bus_Select_3  = 0;
    bus_Select_4  = 0;
    bus_Select_5  = 0;
    bus_Select_6  = 0;
    bus_Select_7  = 0;
    bus_Select_8  = 0;
    bus_Select_9  = 0;
    bus_Select_10 = 0;
    bus_Select_11 = 0;
    bus_Select_12 = 1;
    bus_Select_13 = 1;
    bus_Select_14 = 1;
    bus_Select_15 = 1;
    end
  13 : begin
    bus_Select_0  = 0;
    bus_Select_1  = 0;
    bus_Select_2  = 0;
    bus_Select_3  = 0;
    bus_Select_4  = 0;
    bus_Select_5  = 0;
    bus_Select_6  = 0;
    bus_Select_7  = 0;
    bus_Select_8  = 0;
    bus_Select_9  = 0;
    bus_Select_10 = 0;
    bus_Select_11 = 0;
    bus_Select_12 = 0;
    bus_Select_13 = 1;
    bus_Select_14 = 1;
    bus_Select_15 = 1;
    end
  14 : begin
    bus_Select_0  = 0;
    bus_Select_1  = 0;
    bus_Select_2  = 0;
    bus_Select_3  = 0;
    bus_Select_4  = 0;
    bus_Select_5  = 0;
    bus_Select_6  = 0;
```

```
            bus_Select_7  = 0;
            bus_Select_8  = 0;
            bus_Select_9  = 0;
            bus_Select_10 = 0;
            bus_Select_11 = 0;
            bus_Select_12 = 0;
            bus_Select_13 = 0;
            bus_Select_14 = 1;
            bus_Select_15 = 1;
            end
         15 : begin
            bus_Select_0  = 0;
            bus_Select_1  = 0;
            bus_Select_2  = 0;
            bus_Select_3  = 0;
            bus_Select_4  = 0;
            bus_Select_5  = 0;
            bus_Select_6  = 0;
            bus_Select_7  = 0;
            bus_Select_8  = 0;
            bus_Select_9  = 0;
            bus_Select_10 = 0;
            bus_Select_11 = 0;
            bus_Select_12 = 0;
            bus_Select_13 = 0;
            bus_Select_14 = 0;
            bus_Select_15 = 1;
            end
        default : begin
            bus_Select_0  = 0; //i.e. set to Select Bus A
            bus_Select_1  = 0;
            bus_Select_2  = 0;
            bus_Select_3  = 0;
            bus_Select_4  = 0;
            bus_Select_5  = 0;
            bus_Select_6  = 0;
            bus_Select_7  = 0;
            bus_Select_8  = 0;
            bus_Select_9  = 0;
            bus_Select_10 = 0;
            bus_Select_11 = 0;
            bus_Select_12 = 0;
            bus_Select_13 = 0;
            bus_Select_14 = 0;
            bus_Select_15 = 0;
           end
       endcase
end

//------------- Instantiations -------------------------//

// The 16 4x4 Blocks
four_by_four_block B4x4_00_B (.clk(clk),
                              .reset(reset),
```

```
                        .enable(enable),
                        .update_Row_0(update_Block_Row_0),
                        .update_Row_1(update_Block_Row_1),
                        .update_Row_2(update_Block_Row_2),
                        .update_Row_3(update_Block_Row_3),
                        .bus_Select_0(bus_Select_0),
                        .bus_Select_1(bus_Select_1),
                        .bus_Select_2(bus_Select_2),
                        .bus_Select_3(bus_Select_3),
                    .broadcast_column_1_8bits_A(broadcast_column_1_32bits_A[31:24]),
                    .broadcast_column_2_8bits_A(broadcast_column_1_32bits_A[23:16]),
                    .broadcast_column_3_8bits_A(broadcast_column_1_32bits_A[15:8]),
                    .broadcast_column_4_8bits_A(broadcast_column_1_32bits_A[7:0]),
                    .broadcast_column_1_8bits_B(broadcast_column_1_32bits_B[31:24]),
                    .broadcast_column_2_8bits_B(broadcast_column_1_32bits_B[23:16]),
                    .broadcast_column_3_8bits_B(broadcast_column_1_32bits_B[15:8]),
                    .broadcast_column_4_8bits_B(broadcast_column_1_32bits_B[7:0]),
                    .PSAD_Out(B4x4_00)
                     );

four_by_four_block B4x4_01_B (.clk(clk),
                        .reset(reset),
                        .enable(enable),
                        .update_Row_0(update_Block_Row_0),
                        .update_Row_1(update_Block_Row_1),
                        .update_Row_2(update_Block_Row_2),
                        .update_Row_3(update_Block_Row_3),
                        .bus_Select_0(bus_Select_0),
                        .bus_Select_1(bus_Select_1),
                        .bus_Select_2(bus_Select_2),
                        .bus_Select_3(bus_Select_3),
                    .broadcast_column_1_8bits_A(broadcast_column_2_32bits_A[31:24]),
                    .broadcast_column_2_8bits_A(broadcast_column_2_32bits_A[23:16]),
                    .broadcast_column_3_8bits_A(broadcast_column_2_32bits_A[15:8]),
                    .broadcast_column_4_8bits_A(broadcast_column_2_32bits_A[7:0]),
                    .broadcast_column_1_8bits_B(broadcast_column_2_32bits_B[31:24]),
                    .broadcast_column_2_8bits_B(broadcast_column_2_32bits_B[23:16]),
                    .broadcast_column_3_8bits_B(broadcast_column_2_32bits_B[15:8]),
                    .broadcast_column_4_8bits_B(broadcast_column_2_32bits_B[7:0]),
                    .PSAD_Out(B4x4_01)
                     );

four_by_four_block B4x4_02_B (.clk(clk),
                        .reset(reset),
                        .enable(enable),
                        .update_Row_0(update_Block_Row_0),
                        .update_Row_1(update_Block_Row_1),
                        .update_Row_2(update_Block_Row_2),
                        .update_Row_3(update_Block_Row_3),
                        .bus_Select_0(bus_Select_0),
                        .bus_Select_1(bus_Select_1),
                        .bus_Select_2(bus_Select_2),
                        .bus_Select_3(bus_Select_3),
                    .broadcast_column_1_8bits_A(broadcast_column_3_32bits_A[31:24]),
```

```
                    .broadcast_column_2_8bits_A(broadcast_column_3_32bits_A[23:16]),
                    .broadcast_column_3_8bits_A(broadcast_column_3_32bits_A[15:8]),
                    .broadcast_column_4_8bits_A(broadcast_column_3_32bits_A[7:0]),
                    .broadcast_column_1_8bits_B(broadcast_column_3_32bits_B[31:24]),
                    .broadcast_column_2_8bits_B(broadcast_column_3_32bits_B[23:16]),
                    .broadcast_column_3_8bits_B(broadcast_column_3_32bits_B[15:8]),
                    .broadcast_column_4_8bits_B(broadcast_column_3_32bits_B[7:0]),
                    .PSAD_Out(B4x4_02)
                     );

four_by_four_block B4x4_03_B (.clk(clk),
                          .reset(reset),
                          .enable(enable),
                          .update_Row_0(update_Block_Row_0),
                          .update_Row_1(update_Block_Row_1),
                          .update_Row_2(update_Block_Row_2),
                          .update_Row_3(update_Block_Row_3),
                          .bus_Select_0(bus_Select_0),
                          .bus_Select_1(bus_Select_1),
                          .bus_Select_2(bus_Select_2),
                          .bus_Select_3(bus_Select_3),
                    .broadcast_column_1_8bits_A(broadcast_column_4_32bits_A[31:24]),
                    .broadcast_column_2_8bits_A(broadcast_column_4_32bits_A[23:16]),
                    .broadcast_column_3_8bits_A(broadcast_column_4_32bits_A[15:8]),
                    .broadcast_column_4_8bits_A(broadcast_column_4_32bits_A[7:0]),
                    .broadcast_column_1_8bits_B(broadcast_column_4_32bits_B[31:24]),
                    .broadcast_column_2_8bits_B(broadcast_column_4_32bits_B[23:16]),
                    .broadcast_column_3_8bits_B(broadcast_column_4_32bits_B[15:8]),
                    .broadcast_column_4_8bits_B(broadcast_column_4_32bits_B[7:0]),
                    .PSAD_Out(B4x4_03)
                     );
////////////////////////////////////////////////////////////////////////////////////////////////

four_by_four_block B4x4_10_B (.clk(clk),
                          .reset(reset),
                          .enable(enable),
                          .update_Row_0(update_Block_Row_4),
                          .update_Row_1(update_Block_Row_5),
                          .update_Row_2(update_Block_Row_6),
                          .update_Row_3(update_Block_Row_7),
                          .bus_Select_0(bus_Select_4),
                          .bus_Select_1(bus_Select_5),
                          .bus_Select_2(bus_Select_6),
                          .bus_Select_3(bus_Select_7),
                    .broadcast_column_1_8bits_A(broadcast_column_1_32bits_A[31:24]),
                    .broadcast_column_2_8bits_A(broadcast_column_1_32bits_A[23:16]),
                    .broadcast_column_3_8bits_A(broadcast_column_1_32bits_A[15:8]),
                    .broadcast_column_4_8bits_A(broadcast_column_1_32bits_A[7:0]),
                    .broadcast_column_1_8bits_B(broadcast_column_1_32bits_B[31:24]),
                    .broadcast_column_2_8bits_B(broadcast_column_1_32bits_B[23:16]),
                    .broadcast_column_3_8bits_B(broadcast_column_1_32bits_B[15:8]),
                    .broadcast_column_4_8bits_B(broadcast_column_1_32bits_B[7:0]),
                    .PSAD_Out(B4x4_10_out)
                     );
```

```
four_by_four_block B4x4_11_B (.clk(clk),
                              .reset(reset),
                              .enable(enable),
                              .update_Row_0(update_Block_Row_4),
                              .update_Row_1(update_Block_Row_5),
                              .update_Row_2(update_Block_Row_6),
                              .update_Row_3(update_Block_Row_7),
                              .bus_Select_0(bus_Select_4),
                              .bus_Select_1(bus_Select_5),
                              .bus_Select_2(bus_Select_6),
                              .bus_Select_3(bus_Select_7),
                            .broadcast_column_1_8bits_A(broadcast_column_2_32bits_A[31:24]),
                            .broadcast_column_2_8bits_A(broadcast_column_2_32bits_A[23:16]),
                            .broadcast_column_3_8bits_A(broadcast_column_2_32bits_A[15:8]),
                            .broadcast_column_4_8bits_A(broadcast_column_2_32bits_A[7:0]),
                            .broadcast_column_1_8bits_B(broadcast_column_2_32bits_B[31:24]),
                            .broadcast_column_2_8bits_B(broadcast_column_2_32bits_B[23:16]),
                            .broadcast_column_3_8bits_B(broadcast_column_2_32bits_B[15:8]),
                            .broadcast_column_4_8bits_B(broadcast_column_2_32bits_B[7:0]),
                            .PSAD_Out(B4x4_11_out)
                             );

four_by_four_block B4x4_12_B (.clk(clk),
                              .reset(reset),
                              .enable(enable),
                              .update_Row_0(update_Block_Row_4),
                              .update_Row_1(update_Block_Row_5),
                              .update_Row_2(update_Block_Row_6),
                              .update_Row_3(update_Block_Row_7),
                              .bus_Select_0(bus_Select_4),
                              .bus_Select_1(bus_Select_5),
                              .bus_Select_2(bus_Select_6),
                              .bus_Select_3(bus_Select_7),
                            .broadcast_column_1_8bits_A(broadcast_column_3_32bits_A[31:24]),
                            .broadcast_column_2_8bits_A(broadcast_column_3_32bits_A[23:16]),
                            .broadcast_column_3_8bits_A(broadcast_column_3_32bits_A[15:8]),
                            .broadcast_column_4_8bits_A(broadcast_column_3_32bits_A[7:0]),
                            .broadcast_column_1_8bits_B(broadcast_column_3_32bits_B[31:24]),
                            .broadcast_column_2_8bits_B(broadcast_column_3_32bits_B[23:16]),
                            .broadcast_column_3_8bits_B(broadcast_column_3_32bits_B[15:8]),
                            .broadcast_column_4_8bits_B(broadcast_column_3_32bits_B[7:0]),
                            .PSAD_Out(B4x4_12_out)
                             );

four_by_four_block B4x4_13_B (.clk(clk),
                              .reset(reset),
                              .enable(enable),
                              .update_Row_0(update_Block_Row_4),
                              .update_Row_1(update_Block_Row_5),
                              .update_Row_2(update_Block_Row_6),
                              .update_Row_3(update_Block_Row_7),
                              .bus_Select_0(bus_Select_4),
                              .bus_Select_1(bus_Select_5),
```

```
                        .bus_Select_2(bus_Select_6),
                        .bus_Select_3(bus_Select_7),
                    .broadcast_column_1_8bits_A(broadcast_column_4_32bits_A[31:24]),
                    .broadcast_column_2_8bits_A(broadcast_column_4_32bits_A[23:16]),
                    .broadcast_column_3_8bits_A(broadcast_column_4_32bits_A[15:8]),
                    .broadcast_column_4_8bits_A(broadcast_column_4_32bits_A[7:0]),
                    .broadcast_column_1_8bits_B(broadcast_column_4_32bits_B[31:24]),
                    .broadcast_column_2_8bits_B(broadcast_column_4_32bits_B[23:16]),
                    .broadcast_column_3_8bits_B(broadcast_column_4_32bits_B[15:8]),
                    .broadcast_column_4_8bits_B(broadcast_column_4_32bits_B[7:0]),
                    .PSAD_Out(B4x4_13_out)
                     );
////////////////////////////////////////////////////////////////////////////////////////////////

four_by_four_block B4x4_20_B (.clk(clk),
                        .reset(reset),
                        .enable(enable),
                        .update_Row_0(update_Block_Row_8),
                        .update_Row_1(update_Block_Row_9),
                        .update_Row_2(update_Block_Row_10),
                        .update_Row_3(update_Block_Row_11),
                        .bus_Select_0(bus_Select_8),
                        .bus_Select_1(bus_Select_9),
                        .bus_Select_2(bus_Select_10),
                        .bus_Select_3(bus_Select_11),
                    .broadcast_column_1_8bits_A(broadcast_column_1_32bits_A[31:24]),
                    .broadcast_column_2_8bits_A(broadcast_column_1_32bits_A[23:16]),
                    .broadcast_column_3_8bits_A(broadcast_column_1_32bits_A[15:8]),
                    .broadcast_column_4_8bits_A(broadcast_column_1_32bits_A[7:0]),
                    .broadcast_column_1_8bits_B(broadcast_column_1_32bits_B[31:24]),
                    .broadcast_column_2_8bits_B(broadcast_column_1_32bits_B[23:16]),
                    .broadcast_column_3_8bits_B(broadcast_column_1_32bits_B[15:8]),
                    .broadcast_column_4_8bits_B(broadcast_column_1_32bits_B[7:0]),
                    .PSAD_Out(B4x4_20)
                     );

four_by_four_block B4x4_21_B (.clk(clk),
                        .reset(reset),
                        .enable(enable),
                        .update_Row_0(update_Block_Row_8),
                        .update_Row_1(update_Block_Row_9),
                        .update_Row_2(update_Block_Row_10),
                        .update_Row_3(update_Block_Row_11),
                        .bus_Select_0(bus_Select_8),
                        .bus_Select_1(bus_Select_9),
                        .bus_Select_2(bus_Select_10),
                        .bus_Select_3(bus_Select_11),
                    .broadcast_column_1_8bits_A(broadcast_column_2_32bits_A[31:24]),
                    .broadcast_column_2_8bits_A(broadcast_column_2_32bits_A[23:16]),
                    .broadcast_column_3_8bits_A(broadcast_column_2_32bits_A[15:8]),
                    .broadcast_column_4_8bits_A(broadcast_column_2_32bits_A[7:0]),
                    .broadcast_column_1_8bits_B(broadcast_column_2_32bits_B[31:24]),
                    .broadcast_column_2_8bits_B(broadcast_column_2_32bits_B[23:16]),
                    .broadcast_column_3_8bits_B(broadcast_column_2_32bits_B[15:8]),
```

```
                    .broadcast_column_4_8bits_B(broadcast_column_2_32bits_B[7:0]),
                    .PSAD_Out(B4x4_21)
                     );

four_by_four_block B4x4_22_B (.clk(clk),
                          .reset(reset),
                          .enable(enable),
                          .update_Row_0(update_Block_Row_8),
                          .update_Row_1(update_Block_Row_9),
                          .update_Row_2(update_Block_Row_10),
                          .update_Row_3(update_Block_Row_11),
                          .bus_Select_0(bus_Select_8),
                          .bus_Select_1(bus_Select_9),
                          .bus_Select_2(bus_Select_10),
                          .bus_Select_3(bus_Select_11),
                    .broadcast_column_1_8bits_A(broadcast_column_3_32bits_A[31:24]),
                    .broadcast_column_2_8bits_A(broadcast_column_3_32bits_A[23:16]),
                    .broadcast_column_3_8bits_A(broadcast_column_3_32bits_A[15:8]),
                    .broadcast_column_4_8bits_A(broadcast_column_3_32bits_A[7:0]),
                    .broadcast_column_1_8bits_B(broadcast_column_3_32bits_B[31:24]),
                    .broadcast_column_2_8bits_B(broadcast_column_3_32bits_B[23:16]),
                    .broadcast_column_3_8bits_B(broadcast_column_3_32bits_B[15:8]),
                    .broadcast_column_4_8bits_B(broadcast_column_3_32bits_B[7:0]),
                    .PSAD_Out(B4x4_22)
                     );

four_by_four_block B4x4_23_B (.clk(clk),
                          .reset(reset),
                          .enable(enable),
                          .update_Row_0(update_Block_Row_8),
                          .update_Row_1(update_Block_Row_9),
                          .update_Row_2(update_Block_Row_10),
                          .update_Row_3(update_Block_Row_11),
                          .bus_Select_0(bus_Select_8),
                          .bus_Select_1(bus_Select_9),
                          .bus_Select_2(bus_Select_10),
                          .bus_Select_3(bus_Select_11),
                    .broadcast_column_1_8bits_A(broadcast_column_4_32bits_A[31:24]),
                    .broadcast_column_2_8bits_A(broadcast_column_4_32bits_A[23:16]),
                    .broadcast_column_3_8bits_A(broadcast_column_4_32bits_A[15:8]),
                    .broadcast_column_4_8bits_A(broadcast_column_4_32bits_A[7:0]),
                    .broadcast_column_1_8bits_B(broadcast_column_4_32bits_B[31:24]),
                    .broadcast_column_2_8bits_B(broadcast_column_4_32bits_B[23:16]),
                    .broadcast_column_3_8bits_B(broadcast_column_4_32bits_B[15:8]),
                    .broadcast_column_4_8bits_B(broadcast_column_4_32bits_B[7:0]),
                    .PSAD_Out(B4x4_23)
                     );
////////////////////////////////////////////////////////////////////////////////////////////

four_by_four_block B4x4_30_B (.clk(clk),
                          .reset(reset),
                          .enable(enable),
                          .update_Row_0(update_Block_Row_12),
                          .update_Row_1(update_Block_Row_13),
```

```
                        .update_Row_2(update_Block_Row_14),
                        .update_Row_3(update_Block_Row_15),
                        .bus_Select_0(bus_Select_12),
                        .bus_Select_1(bus_Select_13),
                        .bus_Select_2(bus_Select_14),
                        .bus_Select_3(bus_Select_15),
                    .broadcast_column_1_8bits_A(broadcast_column_1_32bits_A[31:24]),
                    .broadcast_column_2_8bits_A(broadcast_column_1_32bits_A[23:16]),
                    .broadcast_column_3_8bits_A(broadcast_column_1_32bits_A[15:8]),
                    .broadcast_column_4_8bits_A(broadcast_column_1_32bits_A[7:0]),
                    .broadcast_column_1_8bits_B(broadcast_column_1_32bits_B[31:24]),
                    .broadcast_column_2_8bits_B(broadcast_column_1_32bits_B[23:16]),
                    .broadcast_column_3_8bits_B(broadcast_column_1_32bits_B[15:8]),
                    .broadcast_column_4_8bits_B(broadcast_column_1_32bits_B[7:0]),
                    .PSAD_Out(B4x4_30_out)
                     );

four_by_four_block B4x4_31_B (.clk(clk),
                        .reset(reset),
                        .enable(enable),
                        .update_Row_0(update_Block_Row_12),
                        .update_Row_1(update_Block_Row_13),
                        .update_Row_2(update_Block_Row_14),
                        .update_Row_3(update_Block_Row_15),
                        .bus_Select_0(bus_Select_12),
                        .bus_Select_1(bus_Select_13),
                        .bus_Select_2(bus_Select_14),
                        .bus_Select_3(bus_Select_15),
                    .broadcast_column_1_8bits_A(broadcast_column_2_32bits_A[31:24]),
                    .broadcast_column_2_8bits_A(broadcast_column_2_32bits_A[23:16]),
                    .broadcast_column_3_8bits_A(broadcast_column_2_32bits_A[15:8]),
                    .broadcast_column_4_8bits_A(broadcast_column_2_32bits_A[7:0]),
                    .broadcast_column_1_8bits_B(broadcast_column_2_32bits_B[31:24]),
                    .broadcast_column_2_8bits_B(broadcast_column_2_32bits_B[23:16]),
                    .broadcast_column_3_8bits_B(broadcast_column_2_32bits_B[15:8]),
                    .broadcast_column_4_8bits_B(broadcast_column_2_32bits_B[7:0]),
                    .PSAD_Out(B4x4_31_out)
                     );

four_by_four_block B4x4_32_B (.clk(clk),
                        .reset(reset),
                        .enable(enable),
                        .update_Row_0(update_Block_Row_12),
                        .update_Row_1(update_Block_Row_13),
                        .update_Row_2(update_Block_Row_14),
                        .update_Row_3(update_Block_Row_15),
                        .bus_Select_0(bus_Select_12),
                        .bus_Select_1(bus_Select_13),
                        .bus_Select_2(bus_Select_14),
                        .bus_Select_3(bus_Select_15),
                    .broadcast_column_1_8bits_A(broadcast_column_3_32bits_A[31:24]),
                    .broadcast_column_2_8bits_A(broadcast_column_3_32bits_A[23:16]),
                    .broadcast_column_3_8bits_A(broadcast_column_3_32bits_A[15:8]),
                    .broadcast_column_4_8bits_A(broadcast_column_3_32bits_A[7:0]),
```

```
                    .broadcast_column_1_8bits_B(broadcast_column_3_32bits_B[31:24]),
                    .broadcast_column_2_8bits_B(broadcast_column_3_32bits_B[23:16]),
                    .broadcast_column_3_8bits_B(broadcast_column_3_32bits_B[15:8]),
                    .broadcast_column_4_8bits_B(broadcast_column_3_32bits_B[7:0]),
                    .PSAD_Out(B4x4_32_out)
                     );

four_by_four_block B4x4_33_B (.clk(clk),
                          .reset(reset),
                          .enable(enable),
                          .update_Row_0(update_Block_Row_12),
                          .update_Row_1(update_Block_Row_13),
                          .update_Row_2(update_Block_Row_14),
                          .update_Row_3(update_Block_Row_15),
                          .bus_Select_0(bus_Select_12),
                          .bus_Select_1(bus_Select_13),
                          .bus_Select_2(bus_Select_14),
                          .bus_Select_3(bus_Select_15),
                    .broadcast_column_1_8bits_A(broadcast_column_4_32bits_A[31:24]),
                    .broadcast_column_2_8bits_A(broadcast_column_4_32bits_A[23:16]),
                    .broadcast_column_3_8bits_A(broadcast_column_4_32bits_A[15:8]),
                    .broadcast_column_4_8bits_A(broadcast_column_4_32bits_A[7:0]),
                    .broadcast_column_1_8bits_B(broadcast_column_4_32bits_B[31:24]),
                    .broadcast_column_2_8bits_B(broadcast_column_4_32bits_B[23:16]),
                    .broadcast_column_3_8bits_B(broadcast_column_4_32bits_B[15:8]),
                    .broadcast_column_4_8bits_B(broadcast_column_4_32bits_B[7:0]),
                    .PSAD_Out(B4x4_33_out)
                     );

// The 8 4-stage Delay Lines:

four_stage_delay_line B4x4_00_delay (.clk(clk),
                          .reset(reset),
                          .enable(enable),
                          .input_13_bits(B4x4_00),
                          .output_13_bits(B4x4_00_out)
                          );

four_stage_delay_line B4x4_01_delay (.clk(clk),
                          .reset(reset),
                          .enable(enable),
                          .input_13_bits(B4x4_01),
                          .output_13_bits(B4x4_01_out)
                          );

four_stage_delay_line B4x4_02_delay (.clk(clk),
                          .reset(reset),
                          .enable(enable),
                          .input_13_bits(B4x4_02),
                          .output_13_bits(B4x4_02_out)
                          );

four_stage_delay_line B4x4_03_delay (.clk(clk),
                          .reset(reset),
```

```
                        .enable(enable),
                        .input_13_bits(B4x4_03),
                        .output_13_bits(B4x4_03_out)
                        );

four_stage_delay_line B4x4_20_delay (.clk(clk),
                        .reset(reset),
                        .enable(enable),
                        .input_13_bits(B4x4_20),
                        .output_13_bits(B4x4_20_out)
                        );

four_stage_delay_line B4x4_21_delay (.clk(clk),
                        .reset(reset),
                        .enable(enable),
                        .input_13_bits(B4x4_21),
                        .output_13_bits(B4x4_21_out)
                        );

four_stage_delay_line B4x4_22_delay (.clk(clk),
                        .reset(reset),
                        .enable(enable),
                        .input_13_bits(B4x4_22),
                        .output_13_bits(B4x4_22_out)
                        );

four_stage_delay_line B4x4_23_delay (.clk(clk),
                        .reset(reset),
                        .enable(enable),
                        .input_13_bits(B4x4_23),
                        .output_13_bits(B4x4_23_out)
                        );

// The 2 8-stage Delay Lines:

eight_stage_delay_line B8x8_00_delay (.clk(clk),
                         .reset(reset),
                         .enable(enable),
                         .input_15_bits(B8x8_00_out),
                         .output_15_bits(B8x8_00_delayed)
                         );

eight_stage_delay_line B8x8_01_delay (.clk(clk),
                         .reset(reset),
                         .enable(enable),
                         .input_15_bits(B8x8_01_out),
                         .output_15_bits(B8x8_01_delayed)
                         );

//------------- Implementation -------------------------//
/*
// The 8 4x8 SAD outputs
assign B4x8_00_out = {1'b0,B4x4_10_out} + {1'b0,B4x4_00_out};
assign B4x8_01_out = {1'b0,B4x4_11_out} + {1'b0,B4x4_01_out};
```

```
assign B4x8_02_out = {1'b0,B4x4_12_out} + {1'b0,B4x4_02_out};
assign B4x8_03_out = {1'b0,B4x4_13_out} + {1'b0,B4x4_03_out};

assign B4x8_10_out = {1'b0,B4x4_30_out} + {1'b0,B4x4_20_out};
assign B4x8_11_out = {1'b0,B4x4_31_out} + {1'b0,B4x4_21_out};
assign B4x8_12_out = {1'b0,B4x4_32_out} + {1'b0,B4x4_22_out};
assign B4x8_13_out = {1'b0,B4x4_33_out} + {1'b0,B4x4_23_out};

// The 8 8x4 SAD outputs
assign B8x4_00_out = {1'b0,B4x4_00_out} + {1'b0,B4x4_01_out};
assign B8x4_01_out = {1'b0,B4x4_02_out} + {1'b0,B4x4_03_out};
assign B8x4_10_out = {1'b0,B4x4_10_out} + {1'b0,B4x4_11_out};
assign B8x4_11_out = {1'b0,B4x4_12_out} + {1'b0,B4x4_13_out};

assign B8x4_20_out = {1'b0,B4x4_20_out} + {1'b0,B4x4_21_out};
assign B8x4_21_out = {1'b0,B4x4_22_out} + {1'b0,B4x4_23_out};
assign B8x4_30_out = {1'b0,B4x4_30_out} + {1'b0,B4x4_31_out};
assign B8x4_31_out = {1'b0,B4x4_32_out} + {1'b0,B4x4_33_out};

// The 4 8x8 SAD outputs
assign B8x8_00_out = {1'b0,B8x4_00_out} + {1'b0,B8x4_10_out};
assign B8x8_01_out = {1'b0,B8x4_01_out} + {1'b0,B8x4_11_out};
assign B8x8_10_out = {1'b0,B8x4_20_out} + {1'b0,B8x4_30_out};
assign B8x8_11_out = {1'b0,B8x4_21_out} + {1'b0,B8x4_31_out};

// The 2 8x16 SAD outputs
assign B8x16_0_out = {1'b0,B8x8_10_out}     + {1'b0,B8x8_00_delayed};
assign B8x16_1_out = {1'b0,B8x8_01_delayed} + {1'b0,B8x8_11_out};

// The 2 16x8 SAD outputs
assign B16x8_0_out = {1'b0,B8x8_00_out} + {1'b0,B8x8_01_out};
assign B16x8_1_out = {1'b0,B8x8_10_out} + {1'b0,B8x8_11_out};

// The 1 16x16 SAD output
assign B16x16_out = {1'b0,B8x16_0_out} + {1'b0,B8x16_1_out};

endmodule
*/

//------------- Implementation ------------------------//

// The 8 4x8 SAD outputs
always @(posedge clk) begin
      if     (reset)  B4x8_00_out <= 0;
      else if (enable) B4x8_00_out <= {1'b0,B4x4_10_out} + {1'b0,B4x4_00_out};
end
always @(posedge clk) begin
      if     (reset)  B4x8_01_out <= 0;
      else if (enable) B4x8_01_out <= {1'b0,B4x4_11_out} + {1'b0,B4x4_01_out};
end
always @(posedge clk) begin
      if     (reset)  B4x8_02_out <= 0;
      else if (enable) B4x8_02_out <= {1'b0,B4x4_12_out} + {1'b0,B4x4_02_out};
end
```

```
always @(posedge clk) begin
    if    (reset)  B4x8_03_out <= 0;
    else if (enable) B4x8_03_out <= {1'b0,B4x4_13_out} + {1'b0,B4x4_03_out};
end
always @(posedge clk) begin
    if    (reset)  B4x8_10_out <= 0;
    else if (enable) B4x8_10_out <= {1'b0,B4x4_30_out} + {1'b0,B4x4_20_out};
end
always @(posedge clk) begin
    if    (reset)  B4x8_11_out <= 0;
    else if (enable) B4x8_11_out <= {1'b0,B4x4_31_out} + {1'b0,B4x4_21_out};
end
always @(posedge clk) begin
    if    (reset)  B4x8_12_out <= 0;
    else if (enable) B4x8_12_out <= {1'b0,B4x4_32_out} + {1'b0,B4x4_22_out};
end
always @(posedge clk) begin
    if    (reset)  B4x8_13_out <= 0;
    else if (enable) B4x8_13_out <= {1'b0,B4x4_33_out} + {1'b0,B4x4_23_out};
end

// The 8 8x4 SAD outputs
always @(posedge clk) begin
    if    (reset)  B8x4_00_out <= 0;
    else if (enable) B8x4_00_out <= {1'b0,B4x4_00_out} + {1'b0,B4x4_01_out};
end
always @(posedge clk) begin
    if    (reset)  B8x4_01_out <= 0;
    else if (enable) B8x4_01_out <= {1'b0,B4x4_02_out} + {1'b0,B4x4_03_out};
end
always @(posedge clk) begin
    if    (reset)  B8x4_10_out <= 0;
    else if (enable) B8x4_10_out <= {1'b0,B4x4_10_out} + {1'b0,B4x4_11_out};
end
always @(posedge clk) begin
    if    (reset)  B8x4_11_out <= 0;
    else if (enable) B8x4_11_out <= {1'b0,B4x4_12_out} + {1'b0,B4x4_13_out};
end
always @(posedge clk) begin
    if    (reset)  B8x4_20_out <= 0;
    else if (enable) B8x4_20_out <= {1'b0,B4x4_20_out} + {1'b0,B4x4_21_out};
end
always @(posedge clk) begin
    if    (reset)  B8x4_21_out <= 0;
    else if (enable) B8x4_21_out <= {1'b0,B4x4_22_out} + {1'b0,B4x4_23_out};
end
always @(posedge clk) begin
    if    (reset)  B8x4_30_out <= 0;
    else if (enable) B8x4_30_out <= {1'b0,B4x4_30_out} + {1'b0,B4x4_31_out};
end
always @(posedge clk) begin
    if    (reset)  B8x4_31_out <= 0;
    else if (enable) B8x4_31_out <= {1'b0,B4x4_32_out} + {1'b0,B4x4_33_out};
end
```

```
// The 4 8x8 SAD outputs
always @(posedge clk) begin
    if    (reset)  B8x8_00_out <= 0;
    else if (enable) B8x8_00_out <= {1'b0,B8x4_00_out} + {1'b0,B8x4_10_out};
end
always @(posedge clk) begin
    if    (reset)  B8x8_01_out <= 0;
    else if (enable) B8x8_01_out <= {1'b0,B8x4_01_out} + {1'b0,B8x4_11_out};
end
always @(posedge clk) begin
    if    (reset)  B8x8_10_out <= 0;
    else if (enable) B8x8_10_out <= {1'b0,B8x4_20_out} + {1'b0,B8x4_30_out};
end
always @(posedge clk) begin
    if    (reset)  B8x8_11_out <= 0;
    else if (enable) B8x8_11_out <= {1'b0,B8x4_21_out} + {1'b0,B8x4_31_out};
end

// The 2 8x16 SAD outputs
always @(posedge clk) begin
    if    (reset)  B8x16_0_out <= 0;
    else if (enable) B8x16_0_out <= {1'b0,B8x8_10_out} + {1'b0,B8x8_00_delayed};
end
always @(posedge clk) begin
    if    (reset)  B8x16_1_out <= 0;
    else if (enable) B8x16_1_out <= {1'b0,B8x8_01_delayed} + {1'b0,B8x8_11_out};
end

// The 2 16x8 SAD outputs
always @(posedge clk) begin
    if    (reset)  B16x8_0_out <= 0;
    else if (enable) B16x8_0_out <= {1'b0,B8x8_00_out} + {1'b0,B8x8_01_out};
end
always @(posedge clk) begin
    if    (reset)  B16x8_1_out <= 0;
    else if (enable) B16x8_1_out <= {1'b0,B8x8_10_out} + {1'b0,B8x8_11_out};
end

// The 1 16x16 SAD output
always @(posedge clk) begin
    if    (reset)  B16x16_out <= 0;
    else if (enable) B16x16_out <= {1'b0,B8x16_0_out} + {1'b0,B8x16_1_out};
end
endmodule
```

```
/*************************************/
/* SINGLE PORT BLOCK RAM  30         */
/* Theepan Moorthy                   */
/* Ryerson University                */
/* Date : February, 2008             */
/*************************************/
/*************************************/
/*                _____              */
/* clk----|>CLK          |           */
/*        |--------------|           */
/* we-->|WE              |          */
/*   di-->|DataW   DataR |-->do      */
/*    a-->|@R_W          |          */
/*        |_____________|           */
/*************************************/
//Only XST supports RAM inference
//Infers Single Port Block Ram

module spblockram_30 (clk,    //input
                      we,     //input
                       a,     //input
                      di,    //input
                       do     //output
                      );


/***** PARAMETERS ******/

parameter PIXEL_ROWS                = 30;
parameter RAM_DATA_WIDTH            = 128;
parameter RAM_SIZE                  = PIXEL_ROWS;
parameter RAM_ADDRESS_WIDTH    = 5; // 0 to 30 addresses


/******   INPUT AND OUTPUT PORTS   ******/

input clk;
input we;
input  [RAM_ADDRESS_WIDTH-1:0] a;
input  [RAM_DATA_WIDTH-1:0]       di;
output [RAM_DATA_WIDTH-1:0]       do;


/***** INTERNAL VARIABLES *****/

reg [RAM_DATA_WIDTH-1:0]    ram [RAM_SIZE-1:0];
reg [RAM_ADDRESS_WIDTH-1:0] read_a;


/************ PROGRAM ***********/

assign do = ram[read_a];

always @(posedge clk) begin
```

```
    if (we) ram[a] <= di;
    read_a <= a;
end

endmodule
```

```
/*************************************/
/* SINGLE PORT BLOCK RAM  66         */
/* Theepan Moorthy                   */
/* Ryerson University                */
/* Date : February, 2008             */
/*************************************/
/*************************************/
/*                 _____________     */
/* clk----|>CLK               |     */
/*        |--------------------|     */
/* we-->|WE                   |     */
/*   di-->|DataW      DataR |-->do   */
/*    a-->|@R_W               |     */
/*        |____________________|     */
/*************************************/
//Only XST supports RAM inference
//Infers Single Port Block Ram

module spblockram_66 (clk,    //input
                      we,     //input
                       a,     //input
                      di,    //input
                       do     //output
                      );


/***** PARAMETERS ******/

parameter PIXEL_ROWS                = 66;

parameter RAM_DATA_WIDTH        = 128;
parameter RAM_SIZE                  = PIXEL_ROWS;
parameter RAM_ADDRESS_WIDTH   = 7; // 0 to 65 addresses


/******   INPUT AND OUTPUT PORTS   ******/

input clk;
input we;
input  [RAM_ADDRESS_WIDTH-1:0] a;
input  [RAM_DATA_WIDTH-1:0]    di;
output [RAM_DATA_WIDTH-1:0]    do;


/***** INTERNAL VARIABLES *****/

reg [RAM_DATA_WIDTH-1:0]       ram [RAM_SIZE-1:0];
reg [RAM_ADDRESS_WIDTH-1:0] read_a;


/************ PROGRAM ***********/

assign do = ram[read_a];
```

```
always @(posedge clk) begin
    if (we) ram[a] <= di;
    read_a <= a;
end

endmodule
```

```
/************************************/
/* On Chip Memory Unit              */
/* Theepan Moorthy                  */
/* Ryerson University               */
/* Date : February, 2008            */
/************************************/

// This module integrates two 66-depth single port memory blocks and
// two 30-depth single port memory blocks to create a single memory unit
// (consisting of both the A & B memory partitions)

module on_chip_mem (
            //control inputs
              clk,
              reset,
              enable,
                write_enable,
                address, // 6-bits (i.e. 0 to 47 search pixel rows)
                read_Buf_Counter,
                toggle,
                load_top,

                read_L0_L1,
                read_L1_L2,
                read_L2_L3,

                // Input Bus
                  dataIn_32_Bytes,
                // Output Buses
                   dataOut_31_Bytes_A,
                   dataOut_31_Bytes_B
                 );

/******   INPUT AND OUTPUT PORTS   ******/

input clk, reset, enable;
input write_enable;
input [5:0] address;
input [6:0] read_Buf_Counter;
input toggle;

input load_top;
input read_L0_L1;
input read_L1_L2;
input read_L2_L3;

input      [255:0] dataIn_32_Bytes;
output reg [247:0] dataOut_31_Bytes_A;
output reg [247:0] dataOut_31_Bytes_B;


/***** INTERNAL VARIABLES *****/
```

```
wire we_mem_A;
wire we_mem_B;

reg [6:0] mem_1_A_address;
reg [6:0] mem_2_A_address;
reg [4:0] mem_1_B_address;
reg [4:0] mem_2_B_address;

wire [127:0] mem_1_A_out;
wire [127:0] mem_2_A_out;
wire [127:0] mem_1_B_out;
wire [127:0] mem_2_B_out;

reg [5:0] partition_A_Counter;
reg [5:0] nxt_partition_A_Counter;
reg [3:0] partition_B_Counter;
reg [3:0] nxt_partition_B_Counter;

/***************IMPLEMENTATION ********************************/

assign we_mem_A = (write_enable &&   (address<33)) ? 1'b1 : 1'b0;// the we signals are asserted/deasserted
assign we_mem_B = (write_enable && (!(address<33)))? 1'b1 : 1'b0;// one cycle ahead so that they are set and ac-
tive before
                                        // the actual pos edge at which they're required

//------------- Memory Blocks' Output Muxing ------------------------//

always @(*) begin
     if (
          (  (read_L1_L2)&&(address!=33) )
                    ||
          (  (read_L2_L3)&&(address==18) )
        )
          dataOut_31_Bytes_A = {mem_2_A_out,mem_1_A_out[127:8]};
     else dataOut_31_Bytes_A = {mem_1_A_out,mem_2_A_out[127:8]};
end
always @(*) begin
     //if ((read_L2_L3)&&(address>18)&&(read_Buf_Counter<100)) dataOut_31_Bytes_B =
{mem_2_B_out,mem_1_B_out[127:8]};
     if ((read_L2_L3)&&(address>18)) dataOut_31_Bytes_B = {mem_2_B_out,mem_1_B_out[127:8]};
     else                  dataOut_31_Bytes_B = {mem_1_B_out,mem_2_B_out[127:8]};
end

//------------- Address Assignments for the memory Blocks ------------------------//

always @(*) begin
     if (((we_mem_A)&&(load_top))||(!read_L0_L1)) mem_1_A_address = partition_A_Counter + 33;
     else                          mem_1_A_address = partition_A_Counter;
end
always @(*) begin
     if (((we_mem_A)&&(load_top))||(read_L2_L3)) mem_2_A_address = partition_A_Counter + 33;
     else                          mem_2_A_address = partition_A_Counter;
end
always @(*) begin
```

```
    if (((we_mem_B)&&(load_top))||(read_L2_L3)) mem_1_B_address = partition_B_Counter + 15;
    else                          mem_1_B_address = partition_B_Counter;
end
always @(*) begin
    if (
       (   (we_mem_B)&&(load_top)   )
                ||
       ( (read_L2_L3)&&(address<18) )
      )
        mem_2_B_address = partition_B_Counter + 15;
    else mem_2_B_address = partition_B_Counter;
end

//------------- Address Counters for the memory Blocks ------------------------//

always @(posedge clk) begin
    if (reset) partition_A_Counter <= 63; // 63 is used so that the address rolls
                                  // over to 0 when enable goes high
    else       partition_A_Counter <= nxt_partition_A_Counter;
end
always @(*) begin
    if (!enable) nxt_partition_A_Counter = 63;
    else
        if ( (partition_A_Counter==32)
                ||
            (write_enable&&(address==0))
                ||
            (read_Buf_Counter==114)
                ||
            toggle
           )
            nxt_partition_A_Counter = 0;
        else nxt_partition_A_Counter = partition_A_Counter + 1;
end
always @(posedge clk) begin
    if (reset) partition_B_Counter <= 15;
    else       partition_B_Counter <= nxt_partition_B_Counter;
end
always @(*) begin
    if (!enable) nxt_partition_B_Counter = 15;
    else
        if ( (partition_B_Counter==14)
                ||
            (partition_A_Counter==32)
                ||
            (read_Buf_Counter==114)
           )
            nxt_partition_B_Counter = 0;
        else nxt_partition_B_Counter = partition_B_Counter + 1;
end

//------------- Instantiations ------------------------//

spblockram_66 mem_1_A (.clk(clk),
```

```
                .we(we_mem_A),
               .a(mem_1_A_address),
                .di(dataIn_32_Bytes[255:128]),
               .do(mem_1_A_out)
                );

spblockram_66 mem_2_A (.clk(clk),
                .we(we_mem_A),
               .a(mem_2_A_address),
                .di(dataIn_32_Bytes[127:0]),
               .do(mem_2_A_out)
                );

spblockram_30 mem_1_B (.clk(clk),
                .we(we_mem_B),
               .a(mem_1_B_address),
                .di(dataIn_32_Bytes[255:128]),
               .do(mem_1_B_out)
                );

spblockram_30 mem_2_B (.clk(clk),
                .we(we_mem_B),
               .a(mem_2_B_address),
                .di(dataIn_32_Bytes[127:0]),
               .do(mem_2_B_out)
                );

endmodule
```

```
/*************************************/
/* Dual Buffer Unit                  */
/* Theepan Moorthy                   */
/* Ryerson University                */
/* Date : February, 2008             */
/*************************************/

// This module integrates two Memory Modules to form a Double Buffer

module dual_buffer (
                //control inputs
                 clk,
                 reset,
                 enable,
                   address, // 6-bits (i.e. 0 to 47 search pixel rows)
                   read_Buffer_1,
                   // Input Bus
                     dataIn_32_Bytes,
                  // Output Buses
                     dataOut_31_Bytes_A,
                     dataOut_31_Bytes_B
                   );

/******   INPUT AND OUTPUT PORTS   ******/

input clk, reset, enable;
input [5:0] address;
input read_Buffer_1;
input      [255:0] dataIn_32_Bytes;
output reg [247:0] dataOut_31_Bytes_A;
output reg [247:0] dataOut_31_Bytes_B;


/***** INTERNAL VARIABLES *****/

//wire enable_B1;
//wire enable_B2;
wire we_B1;
wire we_B2;
wire load_top;

wire read_L0_L1;
wire read_L1_L2;
wire read_L2_L3;

reg [6:0] read_Buff_Counter;
reg [6:0] nxt_read_Buff_Counter;

reg last_read;
wire toggle;

wire [247:0] B1_Out_A;
wire [247:0] B1_Out_B;
```

```
wire [247:0] B2_Out_A;
wire [247:0] B2_Out_B;


/*******************IMPLEMENTATION ***********************************/

//assign enable_B1 = ((!read_Buffer_1) && (read_Buff_Counter<96)) ? 1'b1 : 1'b0;
//assign enable_B2 = (  read_Buffer_1  && (read_Buff_Counter<96)) ? 1'b1 : 1'b0;
assign we_B1 = ((!read_Buffer_1) && (read_Buff_Counter<96)) ? 1'b1 : 1'b0;
assign we_B2 = (  read_Buffer_1  && (read_Buff_Counter<96)) ? 1'b1 : 1'b0;

assign read_L0_L1 = (read_Buff_Counter<33) ? 1'b1 : 1'b0;
assign read_L1_L2 = ((!read_L0_L1) && (read_Buff_Counter<66)) ? 1'b1 : 1'b0;
assign read_L2_L3 =                         (!(read_Buff_Counter<66)) ? 1'b1 : 1'b0;

assign load_top = (!(read_Buff_Counter<48)) ? 1'b1 : 1'b0;

//------------- Output Muxing ------------------------//

always @(*) begin
     if (read_Buffer_1) dataOut_31_Bytes_A = B1_Out_A;
     else          dataOut_31_Bytes_A = B2_Out_A;
end
always @(*) begin
     if (read_Buffer_1) dataOut_31_Bytes_B = B1_Out_B;
     else          dataOut_31_Bytes_B = B2_Out_B;
end

//------------- Read Buffer Counters ------------------------//

always @(posedge clk) begin
     if (reset) read_Buff_Counter <= 127;
     else       read_Buff_Counter <= nxt_read_Buff_Counter;
end
always @(*) begin
     if (!enable) nxt_read_Buff_Counter = 127;
     else
          if (   (read_Buff_Counter==114) || toggle)
              //(!(read_Buffer_1==last_read)) )
              nxt_read_Buff_Counter = 0;
          else nxt_read_Buff_Counter = read_Buff_Counter + 1;
end

always @(posedge clk) begin
     if (reset) last_read <= 0;
     else       last_read <= read_Buffer_1;
end

assign toggle = (!(read_Buffer_1==last_read)) ? 1 : 0;

//------------- Instantiations ------------------------//

on_chip_mem buff_1 (.clk(clk),
                    .reset(reset),
```

```
                .enable(enable),
               .write_enable(we_B1),
               .address(address),
               .read_Buf_Counter(read_Buff_Counter),
               .toggle(toggle),
               .load_top(load_top),
               .read_L0_L1(read_L0_L1),
               .read_L1_L2(read_L1_L2),
               .read_L2_L3(read_L2_L3),
               .dataIn_32_Bytes(dataIn_32_Bytes),
               .dataOut_31_Bytes_A(B1_Out_A),
               .dataOut_31_Bytes_B(B1_Out_B)
               );

on_chip_mem buff_2 (.clk(clk),
                .reset(reset),
                .enable(enable),
               .write_enable(we_B2),
               .address(address),
               .read_Buf_Counter(read_Buff_Counter),
               .toggle(toggle),
               .load_top(load_top),
               .read_L0_L1(read_L0_L1),
               .read_L1_L2(read_L1_L2),
               .read_L2_L3(read_L2_L3),
               .dataIn_32_Bytes(dataIn_32_Bytes),
               .dataOut_31_Bytes_A(B2_Out_A),
               .dataOut_31_Bytes_B(B2_Out_B)
               );

endmodule
```

```
/************************
 * Theepan Moorthy      *
 * Ryerson University   *
 * Copyright July 2007  *
 * test_PE_AD.v         *
 ************************/

// This Module is used as a Test Bench for the PE_AD module

module test_PE_AD;


/***** PARAMETERS ******/

parameter CLKPERIOD = 2;


//------------- Test Bench Connections ----------------//

reg clk, reset, enable;
reg  [7:0] Cur_Pel;
reg  [7:0] Ref_Pel;

wire [7:0] AD;

//------------- Variables ----------------//

integer Cur_Luminance;
integer Ref_Luminance;

//------------- Instantiation ----------------//

PE_AD DUT (.clk(clk),
        .reset(reset),
        .enable(enable),
        .C(Cur_Pel),
        .R(Ref_Pel),
        .AD(AD)
        );

initial begin
end

/***** CLOCK GENERATION ******/
initial
begin
      clk = 1'b0;
end

always #(CLKPERIOD/2) clk = ~clk;


/***** THIS TASK CREATES A POSITIVE RESET FOR n CLOCK CYCLES *****/
```

```
task reset_for_n_clocks;
input [7:0] number_of_resets;
begin
      @(posedge clk);
      reset = 1'b1;     //activates the positive reset signal
      repeat(number_of_resets) @(posedge clk);
      reset=1'b0;             //remove the reset
end
endtask


/***** THIS TASK CREATES A DELAY FOR n CLOCK CYCLES *****/
task delay_for_n_clocks;
input [15:0] number_of_clock_delays;
begin
      repeat(number_of_clock_delays) @(posedge clk);
end
endtask


/***** THIS TASK INITIALIZES THE FOLLOWING INPUT SIGNALS *****/
task init_inputs;
begin
      clk    = 0;
      reset  = 0;
      enable = 0;

      Cur_Pel = 0;
      Ref_Pel = 0;
end
endtask


/******   RUN THE TEST ******/
initial
begin
      init_inputs;
      reset_for_n_clocks(4);
      delay_for_n_clocks(2);
      enable = 1'b1;
      @(posedge clk);
      for(Cur_Luminance=0; Cur_Luminance<256; Cur_Luminance= Cur_Luminance + 1) begin
            Cur_Pel = Cur_Luminance;
            for(Ref_Luminance=0; Ref_Luminance<256; Ref_Luminance= Ref_Luminance + 1) begin
                  Ref_Pel = Ref_Luminance;
                  @(posedge clk);
            end //ends Ref_Luminance for loop
      end //ends Cur_Luminance for loop

end

endmodule
```

```
/************************
 * Theepan Moorthy      *
 * Ryerson University   *
 * Copyright July 2007  *
 * test_Row_Adder_0.v   *
 ************************/

// This Module is used as a Test Bench for the Row_Adder_0 module

module test_Row_Adder_0;


/***** PARAMETERS ******/

parameter CLKPERIOD = 2;


//------------- Test Bench Connections ---------------------//

reg clk, reset, enable;
reg  [7:0] Cur_Pel_1;
reg  [7:0] Cur_Pel_2;
reg  [7:0] Cur_Pel_3;
reg  [7:0] Cur_Pel_4;
reg  [7:0] Ref_Pel_1;
reg  [7:0] Ref_Pel_2;
reg  [7:0] Ref_Pel_3;
reg  [7:0] Ref_Pel_4;

wire [9:0] Partial_SAD;

//------------- Variables ----------------------//

integer Cur_Luminance;
integer Ref_Luminance;


//------------- Instantiation ----------------------//

Row_Adder_0 DUT (.clk(clk),
            .reset(reset),
            .enable(enable),
            .C1(Cur_Pel_1),
            .C2(Cur_Pel_2),
            .C3(Cur_Pel_3),
            .C4(Cur_Pel_4),
            .R1(Ref_Pel_1),
            .R2(Ref_Pel_2),
            .R3(Ref_Pel_3),
            .R4(Ref_Pel_4),
            .Partial_SAD(Partial_SAD)
            );
```

```
initial begin
end


/***** CLOCK GENERATION ******/
initial
begin
      clk = 1'b0;
end

always #(CLKPERIOD/2) clk = ~clk;


/***** THIS TASK CREATES A POSITIVE RESET FOR n CLOCK CYCLES *****/
task reset_for_n_clocks;
input [7:0] number_of_resets;
begin
      @(posedge clk);
      reset = 1'b1;     //activates the positive reset signal
      repeat(number_of_resets) @(posedge clk);
      reset=1'b0;             //remove the reset
end
endtask


/***** THIS TASK CREATES A DELAY FOR n CLOCK CYCLES *****/
task delay_for_n_clocks;
input [15:0] number_of_clock_delays;
begin
      repeat(number_of_clock_delays) @(posedge clk);
end
endtask


/***** THIS TASK INITIALIZES THE FOLLOWING INPUT SIGNALS *****/
task init_inputs;
begin
      clk    = 0;
      reset  = 0;
      enable = 0;

      Cur_Pel_1 = 0;
      Cur_Pel_2 = 0;
      Cur_Pel_3 = 0;
      Cur_Pel_4 = 0;
      Ref_Pel_1 = 0;
      Ref_Pel_2 = 0;
      Ref_Pel_3 = 0;
      Ref_Pel_4 = 0;
end
endtask


/******   RUN THE TEST ******/
```

```
initial
begin
	init_inputs;
	reset_for_n_clocks(4);
	delay_for_n_clocks(2);
	enable = 1'b1;
	for(Cur_Luminance=0; Cur_Luminance<256; Cur_Luminance= Cur_Luminance + 1) begin
		Cur_Pel_1 = Cur_Luminance;
		Cur_Pel_2 = Cur_Luminance;
		Cur_Pel_3 = Cur_Luminance;
		Cur_Pel_4 = Cur_Luminance;
		for(Ref_Luminance=0; Ref_Luminance<256; Ref_Luminance= Ref_Luminance + 1) begin
			Ref_Pel_1 = Ref_Luminance;
			Ref_Pel_2 = Ref_Luminance;
			Ref_Pel_3 = Ref_Luminance;
			Ref_Pel_4 = Ref_Luminance;
			@(posedge clk);
		end //ends Ref_Luminance for loop
	end //ends Cur_Luminance for loop

end


endmodule
```

```
/***************************
 * Theepan Moorthy         *
 * Ryerson University      *
 * Copyright July 2007     *
 * test_Row_Adder_1_CSA.v  *
 ***************************/

// This Module is used as a Test Bench for the Row_Adder_1_CSA module

module test_Row_Adder_1_CSA;


/***** PARAMETERS ******/

parameter CLKPERIOD = 2;


//------------- Test Bench Connections ---------------------//

reg clk, reset, enable;
reg [7:0] Cur_Pel_1;
reg [7:0] Cur_Pel_2;
reg [7:0] Cur_Pel_3;
reg [7:0] Cur_Pel_4;
reg [7:0] Ref_Pel_1;
reg [7:0] Ref_Pel_2;
reg [7:0] Ref_Pel_3;
reg [7:0] Ref_Pel_4;
reg [9:0] P_SAD_0;

wire [10:0] P_SAD_1;

//------------- Variables ----------------------//

integer Cur_Luminance;
integer Ref_Luminance;


//------------- Instantiation ----------------------//

Row_Adder_1_CSA DUT (.clk(clk),
            .reset(reset),
            .enable(enable),
            .update_Cur_MB(1'b1),
            .C1(Cur_Pel_1),
            .C2(Cur_Pel_2),
            .C3(Cur_Pel_3),
            .C4(Cur_Pel_4),
            .R1(Ref_Pel_1),
            .R2(Ref_Pel_2),
            .R3(Ref_Pel_3),
            .R4(Ref_Pel_4),
            .PSAD_0(P_SAD_0),
```

```
        .PSAD_1(P_SAD_1)
        );

initial begin
end


/***** CLOCK GENERATION ******/
initial
begin
    clk = 1'b0;
end

always #(CLKPERIOD/2) clk = ~clk;


/***** THIS TASK CREATES A POSITIVE RESET FOR n CLOCK CYCLES *****/
task reset_for_n_clocks;
input [7:0] number_of_resets;
begin
    @(posedge clk);
    reset = 1'b1;    //activates the positive reset signal
    repeat(number_of_resets) @(posedge clk);
    reset=1'b0;          //remove the reset
end
endtask


/***** THIS TASK CREATES A DELAY FOR n CLOCK CYCLES *****/
task delay_for_n_clocks;
input [15:0] number_of_clock_delays;
begin
    repeat(number_of_clock_delays) @(posedge clk);
end
endtask


/***** THIS TASK INITIALIZES THE FOLLOWING INPUT SIGNALS *****/
task init_inputs;
begin
    clk    = 0;
    reset  = 0;
    enable = 0;

    Cur_Pel_1 = 149;
    Cur_Pel_2 = 122;
    Cur_Pel_3 = 89;
    Cur_Pel_4 = 228;
    Ref_Pel_1 = 23;
    Ref_Pel_2 = 92;
    Ref_Pel_3 = 37;
    Ref_Pel_4 = 42;
    P_SAD_0   = 425;
end
```

```
endtask

/******   RUN THE TEST ******/
initial
begin
      init_inputs;
      reset_for_n_clocks(4);
      delay_for_n_clocks(2);
      enable = 1'b1;
      for(Cur_Luminance=0; Cur_Luminance<256; Cur_Luminance= Cur_Luminance + 1) begin
            Cur_Pel_1 = Cur_Luminance;
            Cur_Pel_2 = Cur_Luminance;
            Cur_Pel_3 = Cur_Luminance;
            Cur_Pel_4 = Cur_Luminance;
            for(Ref_Luminance=0; Ref_Luminance<256; Ref_Luminance= Ref_Luminance + 1) begin
                  Ref_Pel_1 = Ref_Luminance;
                  Ref_Pel_2 = Ref_Luminance;
                  Ref_Pel_3 = Ref_Luminance;
                  Ref_Pel_4 = Ref_Luminance;
                  @(posedge clk);
            end //ends Ref_Luminance for loop
      end //ends Cur_Luminance for loop

end
endmodule
```

```
/***************************
 * Theepan Moorthy         *
 * Ryerson University      *
 * Copyright July 2007     *
 * test_Row_Adder_2_CSA.v  *
 ***************************/

// This Module is used as a Test Bench for the Row_Adder_2_CSA module

module test_Row_Adder_2_CSA;


/***** PARAMETERS ******/

parameter CLKPERIOD = 2;


//------------- Test Bench Connections ---------------------//

reg clk, reset, enable;
reg [7:0]  Cur_Pel_1;
reg [7:0]  Cur_Pel_2;
reg [7:0]  Cur_Pel_3;
reg [7:0]  Cur_Pel_4;
reg [7:0]  Ref_Pel_1;
reg [7:0]  Ref_Pel_2;
reg [7:0]  Ref_Pel_3;
reg [7:0]  Ref_Pel_4;
reg [10:0] P_SAD_1;

wire [11:0] P_SAD_2;

//------------- Variables ----------------------//

integer Cur_Luminance_1;
integer Cur_Luminance_2;
integer Cur_Luminance_3;
integer Cur_Luminance_4;
integer Ref_Luminance_1;
integer Ref_Luminance_2;
integer Ref_Luminance_3;
integer Ref_Luminance_4;
integer P_SAD_1_Value;

integer AB_1;
integer AB_2;
integer AB_3;
integer AB_4;
reg [11:0] Addition_Compare;

//------------- Instantiation ----------------------//

Row_Adder_2_CSA DUT (.clk(clk),
```

```
        .reset(reset),
        .enable(enable),
        .C1(Cur_Pel_1),
        .C2(Cur_Pel_2),
        .C3(Cur_Pel_3),
        .C4(Cur_Pel_4),
        .R1(Ref_Pel_1),
        .R2(Ref_Pel_2),
        .R3(Ref_Pel_3),
        .R4(Ref_Pel_4),
        .P_SAD_1(P_SAD_1),
        .P_SAD_2(P_SAD_2)
        );


/***** CLOCK GENERATION ******/
initial
begin
    clk = 1'b0;
end

always #(CLKPERIOD/2) clk = ~clk;


/***** THIS TASK CREATES A POSITIVE RESET FOR n CLOCK CYCLES *****/
task reset_for_n_clocks;
input [7:0] number_of_resets;
begin
    @(posedge clk);
    reset = 1'b1;    //activates the positive reset signal
    repeat(number_of_resets) @(posedge clk);
    reset=1'b0;          //remove the reset
end
endtask


/***** THIS TASK CREATES A DELAY FOR n CLOCK CYCLES *****/
task delay_for_n_clocks;
input [15:0] number_of_clock_delays;
begin
    repeat(number_of_clock_delays) @(posedge clk);
end
endtask

/***** THIS EVENT WILL END THE SIMULATION WHEN CALLED UPON *****/
event terminate_sim;
initial begin
    @ (terminate_sim);
    $finish;
end

/***** THIS TASK INITIALIZES THE FOLLOWING INPUT SIGNALS *****/
task init_inputs;
begin
```

```
    clk    = 0;
    reset  = 0;
    enable = 0;

    Cur_Pel_1 = 0;
    Cur_Pel_2 = 0;
    Cur_Pel_3 = 0;
    Cur_Pel_4 = 0;
    Ref_Pel_1 = 0;
    Ref_Pel_2 = 0;
    Ref_Pel_3 = 0;
    Ref_Pel_4 = 0;
    P_SAD_1   = 0;
end
endtask


/******   RUN THE TEST ******/
initial
begin
    init_inputs;
    reset_for_n_clocks(4);
    delay_for_n_clocks(2);
    enable = 1'b1;
/*
    for(P_SAD_1_Value=2040; P_SAD_1_Value<2048; P_SAD_1_Value= P_SAD_1_Value + 1) begin
        P_SAD_1 = P_SAD_1_Value;
    for(Cur_Luminance_1=0; Cur_Luminance_1<1; Cur_Luminance_1= Cur_Luminance_1 + 1) begin
        Cur_Pel_1 = Cur_Luminance_1;
    for(Cur_Luminance_2=0; Cur_Luminance_2<1; Cur_Luminance_2= Cur_Luminance_2 + 1) begin
        Cur_Pel_2 = Cur_Luminance_2;
    for(Cur_Luminance_3=0; Cur_Luminance_3<1; Cur_Luminance_3= Cur_Luminance_3 + 1) begin
        Cur_Pel_3 = Cur_Luminance_3;
    for(Cur_Luminance_4=0; Cur_Luminance_4<1; Cur_Luminance_4= Cur_Luminance_4 + 1) begin
        Cur_Pel_4 = Cur_Luminance_4;
        for(Ref_Luminance_1=255; Ref_Luminance_1<256; Ref_Luminance_1= Ref_Luminance_1 + 1) begin

          Ref_Pel_1 = Ref_Luminance_1;
          for(Ref_Luminance_2=255; Ref_Luminance_2<256; Ref_Luminance_2= Ref_Luminance_2 + 1) begin

            Ref_Pel_2 = Ref_Luminance_2;
            for(Ref_Luminance_3=255; Ref_Luminance_3<256; Ref_Luminance_3= Ref_Luminance_3 + 1) begin

              Ref_Pel_3 = Ref_Luminance_3;
              for(Ref_Luminance_4=240; Ref_Luminance_4<256; Ref_Luminance_4= Ref_Luminance_4 + 1) begin

               Ref_Pel_4 = Ref_Luminance_4;
               if (Cur_Luminance_1>Ref_Luminance_1) AB_1 = Cur_Luminance_1 - Ref_Luminance_1;
               else AB_1 = Ref_Luminance_1 - Cur_Luminance_1;
               if (Cur_Luminance_2>Ref_Luminance_2) AB_2 = Cur_Luminance_2 - Ref_Luminance_2;
               else AB_2 = Ref_Luminance_2 - Cur_Luminance_2;
               if (Cur_Luminance_3>Ref_Luminance_3) AB_3 = Cur_Luminance_3 - Ref_Luminance_3;
               else AB_3 = Ref_Luminance_3 - Cur_Luminance_3;
               if (Cur_Luminance_4>Ref_Luminance_4) AB_4 = Cur_Luminance_4 - Ref_Luminance_4;
```

```
                else AB_4 = Ref_Luminance_4 - Cur_Luminance_4;
                @(posedge clk);
               end //Ref_Luminance_4 loop
              end //Ref_Luminance_3 loop
             end //Ref_Luminance_2 loop
            end //Ref_Luminance_1 loop
      end //Cur_Luminance_4 loop
      end //Cur_Luminance_3 loop
      end //Cur_Luminance_2 loop
      end //Cur_Luminance_1 loop
      end // P_SAD_Value_1 loop
*/

Cur_Pel_1 = 1;
Cur_Pel_2 = 1;
Cur_Pel_3 = 1;
Cur_Pel_4 = 1;

Ref_Pel_1 = 255;
Ref_Pel_2 = 255;
Ref_Pel_3 = 255;
Ref_Pel_4 = 255;
P_SAD_1   = 2047;

end


always @(posedge clk, reset, enable) begin
      if    (reset  == 1'b1) Addition_Compare <= 0;
      else if (enable == 1'b1) Addition_Compare <= AB_1 + AB_2 + AB_3 + AB_4 + P_SAD_1;
end

always @ (posedge clk) begin
      //$display ("P_SAD_2 $d", P_SAD_2);
      if (Addition_Compare != P_SAD_2) begin
            //$display ("DUT Error at time %d", $time);
            //$display ("Expected value %d, Got Value %d", Addition_Compare, P_SAD_2);
            //-> terminate_sim;
      end
end

endmodule
```

```
/************************** *
 * Theepan Moorthy         *
 * Ryerson University      *
 * Copyright July 2007     *
 * test_Row_Adder_3_CSA.v  *
 ***************************/

// This Module is used as a Test Bench for the Row_Adder_3_CSA module

module test_Row_Adder_3_CSA;


/***** PARAMETERS ******/

parameter CLKPERIOD = 2;


//------------- Test Bench Connections ----------------//

reg clk, reset, enable;
reg update_Cur_MB;
reg [7:0]  Cur_Pel_1;
reg [7:0]  Cur_Pel_2;
reg [7:0]  Cur_Pel_3;
reg [7:0]  Cur_Pel_4;
reg [7:0]  Ref_Pel_1;
reg [7:0]  Ref_Pel_2;
reg [7:0]  Ref_Pel_3;
reg [7:0]  Ref_Pel_4;
reg [11:0] PSAD_2;

wire [12:0] PSAD_3;

//------------- Variables ----------------//

integer Cur_Luminance_1;
integer Cur_Luminance_2;
integer Cur_Luminance_3;
integer Cur_Luminance_4;
integer Ref_Luminance_1;
integer Ref_Luminance_2;
integer Ref_Luminance_3;
integer Ref_Luminance_4;
integer PSAD_2_Value;

integer AB_1;
integer AB_2;
integer AB_3;
integer AB_4;
reg [12:0] Addition_Compare;

//------------- Instantiation ----------------//

Row_Adder_3_CSA DUT (.clk(clk),
```

```
            .reset(reset),
            .enable(enable),
             .update_Cur_MB(update_Cur_MB),
            .C1(Cur_Pel_1),
            .C2(Cur_Pel_2),
            .C3(Cur_Pel_3),
            .C4(Cur_Pel_4),
            .R1(Ref_Pel_1),
            .R2(Ref_Pel_2),
            .R3(Ref_Pel_3),
            .R4(Ref_Pel_4),
            .PSAD_2(PSAD_2),
            .PSAD_3(PSAD_3)
            );


/***** CLOCK GENERATION ******/
initial
begin
    clk = 1'b0;
end

always #(CLKPERIOD/2) clk = ~clk;


/***** THIS TASK CREATES A POSITIVE RESET FOR n CLOCK CYCLES *****/
task reset_for_n_clocks;
input [7:0] number_of_resets;
begin
    @(posedge clk);
    reset = 1'b1;    //activates the positive reset signal
    repeat(number_of_resets) @(posedge clk);
    reset=1'b0;          //remove the reset
end
endtask


/***** THIS TASK CREATES A DELAY FOR n CLOCK CYCLES *****/
task delay_for_n_clocks;
input [15:0] number_of_clock_delays;
begin
    repeat(number_of_clock_delays) @(posedge clk);
end
endtask

/***** THIS EVENT WILL END THE SIMULATION WHEN CALLED UPON *****/
event terminate_sim;
initial begin
    @ (terminate_sim);
    $finish;
end

/***** THIS TASK INITIALIZES THE FOLLOWING INPUT SIGNALS *****/
task init_inputs;
```

```
begin
      clk    = 0;
      reset  = 0;
      enable = 0;
      update_Cur_MB = 0;

      Cur_Pel_1 = 0;
      Cur_Pel_2 = 0;
      Cur_Pel_3 = 0;
      Cur_Pel_4 = 0;
      Ref_Pel_1 = 0;
      Ref_Pel_2 = 0;
      Ref_Pel_3 = 0;
      Ref_Pel_4 = 0;
      PSAD_2    = 0;
end
endtask


/******   RUN THE TEST ******/
initial
begin
      init_inputs;
      reset_for_n_clocks(4);
      delay_for_n_clocks(2);
      enable = 1'b1;
      update_Cur_MB = 1'b1;
/*
      for(PSAD_2_Value=2040; PSAD_2_Value<2048; PSAD_2_Value= PSAD_2_Value + 1) begin
            PSAD_2 = PSAD_2_Value;
      for(Cur_Luminance_1=0; Cur_Luminance_1<1; Cur_Luminance_1= Cur_Luminance_1 + 1) begin
            Cur_Pel_1 = Cur_Luminance_1;
      for(Cur_Luminance_2=0; Cur_Luminance_2<1; Cur_Luminance_2= Cur_Luminance_2 + 1) begin
            Cur_Pel_2 = Cur_Luminance_2;
      for(Cur_Luminance_3=0; Cur_Luminance_3<1; Cur_Luminance_3= Cur_Luminance_3 + 1) begin
            Cur_Pel_3 = Cur_Luminance_3;
      for(Cur_Luminance_4=0; Cur_Luminance_4<1; Cur_Luminance_4= Cur_Luminance_4 + 1) begin
            Cur_Pel_4 = Cur_Luminance_4;
            for(Ref_Luminance_1=255; Ref_Luminance_1<256; Ref_Luminance_1= Ref_Luminance_1 + 1) begin

              Ref_Pel_1 = Ref_Luminance_1;
              for(Ref_Luminance_2=255; Ref_Luminance_2<256; Ref_Luminance_2= Ref_Luminance_2 + 1) begin

                Ref_Pel_2 = Ref_Luminance_2;
                for(Ref_Luminance_3=255; Ref_Luminance_3<256; Ref_Luminance_3= Ref_Luminance_3 + 1) begin

                  Ref_Pel_3 = Ref_Luminance_3;
                  for(Ref_Luminance_4=240; Ref_Luminance_4<256; Ref_Luminance_4= Ref_Luminance_4 + 1) begin

                    Ref_Pel_4 = Ref_Luminance_4;
                    if (Cur_Luminance_1>Ref_Luminance_1) AB_1 = Cur_Luminance_1 - Ref_Luminance_1;
                    else AB_1 = Ref_Luminance_1 - Cur_Luminance_1;
                    if (Cur_Luminance_2>Ref_Luminance_2) AB_2 = Cur_Luminance_2 - Ref_Luminance_2;
                    else AB_2 = Ref_Luminance_2 - Cur_Luminance_2;
```

```
                if (Cur_Luminance_3>Ref_Luminance_3) AB_3 = Cur_Luminance_3 - Ref_Luminance_3;
                else AB_3 = Ref_Luminance_3 - Cur_Luminance_3;
                if (Cur_Luminance_4>Ref_Luminance_4) AB_4 = Cur_Luminance_4 - Ref_Luminance_4;
                else AB_4 = Ref_Luminance_4 - Cur_Luminance_4;
                @(posedge clk);
               end //Ref_Luminance_4 loop
             end //Ref_Luminance_3 loop
           end //Ref_Luminance_2 loop
         end //Ref_Luminance_1 loop
    end //Cur_Luminance_4 loop
    end //Cur_Luminance_3 loop
    end //Cur_Luminance_2 loop
    end //Cur_Luminance_1 loop
    end // PSAD_Value_1 loop
*/

Cur_Pel_1 = 0;
Cur_Pel_2 = 0;
Cur_Pel_3 = 0;
Cur_Pel_4 = 0;

Ref_Pel_1 = 255;
Ref_Pel_2 = 255;
Ref_Pel_3 = 255;
Ref_Pel_4 = 255;
PSAD_2    = 4095;
end

always @(posedge clk, reset, enable) begin
    if    (reset  == 1'b1) Addition_Compare <= 0;
    else if (enable == 1'b1) Addition_Compare <= AB_1 + AB_2 + AB_3 + AB_4 + PSAD_2;
end

always @ (posedge clk) begin
    //$display ("PSAD_3 $d", PSAD_3);
    if (Addition_Compare != PSAD_3) begin
        //$display ("DUT Error at time %d", $time);
        //$display ("Expected value %d, Got Value %d", Addition_Compare, PSAD_3);
        //-> terminate_sim;
    end
end
endmodule
```

```
/***************************
 * Theepan Moorthy         *
 * Ryerson University      *
 * Copyright Octoer 2007   *
 * test_main_data_path.v   *
 ***************************/

// This Module is used as a Test Bench to test the main_data_path

module test_main_data_path;


/***** PARAMETERS ******/

parameter CLKPERIOD = 2;

//------------- Test Bench Connections ----------------//

//------- Input Ports ---------//

reg clk;
reg reset;
reg enable;

reg [4:0] update_Cur_MB;
reg [3:0] bus_Select;

reg [31:0] broadcast_column_1_32bits_A;
reg [31:0] broadcast_column_2_32bits_A;
reg [31:0] broadcast_column_3_32bits_A;
reg [31:0] broadcast_column_4_32bits_A;

reg [31:0] broadcast_column_1_32bits_B;
reg [31:0] broadcast_column_2_32bits_B;
reg [31:0] broadcast_column_3_32bits_B;
reg [31:0] broadcast_column_4_32bits_B;

//------ Output Ports -------//

// The 16 4x4 SAD output registers
wire [12:0]
B4x4_00_out, B4x4_01_out, B4x4_02_out, B4x4_03_out,
B4x4_10_out, B4x4_11_out, B4x4_12_out, B4x4_13_out,
B4x4_20_out, B4x4_21_out, B4x4_22_out, B4x4_23_out,
B4x4_30_out, B4x4_31_out, B4x4_32_out, B4x4_33_out;

// The 8 4x8 SAD output registers
wire [13:0]
B4x8_00_out, B4x8_01_out, B4x8_02_out, B4x8_03_out,
B4x8_10_out, B4x8_11_out, B4x8_12_out, B4x8_13_out;

// The 8 8x4 SAD output registers
wire [13:0]
```

```
B8x4_00_out, B8x4_01_out,
B8x4_10_out, B8x4_11_out,
B8x4_20_out, B8x4_21_out,
B8x4_30_out, B8x4_31_out;

// The 4 8x8 SAD output registers
wire [14:0]
B8x8_00_out, B8x8_01_out,
B8x8_10_out, B8x8_11_out;

// The 2 8x16 SAD output registers
wire [15:0] B8x16_0_out, B8x16_1_out;

// The 2 16x8 SAD output registers
wire [15:0] B16x8_0_out, B16x8_1_out;

// The 1 16x16 SAD output register
wire [16:0] B16x16_out;


//------------- Variables ----------------------//

reg [4:0] read_file_update_Cur_MB [0:1119]; // temp array for $readmemb to store update_Cur_MB input values
reg [3:0] read_file_bus_Select    [0:1119];

reg [31:0] read_file_Bus_A_1 [0:1119999]; // temp array for $readmemb to store bus line A_1's values
reg [31:0] read_file_Bus_A_2 [0:1119999];
reg [31:0] read_file_Bus_A_3 [0:1119999];
reg [31:0] read_file_Bus_A_4 [0:1119999];

reg [31:0] read_file_Bus_B_1 [0:1119999]; // temp array for $readmemb to store bus line B_1's values
reg [31:0] read_file_Bus_B_2 [0:1119999];
reg [31:0] read_file_Bus_B_3 [0:1119999];
reg [31:0] read_file_Bus_B_4 [0:1119999];

reg [12:0] read_file_B4x4_top     [0:8711999]; // temp array for $readmemb to read the C code generated
reg [12:0] read_file_B4x4_bottom  [0:8711999]; // SAD values, to be checked against the RTL model output

reg [13:0] read_file_B4x8_top     [0:4355999];
reg [13:0] read_file_B4x8_bottom  [0:4355999];

reg [13:0] read_file_B8x4_top     [0:4355999];
reg [13:0] read_file_B8x4_bottom  [0:4355999];

reg [14:0] read_file_B8x8_top     [0:2177999];
reg [14:0] read_file_B8x8_bottom  [0:2177999];

reg [15:0] read_file_B16x8_top    [0:1088999];
reg [15:0] read_file_B16x8_bottom [0:1088999];

reg [15:0] read_file_B8x16_bottom [0:2177999];

reg [16:0] read_file_B16x16_bottom [0:1088999];
```

```
integer i; //used for FOR loop indexing
integer B4x4_top_offset; // used to access the correct output C SAD data
integer B8x4_top_offset;
integer B8x8_top_offset;
integer B16x8_top_offset;
integer B4x4_bottom_offset;
integer B8x4_bottom_offset;
integer B8x8_bottom_offset;
integer B16x8_bottom_offset;
integer broadcast_column_offset;
integer TestCase;

//------------- Instantiation ----------------------//

main_data_path DUT (// Input Ports
                .clk(clk),
                .reset(reset),
                .enable(enable),
                 .update_Cur_MB(update_Cur_MB),
                .bus_Select(bus_Select),
                .broadcast_column_1_32bits_A(broadcast_column_1_32bits_A),
                .broadcast_column_2_32bits_A(broadcast_column_2_32bits_A),
                .broadcast_column_3_32bits_A(broadcast_column_3_32bits_A),
                .broadcast_column_4_32bits_A(broadcast_column_4_32bits_A),
                .broadcast_column_1_32bits_B(broadcast_column_1_32bits_B),
                .broadcast_column_2_32bits_B(broadcast_column_2_32bits_B),
                .broadcast_column_3_32bits_B(broadcast_column_3_32bits_B),
                .broadcast_column_4_32bits_B(broadcast_column_4_32bits_B),

                // Output Ports
                // The 16 4x4 SAD output registers

.B4x4_00_out(B4x4_00_out),.B4x4_01_out(B4x4_01_out),.B4x4_02_out(B4x4_02_out),.B4x4_03_out(B4x4_03_
out),

.B4x4_10_out(B4x4_10_out),.B4x4_11_out(B4x4_11_out),.B4x4_12_out(B4x4_12_out),.B4x4_13_out(B4x4_13_
out),

.B4x4_20_out(B4x4_20_out),.B4x4_21_out(B4x4_21_out),.B4x4_22_out(B4x4_22_out),.B4x4_23_out(B4x4_23_
out),

.B4x4_30_out(B4x4_30_out),.B4x4_31_out(B4x4_31_out),.B4x4_32_out(B4x4_32_out),.B4x4_33_out(B4x4_33_
out),
                // The 8 4x8 SAD output registers

.B4x8_00_out(B4x8_00_out),.B4x8_01_out(B4x8_01_out),.B4x8_02_out(B4x8_02_out),.B4x8_03_out(B4x8_03_
out),

.B4x8_10_out(B4x8_10_out),.B4x8_11_out(B4x8_11_out),.B4x8_12_out(B4x8_12_out),.B4x8_13_out(B4x8_13_
out),
                // The 8 8x4 SAD output registers
                .B8x4_00_out(B8x4_00_out),.B8x4_01_out(B8x4_01_out),
                .B8x4_10_out(B8x4_10_out),.B8x4_11_out(B8x4_11_out),
                .B8x4_20_out(B8x4_20_out),.B8x4_21_out(B8x4_21_out),
```

```
            .B8x4_30_out(B8x4_30_out),.B8x4_31_out(B8x4_31_out),
            // The 4 8x8 SAD output registers
            .B8x8_00_out(B8x8_00_out),.B8x8_01_out(B8x8_01_out),
            .B8x8_10_out(B8x8_10_out),.B8x8_11_out(B8x8_11_out),
            // The 2 8x16 SAD output registers
            .B8x16_0_out(B8x16_0_out),.B8x16_1_out(B8x16_1_out),
            // The 2 16x8 SAD output registers
            .B16x8_0_out(B16x8_0_out),.B16x8_1_out(B16x8_1_out),
            // The 1 16x16 SAD output register
            .B16x16_out(B16x16_out)
             );


/***** CLOCK GENERATION ******/
initial
begin
    clk = 1'b0;
end

always #(CLKPERIOD/2) clk = ~clk;


/***** THIS TASK CREATES A POSITIVE RESET FOR n CLOCK CYCLES *****/
task reset_for_n_clocks;
input [7:0] number_of_resets;
begin
    @(posedge clk);
    reset = 1'b1;    //activates the positive reset signal
    repeat(number_of_resets) @(posedge clk);
    reset=1'b0;          //remove the reset
end
endtask


/***** THIS TASK CREATES A DELAY FOR n CLOCK CYCLES *****/
task delay_for_n_clocks;
input [15:0] number_of_clock_delays;
begin
    repeat(number_of_clock_delays) @(posedge clk);
end
endtask

/***** THIS EVENT WILL END THE SIMULATION WHEN CALLED UPON *****/
event terminate_sim;
initial begin
    @ (terminate_sim);
    $finish;
end


/***** THIS TASK INITIALIZES THE FOLLOWING INPUT SIGNALS *****/
task init_inputs;
begin
    clk   = 0;
```

```
      reset  = 0;
      enable = 0;
      update_Cur_MB = 0;
      bus_Select = 0;

       broadcast_column_1_32bits_A = 0;
       broadcast_column_2_32bits_A = 0;
       broadcast_column_3_32bits_A = 0;
       broadcast_column_4_32bits_A = 0;

       broadcast_column_1_32bits_B = 0;
       broadcast_column_2_32bits_B = 0;
       broadcast_column_3_32bits_B = 0;
       broadcast_column_4_32bits_B = 0;
end
endtask



/*****    THIS TASK FEEDS IN ALL OF THE TEST INPUT VECTORS              *****/
/*****    AND CHECKS THE OUTPUT SAD VALUES AGAINST THE C OUTPUT FILES    *****/
task feed_inputs_check_output;
begin
      $readmemb("FILES/update_Cur_MB.txt", read_file_update_Cur_MB); //files must be located in the project
folder
      $readmemb("FILES/bus_Select.txt", read_file_bus_Select);

      $readmemb("FILES_LARGE/bus_A_1.txt", read_file_Bus_A_1); // files must be located in the project folder
      $readmemb("FILES_LARGE/bus_A_2.txt", read_file_Bus_A_2);
      $readmemb("FILES_LARGE/bus_A_3.txt", read_file_Bus_A_3);
      $readmemb("FILES_LARGE/bus_A_4.txt", read_file_Bus_A_4);

      $readmemb("FILES_LARGE/bus_B_1.txt", read_file_Bus_B_1); // files must be located in the project folder
      $readmemb("FILES_LARGE/bus_B_2.txt", read_file_Bus_B_2);
      $readmemb("FILES_LARGE/bus_B_3.txt", read_file_Bus_B_3);
      $readmemb("FILES_LARGE/bus_B_4.txt", read_file_Bus_B_4);

      $readmemb("FILES_LARGE/B4x4_top.txt",  read_file_B4x4_top);
      $readmemb("FILES_LARGE/B4x8_top.txt",  read_file_B4x8_top);
      $readmemb("FILES_LARGE/B8x4_top.txt",  read_file_B8x4_top);
      $readmemb("FILES_LARGE/B8x8_top.txt",  read_file_B8x8_top);
      $readmemb("FILES_LARGE/B16x8_top.txt", read_file_B16x8_top);

      $readmemb("FILES_LARGE/B4x4_bottom.txt",   read_file_B4x4_bottom);
      $readmemb("FILES_LARGE/B4x8_bottom.txt",   read_file_B4x8_bottom);
      $readmemb("FILES_LARGE/B8x4_bottom.txt",   read_file_B8x4_bottom);
      $readmemb("FILES_LARGE/B8x8_bottom.txt",   read_file_B8x8_bottom);
      $readmemb("FILES_LARGE/B16x8_bottom.txt",  read_file_B16x8_bottom);
      $readmemb("FILES_LARGE/B8x16_bottom.txt",  read_file_B8x16_bottom);
      $readmemb("FILES_LARGE/B16x16_bottom.txt", read_file_B16x16_bottom);

      for (TestCase=0; TestCase<1000; TestCase=TestCase+1) begin

      for (i=0; i<1120; i=i+1) begin
```

```
        update_Cur_MB = read_file_update_Cur_MB[i+1];
        bus_Select    = read_file_bus_Select[i];

        broadcast_column_offset = i + (1120*TestCase);
        broadcast_column_1_32bits_A = read_file_Bus_A_1[broadcast_column_offset];
        broadcast_column_2_32bits_A = read_file_Bus_A_2[broadcast_column_offset];
        broadcast_column_3_32bits_A = read_file_Bus_A_3[broadcast_column_offset];
        broadcast_column_4_32bits_A = read_file_Bus_A_4[broadcast_column_offset];

        broadcast_column_1_32bits_B = read_file_Bus_B_1[broadcast_column_offset];
        broadcast_column_2_32bits_B = read_file_Bus_B_2[broadcast_column_offset];
        broadcast_column_3_32bits_B = read_file_Bus_B_3[broadcast_column_offset];
        broadcast_column_4_32bits_B = read_file_Bus_B_4[broadcast_column_offset];


        if ((i>24)&(i<=1113)) begin // since the correct value is only being clocked in at cycle 24
                // if we need to sample that value for testing we should read it at cycle 25,
                // so that the register has settled/stabalized with the value that was clocked into
                // it at the end of cycle 23 (i.e. the instantaneous pos-edge start of cycle 24)

        // Test the Top 8 4x4 Blocks
        B4x4_top_offset = ((i-25)*8) + (8712*TestCase);
        if (read_file_B4x4_top[B4x4_top_offset] != B4x4_00_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B4x4_top[B4x4_top_offset], B4x4_00_out);
        end
        if (read_file_B4x4_top[B4x4_top_offset+1] != B4x4_01_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B4x4_top[B4x4_top_offset+1],
B4x4_01_out);
        end
        if (read_file_B4x4_top[B4x4_top_offset+2] != B4x4_02_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B4x4_top[B4x4_top_offset+2],
B4x4_02_out);
        end
        if (read_file_B4x4_top[B4x4_top_offset+3] != B4x4_03_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B4x4_top[B4x4_top_offset+3],
B4x4_03_out);
        end
        if (read_file_B4x4_top[B4x4_top_offset+4] != B4x4_10_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B4x4_top[B4x4_top_offset+4],
B4x4_10_out);
        end
        if (read_file_B4x4_top[B4x4_top_offset+5] != B4x4_11_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B4x4_top[B4x4_top_offset+5],
B4x4_11_out);
        end
        if (read_file_B4x4_top[B4x4_top_offset+6] != B4x4_12_out) begin
            $display ("DUT Error at time %d", $time);
```

```
            $display ("Expected value %d, Got Value %d", read_file_B4x4_top[B4x4_top_offset+6],
B4x4_12_out);
        end
        if (read_file_B4x4_top[B4x4_top_offset+7] != B4x4_13_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B4x4_top[B4x4_top_offset+7],
B4x4_13_out);
        end

        ////////////////////////////////////////////////////////////////////////////////////////////////
        // Test the Top 4 Half of 8x4 Blocks
        B8x4_top_offset = ((i-25)*4) + (4356*TestCase);
        if (read_file_B8x4_top[B8x4_top_offset] != B8x4_00_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B8x4_top[B8x4_top_offset], B8x4_00_out);
        end
        if (read_file_B8x4_top[B8x4_top_offset+1] != B8x4_01_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B8x4_top[B8x4_top_offset+1],
B8x4_01_out);
        end
        if (read_file_B8x4_top[B8x4_top_offset+2] != B8x4_10_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B8x4_top[B8x4_top_offset+2],
B8x4_10_out);
        end
        if (read_file_B8x4_top[B8x4_top_offset+3] != B8x4_11_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B8x4_top[B8x4_top_offset+3],
B8x4_11_out);
        end

        ////////////////////////////////////////////////////////////////////////////////////////////////
        // Test the Top 4 Half of 4x8 Blocks
        if (read_file_B4x8_top[B8x4_top_offset] != B4x8_00_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B4x8_top[B8x4_top_offset], B4x8_00_out);
        end
        if (read_file_B4x8_top[B8x4_top_offset+1] != B4x8_01_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B4x8_top[B8x4_top_offset+1],
B4x8_01_out);
        end
        if (read_file_B4x8_top[B8x4_top_offset+2] != B4x8_02_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B4x8_top[B8x4_top_offset+2],
B4x8_02_out);
        end
        if (read_file_B4x8_top[B8x4_top_offset+3] != B4x8_03_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B4x8_top[B8x4_top_offset+3],
B4x8_03_out);
        end
```

```
        ////////////////////////////////////////////////////////////////////////////////////////////////
        // Test the Top 2 Half of 8x8 Blocks
        B8x8_top_offset = ((i-25)*2) + (2178*TestCase);
        if (read_file_B8x8_top[B8x8_top_offset] != B8x8_00_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B8x8_top[B8x8_top_offset], B8x8_00_out);
        end
        if (read_file_B8x8_top[B8x8_top_offset+1] != B8x8_01_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B8x8_top[B8x8_top_offset+1],
B8x8_01_out);
        end

        ////////////////////////////////////////////////////////////////////////////////////////////////
        // Test the Top 16x8 Block
        B16x8_top_offset = (i-25) + (1089*TestCase);
        if (read_file_B16x8_top[B16x8_top_offset] != B16x8_0_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B16x8_top[B16x8_top_offset],
B16x8_0_out);
        end

        end // ends if i>24

        ////////////////////////////////////////////////////////////////////////////////////////////////

        if (i>32)begin // since the correct value is only being clocked in at cycle 32
                // if we need to sample that value for testing we should read it at cycle 33,
                // so that the register has settled/stabalized with the value that was clocked into
                // it at the end of cycle 31 (i.e. the instantaneous pos-edge start of cycle 32

        // Test the Bottom 8 Half of 4x4 Blocks
        B4x4_bottom_offset = ((i-33)*8) + (8712*TestCase);
        if (read_file_B4x4_bottom[B4x4_bottom_offset] != B4x4_20_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B4x4_bottom[B4x4_bottom_offset],
B4x4_20_out);
        end
        if (read_file_B4x4_bottom[B4x4_bottom_offset+1] != B4x4_21_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B4x4_bottom[B4x4_bottom_offset+1],
B4x4_21_out);
        end
        if (read_file_B4x4_bottom[B4x4_bottom_offset+2] != B4x4_22_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B4x4_bottom[B4x4_bottom_offset+2],
B4x4_22_out);
        end
        if (read_file_B4x4_bottom[B4x4_bottom_offset+3] != B4x4_23_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B4x4_bottom[B4x4_bottom_offset+3],
B4x4_23_out);
        end
        if (read_file_B4x4_bottom[B4x4_bottom_offset+4] != B4x4_30_out) begin
```

```
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B4x4_bottom[B4x4_bottom_offset+4],
B4x4_30_out);
        end
        if (read_file_B4x4_bottom[B4x4_bottom_offset+5] != B4x4_31_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B4x4_bottom[B4x4_bottom_offset+5],
B4x4_31_out);
        end
        if (read_file_B4x4_bottom[B4x4_bottom_offset+6] != B4x4_32_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B4x4_bottom[B4x4_bottom_offset+6],
B4x4_32_out);
        end
        if (read_file_B4x4_bottom[B4x4_bottom_offset+7] != B4x4_33_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B4x4_bottom[B4x4_bottom_offset+7],
B4x4_33_out);
        end

        ////////////////////////////////////////////////////////////////////////////////////////////////
        // Test the Bottom 4 Half of 8x4 Blocks
        B8x4_top_offset = ((i-33)*4) + (4356*TestCase);
        if (read_file_B8x4_bottom[B8x4_bottom_offset] != B8x4_20_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B8x4_bottom[B8x4_bottom_offset],
B8x4_20_out);
        end
        if (read_file_B8x4_bottom[B8x4_bottom_offset+1] != B8x4_21_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B8x4_bottom[B8x4_bottom_offset+1],
B8x4_21_out);
        end
        if (read_file_B8x4_bottom[B8x4_bottom_offset+2] != B8x4_30_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B8x4_bottom[B8x4_bottom_offset+2],
B8x4_30_out);
        end
        if (read_file_B8x4_bottom[B8x4_bottom_offset+3] != B8x4_31_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B8x4_bottom[B8x4_bottom_offset+3],
B8x4_31_out);
        end

        ////////////////////////////////////////////////////////////////////////////////////////////////
        // Test the Bottom 4 Half of 4x8 Blocks
        if (read_file_B4x8_bottom[B8x4_bottom_offset] != B4x8_10_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B4x8_bottom[B8x4_bottom_offset],
B4x8_10_out);
        end
        if (read_file_B4x8_bottom[B8x4_bottom_offset+1] != B4x8_11_out) begin
            $display ("DUT Error at time %d", $time);
```

```
            $display ("Expected value %d, Got Value %d", read_file_B4x8_bottom[B8x4_bottom_offset+1],
B4x8_11_out);
        end
        if (read_file_B4x8_bottom[B8x4_bottom_offset+2] != B4x8_12_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B4x8_bottom[B8x4_bottom_offset+2],
B4x8_12_out);
        end
        if (read_file_B4x8_bottom[B8x4_bottom_offset+3] != B4x8_13_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B4x8_bottom[B8x4_bottom_offset+3],
B4x8_13_out);
        end

        ////////////////////////////////////////////////////////////////////////////////////////////////
        // Test the Bottom 2 Half of 8x8 Blocks
        B8x8_bottom_offset = ((i-33)*2) + (2178*TestCase);
        if (read_file_B8x8_bottom[B8x8_bottom_offset] != B8x8_10_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B8x8_bottom[B8x8_bottom_offset],
B8x8_10_out);
        end
        if (read_file_B8x8_bottom[B8x8_bottom_offset+1] != B8x8_11_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B8x8_bottom[B8x8_bottom_offset+1],
B8x8_11_out);
        end

        ////////////////////////////////////////////////////////////////////////////////////////////////
        // Test the Bottom 16x8 Block
        B16x8_top_offset = (i-33) + (1089*TestCase);
        if (read_file_B16x8_bottom[B16x8_top_offset] != B16x8_1_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B16x8_bottom[B16x8_top_offset],
B16x8_1_out);
        end

        ////////////////////////////////////////////////////////////////////////////////////////////////
        // Test the two 8x16 Blocks
        if (read_file_B8x16_bottom[B8x8_bottom_offset] != B8x16_0_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B8x16_bottom[B8x8_bottom_offset],
B8x16_0_out);
        end
        if (read_file_B8x16_bottom[B8x8_bottom_offset+1] != B8x16_1_out) begin
            $display ("DUT Error at time %d", $time);
            $display ("Expected value %d, Got Value %d", read_file_B8x16_bottom[B8x8_bottom_offset+1],
B8x16_1_out);
        end

        ////////////////////////////////////////////////////////////////////////////////////////////////
        // Test the 16x16 Block
        if (read_file_B16x16_bottom[B16x8_top_offset] != B16x16_out) begin
            $display ("DUT Error at time %d", $time);
```

```
            $display ("Expected value %d, Got Value %d", read_file_B16x16_bottom[B16x8_top_offset],
B16x16_out);
        end

        end // ends if i>32

        @(posedge clk);
    end //ends the i clock cycle for loop
    $display("Input Values have been Fed \n");
    $display("TestCase = %d\n", TestCase);
    update_Cur_MB = 0;
    bus_Select = 0;
    reset_for_n_clocks(2);
    end //ends the 1000 TestCase for loop
end
endtask


/******   RUN THE MAIN_DATA_PATH ******/
initial
begin
    init_inputs;
    reset_for_n_clocks(4);
    delay_for_n_clocks(2);
    enable = 1'b1;
    $display("\n **************DATA-PATH SAD RESULTS*************** \n");
    feed_inputs_check_output;
    $display("");
    $display("**********************************************************************");
    enable = 1'b0;
    delay_for_n_clocks(5);
    //-> terminate_sim;
end
endmodule
```

```
/***************************
 * Theepan Moorthy         *
 * Ryerson University      *
 * Copyright February 2008 *
 * test_dual_buffer.v      *
 ***************************/

// This Module is used as a Test Bench to test the dual_buffer module

module test_dual_buffer;


/***** PARAMETERS ******/

parameter CLKPERIOD = 2;

//------------- Test Bench Connections ---------------------//

//------- Input Ports ---------//

reg clk;
reg reset;
reg enable;

reg [5:0] address;
reg read_Buffer_1;

reg [255:0] dataIn_32_Bytes;

//------ Output Ports -------//

wire [247:0] dataOut_31_Bytes_A;
wire [247:0] dataOut_31_Bytes_B;

reg [6:0] a;
reg we_a;
reg [255:0] a_di;
wire [247:0] a_do;

//------------- Variables ---------------------//

reg [7:0] input_32_Bytes [0:31]; // temp array for $readmemb to input Vectors

integer i; //used for FOR loop indexing
integer input_value = 0;
integer input_value_2 = 0;
integer Clock_Cycle;

//------------- Instantiation ---------------------//

dual_buffer DUT (.clk(clk),
                 .reset(reset),
                 .enable(enable),
```

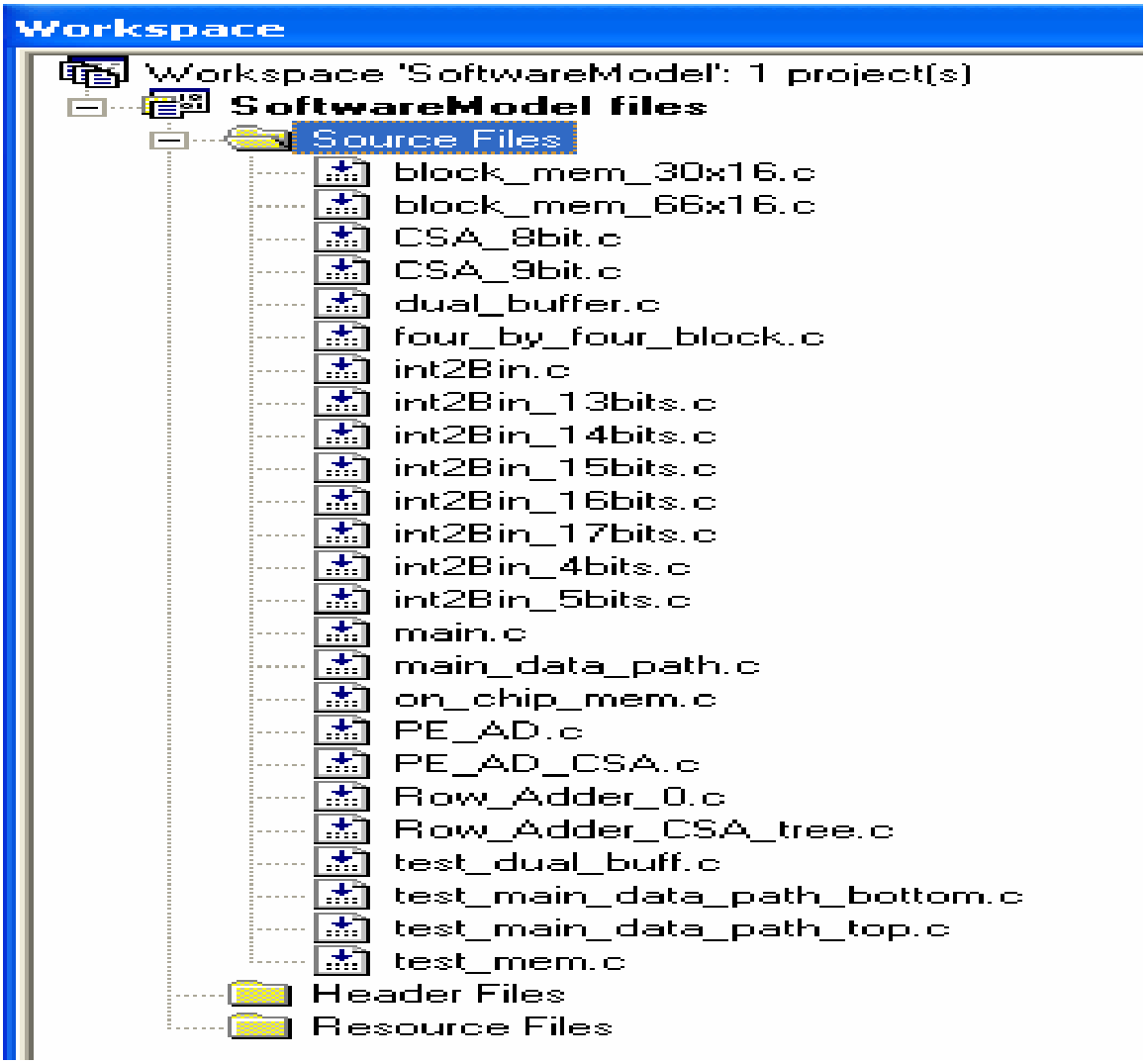
```
            .address(address),
            .read_Buffer_1(read_Buffer_1),
            .dataIn_32_Bytes(dataIn_32_Bytes),
            .dataOut_31_Bytes_A(dataOut_31_Bytes_A),
            .dataOut_31_Bytes_B(dataOut_31_Bytes_B)
            );
/*
spblockram_66 DUT2 (.clk(clk),
                    .we(we_a),
                   .a(a),
                    .di(a_di),
                   .do(a_do)
                    );
 */

/***** CLOCK GENERATION ******/
initial
begin
      clk = 1'b0;
end

always #(CLKPERIOD/2) clk = ~clk;


/***** THIS TASK CREATES A POSITIVE RESET FOR n CLOCK CYCLES *****/
task reset_for_n_clocks;
input [7:0] number_of_resets;
begin
      @(posedge clk);
      reset = 1'b1;     //activates the positive reset signal
      repeat(number_of_resets) @(posedge clk);
      reset=1'b0;             //remove the reset
end
endtask


/***** THIS TASK CREATES A DELAY FOR n CLOCK CYCLES *****/
task delay_for_n_clocks;
input [15:0] number_of_clock_delays;
begin
      repeat(number_of_clock_delays) @(posedge clk);
end
endtask

/***** THIS EVENT WILL END THE SIMULATION WHEN CALLED UPON *****/
event terminate_sim;
initial begin
      @ (terminate_sim);
      $finish;
end


/***** THIS TASK INITIALIZES THE FOLLOWING INPUT SIGNALS *****/
task init_inputs;
```

```
begin
	clk    = 0;
	reset  = 0;
	enable = 0;
	address = 0;
	read_Buffer_1 = 0;
	 dataIn_32_Bytes = 0;
end
endtask


/*****    THIS TASK FEEDS IN ALL OF THE TEST INPUT VECTORS              *****/
/*****    AND CHECKS THE OUTPUT SAD VALUES AGAINST THE C OUTPUT FILES    *****/
task feed_inputs_check_output;
begin
	//$readmemb("FILES/input_pixels.txt", input_32_Bytes); //files must be located in the project folder

	for (Clock_Cycle=0; Clock_Cycle<340; Clock_Cycle=Clock_Cycle+1) begin

	if (Clock_Cycle<96) read_Buffer_1 = 0;// keep it in read_Buffer_2 mode so that Buffer 1 gets loaded first
	else if ((Clock_Cycle>95)&&(Clock_Cycle<211)) read_Buffer_1 = 1;
	else read_Buffer_1 = 0;


	if (Clock_Cycle<143) input_value_2 = 0;
	else input_value_2 = 32;
	for (i=0; i<32; i=i+1) begin
		if (read_Buffer_1==1) input_32_Bytes[i] = input_value_2 + i;
		else input_32_Bytes[i] = input_value;
	end
	//if (Clock_Cycle>95) input_value = input_value + 1;
	input_value = input_value + 1;
	if (input_value==48) input_value = 0;

		@(posedge clk);

		dataIn_32_Bytes[255:248] = input_32_Bytes [0];
		dataIn_32_Bytes[247:240] = input_32_Bytes [1];
		dataIn_32_Bytes[239:232] = input_32_Bytes [2];
		dataIn_32_Bytes[231:224] = input_32_Bytes [3];
		dataIn_32_Bytes[223:216] = input_32_Bytes [4];
		dataIn_32_Bytes[215:208] = input_32_Bytes [5];
		dataIn_32_Bytes[207:200] = input_32_Bytes [6];
		dataIn_32_Bytes[199:192] = input_32_Bytes [7];

		dataIn_32_Bytes[191:184] = input_32_Bytes [8];
		dataIn_32_Bytes[183:176] = input_32_Bytes [9];
		dataIn_32_Bytes[175:168] = input_32_Bytes [10];
		dataIn_32_Bytes[167:160] = input_32_Bytes [11];
		dataIn_32_Bytes[159:152] = input_32_Bytes [12];
		dataIn_32_Bytes[151:144] = input_32_Bytes [13];
		dataIn_32_Bytes[143:136] = input_32_Bytes [14];
		dataIn_32_Bytes[135:128] = input_32_Bytes [15];
```

```
            dataIn_32_Bytes[127:120] = input_32_Bytes [16];
            dataIn_32_Bytes[119:112] = input_32_Bytes [17];
            dataIn_32_Bytes[111:104] = input_32_Bytes [18];
            dataIn_32_Bytes[103:96] = input_32_Bytes [19];
            dataIn_32_Bytes[95:88] = input_32_Bytes [20];
            dataIn_32_Bytes[87:80] = input_32_Bytes [21];
            dataIn_32_Bytes[79:72] = input_32_Bytes [22];
            dataIn_32_Bytes[71:64] = input_32_Bytes [23];

            dataIn_32_Bytes[63:56] = input_32_Bytes [24];
            dataIn_32_Bytes[55:48] = input_32_Bytes [25];
            dataIn_32_Bytes[47:40] = input_32_Bytes [26];
            dataIn_32_Bytes[39:32] = input_32_Bytes [27];
            dataIn_32_Bytes[31:24] = input_32_Bytes [28];
            dataIn_32_Bytes[23:16] = input_32_Bytes [29];
            dataIn_32_Bytes[15:8] = input_32_Bytes [30];
            dataIn_32_Bytes[7:0] = input_32_Bytes [31];


        $display("Clock Cycle = %d\n", Clock_Cycle);

        address = address + 1;
        if (address==48) address = 0;
        if (Clock_Cycle==210) address = 0;
        //if (Clock_Cycle==96) address = 0;

        end //ends the Clock Cycle for loop

        $display("Input Values have been Fed \n");

end
endtask


/******   RUN THE MAIN_DATA_PATH ******/
initial
begin
        init_inputs;
        reset_for_n_clocks(4);
        delay_for_n_clocks(2);
        enable = 1'b1;
        $display("\n ****************** MEMORY UNIT BUS RESULTS  ********************** \n");
        feed_inputs_check_output;
        $display("");
        $dis-
play("*********************************************************************************");
        enable = 1'b0;
        delay_for_n_clocks(5);
        //-> terminate_sim;
end
endmodule
```

7.2 C Code Files

The following screenshot provides an alphabetical listing of all of the C source files that were used to create the Software Model replica of the Verilog RTL designs, which were used to cross-check and verify the hardware designs.



The actual C source code files themselves are given below. There are 21 C code design files and 4 C test code files in total.

```
#include <stdlib.h>
#include <stdio.h>
//-------------------------------------------------------------------------------------------------------------------------------
----------//

// This function implements the read/write memory functionality of a
// 16 byte wide and 30 deep memory block (i.e Memory Partition B)


void block_mem_30x16 ( // inputs
                       int write_enable,
                       int address, // 7-bits (i.e. 0 to 29 addresses)
                       // input memory array address
                       int array_30x16 [30][16],
                       // Input Bus
                       int dataIn_16_Bytes[16],
                       // Output Bus
                       int dataOut_16_Bytes[16]
                      ){

    dataOut_16_Bytes[0] = array_30x16 [address][0];
    dataOut_16_Bytes[1] = array_30x16 [address][1];
    dataOut_16_Bytes[2] = array_30x16 [address][2];
    dataOut_16_Bytes[3] = array_30x16 [address][3];
    dataOut_16_Bytes[4] = array_30x16 [address][4];
    dataOut_16_Bytes[5] = array_30x16 [address][5];
    dataOut_16_Bytes[6] = array_30x16 [address][6];
    dataOut_16_Bytes[7] = array_30x16 [address][7];

    dataOut_16_Bytes[8]  = array_30x16 [address][8];
    dataOut_16_Bytes[9]  = array_30x16 [address][9];
    dataOut_16_Bytes[10] = array_30x16 [address][10];
    dataOut_16_Bytes[11] = array_30x16 [address][11];
    dataOut_16_Bytes[12] = array_30x16 [address][12];
    dataOut_16_Bytes[13] = array_30x16 [address][13];
    dataOut_16_Bytes[14] = array_30x16 [address][14];
    dataOut_16_Bytes[15] = array_30x16 [address][15];

    if (write_enable==1){
        array_30x16[address][0] = dataIn_16_Bytes[0];
        array_30x16[address][1] = dataIn_16_Bytes[1];
        array_30x16[address][2] = dataIn_16_Bytes[2];
        array_30x16[address][3] = dataIn_16_Bytes[3];
        array_30x16[address][4] = dataIn_16_Bytes[4];
        array_30x16[address][5] = dataIn_16_Bytes[5];
        array_30x16[address][6] = dataIn_16_Bytes[6];
        array_30x16[address][7] = dataIn_16_Bytes[7];

        array_30x16[address][8]  = dataIn_16_Bytes[8];
        array_30x16[address][9]  = dataIn_16_Bytes[9];
        array_30x16[address][10] = dataIn_16_Bytes[10];
        array_30x16[address][11] = dataIn_16_Bytes[11];
        array_30x16[address][12] = dataIn_16_Bytes[12];
```

```
        array_30x16[address][13] = dataIn_16_Bytes[13];
        array_30x16[address][14] = dataIn_16_Bytes[14];
        array_30x16[address][15] = dataIn_16_Bytes[15];
    } // ends if write_enable==1

                        } // ends function block_mem_30x16
```

```
#include <stdlib.h>
#include <stdio.h>
//-------------------------------------------------------------------------------------------------------------------------------------------
----------//

// This function implements the read/write memory functionality of a
// 16 byte wide and 66 deep memory block (i.e Memory Partition A)


void block_mem_66x16 ( // inputs
                       int write_enable,
                       int address, // 7-bits (i.e. 0 to 65 addresses)
                       // input memory array address
                       int array_66x16 [66][16],
                       // Input Bus
                       int dataIn_16_Bytes[16],
                       // Output Bus
                       int dataOut_16_Bytes[16]
                      ){

    dataOut_16_Bytes[0] = array_66x16 [address][0];
    dataOut_16_Bytes[1] = array_66x16 [address][1];
    dataOut_16_Bytes[2] = array_66x16 [address][2];
    dataOut_16_Bytes[3] = array_66x16 [address][3];
    dataOut_16_Bytes[4] = array_66x16 [address][4];
    dataOut_16_Bytes[5] = array_66x16 [address][5];
    dataOut_16_Bytes[6] = array_66x16 [address][6];
    dataOut_16_Bytes[7] = array_66x16 [address][7];

    dataOut_16_Bytes[8] = array_66x16 [address][8];
    dataOut_16_Bytes[9]  = array_66x16 [address][9];
    dataOut_16_Bytes[10] = array_66x16 [address][10];
    dataOut_16_Bytes[11] = array_66x16 [address][11];
    dataOut_16_Bytes[12] = array_66x16 [address][12];
    dataOut_16_Bytes[13] = array_66x16 [address][13];
    dataOut_16_Bytes[14] = array_66x16 [address][14];
    dataOut_16_Bytes[15] = array_66x16 [address][15];

    if (write_enable==1){
        array_66x16[address][0] = dataIn_16_Bytes[0];
        //printf("\n %d \n", dataIn_16_Bytes[0]);

        array_66x16[address][1] = dataIn_16_Bytes[1];
        array_66x16[address][2] = dataIn_16_Bytes[2];
        array_66x16[address][3] = dataIn_16_Bytes[3];
        array_66x16[address][4] = dataIn_16_Bytes[4];
        array_66x16[address][5] = dataIn_16_Bytes[5];
        array_66x16[address][6] = dataIn_16_Bytes[6];
        array_66x16[address][7] = dataIn_16_Bytes[7];

        array_66x16[address][8]  = dataIn_16_Bytes[8];
        array_66x16[address][9]  = dataIn_16_Bytes[9];
        array_66x16[address][10] = dataIn_16_Bytes[10];
```

```
        array_66x16[address][11] = dataIn_16_Bytes[11];
        array_66x16[address][12] = dataIn_16_Bytes[12];
        array_66x16[address][13] = dataIn_16_Bytes[13];
        array_66x16[address][14] = dataIn_16_Bytes[14];
        array_66x16[address][15] = dataIn_16_Bytes[15];
    } // ends if write_enable==1

                        } // ends function block_mem_66x16
```

```
#include <stdlib.h>
#include <stdio.h>
//-------------------------------------------------------------------------------------------------------------------------------------------
----------//

// This is an 8bit CSA Adder Block

void CSA_8bit_block (int x, int y, int z, int *Carry, int *Sum){

	int x_truncated, y_truncated, z_truncated;

	// first the first 24 MSBs of the input are chopped off
	// to truly reflect an 8-bit CSA block
	x_truncated = x & 0x000000FF;
	y_truncated = y & 0x000000FF;
	z_truncated = z & 0x000000FF;

	// XOR x, y, and z

	/*
	printf("\n\nx_truncated is: %d\n\n", x_truncated);
	printf("\n\ny_truncated is: %d\n\n", y_truncated);
	printf("\n\nz_truncated is: %d\n\n", z_truncated);
	*/
	*Sum = x_truncated ^ y_truncated ^ z_truncated;

	//printf("\n\nThe Sum is: %d\n\n", *Sum);
	*Carry = (x_truncated&y_truncated) ^
		 (x_truncated&z_truncated) ^
		 (y_truncated&z_truncated);

} //ends CSA_8bit
```

```
#include <stdlib.h>
#include <stdio.h>
//-------------------------------------------------------------------------------------------------------------------------------------------
----------//
// This is an 9bit CSA Adder Block

void CSA_9bit_block (int x, int y, int z, int *Carry, int *Sum){

	int x_truncated, y_truncated, z_truncated;

	// first the first 23 MSBs of the input are chopped off
	// to truly reflect a 9-bit CSA block
	x_truncated = x & 0x000001FF;
	y_truncated = y & 0x000001FF;
	z_truncated = z & 0x000001FF;

	// XOR x, y, and z
	*Sum = x_truncated ^ y_truncated ^ z_truncated;

	*Carry = (x_truncated&y_truncated) ^
		 (x_truncated&z_truncated) ^
		 (y_truncated&z_truncated);

} //ends CSA_9bit
```

```
#include <stdlib.h>
#include <stdio.h>
//-------------------------------------------------------------------------------------------------------------------------------------------
----------//
// This is the dual_buffer design file.

// External Functions located in other Project Files

// Two on_chip_mem modules will be used to create/make up the
// Dual Buffer Memory Unit
void on_chip_mem (// inputs
                              int write_enable,
                              int address, // 6-bits (i.e. 0 to 47 search pixel rows)
                              int load_bottom,
                              int load_top,

                              int read_L0_L1,
                              int read_L1_L2,
                              int read_L2_L3,

                              int block_66x16_mem_1 [66][16],
                              int block_66x16_mem_2 [66][16],
                              int block_30x16_mem_1 [66][16],
                              int block_30x16_mem_2 [66][16],

                              int *partition_A_Counter,
                              int *partition_B_Counter,

                              // Input Bus
                              int dataIn_32_Bytes[32],
                              // Output Buses
                              int dataOut_31_Bytes_A[32],// an extra byte in the array is added
                              int dataOut_31_Bytes_B[32] // for programming convenience
                             );

// ------------------------------------------------------------------------------- //

void dual_buffer (// inputs
                        int initialize,
                        int address, // 6-bits (i.e. 0 to 47 search pixel rows)
                        int read_Buffer_1,
                        int read_Buffer_2,

                        // Input Bus
                        int dataIn_32_Bytes[32],
                        // Output Buses
                        int dataOut_31_Bytes_A[32],// an extra byte in the array is added
                        int dataOut_31_Bytes_B[32] // for programming convenience
                        ){

    // Memory Blocks/Parts that are used to create Buffer 1
    static int block_66x16_mem_1_B1 [66][16];
    static int block_66x16_mem_2_B1 [66][16];
```

```
	static int block_30x16_mem_1_B1 [66][16];
	static int block_30x16_mem_2_B1 [66][16];

	static int partition_A_Counter_B1 = 0;
	static int partition_B_Counter_B1 = 0;

	// Memory Blocks/Parts that are used to create Buffer 2
	static int block_66x16_mem_1_B2 [66][16];
	static int block_66x16_mem_2_B2 [66][16];
	static int block_30x16_mem_1_B2 [66][16];
	static int block_30x16_mem_2_B2 [66][16];

	static int partition_A_Counter_B2 = 0;
	static int partition_B_Counter_B2 = 0;

	// Control Signals for both Memory Blocks
	int   write_enable_B1 = 0, write_enable_B2 = 0;
	int   load_bottom_B1 = 0,  load_bottom_B2 = 0;
	int   load_top_B1 = 0,     load_top_B2 = 0;

	int   read_L0_L1_B1 = 0, read_L0_L1_B2 = 0;
	int   read_L1_L2_B1 = 0, read_L1_L2_B2 = 0;
	int   read_L2_L3_B1 = 0, read_L2_L3_B2 = 0;

	// Internal Variable
	static int load_Counter = 0;
	static int read_Buff_Counter = 0;

// ------------------------------------------------------------------------------------------------//
// Execution

if (initialize==1) {

	write_enable_B1 = 1;
	if (load_Counter<48) load_bottom_B1 = 1;
	else                 load_top_B1    = 1;

	load_Counter++;
	if (load_Counter==96) load_Counter = 0;

	on_chip_mem (// inputs
		write_enable_B1,
		address, // 6-bits (i.e. 0 to 47 search pixel rows)
		load_bottom_B1,
		load_top_B1,
		read_L0_L1_B1,
		read_L1_L2_B1,
		read_L2_L3_B1,
		block_66x16_mem_1_B1,
		block_66x16_mem_2_B1,
		block_30x16_mem_1_B1,
		block_30x16_mem_2_B1,
		&partition_A_Counter_B1,
		&partition_B_Counter_B1,
```

```
                // Input Bus
                dataIn_32_Bytes,
                // Output Buses
                dataOut_31_Bytes_A,
                dataOut_31_Bytes_B
                );
} // ends if initialize==1

if (read_Buffer_1==1) {

    write_enable_B2 = 1;
    if (load_Counter<48) load_bottom_B2 = 1;
    else                 load_top_B2    = 1;
    load_Counter++;
    if (load_Counter==96) load_Counter = 0;

    if (read_Buff_Counter<96){
        on_chip_mem (// inputs
                write_enable_B2,
                address, // 6-bits (i.e. 0 to 47 search pixel rows)
                load_bottom_B2,
                load_top_B2,
                read_L0_L1_B2,
                read_L1_L2_B2,
                read_L2_L3_B2,
                block_66x16_mem_1_B2,
                block_66x16_mem_2_B2,
                block_30x16_mem_1_B2,
                block_30x16_mem_2_B2,
                &partition_A_Counter_B2,
                &partition_B_Counter_B2,
                // Input Bus
                dataIn_32_Bytes,
                // Output Buses
                dataOut_31_Bytes_A,
                dataOut_31_Bytes_B
                );
    }// ends if read_Buff_Counter<96

    if          (read_Buff_Counter<33) read_L0_L1_B1 = 1;
    else if (read_Buff_Counter<66) read_L1_L2_B1 = 1;
    else                           read_L2_L3_B1 = 1;
    read_Buff_Counter++;
    if (read_Buff_Counter==114) read_Buff_Counter = 0; // 114 = (3*33) + 15

    on_chip_mem (// inputs
                write_enable_B1,
                address, // 6-bits (i.e. 0 to 47 search pixel rows)
                load_bottom_B1,
                load_top_B1,
                read_L0_L1_B1,
                read_L1_L2_B1,
                read_L2_L3_B1,
                block_66x16_mem_1_B1,
```

```
                block_66x16_mem_2_B1,
                block_30x16_mem_1_B1,
                block_30x16_mem_2_B1,
                &partition_A_Counter_B1,
                &partition_B_Counter_B1,
                // Input Bus
                dataIn_32_Bytes,
                // Output Buses
                dataOut_31_Bytes_A,
                dataOut_31_Bytes_B
                );
} // ends if read_Buffer_1

if (read_Buffer_2==1) {

    write_enable_B1 = 1;
    if (load_Counter<48) load_bottom_B1 = 1;
    else                 load_top_B1    = 1;
    load_Counter++;
    if (load_Counter==96) load_Counter = 0;
    if (read_Buff_Counter<96){
        on_chip_mem (// inputs
                write_enable_B1,
                address, // 6-bits (i.e. 0 to 47 search pixel rows)
                load_bottom_B1,
                load_top_B1,
                read_L0_L1_B1,
                read_L1_L2_B1,
                read_L2_L3_B1,
                block_66x16_mem_1_B1,
                block_66x16_mem_2_B1,
                block_30x16_mem_1_B1,
                block_30x16_mem_2_B1,
                &partition_A_Counter_B1,
                &partition_B_Counter_B1,
                // Input Bus
                dataIn_32_Bytes,
                // Output Buses
                dataOut_31_Bytes_A,
                dataOut_31_Bytes_B
                );
    }// ends if read_Buff_Counter<96


    //printf("Read Buffer: %d\n", read_Buff_Counter);
    if          (read_Buff_Counter<33) read_L0_L1_B2 = 1;
    else if (read_Buff_Counter<66) read_L1_L2_B2 = 1;
    else                           read_L2_L3_B2 = 1;
    read_Buff_Counter++;
    if (read_Buff_Counter==114) read_Buff_Counter = 0; // 114 = (3*33) + 15

    //printf("Address: %d\n", address);

    on_chip_mem (// inputs
```

```
                write_enable_B2,
                address, // 6-bits (i.e. 0 to 47 search pixel rows)
                load_bottom_B2,
                load_top_B2,
                read_L0_L1_B2,
                read_L1_L2_B2,
                read_L2_L3_B2,
                block_66x16_mem_1_B2,
                block_66x16_mem_2_B2,
                block_30x16_mem_1_B2,
                block_30x16_mem_2_B2,
                &partition_A_Counter_B2,
                &partition_B_Counter_B2,
                // Input Bus
                dataIn_32_Bytes,
                // Output Buses
                dataOut_31_Bytes_A,
                dataOut_31_Bytes_B
                );
} // ends if read_Buffer_2

}// ends function dual_buffer
```

```
#include <stdlib.h>
#include <stdio.h>
//-------------------------------------------------------------------------------------------------------------------------------------------
----------//
FILE *fp;

// This is the Four_by_Four_Block design file.

// External Functions located in other Project Files

void Row_Adder_0 (int clk, int reset, int enable, int update_Cur_MB,
                  int R3_a,int C3_a,int R2_a,int C2_a,int R1_a,int C1_a,int R0_a,int C0_a,
                  int R3_b,int C3_b,int R2_b,int C2_b,int R1_b,int C1_b,int R0_b,int C0_b,
                  int broadcast_bus_select,
                  int *C3_Reg,
                  int *C2_Reg,
                  int *C1_Reg,
                  int *C0_Reg,
                  int *PSAD_0_Register);

void Row_Adder_CSA_tree (int clk, int reset, int enable, int update_Cur_MB,
                  int R3_a,int C3_a,int R2_a,int C2_a,int R1_a,int C1_a,int R0_a,int C0_a,
                  int R3_b,int C3_b,int R2_b,int C2_b,int R1_b,int C1_b,int R0_b,int C0_b,
                  int broadcast_bus_select,
                        int PSAD_In,
                        int *C3_Reg,
                        int *C2_Reg,
                        int *C1_Reg,
                        int *C0_Reg,
                        int *PSAD_Out_Register);


//-----------------------------------------------------------------------------------------------//


// This function acts as a single 4x4 block
// which combines the 4 Row Adder Trees
// and produces 1 final PSAD value

void four_by_four_block (int clk, int reset, int enable, int update_Cur_MB,
                           int broadcast_column_1_a,
                           int broadcast_column_2_a,
                           int broadcast_column_3_a,
                           int broadcast_column_4_a,
                           int broadcast_column_1_b,
                           int broadcast_column_2_b,
                           int broadcast_column_3_b,
                           int broadcast_column_4_b,
                         int broadcast_bus_select_1,
                         int broadcast_bus_select_2,
                         int broadcast_bus_select_3,
                         int broadcast_bus_select_4,
                           int *C3_Row0, int *C2_Row0, int *C1_Row0, int *C0_Row0,
```

```
                        int *C3_Row1, int *C2_Row1, int *C1_Row1, int *C0_Row1,
                        int *C3_Row2, int *C2_Row2, int *C1_Row2, int *C0_Row2,
                        int *C3_Row3, int *C2_Row3, int *C1_Row3, int *C0_Row3,
                        int *PSAD_0,
                        int *PSAD_1,
                        int *PSAD_2,
                        int *PSAD_Out
                        ){

int update_Control_0 = 0;
int update_Control_1 = 0;
int update_Control_2 = 0;
int update_Control_3 = 0;

int PSAD_0_old = *PSAD_0;
int PSAD_1_old = *PSAD_1;
int PSAD_2_old = *PSAD_2;

if     (update_Cur_MB==0) update_Control_0 = 1;
else if (update_Cur_MB==1) update_Control_1 = 1;
else if (update_Cur_MB==2) update_Control_2 = 1;
else if (update_Cur_MB==3) update_Control_3 = 1;

Row_Adder_0 (clk, reset, enable, update_Control_0,
                broadcast_column_1_a, broadcast_column_1_a,
                broadcast_column_2_a, broadcast_column_2_a,
                broadcast_column_3_a, broadcast_column_3_a,
                broadcast_column_4_a, broadcast_column_4_a,
                broadcast_column_1_b, broadcast_column_1_b,
                broadcast_column_2_b, broadcast_column_2_b,
                broadcast_column_3_b, broadcast_column_3_b,
                broadcast_column_4_b, broadcast_column_4_b,
                broadcast_bus_select_1,
                C3_Row0, C2_Row0, C1_Row0, C0_Row0,
                PSAD_0);

Row_Adder_CSA_tree (clk, reset, enable, update_Control_1,
                broadcast_column_1_a, broadcast_column_1_a,
                broadcast_column_2_a, broadcast_column_2_a,
                broadcast_column_3_a, broadcast_column_3_a,
                broadcast_column_4_a, broadcast_column_4_a,
                broadcast_column_1_b, broadcast_column_1_b,
                broadcast_column_2_b, broadcast_column_2_b,
                broadcast_column_3_b, broadcast_column_3_b,
                broadcast_column_4_b, broadcast_column_4_b,
                broadcast_bus_select_2,
                        PSAD_0_old,
                        C3_Row1, C2_Row1, C1_Row1, C0_Row1,
                        PSAD_1);

Row_Adder_CSA_tree (clk, reset, enable, update_Control_2,
                broadcast_column_1_a, broadcast_column_1_a,
                broadcast_column_2_a, broadcast_column_2_a,
                broadcast_column_3_a, broadcast_column_3_a,
```

```
                broadcast_column_4_a, broadcast_column_4_a,
                broadcast_column_1_b, broadcast_column_1_b,
                broadcast_column_2_b, broadcast_column_2_b,
                broadcast_column_3_b, broadcast_column_3_b,
                broadcast_column_4_b, broadcast_column_4_b,
                broadcast_bus_select_3,
                        PSAD_1_old,
                        C3_Row2, C2_Row2, C1_Row2, C0_Row2,
                        PSAD_2);

        Row_Adder_CSA_tree (clk, reset, enable, update_Control_3,
                broadcast_column_1_a, broadcast_column_1_a,
                broadcast_column_2_a, broadcast_column_2_a,
                broadcast_column_3_a, broadcast_column_3_a,
                broadcast_column_4_a, broadcast_column_4_a,
                broadcast_column_1_b, broadcast_column_1_b,
                broadcast_column_2_b, broadcast_column_2_b,
                broadcast_column_3_b, broadcast_column_3_b,
                broadcast_column_4_b, broadcast_column_4_b,
                broadcast_bus_select_4,
                        PSAD_2_old,
                        C3_Row3, C2_Row3, C1_Row3, C0_Row3,
                        PSAD_Out);

}// ends the four_by_four_block module


//-------------------------------------------------------------------------------//
// This function produces a random integer between 0 and 255
int random_int_between_0_and_255(){
        double random_float_between_0_and_1 = (float)rand() / (float)0x7fff;
        int random_int = (int)(random_float_between_0_and_1 * 255);
        if (random_int>255){
                printf("\n\nError with the Random Number Generation.");
                printf("\n\nThe Random Number produced is: %d\n\n", random_int);
                exit(0);
        }
        return random_int;
}// ends the random_into_between_0_and_255 function


//-------------------------------------------------------------------------------//



void test_four_by_four_block(){

        int broadcast_column_1_a;
        int broadcast_column_2_a;
        int broadcast_column_3_a;
        int broadcast_column_4_a;
        int broadcast_column_1_b;
        int broadcast_column_2_b;
        int broadcast_column_3_b;
```

```
    int broadcast_column_4_b;

  int broadcast_bus_select_1 = 0;
    int   broadcast_bus_select_2 = 0;
    int broadcast_bus_select_3 = 0;
    int   broadcast_bus_select_4 = 0;

    int C3_Row0, C2_Row0, C1_Row0, C0_Row0;
    int C3_Row1, C2_Row1, C1_Row1, C0_Row1;
    int C3_Row2, C2_Row2, C1_Row2, C0_Row2;
    int C3_Row3, C2_Row3, C1_Row3, C0_Row3;
    int PSAD_0 = 0;
    int PSAD_1 = 0;
    int PSAD_2 = 0;
    int PSAD_Out = 0;

    int test_Case;
    int update_Cur_MB;

    if ((fp = fopen("output.txt", "w"))==NULL){
              fprintf(fp, "\n\n Can not open the output file. \n\n");
              exit(1);
    }

    for (test_Case = 0; test_Case<10; test_Case++){

    if      (test_Case>3) update_Cur_MB = 4;
    else update_Cur_MB = (test_Case%4);
    //fprintf(fp, "\n\nupdate_Cur_MB with in main test: %d\n\n", update_Cur_MB);

    broadcast_column_1_a = random_int_between_0_and_255();
    broadcast_column_2_a = random_int_between_0_and_255();
    broadcast_column_3_a = random_int_between_0_and_255();
    broadcast_column_4_a = random_int_between_0_and_255();
    broadcast_column_1_b = random_int_between_0_and_255();
    broadcast_column_2_b = random_int_between_0_and_255();
    broadcast_column_3_b = random_int_between_0_and_255();
    broadcast_column_4_b = random_int_between_0_and_255();
    fprintf(fp, "broadcast: %d  %d  %d  %d\n\n", broadcast_column_1_a,
                                                  broadcast_column_2_a,
                                                  broadcast_column_3_a,
                                                  broadcast_column_4_a);

    // 1,0,1 corresponds to clk=1, reset=0,and enable = 1
    four_by_four_block (1,0,1,update_Cur_MB,
                              broadcast_column_1_a,
                              broadcast_column_2_a,
                              broadcast_column_3_a,
                              broadcast_column_4_a,
                              broadcast_column_1_b,
                              broadcast_column_2_b,
                              broadcast_column_3_b,
                              broadcast_column_4_b,
                            broadcast_bus_select_1,
```

```
                        broadcast_bus_select_2,
                        broadcast_bus_select_3,
                        broadcast_bus_select_4,
                          &C3_Row0, &C2_Row0, &C1_Row0, &C0_Row0,
                          &C3_Row1, &C2_Row1, &C1_Row1, &C0_Row1,
                          &C3_Row2, &C2_Row2, &C1_Row2, &C0_Row2,
                          &C3_Row3, &C2_Row3, &C1_Row3, &C0_Row3,
                          &PSAD_0,
                          &PSAD_1,
                          &PSAD_2,
                          &PSAD_Out
                          );

    fprintf(fp, "\n\nAfter Clock Cycle: %d\n\n", (test_Case+1));

    /*
    fprintf(fp, "%d  %d  %d  %d\n\n", C3_Row0,C2_Row0,C1_Row0,C0_Row0);
    fprintf(fp, "%d  %d  %d  %d\n\n", C3_Row1,C2_Row1,C1_Row1,C0_Row1);
    fprintf(fp, "%d  %d  %d  %d\n\n", C3_Row2,C2_Row2,C1_Row2,C0_Row2);
    fprintf(fp, "%d  %d  %d  %d\n\n", C3_Row3,C2_Row3,C1_Row3,C0_Row3);
    */

    fprintf(fp, "PSAD_0: %d\n\n", PSAD_0);
    fprintf(fp, "PSAD_1: %d\n\n", PSAD_1);
    fprintf(fp, "PSAD_2: %d\n\n", PSAD_2);
    fprintf(fp, "PSAD_Out: %d\n\n", PSAD_Out);

    }// ends the test_Case loop (after 1000 test cases)

    fclose(fp);

}// ends test_four_by_four_block()
```

```
#include <stdlib.h>
#include <stdio.h>
//-------------------------------------------------------------------------------------------------------------------------------------------
----------//
FILE *bus_A;

// This function coverts 4, 32 bit integer values
// into their equivalent 32 bit binary string representation.
// Each 32-bit integer is first truncated to 8-bits only
// (this is acceptable, since the int value is always<256),
// then each of the 4 8-bit truncated int values are arranged
// from left to right to form a 32-bit bus line.

char* int2Bin(int a, int b, int c, int d) {
 char *str;
 int cnt = 31;
 str = (char *) malloc(33); /* 32 + 1 , becoz its a 32 bit bin number */
 while ( cnt > -1 ){
     str[cnt]= '0';
     cnt --;
 }

 cnt = 31;
 while (d > 0){
      if (d%2==1){
         str[cnt] = '1';
       }
     cnt--;
      d = d/2 ;
 }

 cnt = 23;
 while (c > 0){
      if (c%2==1){
         str[cnt] = '1';
       }
     cnt--;
      c = c/2 ;
 }

 cnt = 15;
 while (b > 0){
      if (b%2==1){
         str[cnt] = '1';
       }
     cnt--;
      b = b/2 ;
 }

 cnt = 7;
 while (a > 0){
      if (a%2==1){
         str[cnt] = '1';
```

```
        }
      cnt--;
        a = a/2 ;
 }

 str[32] = '\0';
 return str;

} // ends int2Bin


void test_int2Bin(){

      int A = 2;
      int B = 3;
      int C = 4;
      int D = 5;

      if ((bus_A = fopen("bus_A.txt", "w"))==NULL){
                fprintf(bus_A, "\n\n Can not open the bus_A output file. \n\n");
                exit(1);
      }

      fprintf(bus_A,"\nThe Binary Value is: %s\n", int2Bin(A,B,C,D));
}
```

```
#include <stdlib.h>
#include <stdio.h>
//-------------------------------------------------------------------------------------------------------------------------------
----------//
// This function coverts an Integer Value
// into its equivalent 13-bit binary value.

char* int2Bin_13bits(int a) {
 char *str;
 int cnt = 12;
 str = (char *) malloc(14); /* 32 + 1 , because its a 32 bit bin number */
 while ( cnt > -1 ){
     str[cnt]= '0';
     cnt --;
 }

 cnt = 12;
 while (a > 0){
      if (a%2==1){
         str[cnt] = '1';
       }
     cnt--;
       a = a/2 ;
 }

 str[13] = '\0';
 return str;

} // ends int2Bin_13bits
```

```
#include <stdlib.h>
#include <stdio.h>
//-------------------------------------------------------------------------------------------------------------------------------------------
----------//
// This function coverts an Integer Value
// into its equivalent 14-bit binary value.

char* int2Bin_14bits(int a) {
 char *str;
 int cnt = 13;
 str = (char *) malloc(15); /* 32 + 1 , because its a 32 bit bin number */
 while ( cnt > -1 ){
     str[cnt]= '0';
     cnt --;
 }

 cnt = 13;
 while (a > 0){
      if (a%2==1){
         str[cnt] = '1';
       }
     cnt--;
       a = a/2 ;
 }

 str[14] = '\0';
 return str;

} // ends int2Bin_14bits
```

```
#include <stdlib.h>
#include <stdio.h>
//-------------------------------------------------------------------------------------------------------------------------------
----------//
// This function coverts an Integer Value
// into its equivalent 15-bit binary value.

char* int2Bin_15bits(int a) {
 char *str;
 int cnt = 14;
 str = (char *) malloc(16); /* 32 + 1 , because its a 32 bit bin number */
 while ( cnt > -1 ){
     str[cnt]= '0';
     cnt --;
 }

 cnt = 14;
 while (a > 0){
      if (a%2==1){
         str[cnt] = '1';
       }
     cnt--;
       a = a/2 ;
 }

 str[15] = '\0';
 return str;

} // ends int2Bin_15bits
```

```
#include <stdlib.h>
#include <stdio.h>
//-------------------------------------------------------------------------------------------------------------------------------
----------//
// This function coverts an Integer Value
// into its equivalent 16-bit binary value.

char* int2Bin_16bits(int a) {
 char *str;
 int cnt = 15;
 str = (char *) malloc(17); /* 32 + 1 , because its a 32 bit bin number */
 while ( cnt > -1 ){
     str[cnt]= '0';
     cnt --;
 }

 cnt = 15;
 while (a > 0){
      if (a%2==1){
         str[cnt] = '1';
       }
     cnt--;
       a = a/2 ;
 }

 str[16] = '\0';
 return str;

} // ends int2Bin_16bits
```

```
#include <stdlib.h>
#include <stdio.h>
//-------------------------------------------------------------------------------------------------------------------------------------------
----------//
// This function coverts an Integer Value
// into its equivalent 17-bit binary value.

char* int2Bin_17bits(int a) {
 char *str;
 int cnt = 16;
 str = (char *) malloc(18); /* 32 + 1 , because its a 32 bit bin number */
 while ( cnt > -1 ){
     str[cnt]= '0';
     cnt --;
 }

 cnt = 16;
 while (a > 0){
      if (a%2==1){
         str[cnt] = '1';
       }
     cnt--;
       a = a/2 ;
 }

 str[17] = '\0';
 return str;

} // ends int2Bin_17bits
```

```
#include <stdlib.h>
#include <stdio.h>
//-------------------------------------------------------------------------------------------------------------------------------------
----------//
// This function coverts an Integer Value between 15 and 0
// to its equivalent 4-bit binary value.

char* int2Bin_4bits(int a) {
 char *str;
 int cnt = 3;
 str = (char *) malloc(5); /* 32 + 1 , because its a 32 bit bin number */
 while ( cnt > -1 ){
     str[cnt]= '0';
     cnt --;
 }

 cnt = 3;
 while (a > 0){
      if (a%2==1){
         str[cnt] = '1';
       }
     cnt--;
       a = a/2 ;
 }

 str[4] = '\0';
 return str;

} // ends int2Bin_4bits
```

```
#include <stdlib.h>
#include <stdio.h>
//-------------------------------------------------------------------------------------------------------------------------------------------
----------//
// This function coverts an Integer Value between 16 and 0
// to its equivalent 5-bit binary value.

char* int2Bin_5bits(int a) {
 char *str;
 int cnt = 4;
 str = (char *) malloc(6); /* 32 + 1 , because its a 32 bit bin number */
 while ( cnt > -1 ){
     str[cnt]= '0';
     cnt --;
 }

 cnt = 4;
 while (a > 0){
      if (a%2==1){
         str[cnt] = '1';
       }
     cnt--;
       a = a/2 ;
 }

 str[5] = '\0';
 return str;

} // ends int2Bin_5bits
```

```
#include <stdlib.h>
#include <stdio.h>
//-------------------------------------------------------------------------------------------------------------------------------------------
----------//
// This is the main calling program.
// External Functions located in other Project Files

void test_PE_AD();

void test_Row_Adder_CSA_tree();

void test_main_data_path();

void test_main_data_path_bottom();

void test_main_data_path_top();

void test_int2Bin();

void test_mem();

//-----------------------------------------------------------------------------------------//

int main(void)
{

	//test_PE_AD();

	//test_Row_Adder_CSA_tree();

	//test_four_by_four_block();

	//test_main_data_path_top();

	//test_main_data_path_bottom();

	//test_int2Bin();
	printf("\n\nSTARTING...\n\n");

	//test_mem();

	test_dual_buff();

	printf("\n\nIf NO Error Messages are Displayed above\n");
	printf("then the testing was completely Successful!!\n\n");

	return 0;
} //ends main
```

```
#include <stdlib.h>
#include <stdio.h>
//-------------------------------------------------------------------------------------------------------------------------------------------
----------//
FILE *fp;

// This is the main_data_path design file file.

// External Functions located in other Project Files

void four_by_four_block (int clk, int reset, int enable, int update_Cur_MB,
                                    int broadcast_column_1_a,
                                    int broadcast_column_2_a,
                                    int broadcast_column_3_a,
                                    int broadcast_column_4_a,
                                    int broadcast_column_1_b,
                                    int broadcast_column_2_b,
                                    int broadcast_column_3_b,
                                    int broadcast_column_4_b,
                                  int broadcast_bus_select_1,
                                  int broadcast_bus_select_2,
                                  int broadcast_bus_select_3,
                                  int broadcast_bus_select_4,
                                    int *C3_Row0, int *C2_Row0, int *C1_Row0, int *C0_Row0,
                                    int *C3_Row1, int *C2_Row1, int *C1_Row1, int *C0_Row1,
                                    int *C3_Row2, int *C2_Row2, int *C1_Row2, int *C0_Row2,
                                    int *C3_Row3, int *C2_Row3, int *C1_Row3, int *C0_Row3,
                                    int *PSAD_0,
                                    int *PSAD_1,
                                    int *PSAD_2,
                                    int *PSAD_Out
                                    );


//-------------------------------------------------------------------------------------------//

void main_data_path (
                            // Input Wires & Buses
                            int clk, int reset, int enable, int update_Cur_MB,

                            int broadcast_column_1_a,
                            int broadcast_column_2_a,
                            int broadcast_column_3_a,
                            int broadcast_column_4_a,
                            int broadcast_column_5_a,
                            int broadcast_column_6_a,
                            int broadcast_column_7_a,
                            int broadcast_column_8_a,
                            int broadcast_column_9_a,
                            int broadcast_column_10_a,
                            int broadcast_column_11_a,
                            int broadcast_column_12_a,
                            int broadcast_column_13_a,
```

```
int broadcast_column_14_a,
int broadcast_column_15_a,
int broadcast_column_16_a,

int broadcast_column_1_b,
int broadcast_column_2_b,
int broadcast_column_3_b,
int broadcast_column_4_b,
int broadcast_column_5_b,
int broadcast_column_6_b,
int broadcast_column_7_b,
int broadcast_column_8_b,
int broadcast_column_9_b,
int broadcast_column_10_b,
int broadcast_column_11_b,
int broadcast_column_12_b,
int broadcast_column_13_b,
int broadcast_column_14_b,
int broadcast_column_15_b,
int broadcast_column_16_b,

int broadcast_bus_select,

// Ouput Registers

// The 16 4x4 SAD output registers, labeled by row/column
int *B4x4_00_out, int *B4x4_01_out, int *B4x4_02_out, int *B4x4_03_out,
int *B4x4_10_out, int *B4x4_11_out, int *B4x4_12_out, int *B4x4_13_out,
int *B4x4_20_out, int *B4x4_21_out, int *B4x4_22_out, int *B4x4_23_out,
int *B4x4_30_out, int *B4x4_31_out, int *B4x4_32_out, int *B4x4_33_out,

// The 8 4x8 SAD output registers, labeled by row/column
int *B4x8_00_out, int *B4x8_01_out, int *B4x8_02_out, int *B4x8_03_out,
int *B4x8_10_out, int *B4x8_11_out, int *B4x8_12_out, int *B4x8_13_out,

// The 8 8x4 SAD output registers, labeled by row/column
int *B8x4_00_out, int *B8x4_01_out,
int *B8x4_10_out, int *B8x4_11_out,
int *B8x4_20_out, int *B8x4_21_out,
int *B8x4_30_out, int *B8x4_31_out,

// The 4 8x8 SAD output registers, labeled by row/column
int *B8x8_00_out, int *B8x8_01_out,
int *B8x8_10_out, int *B8x8_11_out,

// The 2 8x16 SAD output registers, labeled by column
int *B8x16_0_out, int *B8x16_1_out,

// The 2 16x8 SAD output registers, labeled by row
int *B16x8_0_out, int *B16x8_1_out,

// The 1 16x16 SAD output register
int *B16x16_out
```

){

// Internal Registers

static int C3_Row0_1; static int C2_Row0_1; static int C1_Row0_1; static int C0_Row0_1;
static int C3_Row1_1; static int C2_Row1_1; static int C1_Row1_1; static int C0_Row1_1;
static int C3_Row2_1; static int C2_Row2_1; static int C1_Row2_1; static int C0_Row2_1;
static int C3_Row3_1; static int C2_Row3_1; static int C1_Row3_1; static int C0_Row3_1;
static int C3_Row4_1; static int C2_Row4_1; static int C1_Row4_1; static int C0_Row4_1;
static int C3_Row5_1; static int C2_Row5_1; static int C1_Row5_1; static int C0_Row5_1;
static int C3_Row6_1; static int C2_Row6_1; static int C1_Row6_1; static int C0_Row6_1;
static int C3_Row7_1; static int C2_Row7_1; static int C1_Row7_1; static int C0_Row7_1;
static int C3_Row8_1; static int C2_Row8_1; static int C1_Row8_1; static int C0_Row8_1;
static int C3_Row9_1; static int C2_Row9_1; static int C1_Row9_1; static int C0_Row9_1;
static int C3_Row10_1; static int C2_Row10_1; static int C1_Row10_1; static int C0_Row10_1;
static int C3_Row11_1; static int C2_Row11_1; static int C1_Row11_1; static int C0_Row11_1;
static int C3_Row12_1; static int C2_Row12_1; static int C1_Row12_1; static int C0_Row12_1;
static int C3_Row13_1; static int C2_Row13_1; static int C1_Row13_1; static int C0_Row13_1;
static int C3_Row14_1; static int C2_Row14_1; static int C1_Row14_1; static int C0_Row14_1;
static int C3_Row15_1; static int C2_Row15_1; static int C1_Row15_1; static int C0_Row15_1;

static int C3_Row0_2; static int C2_Row0_2; static int C1_Row0_2; static int C0_Row0_2;
static int C3_Row1_2; static int C2_Row1_2; static int C1_Row1_2; static int C0_Row1_2;
static int C3_Row2_2; static int C2_Row2_2; static int C1_Row2_2; static int C0_Row2_2;
static int C3_Row3_2; static int C2_Row3_2; static int C1_Row3_2; static int C0_Row3_2;
static int C3_Row4_2; static int C2_Row4_2; static int C1_Row4_2; static int C0_Row4_2;
static int C3_Row5_2; static int C2_Row5_2; static int C1_Row5_2; static int C0_Row5_2;
static int C3_Row6_2; static int C2_Row6_2; static int C1_Row6_2; static int C0_Row6_2;
static int C3_Row7_2; static int C2_Row7_2; static int C1_Row7_2; static int C0_Row7_2;
static int C3_Row8_2; static int C2_Row8_2; static int C1_Row8_2; static int C0_Row8_2;
static int C3_Row9_2; static int C2_Row9_2; static int C1_Row9_2; static int C0_Row9_2;
static int C3_Row10_2; static int C2_Row10_2; static int C1_Row10_2; static int C0_Row10_2;
static int C3_Row11_2; static int C2_Row11_2; static int C1_Row11_2; static int C0_Row11_2;
static int C3_Row12_2; static int C2_Row12_2; static int C1_Row12_2; static int C0_Row12_2;
static int C3_Row13_2; static int C2_Row13_2; static int C1_Row13_2; static int C0_Row13_2;
static int C3_Row14_2; static int C2_Row14_2; static int C1_Row14_2; static int C0_Row14_2;
static int C3_Row15_2; static int C2_Row15_2; static int C1_Row15_2; static int C0_Row15_2;

static int C3_Row0_3; static int C2_Row0_3; static int C1_Row0_3; static int C0_Row0_3;
static int C3_Row1_3; static int C2_Row1_3; static int C1_Row1_3; static int C0_Row1_3;
static int C3_Row2_3; static int C2_Row2_3; static int C1_Row2_3; static int C0_Row2_3;
static int C3_Row3_3; static int C2_Row3_3; static int C1_Row3_3; static int C0_Row3_3;
static int C3_Row4_3; static int C2_Row4_3; static int C1_Row4_3; static int C0_Row4_3;
static int C3_Row5_3; static int C2_Row5_3; static int C1_Row5_3; static int C0_Row5_3;
static int C3_Row6_3; static int C2_Row6_3; static int C1_Row6_3; static int C0_Row6_3;
static int C3_Row7_3; static int C2_Row7_3; static int C1_Row7_3; static int C0_Row7_3;
static int C3_Row8_3; static int C2_Row8_3; static int C1_Row8_3; static int C0_Row8_3;
static int C3_Row9_3; static int C2_Row9_3; static int C1_Row9_3; static int C0_Row9_3;
static int C3_Row10_3; static int C2_Row10_3; static int C1_Row10_3; static int C0_Row10_3;
static int C3_Row11_3; static int C2_Row11_3; static int C1_Row11_3; static int C0_Row11_3;
static int C3_Row12_3; static int C2_Row12_3; static int C1_Row12_3; static int C0_Row12_3;
static int C3_Row13_3; static int C2_Row13_3; static int C1_Row13_3; static int C0_Row13_3;
static int C3_Row14_3; static int C2_Row14_3; static int C1_Row14_3; static int C0_Row14_3;
static int C3_Row15_3; static int C2_Row15_3; static int C1_Row15_3; static int C0_Row15_3;

```
static int C3_Row0_4;  static int C2_Row0_4;  static int C1_Row0_4;  static int C0_Row0_4;
static int C3_Row1_4;  static int C2_Row1_4;  static int C1_Row1_4;  static int C0_Row1_4;
static int C3_Row2_4;  static int C2_Row2_4;  static int C1_Row2_4;  static int C0_Row2_4;
static int C3_Row3_4;  static int C2_Row3_4;  static int C1_Row3_4;  static int C0_Row3_4;
static int C3_Row4_4;  static int C2_Row4_4;  static int C1_Row4_4;  static int C0_Row4_4;
static int C3_Row5_4;  static int C2_Row5_4;  static int C1_Row5_4;  static int C0_Row5_4;
static int C3_Row6_4;  static int C2_Row6_4;  static int C1_Row6_4;  static int C0_Row6_4;
static int C3_Row7_4;  static int C2_Row7_4;  static int C1_Row7_4;  static int C0_Row7_4;
static int C3_Row8_4;  static int C2_Row8_4;  static int C1_Row8_4;  static int C0_Row8_4;
static int C3_Row9_4;  static int C2_Row9_4;  static int C1_Row9_4;  static int C0_Row9_4;
static int C3_Row10_4; static int C2_Row10_4; static int C1_Row10_4; static int C0_Row10_4;
static int C3_Row11_4; static int C2_Row11_4; static int C1_Row11_4; static int C0_Row11_4;
static int C3_Row12_4; static int C2_Row12_4; static int C1_Row12_4; static int C0_Row12_4;
static int C3_Row13_4; static int C2_Row13_4; static int C1_Row13_4; static int C0_Row13_4;
static int C3_Row14_4; static int C2_Row14_4; static int C1_Row14_4; static int C0_Row14_4;
static int C3_Row15_4; static int C2_Row15_4; static int C1_Row15_4; static int C0_Row15_4;

static int PSAD_0_11;   static int PSAD_0_12;   static int PSAD_0_13;   static int PSAD_0_14;
static int PSAD_1_11;   static int PSAD_1_12;   static int PSAD_1_13;   static int PSAD_1_14;
static int PSAD_2_11;   static int PSAD_2_12;   static int PSAD_2_13;   static int PSAD_2_14;
static int PSAD_Out_00; static int PSAD_Out_01; static int PSAD_Out_02; static int PSAD_Out_03;

static int PSAD_0_21;   static int PSAD_0_22;   static int PSAD_0_23;   static int PSAD_0_24;
static int PSAD_1_21;   static int PSAD_1_22;   static int PSAD_1_23;   static int PSAD_1_24;
static int PSAD_2_21;   static int PSAD_2_22;   static int PSAD_2_23;   static int PSAD_2_24;
static int PSAD_Out_10; static int PSAD_Out_11; static int PSAD_Out_12; static int PSAD_Out_13;

static int PSAD_0_31;   static int PSAD_0_32;   static int PSAD_0_33;   static int PSAD_0_34;
static int PSAD_1_31;   static int PSAD_1_32;   static int PSAD_1_33;   static int PSAD_1_34;
static int PSAD_2_31;   static int PSAD_2_32;   static int PSAD_2_33;   static int PSAD_2_34;
static int PSAD_Out_20; static int PSAD_Out_21; static int PSAD_Out_22; static int PSAD_Out_23;

static int PSAD_0_41;   static int PSAD_0_42;   static int PSAD_0_43;   static int PSAD_0_44;
static int PSAD_1_41;   static int PSAD_1_42;   static int PSAD_1_43;   static int PSAD_1_44;
static int PSAD_2_41;   static int PSAD_2_42;   static int PSAD_2_43;   static int PSAD_2_44;
static int PSAD_Out_30; static int PSAD_Out_31; static int PSAD_Out_32; static int PSAD_Out_33;

static int delayed4_1_11; static int delayed4_1_12; static int delayed4_1_13; static int delayed4_1_14;
static int delayed4_2_11; static int delayed4_2_12; static int delayed4_2_13; static int delayed4_2_14;
static int delayed4_3_11; static int delayed4_3_12; static int delayed4_3_13; static int delayed4_3_14;
static int delayed4_4_11; static int delayed4_4_12; static int delayed4_4_13; static int delayed4_4_14;

static int delayed4_1_21; static int delayed4_1_22; static int delayed4_1_23; static int delayed4_1_24;
static int delayed4_2_21; static int delayed4_2_22; static int delayed4_2_23; static int delayed4_2_24;
static int delayed4_3_21; static int delayed4_3_22; static int delayed4_3_23; static int delayed4_3_24;
static int delayed4_4_21; static int delayed4_4_22; static int delayed4_4_23; static int delayed4_4_24;

static int delayed8_1_1; static int delayed8_1_2;
static int delayed8_2_1; static int delayed8_2_2;
static int delayed8_3_1; static int delayed8_3_2;
static int delayed8_4_1; static int delayed8_4_2;
static int delayed8_5_1; static int delayed8_5_2;
static int delayed8_6_1; static int delayed8_6_2;
```

```
static int delayed8_7_1; static int delayed8_7_2;
static int delayed8_8_1; static int delayed8_8_2;


int update_Control_row_1 = 4;// initialized to values
int update_Control_row_2 = 4;// that are not used
int update_Control_row_3 = 4;
int update_Control_row_4 = 4;

int broadcast_bus_select_1 = 1;
int broadcast_bus_select_2 = 1;
int broadcast_bus_select_3 = 1;
int broadcast_bus_select_4 = 1;
int broadcast_bus_select_5 = 1;
int broadcast_bus_select_6 = 1;
int broadcast_bus_select_7 = 1;
int broadcast_bus_select_8 = 1;
int broadcast_bus_select_9 = 1;
int broadcast_bus_select_10 = 1;
int broadcast_bus_select_11 = 1;
int broadcast_bus_select_12 = 1;
int broadcast_bus_select_13 = 1;
int broadcast_bus_select_14 = 1;
int broadcast_bus_select_15 = 1;
int broadcast_bus_select_16 = 1;

if     (update_Cur_MB==0)  update_Control_row_1 = 0;
else if (update_Cur_MB==1)  update_Control_row_1 = 1;
else if (update_Cur_MB==2)  update_Control_row_1 = 2;
else if (update_Cur_MB==3)  update_Control_row_1 = 3;
else if (update_Cur_MB==4)  update_Control_row_2 = 0;
else if (update_Cur_MB==5)  update_Control_row_2 = 1;
else if (update_Cur_MB==6)  update_Control_row_2 = 2;
else if (update_Cur_MB==7)  update_Control_row_2 = 3;
else if (update_Cur_MB==8)  update_Control_row_3 = 0;
else if (update_Cur_MB==9)  update_Control_row_3 = 1;
else if (update_Cur_MB==10) update_Control_row_3 = 2;
else if (update_Cur_MB==11) update_Control_row_3 = 3;
else if (update_Cur_MB==12) update_Control_row_4 = 0;
else if (update_Cur_MB==13) update_Control_row_4 = 1;
else if (update_Cur_MB==14) update_Control_row_4 = 2;
else if (update_Cur_MB==15) update_Control_row_4 = 3;

if     (broadcast_bus_select==0){
      broadcast_bus_select_1 = 0;
      broadcast_bus_select_2 = 0;
      broadcast_bus_select_3 = 0;
      broadcast_bus_select_4 = 0;
      broadcast_bus_select_5 = 0;
      broadcast_bus_select_6 = 0;
      broadcast_bus_select_7 = 0;
      broadcast_bus_select_8 = 0;
      broadcast_bus_select_9 = 0;
      broadcast_bus_select_10 = 0;
```

```
        broadcast_bus_select_11 = 0;
        broadcast_bus_select_12 = 0;
        broadcast_bus_select_13 = 0;
        broadcast_bus_select_14 = 0;
        broadcast_bus_select_15 = 0;
        broadcast_bus_select_16 = 0;
    }
    else if (broadcast_bus_select==1){
        broadcast_bus_select_1 = 0;
    }
    else if (broadcast_bus_select==2){
        broadcast_bus_select_1 = 0;
        broadcast_bus_select_2 = 0;
    }
    else if (broadcast_bus_select==3){
        broadcast_bus_select_1 = 0;
        broadcast_bus_select_2 = 0;
        broadcast_bus_select_3 = 0;
    }
    else if (broadcast_bus_select==4){
        broadcast_bus_select_1 = 0;
        broadcast_bus_select_2 = 0;
        broadcast_bus_select_3 = 0;
        broadcast_bus_select_4 = 0;
    }
    else if (broadcast_bus_select==5){
        broadcast_bus_select_1 = 0;
        broadcast_bus_select_2 = 0;
        broadcast_bus_select_3 = 0;
        broadcast_bus_select_4 = 0;
        broadcast_bus_select_5 = 0;
    }
    else if (broadcast_bus_select==6){
        broadcast_bus_select_1 = 0;
        broadcast_bus_select_2 = 0;
        broadcast_bus_select_3 = 0;
        broadcast_bus_select_4 = 0;
        broadcast_bus_select_5 = 0;
        broadcast_bus_select_6 = 0;
    }
    else if (broadcast_bus_select==7){
        broadcast_bus_select_1 = 0;
        broadcast_bus_select_2 = 0;
        broadcast_bus_select_3 = 0;
        broadcast_bus_select_4 = 0;
        broadcast_bus_select_5 = 0;
        broadcast_bus_select_6 = 0;
        broadcast_bus_select_7 = 0;
    }
    else if (broadcast_bus_select==8){
        broadcast_bus_select_1 = 0;
        broadcast_bus_select_2 = 0;
        broadcast_bus_select_3 = 0;
        broadcast_bus_select_4 = 0;
```

```
        broadcast_bus_select_5 = 0;
        broadcast_bus_select_6 = 0;
        broadcast_bus_select_7 = 0;
        broadcast_bus_select_8 = 0;
}
else if (broadcast_bus_select==9){
        broadcast_bus_select_1 = 0;
        broadcast_bus_select_2 = 0;
        broadcast_bus_select_3 = 0;
        broadcast_bus_select_4 = 0;
        broadcast_bus_select_5 = 0;
        broadcast_bus_select_6 = 0;
        broadcast_bus_select_7 = 0;
        broadcast_bus_select_8 = 0;
        broadcast_bus_select_9 = 0;
}
else if (broadcast_bus_select==10){
        broadcast_bus_select_1 = 0;
        broadcast_bus_select_2 = 0;
        broadcast_bus_select_3 = 0;
        broadcast_bus_select_4 = 0;
        broadcast_bus_select_5 = 0;
        broadcast_bus_select_6 = 0;
        broadcast_bus_select_7 = 0;
        broadcast_bus_select_8 = 0;
        broadcast_bus_select_9 = 0;
        broadcast_bus_select_10 = 0;
}
else if (broadcast_bus_select==11){
        broadcast_bus_select_1 = 0;
        broadcast_bus_select_2 = 0;
        broadcast_bus_select_3 = 0;
        broadcast_bus_select_4 = 0;
        broadcast_bus_select_5 = 0;
        broadcast_bus_select_6 = 0;
        broadcast_bus_select_7 = 0;
        broadcast_bus_select_8 = 0;
        broadcast_bus_select_9 = 0;
        broadcast_bus_select_10 = 0;
        broadcast_bus_select_11 = 0;
}
else if (broadcast_bus_select==12){
        broadcast_bus_select_1 = 0;
        broadcast_bus_select_2 = 0;
        broadcast_bus_select_3 = 0;
        broadcast_bus_select_4 = 0;
        broadcast_bus_select_5 = 0;
        broadcast_bus_select_6 = 0;
        broadcast_bus_select_7 = 0;
        broadcast_bus_select_8 = 0;
        broadcast_bus_select_9 = 0;
        broadcast_bus_select_10 = 0;
        broadcast_bus_select_11 = 0;
        broadcast_bus_select_12 = 0;
```

```
}
else if (broadcast_bus_select==13){
	broadcast_bus_select_1 = 0;
	broadcast_bus_select_2 = 0;
	broadcast_bus_select_3 = 0;
	broadcast_bus_select_4 = 0;
	broadcast_bus_select_5 = 0;
	broadcast_bus_select_6 = 0;
	broadcast_bus_select_7 = 0;
	broadcast_bus_select_8 = 0;
	broadcast_bus_select_9 = 0;
	broadcast_bus_select_10 = 0;
	broadcast_bus_select_11 = 0;
	broadcast_bus_select_12 = 0;
	broadcast_bus_select_13 = 0;
}
else if (broadcast_bus_select==14){
	broadcast_bus_select_1 = 0;
	broadcast_bus_select_2 = 0;
	broadcast_bus_select_3 = 0;
	broadcast_bus_select_4 = 0;
	broadcast_bus_select_5 = 0;
	broadcast_bus_select_6 = 0;
	broadcast_bus_select_7 = 0;
	broadcast_bus_select_8 = 0;
	broadcast_bus_select_9 = 0;
	broadcast_bus_select_10 = 0;
	broadcast_bus_select_11 = 0;
	broadcast_bus_select_12 = 0;
	broadcast_bus_select_13 = 0;
	broadcast_bus_select_14 = 0;
}
else if (broadcast_bus_select==15){
	broadcast_bus_select_1 = 0;
	broadcast_bus_select_2 = 0;
	broadcast_bus_select_3 = 0;
	broadcast_bus_select_4 = 0;
	broadcast_bus_select_5 = 0;
	broadcast_bus_select_6 = 0;
	broadcast_bus_select_7 = 0;
	broadcast_bus_select_8 = 0;
	broadcast_bus_select_9 = 0;
	broadcast_bus_select_10 = 0;
	broadcast_bus_select_11 = 0;
	broadcast_bus_select_12 = 0;
	broadcast_bus_select_13 = 0;
	broadcast_bus_select_14 = 0;
	broadcast_bus_select_15 = 0;
}

// --------------------------- Instantiations ---------------------------------------


// 4x4: 1,1
```

```
 four_by_four_block (clk, reset, enable, update_Control_row_1,
                                broadcast_column_1_a,
                                broadcast_column_2_a,
                                broadcast_column_3_a,
                                broadcast_column_4_a,
                                broadcast_column_1_b,
                                broadcast_column_2_b,
                                broadcast_column_3_b,
                                broadcast_column_4_b,
                                broadcast_bus_select_1,
                                broadcast_bus_select_2,
                                broadcast_bus_select_3,
                                broadcast_bus_select_4,
                                &C3_Row0_1, &C2_Row0_1, &C1_Row0_1, &C0_Row0_1,
                                &C3_Row1_1, &C2_Row1_1, &C1_Row1_1, &C0_Row1_1,
                                &C3_Row2_1, &C2_Row2_1, &C1_Row2_1, &C0_Row2_1,
                                &C3_Row3_1, &C2_Row3_1, &C1_Row3_1, &C0_Row3_1,
                                &PSAD_0_11,
                                &PSAD_1_11,
                                &PSAD_2_11,
                                &PSAD_Out_00
                                );

// 4x4: 1,2
 four_by_four_block (clk, reset, enable, update_Control_row_1,
                                broadcast_column_5_a,
                                broadcast_column_6_a,
                                broadcast_column_7_a,
                                broadcast_column_8_a,
                                broadcast_column_5_b,
                                broadcast_column_6_b,
                                broadcast_column_7_b,
                                broadcast_column_8_b,
                                broadcast_bus_select_1,
                                broadcast_bus_select_2,
                                broadcast_bus_select_3,
                                broadcast_bus_select_4,
                                &C3_Row0_2, &C2_Row0_2, &C1_Row0_2, &C0_Row0_2,
                                &C3_Row1_2, &C2_Row1_2, &C1_Row1_2, &C0_Row1_2,
                                &C3_Row2_2, &C2_Row2_2, &C1_Row2_2, &C0_Row2_2,
                                &C3_Row3_2, &C2_Row3_2, &C1_Row3_2, &C0_Row3_2,
                                &PSAD_0_12,
                                &PSAD_1_12,
                                &PSAD_2_12,
                                &PSAD_Out_01
                                );

// 4x4: 1,3
 four_by_four_block (clk, reset, enable, update_Control_row_1,
                                broadcast_column_9_a,
                                broadcast_column_10_a,
                                broadcast_column_11_a,
                                broadcast_column_12_a,
                                broadcast_column_9_b,
```

```
                                broadcast_column_10_b,
                                broadcast_column_11_b,
                                broadcast_column_12_b,
                                broadcast_bus_select_1,
                                broadcast_bus_select_2,
                                broadcast_bus_select_3,
                                broadcast_bus_select_4,
                                &C3_Row0_3, &C2_Row0_3, &C1_Row0_3, &C0_Row0_3,
                                &C3_Row1_3, &C2_Row1_3, &C1_Row1_3, &C0_Row1_3,
                                &C3_Row2_3, &C2_Row2_3, &C1_Row2_3, &C0_Row2_3,
                                &C3_Row3_3, &C2_Row3_3, &C1_Row3_3, &C0_Row3_3,
                                &PSAD_0_13,
                                &PSAD_1_13,
                                &PSAD_2_13,
                                &PSAD_Out_02
                                );

// 4x4: 1,4
 four_by_four_block (clk, reset, enable, update_Control_row_1,
                                broadcast_column_13_a,
                                broadcast_column_14_a,
                                broadcast_column_15_a,
                                broadcast_column_16_a,
                                broadcast_column_13_b,
                                broadcast_column_14_b,
                                broadcast_column_15_b,
                                broadcast_column_16_b,
                                broadcast_bus_select_1,
                                broadcast_bus_select_2,
                                broadcast_bus_select_3,
                                broadcast_bus_select_4,
                                &C3_Row0_4, &C2_Row0_4, &C1_Row0_4, &C0_Row0_4,
                                &C3_Row1_4, &C2_Row1_4, &C1_Row1_4, &C0_Row1_4,
                                &C3_Row2_4, &C2_Row2_4, &C1_Row2_4, &C0_Row2_4,
                                &C3_Row3_4, &C2_Row3_4, &C1_Row3_4, &C0_Row3_4,
                                &PSAD_0_14,
                                &PSAD_1_14,
                                &PSAD_2_14,
                                &PSAD_Out_03
                                );

//--------------------------------------------------------------------------------

// 4x4: 2,1
 four_by_four_block (clk, reset, enable, update_Control_row_2,
                                broadcast_column_1_a,
                                broadcast_column_2_a,
                                broadcast_column_3_a,
                                broadcast_column_4_a,
                                broadcast_column_1_b,
                                broadcast_column_2_b,
                                broadcast_column_3_b,
                                broadcast_column_4_b,
                                broadcast_bus_select_5,
```

```
                                broadcast_bus_select_6,
                                broadcast_bus_select_7,
                                broadcast_bus_select_8,
                                &C3_Row4_1, &C2_Row4_1, &C1_Row4_1, &C0_Row4_1,
                                &C3_Row5_1, &C2_Row5_1, &C1_Row5_1, &C0_Row5_1,
                                &C3_Row6_1, &C2_Row6_1, &C1_Row6_1, &C0_Row6_1,
                                &C3_Row7_1, &C2_Row7_1, &C1_Row7_1, &C0_Row7_1,
                                &PSAD_0_21,
                                &PSAD_1_21,
                                &PSAD_2_21,
                                &PSAD_Out_10
                                );

// 4x4: 2,2
 four_by_four_block (clk, reset, enable, update_Control_row_2,
                                broadcast_column_5_a,
                                broadcast_column_6_a,
                                broadcast_column_7_a,
                                broadcast_column_8_a,
                                broadcast_column_5_b,
                                broadcast_column_6_b,
                                broadcast_column_7_b,
                                broadcast_column_8_b,
                                broadcast_bus_select_5,
                                broadcast_bus_select_6,
                                broadcast_bus_select_7,
                                broadcast_bus_select_8,
                                &C3_Row4_2, &C2_Row4_2, &C1_Row4_2, &C0_Row4_2,
                                &C3_Row5_2, &C2_Row5_2, &C1_Row5_2, &C0_Row5_2,
                                &C3_Row6_2, &C2_Row6_2, &C1_Row6_2, &C0_Row6_2,
                                &C3_Row7_2, &C2_Row7_2, &C1_Row7_2, &C0_Row7_2,
                                &PSAD_0_22,
                                &PSAD_1_22,
                                &PSAD_2_22,
                                &PSAD_Out_11
                                );

// 4x4: 2,3
 four_by_four_block (clk, reset, enable, update_Control_row_2,
                                broadcast_column_9_a,
                                broadcast_column_10_a,
                                broadcast_column_11_a,
                                broadcast_column_12_a,
                                broadcast_column_9_b,
                                broadcast_column_10_b,
                                broadcast_column_11_b,
                                broadcast_column_12_b,
                                broadcast_bus_select_5,
                                broadcast_bus_select_6,
                                broadcast_bus_select_7,
                                broadcast_bus_select_8,
                                &C3_Row4_3, &C2_Row4_3, &C1_Row4_3, &C0_Row4_3,
                                &C3_Row5_3, &C2_Row5_3, &C1_Row5_3, &C0_Row5_3,
                                &C3_Row6_3, &C2_Row6_3, &C1_Row6_3, &C0_Row6_3,
```

```
                                &C3_Row7_3, &C2_Row7_3, &C1_Row7_3, &C0_Row7_3,
                                &PSAD_0_23,
                                &PSAD_1_23,
                                &PSAD_2_23,
                                &PSAD_Out_12
                                );

// 4x4: 2,4
 four_by_four_block (clk, reset, enable, update_Control_row_2,
                                broadcast_column_13_a,
                                broadcast_column_14_a,
                                broadcast_column_15_a,
                                broadcast_column_16_a,
                                broadcast_column_13_b,
                                broadcast_column_14_b,
                                broadcast_column_15_b,
                                broadcast_column_16_b,
                                broadcast_bus_select_5,
                                broadcast_bus_select_6,
                                broadcast_bus_select_7,
                                broadcast_bus_select_8,
                                &C3_Row4_4, &C2_Row4_4, &C1_Row4_4, &C0_Row4_4,
                                &C3_Row5_4, &C2_Row5_4, &C1_Row5_4, &C0_Row5_4,
                                &C3_Row6_4, &C2_Row6_4, &C1_Row6_4, &C0_Row6_4,
                                &C3_Row7_4, &C2_Row7_4, &C1_Row7_4, &C0_Row7_4,
                                &PSAD_0_24,
                                &PSAD_1_24,
                                &PSAD_2_24,
                                &PSAD_Out_13
                                );

//--------------------------------------------------------------------------------

//--------------------------------------------------------------------------------

// 4x4: 3,1
 four_by_four_block (clk, reset, enable, update_Control_row_3,
                                broadcast_column_1_a,
                                broadcast_column_2_a,
                                broadcast_column_3_a,
                                broadcast_column_4_a,
                                broadcast_column_1_b,
                                broadcast_column_2_b,
                                broadcast_column_3_b,
                                broadcast_column_4_b,
                                broadcast_bus_select_9,
                                broadcast_bus_select_10,
                                broadcast_bus_select_11,
                                broadcast_bus_select_12,
                                &C3_Row8_1,  &C2_Row8_1,  &C1_Row8_1,  &C0_Row8_1,
                                &C3_Row9_1,  &C2_Row9_1,  &C1_Row9_1,  &C0_Row9_1,
                                &C3_Row10_1, &C2_Row10_1, &C1_Row10_1, &C0_Row10_1,
                                &C3_Row11_1, &C2_Row11_1, &C1_Row11_1, &C0_Row11_1,
                                &PSAD_0_31,
```

```
                        &PSAD_1_31,
                        &PSAD_2_31,
                        &PSAD_Out_20
                        );

// 4x4: 3,2
 four_by_four_block (clk, reset, enable, update_Control_row_3,
                        broadcast_column_5_a,
                        broadcast_column_6_a,
                        broadcast_column_7_a,
                        broadcast_column_8_a,
                        broadcast_column_5_b,
                        broadcast_column_6_b,
                        broadcast_column_7_b,
                        broadcast_column_8_b,
                        broadcast_bus_select_9,
                        broadcast_bus_select_10,
                        broadcast_bus_select_11,
                        broadcast_bus_select_12,
                        &C3_Row8_2,  &C2_Row8_2,  &C1_Row8_2,  &C0_Row8_2,
                        &C3_Row9_2,  &C2_Row9_2,  &C1_Row9_2,  &C0_Row9_2,
                        &C3_Row10_2, &C2_Row10_2, &C1_Row10_2, &C0_Row10_2,
                        &C3_Row11_2, &C2_Row11_2, &C1_Row11_2, &C0_Row11_2,
                        &PSAD_0_32,
                        &PSAD_1_32,
                        &PSAD_2_32,
                        &PSAD_Out_21
                        );

// 4x4: 3,3
 four_by_four_block (clk, reset, enable, update_Control_row_3,
                        broadcast_column_9_a,
                        broadcast_column_10_a,
                        broadcast_column_11_a,
                        broadcast_column_12_a,
                        broadcast_column_9_b,
                        broadcast_column_10_b,
                        broadcast_column_11_b,
                        broadcast_column_12_b,
                        broadcast_bus_select_9,
                        broadcast_bus_select_10,
                        broadcast_bus_select_11,
                        broadcast_bus_select_12,
                        &C3_Row8_3,  &C2_Row8_3,  &C1_Row8_3,  &C0_Row8_3,
                        &C3_Row9_3,  &C2_Row9_3,  &C1_Row9_3,  &C0_Row9_3,
                        &C3_Row10_3, &C2_Row10_3, &C1_Row10_3, &C0_Row10_3,
                        &C3_Row11_3, &C2_Row11_3, &C1_Row11_3, &C0_Row11_3,
                        &PSAD_0_33,
                        &PSAD_1_33,
                        &PSAD_2_33,
                        &PSAD_Out_22
                        );

// 4x4: 3,4
```

```
 four_by_four_block (clk, reset, enable, update_Control_row_3,
                                broadcast_column_13_a,
                                broadcast_column_14_a,
                                broadcast_column_15_a,
                                broadcast_column_16_a,
                                broadcast_column_13_b,
                                broadcast_column_14_b,
                                broadcast_column_15_b,
                                broadcast_column_16_b,
                                broadcast_bus_select_9,
                                broadcast_bus_select_10,
                                broadcast_bus_select_11,
                                broadcast_bus_select_12,
                                &C3_Row8_4,  &C2_Row8_4,  &C1_Row8_4,  &C0_Row8_4,
                                &C3_Row9_4,  &C2_Row9_4,  &C1_Row9_4,  &C0_Row9_4,
                                &C3_Row10_4, &C2_Row10_4, &C1_Row10_4, &C0_Row10_4,
                                &C3_Row11_4, &C2_Row11_4, &C1_Row11_4, &C0_Row11_4,
                                &PSAD_0_34,
                                &PSAD_1_34,
                                &PSAD_2_34,
                                &PSAD_Out_23
                                );

//--------------------------------------------------------------------------------
//--------------------------------------------------------------------------------

// 4x4: 4,1
 four_by_four_block (clk, reset, enable, update_Control_row_4,
                                broadcast_column_1_a,
                                broadcast_column_2_a,
                                broadcast_column_3_a,
                                broadcast_column_4_a,
                                broadcast_column_1_b,
                                broadcast_column_2_b,
                                broadcast_column_3_b,
                                broadcast_column_4_b,
                                broadcast_bus_select_13,
                                broadcast_bus_select_14,
                                broadcast_bus_select_15,
                                broadcast_bus_select_16,
                                &C3_Row12_1, &C2_Row12_1, &C1_Row12_1, &C0_Row12_1,
                                &C3_Row13_1, &C2_Row13_1, &C1_Row13_1, &C0_Row13_1,
                                &C3_Row14_1, &C2_Row14_1, &C1_Row14_1, &C0_Row14_1,
                                &C3_Row15_1, &C2_Row15_1, &C1_Row15_1, &C0_Row15_1,
                                &PSAD_0_41,
                                &PSAD_1_41,
                                &PSAD_2_41,
                                &PSAD_Out_30
                                );

// 4x4: 4,2
 four_by_four_block (clk, reset, enable, update_Control_row_4,
                                broadcast_column_5_a,
                                broadcast_column_6_a,
```

```
                                broadcast_column_7_a,
                                broadcast_column_8_a,
                                broadcast_column_5_b,
                                broadcast_column_6_b,
                                broadcast_column_7_b,
                                broadcast_column_8_b,
                                broadcast_bus_select_13,
                                broadcast_bus_select_14,
                                broadcast_bus_select_15,
                                broadcast_bus_select_16,
                                &C3_Row12_2, &C2_Row12_2, &C1_Row12_2, &C0_Row12_2,
                                &C3_Row13_2, &C2_Row13_2, &C1_Row13_2, &C0_Row13_2,
                                &C3_Row14_2, &C2_Row14_2, &C1_Row14_2, &C0_Row14_2,
                                &C3_Row15_2, &C2_Row15_2, &C1_Row15_2, &C0_Row15_2,
                                &PSAD_0_42,
                                &PSAD_1_42,
                                &PSAD_2_42,
                                &PSAD_Out_31
                                );

// 4x4: 4,3
 four_by_four_block (clk, reset, enable, update_Control_row_4,
                                broadcast_column_9_a,
                                broadcast_column_10_a,
                                broadcast_column_11_a,
                                broadcast_column_12_a,
                                broadcast_column_9_b,
                                broadcast_column_10_b,
                                broadcast_column_11_b,
                                broadcast_column_12_b,
                                broadcast_bus_select_13,
                                broadcast_bus_select_14,
                                broadcast_bus_select_15,
                                broadcast_bus_select_16,
                                &C3_Row12_3, &C2_Row12_3, &C1_Row12_3, &C0_Row12_3,
                                &C3_Row13_3, &C2_Row13_3, &C1_Row13_3, &C0_Row13_3,
                                &C3_Row14_3, &C2_Row14_3, &C1_Row14_3, &C0_Row14_3,
                                &C3_Row15_3, &C2_Row15_3, &C1_Row15_3, &C0_Row15_3,
                                &PSAD_0_43,
                                &PSAD_1_43,
                                &PSAD_2_43,
                                &PSAD_Out_32
                                );

// 4x4: 4,4
 four_by_four_block (clk, reset, enable, update_Control_row_4,
                                broadcast_column_13_a,
                                broadcast_column_14_a,
                                broadcast_column_15_a,
                                broadcast_column_16_a,
                                broadcast_column_13_b,
                                broadcast_column_14_b,
                                broadcast_column_15_b,
                                broadcast_column_16_b,
```

```
                        broadcast_bus_select_13,
                        broadcast_bus_select_14,
                        broadcast_bus_select_15,
                        broadcast_bus_select_16,
                        &C3_Row12_4, &C2_Row12_4, &C1_Row12_4, &C0_Row12_4,
                        &C3_Row13_4, &C2_Row13_4, &C1_Row13_4, &C0_Row13_4,
                        &C3_Row14_4, &C2_Row14_4, &C1_Row14_4, &C0_Row14_4,
                        &C3_Row15_4, &C2_Row15_4, &C1_Row15_4, &C0_Row15_4,
                        &PSAD_0_44,
                        &PSAD_1_44,
                        &PSAD_2_44,
                        &PSAD_Out_33
                        );

//-------------------------------------------------------------------------------

//-------------------------------------------------------------------------------

// The first 4 on the first row of 16 4x4 SAD output registers +
// updating the delay registers that feed them

*B4x4_00_out = delayed4_4_11;

delayed4_4_11 = delayed4_3_11;
delayed4_3_11 = delayed4_2_11;
delayed4_2_11 = delayed4_1_11;
delayed4_1_11 = PSAD_Out_00;

*B4x4_01_out = delayed4_4_12;

delayed4_4_12 = delayed4_3_12;
delayed4_3_12 = delayed4_2_12;
delayed4_2_12 = delayed4_1_12;
delayed4_1_12 = PSAD_Out_01;

*B4x4_02_out = delayed4_4_13;

delayed4_4_13 = delayed4_3_13;
delayed4_3_13 = delayed4_2_13;
delayed4_2_13 = delayed4_1_13;
delayed4_1_13 = PSAD_Out_02;

*B4x4_03_out = delayed4_4_14;

delayed4_4_14 = delayed4_3_14;
delayed4_3_14 = delayed4_2_14;
delayed4_2_14 = delayed4_1_14;
delayed4_1_14 = PSAD_Out_03;

//-----------------------------------------//
// The 2nd row of 16 4x4 SAD output registers

*B4x4_10_out = PSAD_Out_10;
*B4x4_11_out = PSAD_Out_11;
```

```
*B4x4_12_out = PSAD_Out_12;
*B4x4_13_out = PSAD_Out_13;

// The 3rd row of 16 4x4 SAD output registers +
// updating the delay registers that feed them

*B4x4_20_out = delayed4_4_21;

delayed4_4_21 = delayed4_3_21;
delayed4_3_21 = delayed4_2_21;
delayed4_2_21 = delayed4_1_21;
delayed4_1_21 = PSAD_Out_20;

*B4x4_21_out = delayed4_4_22;

delayed4_4_22 = delayed4_3_22;
delayed4_3_22 = delayed4_2_22;
delayed4_2_22 = delayed4_1_22;
delayed4_1_22 = PSAD_Out_21;

*B4x4_22_out = delayed4_4_23;

delayed4_4_23 = delayed4_3_23;
delayed4_3_23 = delayed4_2_23;
delayed4_2_23 = delayed4_1_23;
delayed4_1_23 = PSAD_Out_22;

*B4x4_23_out = delayed4_4_24;

delayed4_4_24 = delayed4_3_24;
delayed4_3_24 = delayed4_2_24;
delayed4_2_24 = delayed4_1_24;
delayed4_1_24 = PSAD_Out_23;

//-------------------------------------------//
// The 4th row of 16 4x4 SAD output registers

*B4x4_30_out = PSAD_Out_30;
*B4x4_31_out = PSAD_Out_31;
*B4x4_32_out = PSAD_Out_32;
*B4x4_33_out = PSAD_Out_33;

//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------

// The 8 4x8 SAD output registers, labeled by row/column

*B4x8_00_out = *B4x4_00_out + *B4x4_10_out;
*B4x8_01_out = *B4x4_01_out + *B4x4_11_out;
*B4x8_02_out = *B4x4_02_out + *B4x4_12_out;
*B4x8_03_out = *B4x4_03_out + *B4x4_13_out;

*B4x8_10_out = *B4x4_20_out + *B4x4_30_out;
```

```
*B4x8_11_out = *B4x4_21_out + *B4x4_31_out;
*B4x8_12_out = *B4x4_22_out + *B4x4_32_out;
*B4x8_13_out = *B4x4_23_out + *B4x4_33_out;

// The 8 8x4 SAD output registers, labeled by row/column

*B8x4_00_out = *B4x4_00_out + *B4x4_01_out;
*B8x4_01_out = *B4x4_02_out + *B4x4_03_out;

*B8x4_10_out = *B4x4_10_out + *B4x4_11_out;
*B8x4_11_out = *B4x4_12_out + *B4x4_13_out;

*B8x4_20_out = *B4x4_20_out + *B4x4_21_out;
*B8x4_21_out = *B4x4_22_out + *B4x4_23_out;

*B8x4_30_out = *B4x4_30_out + *B4x4_31_out;
*B8x4_31_out = *B4x4_32_out + *B4x4_33_out;

// The 4 8x8 SAD output registers, labeled by row/column

*B8x8_00_out = *B8x4_00_out + *B8x4_10_out;
*B8x8_01_out = *B8x4_01_out + *B8x4_11_out;

*B8x8_10_out = *B8x4_20_out + *B8x4_30_out;
*B8x8_11_out = *B8x4_21_out + *B8x4_31_out;

// The 2 8x16 SAD output registers, labeled by column

*B8x16_0_out = delayed8_8_1 + *B8x8_10_out;
*B8x16_1_out = delayed8_8_2 + *B8x8_11_out;

delayed8_8_1 = delayed8_7_1; delayed8_8_2 = delayed8_7_2;
delayed8_7_1 = delayed8_6_1; delayed8_7_2 = delayed8_6_2;
delayed8_6_1 = delayed8_5_1; delayed8_6_2 = delayed8_5_2;
delayed8_5_1 = delayed8_4_1; delayed8_5_2 = delayed8_4_2;
delayed8_4_1 = delayed8_3_1; delayed8_4_2 = delayed8_3_2;
delayed8_3_1 = delayed8_2_1; delayed8_3_2 = delayed8_2_2;
delayed8_2_1 = delayed8_1_1; delayed8_2_2 = delayed8_1_2;

delayed8_1_1 = *B8x8_00_out; delayed8_1_2 = *B8x8_01_out;

// The 2 16x8 SAD output registers, labeled by row
*B16x8_0_out = *B8x8_00_out + *B8x8_01_out;
*B16x8_1_out = *B8x8_10_out + *B8x8_11_out;

// The 1 16x16 SAD output register
*B16x16_out = *B8x16_0_out + *B8x16_1_out;


}// ends the main_data_path module
```

```
#include <stdlib.h>
#include <stdio.h>
//-------------------------------------------------------------------------------------------------------------------------------------
----------//
// This (on_chip_mem) function uses 2 block_mem_66x16 module and 2 block_mem30x16 module
// memory functions to create a sinlge memory module (containing both A & B memory partitions)


// External Functions located in other Project Files

// Memory Partition A
void block_mem_66x16 ( // inputs
                       int write_enable,
                       int address, // 7-bits (i.e. 0 to 65 addresses)
                       // input memory array address
                       int array_66x16 [66][16],
                       // Input Bus
                       int dataIn_16_Bytes[16],
                       // Output Bus
                       int dataOut_16_Bytes[16]
                      );

// Memory Partion B
void block_mem_30x16 ( // inputs
                       int write_enable,
                       int address, // 5-bits (i.e. 0 to 29 addresses)
                       // input memory array address
                       int array_30x16 [30][16],
                       // Input Bus
                       int dataIn_16_Bytes[16],
                       // Output Bus
                       int dataOut_16_Bytes[16]
                      );

// ------------------------------------------------------------------------------ //

void on_chip_mem (// inputs
                       int write_enable,
                       int address, // 6-bits (i.e. 0 to 47 search pixel rows)
                       int load_bottom,
                       int load_top,

                       int read_L0_L1,
                       int read_L1_L2,
                       int read_L2_L3,

                       int block_66x16_mem_1 [66][16],
                       int block_66x16_mem_2 [66][16],
                       int block_30x16_mem_1 [66][16],
                       int block_30x16_mem_2 [66][16],

                       int *partition_A_Counter,
                       int *partition_B_Counter,
```

```
                        // Input Bus
                        int dataIn_32_Bytes[32],
                        // Output Buses
                        int dataOut_31_Bytes_A[32],// an extra byte in the array is added
                        int dataOut_31_Bytes_B[32] // for programming convenience
                       ){

    //static   int block_66x16_mem_1 [66][16];
    //static   int block_66x16_mem_2 [66][16];
    //static   int block_30x16_mem_1 [66][16];
    //static   int block_30x16_mem_2 [66][16];
    //static int partition_A_Counter;
    //static int partition_B_Counter;

    int mem_bus_1_A[16];
    int mem_bus_2_A[16];
    int mem_bus_1_B[16];
    int mem_bus_2_B[16];

    int index = 0;

// ---------------------------------------------------------------------------------------------//
// Reading Out Memory Partitions A & B

    // Logical L0_L1 type of switching for Partition B has to always
    // Lag Partition A switches by at least 15 cycles
    // i.e. Partition A switches to the next logical mem set before Partition B has to

    if (read_L0_L1==1) {
        //printf("\nInitial A Counter: %d\n", *partition_A_Counter);
        block_mem_66x16 (0, *partition_A_Counter, block_66x16_mem_1, dataIn_32_Bytes, mem_bus_1_A);
        block_mem_66x16 (0, *partition_A_Counter, block_66x16_mem_2, dataIn_32_Bytes, mem_bus_2_A);
        *partition_A_Counter = *partition_A_Counter + 1;
        //printf("\nSecond A Counter: %d\n", *partition_A_Counter);
        for (index=0; index<16; index++){
            dataOut_31_Bytes_A[index]=mem_bus_1_A[index];
            dataOut_31_Bytes_A[index+16]=mem_bus_2_A[index];
        }
    } // ends if read_L0_L1==1
    if (read_L1_L2==1) {
        block_mem_66x16 (0, (*partition_A_Counter+33), block_66x16_mem_1, dataIn_32_Bytes,
mem_bus_1_A);
        block_mem_66x16 (0, *partition_A_Counter, block_66x16_mem_2, dataIn_32_Bytes, mem_bus_2_A);
        block_mem_30x16 (0, *partition_B_Counter, block_30x16_mem_1, dataIn_32_Bytes, mem_bus_1_B);
        block_mem_30x16 (0, *partition_B_Counter, block_30x16_mem_2, dataIn_32_Bytes, mem_bus_2_B);
        *partition_A_Counter = *partition_A_Counter + 1;
        *partition_B_Counter = *partition_B_Counter + 1;
        for (index=0; index<16; index++){
            dataOut_31_Bytes_A[index]=mem_bus_2_A[index];
            dataOut_31_Bytes_A[index+16]=mem_bus_1_A[index];
            dataOut_31_Bytes_B[index]=mem_bus_1_B[index];
            dataOut_31_Bytes_B[index+16]=mem_bus_2_B[index];
        }
```

```
    } // ends if read_L1_L2==1
    if (read_L2_L3==1) {
        //printf("\nThe address: %d\n", address);
        block_mem_66x16 (0, (*partition_A_Counter+33), block_66x16_mem_1, dataIn_32_Bytes,
mem_bus_1_A);
        block_mem_66x16 (0, (*partition_A_Counter+33), block_66x16_mem_2, dataIn_32_Bytes,
mem_bus_2_A);
        for (index=0; index<16; index++){
            dataOut_31_Bytes_A[index]=mem_bus_1_A[index];
            dataOut_31_Bytes_A[index+16]=mem_bus_2_A[index];
        }

        if (address>17){
            //printf("\nThe address: %d\n", address);
            block_mem_30x16 (0, (*partition_B_Counter+15), block_30x16_mem_1, dataIn_32_Bytes,
mem_bus_1_B);
            block_mem_30x16 (0, *partition_B_Counter, block_30x16_mem_2, dataIn_32_Bytes,
mem_bus_2_B);

            *partition_B_Counter = *partition_B_Counter + 1;
            for (index=0; index<16; index++){
                dataOut_31_Bytes_B[index]=mem_bus_2_B[index];
                dataOut_31_Bytes_B[index+16]=mem_bus_1_B[index];
            }
        }
        else {
            block_mem_30x16 (0, (*partition_B_Counter+15), block_30x16_mem_1, dataIn_32_Bytes,
mem_bus_1_B);
            block_mem_30x16 (0, (*partition_B_Counter+15), block_30x16_mem_2, dataIn_32_Bytes,
mem_bus_2_B);
            *partition_B_Counter = *partition_B_Counter + 1;
            for (index=0; index<16; index++){
                dataOut_31_Bytes_B[index]=mem_bus_1_B[index];
                dataOut_31_Bytes_B[index+16]=mem_bus_2_B[index];
            }
        }

        *partition_A_Counter = *partition_A_Counter + 1;
    } // ends if read_L2_L3==1

if (*partition_A_Counter==33){
    *partition_A_Counter = 0;
    *partition_B_Counter = 0;
}
if (*partition_B_Counter==15) *partition_B_Counter = 0;
// ------------------------------------------------------------------------------------------//

// ------------------------------------------------------------------------------------------//
// The Writing/Loading Process

    if (write_enable==1){
if (address<33){
    if (load_bottom==1) {
        block_mem_66x16 (1,address,block_66x16_mem_1,dataIn_32_Bytes,mem_bus_1_A);
```

```
            block_mem_66x16 (1,address,block_66x16_mem_2,(dataIn_32_Bytes+16),mem_bus_2_A);
        } // ends if load_bottom==1
        if (load_top==1) {
            block_mem_66x16 (1,(address+33),block_66x16_mem_1,dataIn_32_Bytes,mem_bus_1_A);
            //printf("dataIn: %d\n", dataIn_32_Bytes[0]);
            block_mem_66x16 (1,(address+33),block_66x16_mem_2,(dataIn_32_Bytes+16),mem_bus_2_A);
        } // ends if load_top==1
} // ends if address<33
else{
        if (load_bottom==1) {
            block_mem_30x16 (1,(address-33),// Memory Partition B begins at
                                            // Reference row 33, thus 33 is deducted
                                            // to offset the external starting address down to 0
                        block_30x16_mem_1,dataIn_32_Bytes,mem_bus_1_B);
            block_mem_30x16 (1,(address-33),block_30x16_mem_2,(dataIn_32_Bytes+16),mem_bus_2_B);
        } // ends if load_bottom==1
        if (load_top==1) {
            block_mem_30x16 (1,((address-33)+15),block_30x16_mem_1,dataIn_32_Bytes,mem_bus_1_B);
            block_mem_30x16 (1,((address-33)+15),block_30x16_mem_2,(dataIn_32_Bytes+16),mem_bus_2_B);
        } // ends if load_top==1
} // ends else if address>=34
        }// ends if write_enable==1
// ------------------------------------------------------------------------------------------//

}// ends function on_chip_mem
```

```
#include <stdlib.h>
#include <stdio.h>

FILE *fp;
//-------------------------------------------------------------------------------------------------------------------------------
----------//
// This function produces an 8-bit Absolute Difference Result
// for R and C.  It is used only in Row 0 where further CSA
// modules are not used

int PE_AD (int clk, int reset, int update_Cur_MB, int R, int C,
              int *C_Register){

      int Nine_bit_sum = 0;

      int s8_bit = 0;
      int s8_negated_8bit_vector = 0;

      int ABS_temp  = 0;
      int ABS_final = 0;

      int C_Not = 0;

      if (clk==1){
            if      (reset==1)        *C_Register = 0;
            else if (update_Cur_MB==1) *C_Register = C;
      }// ends clocking C_Register

      //fprintf(fp, "\n\nupdate_Cur_MB with in PE_AD %d", update_Cur_MB);

      // This line only negates the last 8 bits of C_Register
      // and sets all of the preceding 24 bits to 0
      C_Not = ~(*C_Register | 0xFFFFFF00);

      Nine_bit_sum = R + C_Not;

      s8_bit = (Nine_bit_sum & 0x00000100)>>8;
      s8_negated_8bit_vector = (s8_bit==1) ? 0 : 0x000000FF;

      ABS_temp = (s8_negated_8bit_vector ^ Nine_bit_sum) & 0x000000FF;
      // the extra AND operation at the end is used to clear
      // the un-used 9th MSB

      return ABS_final = ABS_temp + s8_bit;

}// ends PE_AD function

//-------------------------------------------------------------------------------------------//

void test_PE_AD(){

      int R = 0;
      int C = 0;
```

```
    int C_Register = 0;

    for (R=0; R<256; R++){
        for (C=0; C<256; C++){
            if (PE_AD(1,0,1,R,C,&C_Register) != abs(R-C))
                printf("\nError, Expecting: %d Got: %d\n\n", abs(R-C), PE_AD(1,0,1,R,C,&C_Register));
        }// ends C loop
    }// ends R loop
}// ends test_PE_AD
```

```
#include <stdlib.h>
#include <stdio.h>
//-------------------------------------------------------------------------------------------------------------------------------------------
----------//

// This function produces a 9-bit Absolute Difference Result
// for R and C using an internal-representation represented
// by a 1-bit MSB carry bit and remaining 8-bit ABS bits.
// This module is used to feed the CSA blocks used in the Row Adder Trees
// for Rows 1, 2, & 3

void PE_AD_CSA (int clk, int reset, int update_Cur_MB, int C, int *C_Register,
                        int R, int *Carry, int *ABS){

      int Nine_bit_sum = 0;

      int s8_negated_8bit_vector = 0;

      int C_Not = 0;

      if (clk==1){
            if      (reset==1)        *C_Register = 0;
            else if (update_Cur_MB==1) *C_Register = C;
      }// ends clock C_Registered

      // This line only negates the last 8 bits of C_Register
      // and sets all of the preceding 24 bits to 0
      C_Not = ~(*C_Register | 0xFFFFFF00);

      Nine_bit_sum = R + C_Not;

      *Carry = (Nine_bit_sum & 0x00000100)>>8;
      s8_negated_8bit_vector = (*Carry==1) ? 0 : 0x000000FF;

      *ABS = (s8_negated_8bit_vector ^ Nine_bit_sum) & 0x000000FF;
      // the extra AND operation at the end is used to clear
      // the un-used 9th MSB

}// ends PE_AD_CSA function
```

```
#include <stdlib.h>
#include <stdio.h>
//-------------------------------------------------------------------------------------------------------------------------------------------
----------//
FILE *fp;

// This is the Row_Adder file.
// External Sub-Modules, located in other Project Files

int PE_AD (int clk, int reset, int update_Cur_MB, int R, int C, int *C_Register);



// This functions combines the sum of each of the
// 4 PE Absolute Differece values in Row 0 of each
// 4x4 Block

void Row_Adder_0 (int clk, int reset, int enable, int update_Cur_MB,
                  int R3_a,int C3_a,int R2_a,int C2_a,int R1_a,int C1_a,int R0_a,int C0_a,
                  int R3_b,int C3_b,int R2_b,int C2_b,int R1_b,int C1_b,int R0_b,int C0_b,
                  int broadcast_bus_select,
                  int *C3_Reg,
                  int *C2_Reg,
                  int *C1_Reg,
                  int *C0_Reg,
                  int *PSAD_0_Register){

    int PSAD_0 = 0;
    int row_sum_only = 0;
    int R3,C3,R2,C2,R1,C1,R0,C0;

    if (broadcast_bus_select==0){
        R3 = R3_a;
        C3 = C3_a;
        R2 = R2_a;
        C2 = C2_a;
        R1 = R1_a;
        C1 = C1_a;
        R0 = R0_a;
        C0 = C0_a;
    }//ends if
    else{
        R3 = R3_b;
        C3 = C3_b;
        R2 = R2_b;
        C2 = C2_b;
        R1 = R1_b;
        C1 = C1_b;
        R0 = R0_b;
        C0 = C0_b;
    }//ends else

    PSAD_0 = PE_AD(clk,reset,update_Cur_MB,R3,C3,C3_Reg) +
```

```
                PE_AD(clk,reset,update_Cur_MB,R2,C2,C2_Reg) +
                PE_AD(clk,reset,update_Cur_MB,R1,C1,C1_Reg) +
                PE_AD(clk,reset,update_Cur_MB,R0,C0,C0_Reg);

    if (clk==1){
        if    (reset==1)  *PSAD_0_Register = 0;
        else if (enable==1) *PSAD_0_Register = PSAD_0;
    }// ends clocking PSAD_0_Register

    //for testing purposes only

    //printf("PSAD_0 in Row_Adder_0: %d\n\n", PSAD_0);
    row_sum_only = abs(R3-*C3_Reg) + abs(R2-*C2_Reg) + abs(R1-*C1_Reg) + abs(R0-*C0_Reg);
    //fprintf(fp, "\nThe row sum only is: %d", row_sum_only);

} //ends Row_0
```

```
#include <stdlib.h>
#include <stdio.h>
//-------------------------------------------------------------------------------------------------------------------------------------
----------//

FILE *fp;

// This is the Row_Adder_CSA_Tree file
// External Sub-Modules, located in other Project Files

void PE_AD_CSA (int clk, int reset, int update_Cur_MB, int C, int *C_Register,
                int R, int *Carry, int *ABS);

void CSA_8bit_block (int x, int y, int z, int *Carry, int *Sum);

void CSA_9bit_block (int x, int y, int z, int *Carry, int *Sum);

//-----------------------------------------------------------------------------------------//

// This function acts as the Row Adder Module
// for the last three rows, namely Row 1, 2, and 3
// in the 4x4 Block

void Row_Adder_CSA_tree (int clk, int reset, int enable, int update_Cur_MB,
                         int R3_a,int C3_a,int R2_a,int C2_a,int R1_a,int C1_a,int R0_a,int C0_a,
                         int R3_b,int C3_b,int R2_b,int C2_b,int R1_b,int C1_b,int R0_b,int C0_b,
                         int broadcast_bus_select,
                                 int PSAD_In,
                                 int *C3_Reg,
                                 int *C2_Reg,
                                 int *C1_Reg,
                                 int *C0_Reg,
                                 int *PSAD_Out_Register){

    int PSAD_Out = 0;

    int ABS3, ABS2, ABS1, ABS0;
    int Carry_3, Carry_2, Carry_1, Carry_0;

    //outputs of the 3 CSA blocks
    int CSA_1_Carry, CSA_1_Sum;
    int CSA_2_Carry, CSA_2_Sum;
    int CSA_3_Carry, CSA_3_Sum;

    int CSA_3_x, CSA_3_y, CSA_3_z;//inputs to the 3rd CSA Block

    int PSAD_In_8th_bit = PSAD_In & 0x00000100;
    int PSAD_In_Least_9bits_truncated = PSAD_In & 0xFFFFFE00;

    int row_sum_only;// used only for testing

    int R3,C3,R2,C2,R1,C1,R0,C0;
```

```
if (broadcast_bus_select==0){
      R3 = R3_a;
      C3 = C3_a;
      R2 = R2_a;
      C2 = C2_a;
      R1 = R1_a;
      C1 = C1_a;
      R0 = R0_a;
      C0 = C0_a;
}//ends if
else{
      R3 = R3_b;
      C3 = C3_b;
      R2 = R2_b;
      C2 = C2_b;
      R1 = R1_b;
      C1 = C1_b;
      R0 = R0_b;
      C0 = C0_b;
}//ends else

PE_AD_CSA (clk,reset,update_Cur_MB, C3,C3_Reg, R3, &Carry_3, &ABS0);
PE_AD_CSA (clk,reset,update_Cur_MB, C2,C2_Reg, R2, &Carry_2, &ABS1);
PE_AD_CSA (clk,reset,update_Cur_MB, C1,C1_Reg, R1, &Carry_1, &ABS2);
PE_AD_CSA (clk,reset,update_Cur_MB, C0,C0_Reg, R0, &Carry_0, &ABS3);

/*
printf("%d  %d ABS: %d Carry: %d\n\n", *C3_Reg, R3, ABS3, Carry_3);
printf("%d  %d ABS: %d Carry: %d\n\n", *C2_Reg, R2, ABS2, Carry_2);
printf("%d  %d ABS: %d Carry: %d\n\n", *C1_Reg, R1, ABS1, Carry_1);
printf("%d  %d ABS: %d Carry: %d\n\n", *C0_Reg, R0, ABS0, Carry_0);
*/

//CSA Block 1
CSA_8bit_block(ABS3, ABS2, ABS1, &CSA_1_Carry, &CSA_1_Sum);
//printf("\n\nCSA 1 Carry: %d  CSA 1 Sum: %d\n\n", CSA_1_Carry, CSA_1_Sum);

//CSA Block 2
CSA_8bit_block(CSA_1_Sum, ABS0, PSAD_In, &CSA_2_Carry, &CSA_2_Sum);
//printf("\n\nCSA 2 Carry: %d  CSA 2 Sum: %d\n\n", CSA_2_Carry, CSA_2_Sum);

CSA_3_x = (CSA_1_Carry<<1) | Carry_3;
CSA_3_y = (CSA_2_Carry<<1) | Carry_2;
CSA_3_z = PSAD_In_8th_bit | CSA_2_Sum;

//CSA Block 3
CSA_9bit_block(CSA_3_x,CSA_3_y,CSA_3_z, &CSA_3_Carry, &CSA_3_Sum);
//printf("\n\nCSA 3 Carry: %d  CSA 3 Sum: %d\n\n", CSA_3_Carry, CSA_3_Sum);

PSAD_Out =   ((CSA_3_Carry<<1) | Carry_1)
           + (PSAD_In_Least_9bits_truncated | CSA_3_Sum)
           + Carry_0;
//printf("\n\nCarry_1: %d  Carry_0: %d\n\n", PSAD_In_Least_9bits_truncated, Carry_0);
```

```
    if (clk==1){
        if     (reset==1)  *PSAD_Out_Register = 0;
        else if (enable==1) *PSAD_Out_Register = PSAD_Out;
    }// ends clocking PSAD_0_Register

    //for testing purposes only

    row_sum_only = abs(R3-*C3_Reg) + abs(R2-*C2_Reg) + abs(R1-*C1_Reg) + abs(R0-*C0_Reg);
    //fprintf(fp, "\nThe row sum only is: %d", row_sum_only);

    /*
    if (PSAD_In==425){
        printf("\n\nWhen PSAD_In is 425: %d + %d = %d but PSAD_Out is: %d\n\n", row_sum_only,
                                                                    PSAD_In,
                                                                    (row_sum_only + PSAD_In),
                                                                    PSAD_Out);
        printf("\n\n%d  %d %d\n\n", *C3_Reg, R3, ABS3);
        printf("%d  %d %d\n\n", *C2_Reg, R2, ABS2);
        printf("%d  %d %d\n\n", *C1_Reg, R1, ABS1);
        printf("%d  %d %d\n\n", *C0_Reg, R0, ABS0);
    }//ends if
    */

} //ends Row_Adder_CSA_tree

//-------------------------------------------------------------------------------------------//

void test_Row_Adder_CSA_tree(){

    int R3_a,C3_a,R2_a,C2_a,R1_a,C1_a,R0_a,C0_a;
    int R3_b,C3_b,R2_b,C2_b,R1_b,C1_b,R0_b,C0_b;
    int broadcast_bus_select = 0;
    int PSAD_In, PSAD_Out;

    int Expected = 0;

    int C3_Reg;
    int C2_Reg;
    int C1_Reg;
    int C0_Reg;

    const int PSAD_IN_MAX = 4095;
/*
    for (PSAD_In=0; PSAD_In<=PSAD_IN_MAX; PSAD_In++){
        printf("\nPSAD_In: %d\n\n", PSAD_In);
    for (R3=0; R3<256; R3++){
        printf("\nR3: %d\n\n", R3);
    for (C3=0; C3<256; C3++){
        printf("\nC3: %d\n\n", C3);
    for (R2=0; R2<256; R2++){
    for (C2=0; C2<256; C2++){
    for (R1=0; R1<256; R1++){
    for (C1=0; C1<256; C1++){
    for (R0=0; R0<256; R0++){
```

```
    for (C0=0; C0<256; C0++){
*/

    PSAD_In=425;

    C3_a = 149;
    R3_a = 23;
    C2_a = 122;
    R2_a = 92;
    C1_a = 89;
    R1_a = 37;
    C0_a = 228;
    R0_a = 42;

    C3_b = 149;
    R3_b = 23;
    C2_b = 122;
    R2_b = 92;
    C1_b = 89;
    R1_b = 37;
    C0_b = 228;
    R0_b = 42;

/*
    R3=149;
    C3=23;
    R2=122;
    C2=92;
    R1=89;
    C1=37;
    R0=228;
    C0=42;
*/
    // 1,0,1,1 corresponds to clk=1, reset=0, enable = 1, and update_Cur_MB = 1
    Row_Adder_CSA_tree(1,0,1,1,
                       C3_a,R3_a,C2_a,R2_a,C1_a,R1_a,C0_a,R0_a,
                       C3_b,R3_b,C2_b,R2_b,C1_b,R1_b,C0_b,R0_b,
                       broadcast_bus_select,
                       PSAD_In,
                       &C3_Reg,
                       &C2_Reg,
                       &C1_Reg,
                       &C0_Reg,
                       &PSAD_Out);

    Expected = abs(R3_a-C3_a) + abs(R2_a-C2_a) + abs(R1_a-C1_a) + abs(R0_a-C0_a) + PSAD_In;

    if (PSAD_Out != Expected){
        printf("\nError, Expecting: %d Got: %d\n\n", Expected, PSAD_Out);
        //exit(1);
    }
/*
    }// ends C0 loop
    }// ends R0 loop
```

```
	}// ends C1 loop
	}// ends R1 loop
	}// ends C2 loop
	}// ends R2 loop
	}// ends C3 loop
	}// ends R3 loop
	}// ends PSAD_In loop
*/
}// ends test_Row_Adder_CSA_tree
```

```
#include <stdlib.h>
#include <stdio.h>
//-------------------------------------------------------------------------------------------------------------------------------
----------//
FILE *mem_A_file;
FILE *mem_B_file;
FILE *ref_pixels;
FILE *ref_pixels_2;

// This is the dual buffers' test file.
// External Functions located in other Project Files

//-------------------------------------------------------------------------------------------//
void dual_buffer (// inputs
                        int initialize,
                        int address, // 6-bits (i.e. 0 to 47 search pixel rows)
                        int read_Buffer_1,
                        int read_Buffer_2,

                        // Input Bus
                        int dataIn_32_Bytes[32],
                        // Output Buses
                        int dataOut_31_Bytes_A[32],// an extra byte in the array is added
                        int dataOut_31_Bytes_B[32] // for programming convenience
                        );

int random_int_between_0_and_255();

//-------------------------------------------------------------------------------------------//


void test_dual_buff(){

// Input Wires & Buses
int initialize,
      address, // 6-bits (i.e. 0 to 47 search pixel rows)
      read_Buffer_1,
      read_Buffer_2;

// Input Bus
int input_BUS_32bytes[32];

// Ouput Buses
int output_BUS_31bytes_A[32]; // an extra byte in the array is added
int output_BUS_31bytes_B[32]; // for programming convenience

int Test_Case = 0;

if ((mem_A_file = fopen("Bus_A.txt", "w"))==NULL){
                fprintf(mem_A_file, "\n\n Can not open the output file. \n\n");
                exit(1);
}
```

```
if ((mem_B_file = fopen("Bus_B.txt", "w"))==NULL){
                fprintf(mem_B_file, "\n\n Can not open the output file. \n\n");
                exit(1);
}
if ((ref_pixels = fopen("ref_pixels.txt", "w"))==NULL){
                printf("\n\n Can not open the ref_pixels output file. \n\n");
                exit(1);
}
if ((ref_pixels_2 = fopen("ref_pixels_buf2.txt", "w"))==NULL){
                printf("\n\n Can not open the ref_pixels_buf2 output file. \n\n");
                exit(1);
}

//-------------------------------------------------------------------------------------------------------//

for (Test_Case = 0; Test_Case<1; Test_Case++){

      int clock_Cycle = 0;
      int reference_Data[48][64];
      int reference_Data_2[48][64];
      int i, j, k = 0;
      int value, value_A, value_B;

      address = 0;

      fprintf(ref_pixels, "\n");
      for (i=0; i<48; i++){
            for (j=0; j<64; j++){
                  reference_Data[i][j] = (i*100)+j;
                  value = reference_Data[i][j];
                  if      (value<10)   fprintf(ref_pixels, "   %d  ", value);
                  else if (value<100)  fprintf(ref_pixels, "  %d  ", value);
                  else if (value<1000) fprintf(ref_pixels, " %d  ", value);
                  else                 fprintf(ref_pixels, "%d  ", value);
            }
            fprintf(ref_pixels, "\n");
      }

      fprintf(ref_pixels_2, "\n");
      for (i=0; i<48; i++){
            for (j=0; j<64; j++){
                  reference_Data_2[i][j] = k;
                  k++;
                  value = reference_Data_2[i][j];
                  if      (value<10)   fprintf(ref_pixels_2, "   %d  ", value);
                  else if (value<100)  fprintf(ref_pixels_2, "  %d  ", value);
                  else if (value<1000) fprintf(ref_pixels_2, " %d  ", value);
                  else                 fprintf(ref_pixels_2, "%d  ", value);
            }
            fprintf(ref_pixels_2, "\n");
      }

//-------------------------------------------------------------------------------//
```

```
    for (clock_Cycle = 0; clock_Cycle<324; clock_Cycle++){// 210 = 48 + 48 +  (3*33) + 15
                                                          // 324 = 48 + 48 + 2(3*33 + 15)

        if (clock_Cycle<96) initialize = 1;
        else                initialize = 0;

        if (clock_Cycle<48){
            for (i=0; i<32; i++){
                input_BUS_32bytes[i] = reference_Data[address][i];
            }
        }
        if ((clock_Cycle>47) && (clock_Cycle<96)){
             for (i=0; i<32; i++){
                input_BUS_32bytes[i] = reference_Data[address][i+32];
             }
        }
        if ((clock_Cycle>95) && (clock_Cycle<144)){
            //printf("\n %d \n", address);
            for (i=0; i<32; i++){
                input_BUS_32bytes[i] = reference_Data_2[address][i];
                //printf(" %d ", input_BUS_32bytes[i]);
            }
        }
        if ((clock_Cycle>143) && (clock_Cycle<192)){
             for (i=0; i<32; i++){
                input_BUS_32bytes[i] = reference_Data_2[address][i+32];
             }
        }

        if (clock_Cycle>95){
            read_Buffer_1 = 1;
            read_Buffer_2 = 0;
        }

        if (clock_Cycle==210) address = 0;
        if (clock_Cycle>209){
            read_Buffer_1 = 0;
            read_Buffer_2 = 1;
        }

//-------------------------------------------------------------------------------------------------------//
// Instantiation of the Device Unter Test (DUT) i.e. the dual buffer

    dual_buffer (// inputs
                 initialize,
                 address, // 6-bits (i.e. 0 to 47 search pixel rows)
                 read_Buffer_1,
                 read_Buffer_2,

                 // Input Bus
                 input_BUS_32bytes,
                 // Output Buses
                 output_BUS_31bytes_A,// an extra byte in the array is added
                 output_BUS_31bytes_B // for programming convenience
```

```
                    );

//-------------------------------------------------------------------------------------------------//

    if (clock_Cycle>95){

    fprintf(mem_A_file, "\n");
    fprintf(mem_B_file, "\n");
    if (clock_Cycle<100){
        fprintf(mem_A_file, " %d  ", clock_Cycle);
        fprintf(mem_B_file, " %d  ", clock_Cycle);
    }
    else{
        fprintf(mem_A_file, "%d  ", clock_Cycle);
        fprintf(mem_B_file, "%d  ", clock_Cycle);
    }

    for (i=0; i<31; i++){
        value_A = output_BUS_31bytes_A[i];
        value_B = output_BUS_31bytes_B[i];
            if      (value_A<10)   fprintf(mem_A_file, "   %d  ", value_A);
            else if (value_A<100)  fprintf(mem_A_file, "  %d  ", value_A);
            else if (value_A<1000) fprintf(mem_A_file, " %d  ", value_A);
            else                   fprintf(mem_A_file, "%d  ", value_A);
            if      (value_B<10)   fprintf(mem_B_file, "   %d  ", value_B);
            else if (value_B<100)  fprintf(mem_B_file, "  %d  ", value_B);
            else if (value_B<1000) fprintf(mem_B_file, " %d  ", value_B);
            else                   fprintf(mem_B_file, "%d  ", value_B);
    }

    if (clock_Cycle<209){
    if ((clock_Cycle%33)==29){
        fprintf(mem_A_file, "\n\n");
        fprintf(mem_B_file, "\n\n");
    }
    }// ends if clock <209
    if (clock_Cycle==209){
        fprintf(mem_A_file, "\n\n");
        fprintf(mem_B_file, "\n\n");
    }
    if (clock_Cycle>209){
    if ((clock_Cycle%33)==11){
        fprintf(mem_A_file, "\n\n");
        fprintf(mem_B_file, "\n\n");
    }
    }// ends if clock >209

    }// ends if clock_Cycle>95


        address++;
        if (address==48) address = 0;
    }// ends the clock_Cyle for loop
```

```
}// ends the Test Case for loop
}// ends the main void test_mem() function
```

```
#include <stdlib.h>
#include <stdio.h>
//-------------------------------------------------------------------------------------------------------------------------------------------
----------//
FILE *fp;

FILE *B4x4_top;
FILE *B4x8_top;
FILE *B8x4_top;
FILE *B8x8_top;
FILE *B16x8_top;

// This is the main data path test file
// External Functions located in other Project Files

char* int2Bin_13bits(int a); //used to convert the 16 13-bit B4x4 SAD outputs

char* int2Bin_14bits(int a); //used to convert the 16 14-bit 4x8 & 8x4 SAD outputs

char* int2Bin_15bits(int a); //used to convert the  4 15-bit 8x8 SAD outputs

char* int2Bin_16bits(int a); //used to convert the  4 16-bit 8x16 & 16x8 SAD outputs

void main_data_path (
                        // Input Wires & Buses
                        int clk, int reset, int enable, int update_Cur_MB,

                        int broadcast_column_1_a,
                        int broadcast_column_2_a,
                        int broadcast_column_3_a,
                        int broadcast_column_4_a,
                        int broadcast_column_5_a,
                        int broadcast_column_6_a,
                        int broadcast_column_7_a,
                        int broadcast_column_8_a,
                        int broadcast_column_9_a,
                        int broadcast_column_10_a,
                        int broadcast_column_11_a,
                        int broadcast_column_12_a,
                        int broadcast_column_13_a,
                        int broadcast_column_14_a,
                        int broadcast_column_15_a,
                        int broadcast_column_16_a,

                        int broadcast_column_1_b,
                        int broadcast_column_2_b,
                        int broadcast_column_3_b,
                        int broadcast_column_4_b,
                        int broadcast_column_5_b,
                        int broadcast_column_6_b,
                        int broadcast_column_7_b,
                        int broadcast_column_8_b,
                        int broadcast_column_9_b,
```

```
                int broadcast_column_10_b,
                int broadcast_column_11_b,
                int broadcast_column_12_b,
                int broadcast_column_13_b,
                int broadcast_column_14_b,
                int broadcast_column_15_b,
                int broadcast_column_16_b,

                int broadcast_bus_select,

                // Ouput Registers

                // The 16 4x4 SAD output registers, labeled by row/column
                int *B4x4_00_out, int *B4x4_01_out, int *B4x4_02_out, int *B4x4_03_out,
                int *B4x4_10_out, int *B4x4_11_out, int *B4x4_12_out, int *B4x4_13_out,
                int *B4x4_20_out, int *B4x4_21_out, int *B4x4_22_out, int *B4x4_23_out,
                int *B4x4_30_out, int *B4x4_31_out, int *B4x4_32_out, int *B4x4_33_out,

                // The 8 4x8 SAD output registers, labeled by row/column
                int *B4x8_00_out, int *B4x8_01_out, int *B4x8_02_out, int *B4x8_03_out,
                int *B4x8_10_out, int *B4x8_11_out, int *B4x8_12_out, int *B4x8_13_out,

                // The 8 8x4 SAD output registers, labeled by row/column
                int *B8x4_00_out, int *B8x4_01_out,
                int *B8x4_10_out, int *B8x4_11_out,
                int *B8x4_20_out, int *B8x4_21_out,
                int *B8x4_30_out, int *B8x4_31_out,

                // The 4 8x8 SAD output registers, labeled by row/column
                int *B8x8_00_out, int *B8x8_01_out,
                int *B8x8_10_out, int *B8x8_11_out,

                // The 2 8x16 SAD output registers, labeled by column
                int *B8x16_0_out, int *B8x16_1_out,

                // The 2 16x8 SAD output registers, labeled by row
                int *B16x8_0_out, int *B16x8_1_out,

                // The 1 16x16 SAD output register
                int *B16x16_out

                );


//-------------------------------------------------------------------------------------------//

int random_int_between_0_and_255();

//-------------------------------------------------------------------------------------------//


void test_main_data_path_top(){

// Input Wires & Buses
```

```
int clk, reset, enable, update_Cur_MB;

int broadcast_column_1_a;
int broadcast_column_2_a;
int broadcast_column_3_a;
int broadcast_column_4_a;
int broadcast_column_5_a;
int broadcast_column_6_a;
int broadcast_column_7_a;
int broadcast_column_8_a;
int broadcast_column_9_a;
int broadcast_column_10_a;
int broadcast_column_11_a;
int broadcast_column_12_a;
int broadcast_column_13_a;
int broadcast_column_14_a;
int broadcast_column_15_a;
int broadcast_column_16_a;

int broadcast_column_1_b = 0;
int broadcast_column_2_b = 0;
int broadcast_column_3_b = 0;
int broadcast_column_4_b = 0;
int broadcast_column_5_b = 0;
int broadcast_column_6_b = 0;
int broadcast_column_7_b = 0;
int broadcast_column_8_b = 0;
int broadcast_column_9_b = 0;
int broadcast_column_10_b = 0;
int broadcast_column_11_b = 0;
int broadcast_column_12_b = 0;
int broadcast_column_13_b = 0;
int broadcast_column_14_b = 0;
int broadcast_column_15_b = 0;
int broadcast_column_16_b = 0;

int broadcast_bus_select;

// Ouput Registers

// The 16 4x4 SAD output registers; labeled by row/column
int B4x4_00_out; int B4x4_01_out; int B4x4_02_out; int B4x4_03_out;
int B4x4_10_out; int B4x4_11_out; int B4x4_12_out; int B4x4_13_out;
int B4x4_20_out; int B4x4_21_out; int B4x4_22_out; int B4x4_23_out;
int B4x4_30_out; int B4x4_31_out; int B4x4_32_out; int B4x4_33_out;

// The 8 4x8 SAD output registers; labeled by row/column
int B4x8_00_out; int B4x8_01_out; int B4x8_02_out; int B4x8_03_out;
int B4x8_10_out; int B4x8_11_out; int B4x8_12_out; int B4x8_13_out;

// The 8 8x4 SAD output registers; labeled by row/column
int B8x4_00_out; int B8x4_01_out;
int B8x4_10_out; int B8x4_11_out;
int B8x4_20_out; int B8x4_21_out;
```

```
int B8x4_30_out; int B8x4_31_out;

// The 4 8x8 SAD output registers; labeled by row/column
int B8x8_00_out; int B8x8_01_out;
int B8x8_10_out; int B8x8_11_out;

// The 2 8x16 SAD output registers; labeled by column
int B8x16_0_out; int B8x16_1_out;

// The 2 16x8 SAD output registers; labeled by row
int B16x8_0_out; int B16x8_1_out;

// The 1 16x16 SAD output register
int B16x16_out;

// External Registers used for Equivalency Checking....
// Only Registers contained within the upper 8x8 area
// of the whole 16x16 datapath are included here
// since this test file (i.e. test_main_data_path_TOP)
// is exclusively only for the upper 19 MVs out of the
// total available 41 MVs.

// The Upper 8 of the 16 4x4 SAD registers; labeled by row/column
int Ex_B4x4_00; int Ex_B4x4_01; int Ex_B4x4_02; int Ex_B4x4_03;
int Ex_B4x4_10; int Ex_B4x4_11; int Ex_B4x4_12; int Ex_B4x4_13;

// The Upper 4 of the 8 4x8 SAD registers; labeled by row/column
int Ex_B4x8_00; int Ex_B4x8_01; int Ex_B4x8_02; int Ex_B4x8_03;

// The Upper 4 of the 8 8x4 SAD registers; labeled by row/column
int Ex_B8x4_00; int Ex_B8x4_01;
int Ex_B8x4_10; int Ex_B8x4_11;

// The Upper 2 of the 4 8x8 SAD registers; labeled by row/column
int Ex_B8x8_00; int Ex_B8x8_01;

// The Upper 1 of the 2 16x8 SAD registers; labeled by row
int Ex_B16x8_0;

int Test_Case = 0;

if ((fp = fopen("output.txt", "w"))==NULL){
                fprintf(fp, "\n\n Can not open the output file. \n\n");
                exit(1);
}

// files created to record the ouput SAD values for each varying Block
// inorder to do formal equivalency cheching against the Verilog RTL model
if ((B4x4_top = fopen("B4x4_top.txt", "w"))==NULL){
                printf("\n\n Can not open the B4x4_top output file. \n\n");
                exit(1);
}
if ((B4x8_top = fopen("B4x8_top.txt", "w"))==NULL){
                printf("\n\n Can not open the B4x8_top output file. \n\n");
```

```
                exit(1);
}
if ((B8x4_top = fopen("B8x4_top.txt", "w"))==NULL){
                printf("\n\n Can not open the B8x4_top output file. \n\n");
                exit(1);
}
if ((B8x8_top = fopen("B8x8_top.txt", "w"))==NULL){
                printf("\n\n Can not open the B8x8_top output file. \n\n");
                exit(1);
}
if ((B16x8_top = fopen("B16x8_top.txt", "w"))==NULL){
                printf("\n\n Can not open the B16x8_top output file. \n\n");
                exit(1);
}

//-------------------------------------------------------------------------------------------------------//

for (Test_Case = 0; Test_Case<1000; Test_Case++){

      int clock_Cycle = 0;
      int reference_Cycle = clock_Cycle - 16;
      int broadcast_index = 32;

      int current_MB[16][16];
      int reference_Data[48][48];

      int cur_x, cur_y = 0;
      int ref_x, ref_y = 0;
      int ref_Block_x = 0;
      int ref_Block_y = 0;

      int i, j = 0;
      int broadcast_a_row    =  0;
      int broadcast_a_column =  0;
      int broadcast_b_row    = 33;
      int broadcast_b_column =  0;

      clk    = 1;
      reset  = 0;
      enable = 1;
      update_Cur_MB = 16;

      broadcast_bus_select = 0;


      for (i=0; i<16; i++){
            //fprintf(fp, "\n");
            for (j=0; j<16; j++){
                  current_MB[i][j] = random_int_between_0_and_255();
                  //if (current_MB[i][j]<10) fprintf(fp, "  %d   ", current_MB[i][j]);
                  //else if (current_MB[i][j]<100) fprintf(fp, " %d   ", current_MB[i][j]);
                  //else fprintf(fp, "%d   ", current_MB[i][j]);
            }
      }
```

```
	for (i=0; i<48; i++){
		//fprintf(fp, "\n");
		for (j=0; j<48; j++){
			reference_Data[i][j] = random_int_between_0_and_255();
			//if (current_MB[i][j]<10) fprintf(fp, "  %d   ", current_MB[i][j]);
			//else if (current_MB[i][j]<100) fprintf(fp, " %d   ", current_MB[i][j]);
			//else fprintf(fp, "%d   ", current_MB[i][j]);
		}
	}

//-------------------------------------------------------------------------------//

	for (clock_Cycle = 0; clock_Cycle<(16+15+1089); clock_Cycle++){

	if (clock_Cycle<16) update_Cur_MB = clock_Cycle;
	else {
		update_Cur_MB = 16;
		reference_Cycle = clock_Cycle - 16;
	}

	if (reference_Cycle==broadcast_index+16){
		broadcast_bus_select = 0;
		broadcast_index += 33;
	}

	if (clock_Cycle<16) {
		broadcast_column_1_a  = current_MB[clock_Cycle][0];
		broadcast_column_2_a  = current_MB[clock_Cycle][1];
		broadcast_column_3_a  = current_MB[clock_Cycle][2];
		broadcast_column_4_a  = current_MB[clock_Cycle][3];
		broadcast_column_5_a  = current_MB[clock_Cycle][4];
		broadcast_column_6_a  = current_MB[clock_Cycle][5];
		broadcast_column_7_a  = current_MB[clock_Cycle][6];
		broadcast_column_8_a  = current_MB[clock_Cycle][7];
		broadcast_column_9_a  = current_MB[clock_Cycle][8];
		broadcast_column_10_a = current_MB[clock_Cycle][9];
		broadcast_column_11_a = current_MB[clock_Cycle][10];
		broadcast_column_12_a = current_MB[clock_Cycle][11];
		broadcast_column_13_a = current_MB[clock_Cycle][12];
		broadcast_column_14_a = current_MB[clock_Cycle][13];
		broadcast_column_15_a = current_MB[clock_Cycle][14];
		broadcast_column_16_a = current_MB[clock_Cycle][15];
	}//ends if clock_Cycle<16
	else {
		broadcast_column_1_a  = reference_Data[broadcast_a_row][broadcast_a_column+0];
		broadcast_column_2_a  = reference_Data[broadcast_a_row][broadcast_a_column+1];
		broadcast_column_3_a  = reference_Data[broadcast_a_row][broadcast_a_column+2];
		broadcast_column_4_a  = reference_Data[broadcast_a_row][broadcast_a_column+3];
		broadcast_column_5_a  = reference_Data[broadcast_a_row][broadcast_a_column+4];
		broadcast_column_6_a  = reference_Data[broadcast_a_row][broadcast_a_column+5];
		broadcast_column_7_a  = reference_Data[broadcast_a_row][broadcast_a_column+6];
		broadcast_column_8_a  = reference_Data[broadcast_a_row][broadcast_a_column+7];
		broadcast_column_9_a  = reference_Data[broadcast_a_row][broadcast_a_column+8];
```

```
        broadcast_column_10_a = reference_Data[broadcast_a_row][broadcast_a_column+9];
        broadcast_column_11_a = reference_Data[broadcast_a_row][broadcast_a_column+10];
        broadcast_column_12_a = reference_Data[broadcast_a_row][broadcast_a_column+11];
        broadcast_column_13_a = reference_Data[broadcast_a_row][broadcast_a_column+12];
        broadcast_column_14_a = reference_Data[broadcast_a_row][broadcast_a_column+13];
        broadcast_column_15_a = reference_Data[broadcast_a_row][broadcast_a_column+14];
        broadcast_column_16_a = reference_Data[broadcast_a_row][broadcast_a_column+15];

        broadcast_a_row++;
        if (broadcast_a_row==33){
            broadcast_a_row = 0;
            broadcast_a_column++;
        }
    }


    if (reference_Cycle>broadcast_index) {

        broadcast_bus_select = reference_Cycle-broadcast_index;

        broadcast_column_1_b  = reference_Data[broadcast_b_row][broadcast_b_column+0];
        broadcast_column_2_b  = reference_Data[broadcast_b_row][broadcast_b_column+1];
        broadcast_column_3_b  = reference_Data[broadcast_b_row][broadcast_b_column+2];
        broadcast_column_4_b  = reference_Data[broadcast_b_row][broadcast_b_column+3];
        broadcast_column_5_b  = reference_Data[broadcast_b_row][broadcast_b_column+4];
        broadcast_column_6_b  = reference_Data[broadcast_b_row][broadcast_b_column+5];
        broadcast_column_7_b  = reference_Data[broadcast_b_row][broadcast_b_column+6];
        broadcast_column_8_b  = reference_Data[broadcast_b_row][broadcast_b_column+7];
        broadcast_column_9_b  = reference_Data[broadcast_b_row][broadcast_b_column+8];
        broadcast_column_10_b = reference_Data[broadcast_b_row][broadcast_b_column+9];
        broadcast_column_11_b = reference_Data[broadcast_b_row][broadcast_b_column+10];
        broadcast_column_12_b = reference_Data[broadcast_b_row][broadcast_b_column+11];
        broadcast_column_13_b = reference_Data[broadcast_b_row][broadcast_b_column+12];
        broadcast_column_14_b = reference_Data[broadcast_b_row][broadcast_b_column+13];
        broadcast_column_15_b = reference_Data[broadcast_b_row][broadcast_b_column+14];
        broadcast_column_16_b = reference_Data[broadcast_b_row][broadcast_b_column+15];

        broadcast_b_row++;
        if (broadcast_b_row==48){
            broadcast_b_row = 33;
            broadcast_b_column++;
        }
    }//ends if reference_Cycle>broadcast_index
    else {
        broadcast_column_1_b  = 0;
        broadcast_column_2_b  = 0;
        broadcast_column_3_b  = 0;
        broadcast_column_4_b  = 0;
        broadcast_column_5_b  = 0;
        broadcast_column_6_b  = 0;
        broadcast_column_7_b  = 0;
        broadcast_column_8_b  = 0;
        broadcast_column_9_b  = 0;
        broadcast_column_10_b = 0;
```

```
        broadcast_column_11_b = 0;
        broadcast_column_12_b = 0;
        broadcast_column_13_b = 0;
        broadcast_column_14_b = 0;
        broadcast_column_15_b = 0;
        broadcast_column_16_b = 0;
    }

//-------------------------------------------------------------------------------------------------//
// Instantiation

    main_data_path (
                    // Input Wires & Buses
                    clk, reset, enable, update_Cur_MB,

                    broadcast_column_1_a,
                    broadcast_column_2_a,
                    broadcast_column_3_a,
                    broadcast_column_4_a,
                    broadcast_column_5_a,
                    broadcast_column_6_a,
                    broadcast_column_7_a,
                    broadcast_column_8_a,
                    broadcast_column_9_a,
                    broadcast_column_10_a,
                    broadcast_column_11_a,
                    broadcast_column_12_a,
                    broadcast_column_13_a,
                    broadcast_column_14_a,
                    broadcast_column_15_a,
                    broadcast_column_16_a,

                    broadcast_column_1_b,
                    broadcast_column_2_b,
                    broadcast_column_3_b,
                    broadcast_column_4_b,
                    broadcast_column_5_b,
                    broadcast_column_6_b,
                    broadcast_column_7_b,
                    broadcast_column_8_b,
                    broadcast_column_9_b,
                    broadcast_column_10_b,
                    broadcast_column_11_b,
                    broadcast_column_12_b,
                    broadcast_column_13_b,
                    broadcast_column_14_b,
                    broadcast_column_15_b,
                    broadcast_column_16_b,

                    broadcast_bus_select,

                    // Ouput Registers

                    // The 16 4x4 SAD output registers, labeled by row/column
```

```
                        &B4x4_00_out, &B4x4_01_out, &B4x4_02_out, &B4x4_03_out,
                        &B4x4_10_out, &B4x4_11_out, &B4x4_12_out, &B4x4_13_out,
                        &B4x4_20_out, &B4x4_21_out, &B4x4_22_out, &B4x4_23_out,
                        &B4x4_30_out, &B4x4_31_out, &B4x4_32_out, &B4x4_33_out,

                        // The 8 4x8 SAD output registers, labeled by row/column
                        &B4x8_00_out, &B4x8_01_out, &B4x8_02_out, &B4x8_03_out,
                        &B4x8_10_out, &B4x8_11_out, &B4x8_12_out, &B4x8_13_out,

                        // The 8 8x4 SAD output registers, labeled by row/column
                        &B8x4_00_out, &B8x4_01_out,
                        &B8x4_10_out, &B8x4_11_out,
                        &B8x4_20_out, &B8x4_21_out,
                        &B8x4_30_out, &B8x4_31_out,

                        // The 4 8x8 SAD output registers, labeled by row/column
                        &B8x8_00_out, &B8x8_01_out,
                        &B8x8_10_out, &B8x8_11_out,

                        // The 2 8x16 SAD output registers, labeled by column
                        &B8x16_0_out, &B8x16_1_out,

                        // The 2 16x8 SAD output registers, labeled by row
                        &B16x8_0_out, &B16x8_1_out,

                        // The 1 16x16 SAD output register
                        &B16x16_out
                );

//-----------------------------------------------------------------------------------------------//

// Independent External Calculation of the 19 (Upper Portion Only) MVs using the simplest form
// possible, in order to cross-reference them with their equivalent internal registers
// for the purpose of error checking begins.....

if (reference_Cycle<8){
      ref_Block_x = 0;
      ref_Block_y = 0;
}//ends if
else {
      ref_Block_y++;
      if (ref_Block_y==33){
            ref_Block_y = 0;
            ref_Block_x++;
      }
}

cur_x = 0;
cur_y = 0;
ref_x = 0;
ref_y = 0;

// The Upper 8 of the 16 4x4 SAD Equivalency Checking registers:
```

```
Ex_B4x4_00 = abs(current_MB[cur_y+0][cur_x+0] - refer-
ence_Data[ref_Block_y+ref_y+0][ref_Block_x+ref_x+0]) +
                abs(current_MB[cur_y+0][cur_x+1] - reference_Data[ref_Block_y+ref_y+0][ref_Block_x+ref_x+1])
+
                abs(current_MB[cur_y+0][cur_x+2] - reference_Data[ref_Block_y+ref_y+0][ref_Block_x+ref_x+2])
+
                abs(current_MB[cur_y+0][cur_x+3] - reference_Data[ref_Block_y+ref_y+0][ref_Block_x+ref_x+3])
+
                abs(current_MB[cur_y+1][cur_x+0] - reference_Data[ref_Block_y+ref_y+1][ref_Block_x+ref_x+0])
+
                abs(current_MB[cur_y+1][cur_x+1] - reference_Data[ref_Block_y+ref_y+1][ref_Block_x+ref_x+1])
+
                abs(current_MB[cur_y+1][cur_x+2] - reference_Data[ref_Block_y+ref_y+1][ref_Block_x+ref_x+2])
+
                abs(current_MB[cur_y+1][cur_x+3] - reference_Data[ref_Block_y+ref_y+1][ref_Block_x+ref_x+3])
+
                abs(current_MB[cur_y+2][cur_x+0] - reference_Data[ref_Block_y+ref_y+2][ref_Block_x+ref_x+0])
+
                abs(current_MB[cur_y+2][cur_x+1] - reference_Data[ref_Block_y+ref_y+2][ref_Block_x+ref_x+1])
+
                abs(current_MB[cur_y+2][cur_x+2] - reference_Data[ref_Block_y+ref_y+2][ref_Block_x+ref_x+2])
+
                abs(current_MB[cur_y+2][cur_x+3] - reference_Data[ref_Block_y+ref_y+2][ref_Block_x+ref_x+3])
+
                abs(current_MB[cur_y+3][cur_x+0] - reference_Data[ref_Block_y+ref_y+3][ref_Block_x+ref_x+0])
+
                abs(current_MB[cur_y+3][cur_x+1] - reference_Data[ref_Block_y+ref_y+3][ref_Block_x+ref_x+1])
+
                abs(current_MB[cur_y+3][cur_x+2] - reference_Data[ref_Block_y+ref_y+3][ref_Block_x+ref_x+2])
+
                abs(current_MB[cur_y+3][cur_x+3] - reference_Data[ref_Block_y+ref_y+3][ref_Block_x+ref_x+3]);

cur_x += 4;
ref_x += 4;

Ex_B4x4_01 = abs(current_MB[cur_y+0][cur_x+0] - refer-
ence_Data[ref_Block_y+ref_y+0][ref_Block_x+ref_x+0]) +
                abs(current_MB[cur_y+0][cur_x+1] - reference_Data[ref_Block_y+ref_y+0][ref_Block_x+ref_x+1])
+
                abs(current_MB[cur_y+0][cur_x+2] - reference_Data[ref_Block_y+ref_y+0][ref_Block_x+ref_x+2])
+
                abs(current_MB[cur_y+0][cur_x+3] - reference_Data[ref_Block_y+ref_y+0][ref_Block_x+ref_x+3])
+
                abs(current_MB[cur_y+1][cur_x+0] - reference_Data[ref_Block_y+ref_y+1][ref_Block_x+ref_x+0])
+
                abs(current_MB[cur_y+1][cur_x+1] - reference_Data[ref_Block_y+ref_y+1][ref_Block_x+ref_x+1])
+
                abs(current_MB[cur_y+1][cur_x+2] - reference_Data[ref_Block_y+ref_y+1][ref_Block_x+ref_x+2])
+
                abs(current_MB[cur_y+1][cur_x+3] - reference_Data[ref_Block_y+ref_y+1][ref_Block_x+ref_x+3])
+
                abs(current_MB[cur_y+2][cur_x+0] - reference_Data[ref_Block_y+ref_y+2][ref_Block_x+ref_x+0])
+
```

```
                abs(current_MB[cur_y+2][cur_x+1] - reference_Data[ref_Block_y+ref_y+2][ref_Block_x+ref_x+1])
+
                abs(current_MB[cur_y+2][cur_x+2] - reference_Data[ref_Block_y+ref_y+2][ref_Block_x+ref_x+2])
+
                abs(current_MB[cur_y+2][cur_x+3] - reference_Data[ref_Block_y+ref_y+2][ref_Block_x+ref_x+3])
+
                abs(current_MB[cur_y+3][cur_x+0] - reference_Data[ref_Block_y+ref_y+3][ref_Block_x+ref_x+0])
+
                abs(current_MB[cur_y+3][cur_x+1] - reference_Data[ref_Block_y+ref_y+3][ref_Block_x+ref_x+1])
+
                abs(current_MB[cur_y+3][cur_x+2] - reference_Data[ref_Block_y+ref_y+3][ref_Block_x+ref_x+2])
+
                abs(current_MB[cur_y+3][cur_x+3] - reference_Data[ref_Block_y+ref_y+3][ref_Block_x+ref_x+3]);

cur_x += 4;
ref_x += 4;

Ex_B4x4_02 = abs(current_MB[cur_y+0][cur_x+0] - refer-
ence_Data[ref_Block_y+ref_y+0][ref_Block_x+ref_x+0]) +
                abs(current_MB[cur_y+0][cur_x+1] - reference_Data[ref_Block_y+ref_y+0][ref_Block_x+ref_x+1])
+
                abs(current_MB[cur_y+0][cur_x+2] - reference_Data[ref_Block_y+ref_y+0][ref_Block_x+ref_x+2])
+
                abs(current_MB[cur_y+0][cur_x+3] - reference_Data[ref_Block_y+ref_y+0][ref_Block_x+ref_x+3])
+
                abs(current_MB[cur_y+1][cur_x+0] - reference_Data[ref_Block_y+ref_y+1][ref_Block_x+ref_x+0])
+
                abs(current_MB[cur_y+1][cur_x+1] - reference_Data[ref_Block_y+ref_y+1][ref_Block_x+ref_x+1])
+
                abs(current_MB[cur_y+1][cur_x+2] - reference_Data[ref_Block_y+ref_y+1][ref_Block_x+ref_x+2])
+
                abs(current_MB[cur_y+1][cur_x+3] - reference_Data[ref_Block_y+ref_y+1][ref_Block_x+ref_x+3])
+
                abs(current_MB[cur_y+2][cur_x+0] - reference_Data[ref_Block_y+ref_y+2][ref_Block_x+ref_x+0])
+
                abs(current_MB[cur_y+2][cur_x+1] - reference_Data[ref_Block_y+ref_y+2][ref_Block_x+ref_x+1])
+
                abs(current_MB[cur_y+2][cur_x+2] - reference_Data[ref_Block_y+ref_y+2][ref_Block_x+ref_x+2])
+
                abs(current_MB[cur_y+2][cur_x+3] - reference_Data[ref_Block_y+ref_y+2][ref_Block_x+ref_x+3])
+
                abs(current_MB[cur_y+3][cur_x+0] - reference_Data[ref_Block_y+ref_y+3][ref_Block_x+ref_x+0])
+
                abs(current_MB[cur_y+3][cur_x+1] - reference_Data[ref_Block_y+ref_y+3][ref_Block_x+ref_x+1])
+
                abs(current_MB[cur_y+3][cur_x+2] - reference_Data[ref_Block_y+ref_y+3][ref_Block_x+ref_x+2])
+
                abs(current_MB[cur_y+3][cur_x+3] - reference_Data[ref_Block_y+ref_y+3][ref_Block_x+ref_x+3]);

cur_x += 4;
ref_x += 4;

Ex_B4x4_03 = abs(current_MB[cur_y+0][cur_x+0] - refer-
ence_Data[ref_Block_y+ref_y+0][ref_Block_x+ref_x+0]) +
```

```
                abs(current_MB[cur_y+0][cur_x+1] - reference_Data[ref_Block_y+ref_y+0][ref_Block_x+ref_x+1])
+
                abs(current_MB[cur_y+0][cur_x+2] - reference_Data[ref_Block_y+ref_y+0][ref_Block_x+ref_x+2])
+
                abs(current_MB[cur_y+0][cur_x+3] - reference_Data[ref_Block_y+ref_y+0][ref_Block_x+ref_x+3])
+
                abs(current_MB[cur_y+1][cur_x+0] - reference_Data[ref_Block_y+ref_y+1][ref_Block_x+ref_x+0])
+
                abs(current_MB[cur_y+1][cur_x+1] - reference_Data[ref_Block_y+ref_y+1][ref_Block_x+ref_x+1])
+
                abs(current_MB[cur_y+1][cur_x+2] - reference_Data[ref_Block_y+ref_y+1][ref_Block_x+ref_x+2])
+
                abs(current_MB[cur_y+1][cur_x+3] - reference_Data[ref_Block_y+ref_y+1][ref_Block_x+ref_x+3])
+
                abs(current_MB[cur_y+2][cur_x+0] - reference_Data[ref_Block_y+ref_y+2][ref_Block_x+ref_x+0])
+
                abs(current_MB[cur_y+2][cur_x+1] - reference_Data[ref_Block_y+ref_y+2][ref_Block_x+ref_x+1])
+
                abs(current_MB[cur_y+2][cur_x+2] - reference_Data[ref_Block_y+ref_y+2][ref_Block_x+ref_x+2])
+
                abs(current_MB[cur_y+2][cur_x+3] - reference_Data[ref_Block_y+ref_y+2][ref_Block_x+ref_x+3])
+
                abs(current_MB[cur_y+3][cur_x+0] - reference_Data[ref_Block_y+ref_y+3][ref_Block_x+ref_x+0])
+
                abs(current_MB[cur_y+3][cur_x+1] - reference_Data[ref_Block_y+ref_y+3][ref_Block_x+ref_x+1])
+
                abs(current_MB[cur_y+3][cur_x+2] - reference_Data[ref_Block_y+ref_y+3][ref_Block_x+ref_x+2])
+
                abs(current_MB[cur_y+3][cur_x+3] - reference_Data[ref_Block_y+ref_y+3][ref_Block_x+ref_x+3]);

//-------------------------------------------------------------------------------//

cur_x = 0;
ref_x = 0;
cur_y += 4;
ref_y += 4;

//-------------------------------------------------------------------------------//

Ex_B4x4_10 = abs(current_MB[cur_y+0][cur_x+0] - refer-
ence_Data[ref_Block_y+ref_y+0][ref_Block_x+ref_x+0]) +
                abs(current_MB[cur_y+0][cur_x+1] - reference_Data[ref_Block_y+ref_y+0][ref_Block_x+ref_x+1])
+
                abs(current_MB[cur_y+0][cur_x+2] - reference_Data[ref_Block_y+ref_y+0][ref_Block_x+ref_x+2])
+
                abs(current_MB[cur_y+0][cur_x+3] - reference_Data[ref_Block_y+ref_y+0][ref_Block_x+ref_x+3])
+
                abs(current_MB[cur_y+1][cur_x+0] - reference_Data[ref_Block_y+ref_y+1][ref_Block_x+ref_x+0])
+
                abs(current_MB[cur_y+1][cur_x+1] - reference_Data[ref_Block_y+ref_y+1][ref_Block_x+ref_x+1])
+
                abs(current_MB[cur_y+1][cur_x+2] - reference_Data[ref_Block_y+ref_y+1][ref_Block_x+ref_x+2])
+
```

```
                abs(current_MB[cur_y+1][cur_x+3] - reference_Data[ref_Block_y+ref_y+1][ref_Block_x+ref_x+3])
+
                abs(current_MB[cur_y+2][cur_x+0] - reference_Data[ref_Block_y+ref_y+2][ref_Block_x+ref_x+0])
+
                abs(current_MB[cur_y+2][cur_x+1] - reference_Data[ref_Block_y+ref_y+2][ref_Block_x+ref_x+1])
+
                abs(current_MB[cur_y+2][cur_x+2] - reference_Data[ref_Block_y+ref_y+2][ref_Block_x+ref_x+2])
+
                abs(current_MB[cur_y+2][cur_x+3] - reference_Data[ref_Block_y+ref_y+2][ref_Block_x+ref_x+3])
+
                abs(current_MB[cur_y+3][cur_x+0] - reference_Data[ref_Block_y+ref_y+3][ref_Block_x+ref_x+0])
+
                abs(current_MB[cur_y+3][cur_x+1] - reference_Data[ref_Block_y+ref_y+3][ref_Block_x+ref_x+1])
+
                abs(current_MB[cur_y+3][cur_x+2] - reference_Data[ref_Block_y+ref_y+3][ref_Block_x+ref_x+2])
+
                abs(current_MB[cur_y+3][cur_x+3] - reference_Data[ref_Block_y+ref_y+3][ref_Block_x+ref_x+3]);

cur_x += 4;
ref_x += 4;

Ex_B4x4_11 = abs(current_MB[cur_y+0][cur_x+0] - refer-
ence_Data[ref_Block_y+ref_y+0][ref_Block_x+ref_x+0]) +
                abs(current_MB[cur_y+0][cur_x+1] - reference_Data[ref_Block_y+ref_y+0][ref_Block_x+ref_x+1])
+
                abs(current_MB[cur_y+0][cur_x+2] - reference_Data[ref_Block_y+ref_y+0][ref_Block_x+ref_x+2])
+
                abs(current_MB[cur_y+0][cur_x+3] - reference_Data[ref_Block_y+ref_y+0][ref_Block_x+ref_x+3])
+
                abs(current_MB[cur_y+1][cur_x+0] - reference_Data[ref_Block_y+ref_y+1][ref_Block_x+ref_x+0])
+
                abs(current_MB[cur_y+1][cur_x+1] - reference_Data[ref_Block_y+ref_y+1][ref_Block_x+ref_x+1])
+
                abs(current_MB[cur_y+1][cur_x+2] - reference_Data[ref_Block_y+ref_y+1][ref_Block_x+ref_x+2])
+
                abs(current_MB[cur_y+1][cur_x+3] - reference_Data[ref_Block_y+ref_y+1][ref_Block_x+ref_x+3])
+
                abs(current_MB[cur_y+2][cur_x+0] - reference_Data[ref_Block_y+ref_y+2][ref_Block_x+ref_x+0])
+
                abs(current_MB[cur_y+2][cur_x+1] - reference_Data[ref_Block_y+ref_y+2][ref_Block_x+ref_x+1])
+
                abs(current_MB[cur_y+2][cur_x+2] - reference_Data[ref_Block_y+ref_y+2][ref_Block_x+ref_x+2])
+
                abs(current_MB[cur_y+2][cur_x+3] - reference_Data[ref_Block_y+ref_y+2][ref_Block_x+ref_x+3])
+
                abs(current_MB[cur_y+3][cur_x+0] - reference_Data[ref_Block_y+ref_y+3][ref_Block_x+ref_x+0])
+
                abs(current_MB[cur_y+3][cur_x+1] - reference_Data[ref_Block_y+ref_y+3][ref_Block_x+ref_x+1])
+
                abs(current_MB[cur_y+3][cur_x+2] - reference_Data[ref_Block_y+ref_y+3][ref_Block_x+ref_x+2])
+
                abs(current_MB[cur_y+3][cur_x+3] - reference_Data[ref_Block_y+ref_y+3][ref_Block_x+ref_x+3]);

cur_x += 4;
```

```
ref_x += 4;

Ex_B4x4_12 = abs(current_MB[cur_y+0][cur_x+0] - refer-
ence_Data[ref_Block_y+ref_y+0][ref_Block_x+ref_x+0]) +
                abs(current_MB[cur_y+0][cur_x+1] - reference_Data[ref_Block_y+ref_y+0][ref_Block_x+ref_x+1])
+
                abs(current_MB[cur_y+0][cur_x+2] - reference_Data[ref_Block_y+ref_y+0][ref_Block_x+ref_x+2])
+
                abs(current_MB[cur_y+0][cur_x+3] - reference_Data[ref_Block_y+ref_y+0][ref_Block_x+ref_x+3])
+
                abs(current_MB[cur_y+1][cur_x+0] - reference_Data[ref_Block_y+ref_y+1][ref_Block_x+ref_x+0])
+
                abs(current_MB[cur_y+1][cur_x+1] - reference_Data[ref_Block_y+ref_y+1][ref_Block_x+ref_x+1])
+
                abs(current_MB[cur_y+1][cur_x+2] - reference_Data[ref_Block_y+ref_y+1][ref_Block_x+ref_x+2])
+
                abs(current_MB[cur_y+1][cur_x+3] - reference_Data[ref_Block_y+ref_y+1][ref_Block_x+ref_x+3])
+
                abs(current_MB[cur_y+2][cur_x+0] - reference_Data[ref_Block_y+ref_y+2][ref_Block_x+ref_x+0])
+
                abs(current_MB[cur_y+2][cur_x+1] - reference_Data[ref_Block_y+ref_y+2][ref_Block_x+ref_x+1])
+
                abs(current_MB[cur_y+2][cur_x+2] - reference_Data[ref_Block_y+ref_y+2][ref_Block_x+ref_x+2])
+
                abs(current_MB[cur_y+2][cur_x+3] - reference_Data[ref_Block_y+ref_y+2][ref_Block_x+ref_x+3])
+
                abs(current_MB[cur_y+3][cur_x+0] - reference_Data[ref_Block_y+ref_y+3][ref_Block_x+ref_x+0])
+
                abs(current_MB[cur_y+3][cur_x+1] - reference_Data[ref_Block_y+ref_y+3][ref_Block_x+ref_x+1])
+
                abs(current_MB[cur_y+3][cur_x+2] - reference_Data[ref_Block_y+ref_y+3][ref_Block_x+ref_x+2])
+
                abs(current_MB[cur_y+3][cur_x+3] - reference_Data[ref_Block_y+ref_y+3][ref_Block_x+ref_x+3]);

cur_x += 4;
ref_x += 4;

Ex_B4x4_13 = abs(current_MB[cur_y+0][cur_x+0] - refer-
ence_Data[ref_Block_y+ref_y+0][ref_Block_x+ref_x+0]) +
                abs(current_MB[cur_y+0][cur_x+1] - reference_Data[ref_Block_y+ref_y+0][ref_Block_x+ref_x+1])
+
                abs(current_MB[cur_y+0][cur_x+2] - reference_Data[ref_Block_y+ref_y+0][ref_Block_x+ref_x+2])
+
                abs(current_MB[cur_y+0][cur_x+3] - reference_Data[ref_Block_y+ref_y+0][ref_Block_x+ref_x+3])
+
                abs(current_MB[cur_y+1][cur_x+0] - reference_Data[ref_Block_y+ref_y+1][ref_Block_x+ref_x+0])
+
                abs(current_MB[cur_y+1][cur_x+1] - reference_Data[ref_Block_y+ref_y+1][ref_Block_x+ref_x+1])
+
                abs(current_MB[cur_y+1][cur_x+2] - reference_Data[ref_Block_y+ref_y+1][ref_Block_x+ref_x+2])
+
                abs(current_MB[cur_y+1][cur_x+3] - reference_Data[ref_Block_y+ref_y+1][ref_Block_x+ref_x+3])
+
```

```
                abs(current_MB[cur_y+2][cur_x+0] - reference_Data[ref_Block_y+ref_y+2][ref_Block_x+ref_x+0])
+
                abs(current_MB[cur_y+2][cur_x+1] - reference_Data[ref_Block_y+ref_y+2][ref_Block_x+ref_x+1])
+
                abs(current_MB[cur_y+2][cur_x+2] - reference_Data[ref_Block_y+ref_y+2][ref_Block_x+ref_x+2])
+
                abs(current_MB[cur_y+2][cur_x+3] - reference_Data[ref_Block_y+ref_y+2][ref_Block_x+ref_x+3])
+
                abs(current_MB[cur_y+3][cur_x+0] - reference_Data[ref_Block_y+ref_y+3][ref_Block_x+ref_x+0])
+
                abs(current_MB[cur_y+3][cur_x+1] - reference_Data[ref_Block_y+ref_y+3][ref_Block_x+ref_x+1])
+
                abs(current_MB[cur_y+3][cur_x+2] - reference_Data[ref_Block_y+ref_y+3][ref_Block_x+ref_x+2])
+
                abs(current_MB[cur_y+3][cur_x+3] - reference_Data[ref_Block_y+ref_y+3][ref_Block_x+ref_x+3]);

//-------------------------------------------------------------------------------------------------------//

// The Upper 4 of the 8 4x8 SAD Equivalency Checking registers

Ex_B4x8_00 = Ex_B4x4_00 + Ex_B4x4_10;
Ex_B4x8_01 = Ex_B4x4_01 + Ex_B4x4_11;
Ex_B4x8_02 = Ex_B4x4_02 + Ex_B4x4_12;
Ex_B4x8_03 = Ex_B4x4_03 + Ex_B4x4_13;

// The Upper 4 of the 8 8x4 SAD Equivalency Checking registers

Ex_B8x4_00 = Ex_B4x4_00 + Ex_B4x4_01;
Ex_B8x4_01 = Ex_B4x4_02 + Ex_B4x4_03;

Ex_B8x4_10 = Ex_B4x4_10 + Ex_B4x4_11;
Ex_B8x4_11 = Ex_B4x4_12 + Ex_B4x4_13;

// The Upper 2 of the 4 8x8 SAD Equivalency Checking registers

Ex_B8x8_00 = Ex_B8x4_00 + Ex_B8x4_10;
Ex_B8x8_01 = Ex_B8x4_01 + Ex_B8x4_11;

// The Upper 1 of the 2 16x8 SAD Equivalency Checking registers

Ex_B16x8_0 = Ex_B8x8_00 + Ex_B8x8_01;

//-----------------------------------------------------------------------------------//

// Equivalency Checking Code:

if ((reference_Cycle>=7)&(reference_Cycle<1096)){
    //the upper 8 half of the B4x4 blocks (i.e. the top 8 out of 16) are checked here
    if (B4x4_00_out!=Ex_B4x4_00) printf("\n\nError B4x4_00! Expected: %d Got: %d\n\n", Ex_B4x4_00,
B4x4_00_out);
    if (B4x4_01_out!=Ex_B4x4_01) printf("\n\nError B4x4_01! Expected: %d Got: %d\n\n", Ex_B4x4_01,
B4x4_01_out);
    if (B4x4_02_out!=Ex_B4x4_02) printf("\n\nError B4x4_02! Expected: %d Got: %d\n\n", Ex_B4x4_02,
B4x4_02_out);
```

```
    if (B4x4_03_out!=Ex_B4x4_03) printf("\n\nError B4x4_03! Expected: %d Got: %d\n\n", Ex_B4x4_03,
B4x4_03_out);
    if (B4x4_10_out!=Ex_B4x4_10) printf("\n\nError B4x4_10! Expected: %d Got: %d\n\n", Ex_B4x4_10,
B4x4_10_out);
    if (B4x4_11_out!=Ex_B4x4_11) printf("\n\nError B4x4_11! Expected: %d Got: %d\n\n", Ex_B4x4_11,
B4x4_11_out);
    if (B4x4_12_out!=Ex_B4x4_12) printf("\n\nError B4x4_12! Expected: %d Got: %d\n\n", Ex_B4x4_12,
B4x4_12_out);
    if (B4x4_13_out!=Ex_B4x4_13) printf("\n\nError B4x4_13! Expected: %d Got: %d\n\n", Ex_B4x4_13,
B4x4_13_out);
    fprintf(B4x4_top, "%s\n", int2Bin_13bits(B4x4_00_out));
    //fprintf(B4x4_top, "%s     %d\n", int2Bin_13bits(B4x4_00_out), B4x4_00_out);
    fprintf(B4x4_top, "%s\n", int2Bin_13bits(B4x4_01_out));
    fprintf(B4x4_top, "%s\n", int2Bin_13bits(B4x4_02_out));
    fprintf(B4x4_top, "%s\n", int2Bin_13bits(B4x4_03_out));
    fprintf(B4x4_top, "%s\n", int2Bin_13bits(B4x4_10_out));
    fprintf(B4x4_top, "%s\n", int2Bin_13bits(B4x4_11_out));
    fprintf(B4x4_top, "%s\n", int2Bin_13bits(B4x4_12_out));
    fprintf(B4x4_top, "%s\n", int2Bin_13bits(B4x4_13_out));

    // the upper 4 of the 8 B4x8 blocks
    if (B4x8_00_out!=Ex_B4x8_00) printf("\n\nError B4x8_00! Expected: %d Got: %d\n\n", Ex_B4x8_00,
B4x8_00_out);
    if (B4x8_01_out!=Ex_B4x8_01) printf("\n\nError B4x8_01! Expected: %d Got: %d\n\n", Ex_B4x8_01,
B4x8_01_out);
    if (B4x8_02_out!=Ex_B4x8_02) printf("\n\nError B4x8_02! Expected: %d Got: %d\n\n", Ex_B4x8_02,
B4x8_02_out);
    if (B4x8_03_out!=Ex_B4x8_03) printf("\n\nError B4x8_03! Expected: %d Got: %d\n\n", Ex_B4x8_03,
B4x8_03_out);
    fprintf(B4x8_top, "%s\n", int2Bin_14bits(B4x8_00_out));
    fprintf(B4x8_top, "%s\n", int2Bin_14bits(B4x8_01_out));
    fprintf(B4x8_top, "%s\n", int2Bin_14bits(B4x8_02_out));
    fprintf(B4x8_top, "%s\n", int2Bin_14bits(B4x8_03_out));

    // the upper 4 of the 8 B8x4 blocks
    if (B8x4_00_out!=Ex_B8x4_00) printf("\n\nError B8x4_00! Expected: %d Got: %d\n\n", Ex_B8x4_00,
B8x4_00_out);
    if (B8x4_01_out!=Ex_B8x4_01) printf("\n\nError B8x4_01! Expected: %d Got: %d\n\n", Ex_B8x4_01,
B8x4_01_out);
    if (B8x4_10_out!=Ex_B8x4_10) printf("\n\nError B8x4_10! Expected: %d Got: %d\n\n", Ex_B8x4_10,
B8x4_10_out);
    if (B8x4_11_out!=Ex_B8x4_11) printf("\n\nError B8x4_11! Expected: %d Got: %d\n\n", Ex_B8x4_11,
B8x4_11_out);
    fprintf(B8x4_top, "%s\n", int2Bin_14bits(B8x4_00_out));
    fprintf(B8x4_top, "%s\n", int2Bin_14bits(B8x4_01_out));
    fprintf(B8x4_top, "%s\n", int2Bin_14bits(B8x4_10_out));
    fprintf(B8x4_top, "%s\n", int2Bin_14bits(B8x4_11_out));

    // the upper 2 of the 4 B8x8 blocks
    if (B8x8_00_out!=Ex_B8x8_00) printf("\n\nError B8x8_00! Expected: %d Got: %d\n\n", Ex_B8x8_00,
B8x8_00_out);
    if (B8x8_01_out!=Ex_B8x8_01) printf("\n\nError B8x8_01! Expected: %d Got: %d\n\n", Ex_B8x8_01,
B8x8_01_out);
    fprintf(B8x8_top, "%s\n", int2Bin_15bits(B8x8_00_out));
```

```
      fprintf(B8x8_top, "%s\n", int2Bin_15bits(B8x8_01_out));

      // the upper 1 of the 2 B16x8 blocks
      if (B16x8_0_out!=Ex_B16x8_0) printf("\n\nError B16x8_0! Expected: %d Got: %d\n\n", Ex_B16x8_0,
B16x8_0_out);
      fprintf(B16x8_top, "%s\n", int2Bin_16bits(B16x8_0_out));
}

      }// ends the clock_Cycle loop
}//ends the Test_Case loop of 1000

      fclose(fp);
}// ends test_main_data_path_top()
```

```
#include <stdlib.h>
#include <stdio.h>

FILE *mem_A_file;
FILE *mem_B_file;
FILE *ref_pixels;

// This Function tests the on-chip memory unit.
// External Functions located in other Project Files

//-------------------------------------------------------------------------------------------//
void on_chip_mem (// inputs
                  int write_enable,
                  int address, // 6-bits (i.e. 0 to 47 search pixel rows)
                  int load_bottom,
                  int load_top,

                  int read_L0_L1,
                  int read_L1_L2,
                  int read_L2_L3,

                  int block_66x16_mem_1 [66][16],
                  int block_66x16_mem_2 [66][16],
                  int block_30x16_mem_1 [66][16],
                  int block_30x16_mem_2 [66][16],

                  int *partition_A_Counter,
                  int *partition_B_Counter,

                  // Input Bus
                  int dataIn_32_Bytes[32],
                  // Output Buses
                  int dataOut_31_Bytes_A[32],// an extra byte in the array is added
                  int dataOut_31_Bytes_B[32] // for programming convenience
                 );

int random_int_between_0_and_255();

//-------------------------------------------------------------------------------------------//


void test_mem(){

// The Static Memory Blocks that are Used
int block_66x16_mem_1 [66][16];
int block_66x16_mem_2 [66][16];
int block_30x16_mem_1 [66][16];
int block_30x16_mem_2 [66][16];

// Input Wires & Buses
int write_enable,
    address, // 6-bits (i.e. 0 to 47 search pixel rows)
```

```
	load_bottom,
	load_top;

int read_L0_L1, read_L1_L2, read_L2_L3;

// Input Bus
int input_BUS_32bytes[32];

// Ouput Buses
int output_BUS_31bytes_A[32]; // an extra byte in the array is added
int output_BUS_31bytes_B[32]; // for programming convenience

int Test_Case = 0;

if ((mem_A_file = fopen("mem_A_out.txt", "w"))==NULL){
		fprintf(mem_A_file, "\n\n Can not open the output file. \n\n");
		exit(1);
}
if ((mem_B_file = fopen("mem_B_out.txt", "w"))==NULL){
		fprintf(mem_B_file, "\n\n Can not open the output file. \n\n");
		exit(1);
}

if ((ref_pixels = fopen("ref_pixels.txt", "w"))==NULL){
		printf("\n\n Can not open the ref_pixels output file. \n\n");
		exit(1);
}

//-------------------------------------------------------------------------------------------------------------//

for (Test_Case = 0; Test_Case<1; Test_Case++){

	int clock_Cycle = 0;
	int reference_Data[48][64];
	int i, j = 0;
	int value, value_A, value_B;

	int partition_A_Counter = 0;
	int partition_B_Counter = 0;

	write_enable = 0;
	address	= 0;
	load_bottom  = 0;
	load_top = 0;

	read_L0_L1 = 0;
	read_L1_L2 = 0;
	read_L2_L3 = 0;

	fprintf(ref_pixels, "\n");
	for (i=0; i<48; i++){
		for (j=0; j<64; j++){
			reference_Data[i][j] = (i*100)+j;
			value = reference_Data[i][j];
```

```
            if      (value<10)   fprintf(ref_pixels, "   %d  ", value);
            else if (value<100)  fprintf(ref_pixels, "  %d  ", value);
            else if (value<1000) fprintf(ref_pixels, " %d  ", value);
            else                 fprintf(ref_pixels, "%d  ", value);
        }
        fprintf(ref_pixels, "\n");
    }

//-------------------------------------------------------------------------------//

    for (clock_Cycle = 0; clock_Cycle<210; clock_Cycle++){// 210 = 48 + 48 + (3*33) + 15

        if (clock_Cycle<96) write_enable = 1;
        else                write_enable = 0;

        if (clock_Cycle<48){
            load_bottom = 1;
            for (i=0; i<32; i++){
                input_BUS_32bytes[i] = reference_Data[clock_Cycle][i];
            }
        }
        else                load_bottom = 0;

        if ((clock_Cycle>47) && (clock_Cycle<96)){
             load_top       = 1;
             for (i=0; i<32; i++){
                input_BUS_32bytes[i] = reference_Data[clock_Cycle-48][i+32];
             }
        }
        else load_top = 0;

        if ((clock_Cycle>95) && (clock_Cycle<129))
                read_L0_L1 = 1;
        else read_L0_L1 = 0;

        if ((clock_Cycle>128) && (clock_Cycle<162))
                read_L1_L2 = 1;
        else read_L1_L2 = 0;

        if (clock_Cycle>161)
                read_L2_L3 = 1;
        else read_L2_L3 = 0;

//-------------------------------------------------------------------------------------------------------//
// Instantiation of the Device Unter Test (DUT) i.e. the on chip memory

    on_chip_mem (// inputs
                    write_enable,
                    address, // 6-bits (i.e. 0 to 47 search pixel rows)
                    load_bottom,
                    load_top,

                    read_L0_L1,
                    read_L1_L2,
```

```
                        read_L2_L3,

                        block_66x16_mem_1,
                        block_66x16_mem_2,
                        block_30x16_mem_1,
                        block_30x16_mem_2,

                        &partition_A_Counter,
                        &partition_B_Counter,

                        // Input Bus
                        input_BUS_32bytes,
                        // Output Buses
                        output_BUS_31bytes_A,
                        output_BUS_31bytes_B
                       );

//-------------------------------------------------------------------------------------------------------//

    if (clock_Cycle>95){

    fprintf(mem_A_file, "\n");
    fprintf(mem_B_file, "\n");
    if (clock_Cycle<100){
        fprintf(mem_A_file, " %d  ", clock_Cycle);
        fprintf(mem_B_file, " %d  ", clock_Cycle);
    }
    else{
        fprintf(mem_A_file, "%d  ", clock_Cycle);
        fprintf(mem_B_file, "%d  ", clock_Cycle);
    }

    for (i=0; i<31; i++){
        value_A = output_BUS_31bytes_A[i];
        value_B = output_BUS_31bytes_B[i];
            if     (value_A<10)   fprintf(mem_A_file, "   %d  ", value_A);
            else if (value_A<100)  fprintf(mem_A_file, "  %d  ", value_A);
            else if (value_A<1000) fprintf(mem_A_file, " %d  ", value_A);
            else                   fprintf(mem_A_file, "%d  ", value_A);
            if     (value_B<10)   fprintf(mem_B_file, "   %d  ", value_B);
            else if (value_B<100)  fprintf(mem_B_file, "  %d  ", value_B);
            else if (value_B<1000) fprintf(mem_B_file, " %d  ", value_B);
            else                   fprintf(mem_B_file, "%d  ", value_B);
    }

    if ((clock_Cycle==128) || (clock_Cycle==161) || (clock_Cycle==194) || (clock_Cycle==227)){
        fprintf(mem_A_file, "\n\n");
        fprintf(mem_B_file, "\n\n");
    }

    }// ends if clock_Cycle>95


        address++;
```

```
            if (address==48) address = 0;
        }// ends the clock_Cyle for loop

}// ends the Test Case for loop

}// ends the main void test_mem() function
```

7.3 Condensed Synthesis Reports

This section contains the shortened Synthesis Report file data for 6 designs in total. The first five reports are for the designs with 1, 2, 4, 8, and 16 PPUs utilized (all with external memory assumed). The last 6th report gives the condensed synthesis results for a complete system design that uses 16 PPUs and double-buffering to store two search windows simultaneously on-chip.

First design results with 1 PPU utilized:

```
Release 9.2i - xst J.36
Copyright (c) 1995-2007 Xilinx, Inc.  All rights reserved.
--> Parameter TMPDIR set to ./xst/projnav.tmp
CPU : 0.00 / 0.20 s | Elapsed : 0.00 / 0.00 s

--> Parameter xsthdpdir set to ./xst
CPU : 0.00 / 0.20 s | Elapsed : 0.00 / 0.00 s

--> Reading design: main_data_path.prj


=========================================================================
*                     Synthesis Options Summary                         *
=========================================================================
---- Source Parameters
Input File Name                    : "main_data_path.prj"
Input Format                       : mixed
Ignore Synthesis Constraint File   : NO

---- Target Parameters
Output File Name                   : "main_data_path"
Output Format                      : NGC
Target Device                      : xc5vlx330-2-ff1760

---- Source Options
Top Module Name                    : main_data_path
Automatic FSM Extraction           : YES
FSM Encoding Algorithm             : Auto
Safe Implementation                : No
FSM Style                          : lut
RAM Extraction                     : Yes
RAM Style                          : Auto
ROM Extraction                     : Yes
Mux Style                          : Auto
Decoder Extraction                 : YES
Priority Encoder Extraction        : YES
Shift Register Extraction          : YES
Logical Shifter Extraction         : YES
```

```
XOR Collapsing                  : YES
ROM Style                       : Auto
Mux Extraction                  : YES
Resource Sharing                : YES
Asynchronous To Synchronous     : NO
Use DSP Block                   : auto
Automatic Register Balancing    : No

---- Target Options
Add IO Buffers                  : YES
Global Maximum Fanout           : 100000
Add Generic Clock Buffer(BUFG)  : 32
Register Duplication            : YES
Slice Packing                   : YES
Optimize Instantiated Primitives : NO
Use Clock Enable                : Auto
Use Synchronous Set             : Auto
Use Synchronous Reset           : Auto
Pack IO Registers into IOBs     : auto
Equivalent register Removal     : YES

---- General Options
Optimization Goal               : Speed
Optimization Effort             : 1
Power Reduction                 : NO
Library Search Order            : main_data_path.lso
Keep Hierarchy                  : NO
RTL Output                      : Yes
Global Optimization             : AllClockNets
Read Cores                      : YES
Write Timing Constraints        : NO
Cross Clock Analysis            : NO
Hierarchy Separator             : /
Bus Delimiter                   : <>
Case Specifier                  : maintain
Slice Utilization Ratio         : 100
BRAM Utilization Ratio          : 100
DSP48 Utilization Ratio         : 100
Verilog 2001                    : YES
Auto BRAM Packing               : NO
Slice Utilization Ratio Delta   : 5

=========================================================================


=========================================================================
*                     HDL Compilation                   *
=========================================================================
=========================================================================
*                Design Hierarchy Analysis              *
=========================================================================
Analyzing hierarchy for module <main_data_path> in library <work>.

Analyzing hierarchy for module <four_by_four_block> in library <work>.
```

Analyzing hierarchy for module <four_stage_delay_line> in library <work>.

Analyzing hierarchy for module <eight_stage_delay_line> in library <work>.

Analyzing hierarchy for module <Row_Adder_0> in library <work>.

Analyzing hierarchy for module <Row_Adder_1_CSA> in library <work>.

Analyzing hierarchy for module <Row_Adder_2_CSA> in library <work>.

Analyzing hierarchy for module <Row_Adder_3_CSA> in library <work>.

Analyzing hierarchy for module <PE_AD> in library <work>.

Analyzing hierarchy for module <PE_AD_CSA> in library <work>.

Analyzing hierarchy for module <CSA_8> in library <work>.

Analyzing hierarchy for module <CSA_9> in library <work>.

Analyzing hierarchy for module <CSA_Full_Adder> in library <work>.

Unit <CU_1> synthesized.

==========

==========
* Advanced HDL Synthesis *
==========

Loading device for application Rf_Device from file '5vlx330.nph' in environment C:\Xilinx92i.

==========
Advanced HDL Synthesis Report

Macro Statistics
ROMs : 1
16x15-bit ROM : 1
Adders/Subtractors : 441
10-bit adder : 16
11-bit adder carry in : 16
12-bit adder carry in : 16
13-bit adder carry in : 16
14-bit adder : 16
15-bit adder : 4
16-bit adder : 4
17-bit adder : 1
8-bit adder : 64
9-bit adder : 288
Registers : 393
10-bit register : 16
11-bit register : 16
12-bit register : 16

```
 13-bit register                       : 48
 14-bit register                       : 16
 15-bit register                       : 20
 16-bit register                       : 4
 17-bit register                       : 1
 8-bit register                        : 256
# Xors                                 : 2656
 1-bit xor3                            : 2400
 8-bit xor2                            : 256


=========================================================================
*                  Low Level Synthesis                  *
=========================================================================

Optimizing unit <main_data_path> ...

Optimizing unit <four_stage_delay_line> ...

Optimizing unit <eight_stage_delay_line> ...

Optimizing unit <PE_AD_CSA> ...

Optimizing unit <Row_Adder_0> ...

Optimizing unit <Row_Adder_1_CSA> ...

Optimizing unit <Row_Adder_2_CSA> ...

Optimizing unit <Row_Adder_3_CSA> ...

Optimizing unit <four_by_four_block> ...

Mapping all equations...
Building and optimizing final netlist ...
Found area constraint ratio of 100 (+ 5) on block main_data_path, actual ratio is 6.

Final Macro Processing ...

Processing Unit <CU_8> :
Unit <CU_8> processed.

=========================================================================
Final Register Report

Macro Statistics
# Registers                           : 3283
 Flip-Flops                           : 3283
# Shift Registers                      : 134
 3-bit shift register                  : 104
 7-bit shift register                  : 30

=========================================================================
```

```
=========================================================================
*                      Partition Report                       *
=========================================================================

Partition Implementation Status
-------------------------------

  No Partitions were found in this design.

-------------------------------

=========================================================================
*                      Final Report                           *
=========================================================================
Final Results
RTL Top Level Output File Name     : main_data_path.ngr
Top Level Output File Name         : main_data_path
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 841

Cell Usage :
# BELS                             : 14659
#      GND                         : 1
#      LUT1                        : 48
#      LUT2                        : 2004
#      LUT3                        : 400
#      LUT4                        : 1838
#      LUT5                        : 424
#      LUT6                        : 3864
#      MUXCY                       : 3188
#      VCC                         : 1
#      XORCY                       : 2891
# FlipFlops/Latches                : 3417
#      FDE                         : 134
#      FDRE                        : 3283
# Shift Registers                  : 134
#      SRLC32E                     : 134
# Clock Buffers                    : 2
#      BUFGP                       : 2
# IO Buffers                       : 839
#      IBUF                        : 266
#      OBUF                        : 573
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 5vlx330ff1760-2
```

```
Slice Logic Utilization:
 Number of Slice Registers:          3417  out of  207360     1%
 Number of Slice LUTs:               8712  out of  207360     4%
    Number used as Logic:            8578  out of  207360     4%
    Number used as Memory:            134  out of  54720     0%
      Number used as SRL:             134

Slice Logic Distribution:
 Number of Bit Slices used:         10175
   Number with an unused Flip Flop    6758  out of  10175    66%
   Number with an unused LUT:         1463  out of  10175    14%
   Number of fully used Bit Slices:   1954  out of  10175    19%

IO Utilization:
 Number of IOs:                  841
 Number of bonded IOBs:              841  out of   1200    70%

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:             2  out of     32     6%

---------------------------
Partition Resource Summary:
---------------------------

  No Partitions were found in this design.

---------------------------


=================================================================================
TIMING REPORT

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
      FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
      GENERATED AFTER PLACE-and-ROUTE.

Clock Information:
------------------
-----------------------------------+------------------------+-------+
Clock Signal                       | Clock buffer(FF name)  | Load  |
-----------------------------------+------------------------+-------+
clk                                | BUFGP                  | 3551  |
-----------------------------------+------------------------+-------+

Asynchronous Control Signals Information:
----------------------------------------
No asynchronous control signals found in this design

Timing Summary:
---------------
Speed Grade: -2

   Minimum period: 4.983ns (Maximum Frequency: 200.690MHz)
   Minimum input arrival time before clock: 6.380ns
```

```
   Maximum output required time after clock: 2.835ns
   Maximum combinational path delay: No path found

Timing Detail:
--------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default period analysis for Clock 'clk'
  Clock period: 4.983ns (frequency: 200.690MHz)
  Total number of paths / destination ports: 1157262 / 1503
-------------------------------------------------------------------------
Delay:               4.983ns (Levels of Logic = 16)
  Source:            B4x4_03_B/PSAD_0/AD3/C_Registered_0 (FF)
  Destination:       B4x4_03_B/PSAD_0/PSAD_0_8 (FF)
  Source Clock:      clk rising
  Destination Clock: clk rising

  Data Path: B4x4_03_B/PSAD_0/AD3/C_Registered_0 to B4x4_03_B/PSAD_0/PSAD_0_8
                            Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDRE:C->Q             9   0.396   0.449  B4x4_03_B/PSAD_0/AD3/C_Registered_0
(B4x4_03_B/PSAD_0/AD3/C_Registered_0)
     LUT2:I1->O            3   0.086   0.000  B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_lut<0>
(B4x4_03_B/PSAD_0/AD3/Nine_Bit_Sum<0>)
     MUXCY:S->O            1   0.305   0.000  B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<0>
(B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<0>)
     MUXCY:CI->O           1   0.023   0.000  B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<1>
(B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<1>)
     MUXCY:CI->O           1   0.023   0.000  B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<2>
(B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<2>)
     MUXCY:CI->O           1   0.023   0.000  B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<3>
(B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<3>)
     MUXCY:CI->O           1   0.023   0.000  B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<4>
(B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<4>)
     MUXCY:CI->O           1   0.023   0.000  B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<5>
(B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<5>)
     MUXCY:CI->O           1   0.023   0.000  B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<6>
(B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<6>)
     MUXCY:CI->O           9   0.222   0.938  B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<7>
(B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<7>)
     LUT6:I0->O            1   0.086   0.487  B4x4_03_B/PSAD_0/AD3_Out<5>1
(B4x4_03_B/PSAD_0/AD3_Out<5>)
     LUT2:I0->O            1   0.086   0.000  B4x4_03_B/PSAD_0/Madd_Nine_Bit_Sum_2_lut<5>
(B4x4_03_B/PSAD_0/N50)
     MUXCY:S->O            1   0.305   0.000  B4x4_03_B/PSAD_0/Madd_Nine_Bit_Sum_2_cy<5>
(B4x4_03_B/PSAD_0/Madd_Nine_Bit_Sum_2_cy<5>)
     XORCY:CI->O           3   0.300   0.496  B4x4_03_B/PSAD_0/Madd_Nine_Bit_Sum_2_xor<6>
(B4x4_03_B/PSAD_0/Nine_Bit_Sum_2<6>)
     LUT6:I4->O            1   0.086   0.000  B4x4_03_B/PSAD_0/Madd_PSAD_0_add0000_Madd_lut<7>
(B4x4_03_B/PSAD_0/N59)
     MUXCY:S->O            1   0.305   0.000  B4x4_03_B/PSAD_0/Madd_PSAD_0_add0000_Madd_cy<7>
(B4x4_03_B/PSAD_0/Madd_PSAD_0_add0000_Madd_cy<7>)
```

```
    XORCY:CI->O           1   0.300   0.000  B4x4_03_B/PSAD_0/Madd_PSAD_0_add0000_Madd_xor<8>
(B4x4_03_B/PSAD_0/PSAD_0_add0000<8>)
    FDRE:D                    -0.022          B4x4_03_B/PSAD_0/PSAD_0_8
    ----------------------------------------
    Total                     4.983ns (2.613ns logic, 2.370ns route)
                              (52.4% logic, 47.6% route)

=========================================================================
Timing constraint: Default OFFSET IN BEFORE for Clock 'clk'
  Total number of paths / destination ports: 12271122 / 9618
-------------------------------------------------------------------------
Offset:              6.380ns (Levels of Logic = 18)
  Source:            bus_Select<3> (PAD)
  Destination:       B4x4_33_B/PSAD_0/PSAD_0_8 (FF)
  Destination Clock: clk rising

  Data Path: bus_Select<3> to B4x4_33_B/PSAD_0/PSAD_0_8
                            Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
    IBUF:I->O          2190   0.694   0.844  bus_Select_3_IBUF (bus_Select_3_IBUF)
    LUT4:I0->O          184   0.086   0.619  Mrom_bus_Select_rom000041 (Mrom_bus_Select_rom00003)
    LUT4:I2->O            4   0.086   0.000  B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_lut<0>
(B4x4_33_B/PSAD_0/AD3/Nine_Bit_Sum<0>)
    MUXCY:S->O            1   0.305   0.000  B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<0>
(B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<0>)
    MUXCY:CI->O           1   0.023   0.000  B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<1>
(B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<1>)
    MUXCY:CI->O           1   0.023   0.000  B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<2>
(B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<2>)
    MUXCY:CI->O           1   0.023   0.000  B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<3>
(B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<3>)
    MUXCY:CI->O           1   0.023   0.000  B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<4>
(B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<4>)
    MUXCY:CI->O           1   0.023   0.000  B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<5>
(B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<5>)
    MUXCY:CI->O           1   0.023   0.000  B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<6>
(B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<6>)
    MUXCY:CI->O           9   0.222   0.938  B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<7>
(B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<7>)
    LUT6:I0->O            1   0.086   0.487  B4x4_33_B/PSAD_0/AD3_Out<5>1
(B4x4_33_B/PSAD_0/AD3_Out<5>)
    LUT2:I0->O            1   0.086   0.000  B4x4_33_B/PSAD_0/Madd_Nine_Bit_Sum_2_lut<5>
(B4x4_33_B/PSAD_0/N50)
    MUXCY:S->O            1   0.305   0.000  B4x4_33_B/PSAD_0/Madd_Nine_Bit_Sum_2_cy<5>
(B4x4_33_B/PSAD_0/Madd_Nine_Bit_Sum_2_cy<5>)
    XORCY:CI->O           3   0.300   0.496  B4x4_33_B/PSAD_0/Madd_Nine_Bit_Sum_2_xor<6>
(B4x4_33_B/PSAD_0/Nine_Bit_Sum_2<6>)
    LUT6:I4->O            1   0.086   0.000  B4x4_33_B/PSAD_0/Madd_PSAD_0_add0000_Madd_lut<7>
(B4x4_33_B/PSAD_0/N59)
    MUXCY:S->O            1   0.305   0.000  B4x4_33_B/PSAD_0/Madd_PSAD_0_add0000_Madd_cy<7>
(B4x4_33_B/PSAD_0/Madd_PSAD_0_add0000_Madd_cy<7>)
    XORCY:CI->O           1   0.300   0.000  B4x4_33_B/PSAD_0/Madd_PSAD_0_add0000_Madd_xor<8>
(B4x4_33_B/PSAD_0/PSAD_0_add0000<8>)
```

```
    FDRE:D                 -0.022          B4x4_33_B/PSAD_0/PSAD_0_8
    ----------------------------------------
    Total                  6.380ns (2.997ns logic, 3.383ns route)
                              (47.0% logic, 53.0% route)

=========================================================================
Timing constraint: Default OFFSET OUT AFTER for Clock 'clk'
  Total number of paths / destination ports: 573 / 573
-------------------------------------------------------------------------
Offset:              2.835ns (Levels of Logic = 1)
  Source:            B4x4_10_B/PSAD_3/PSAD_3_12 (FF)
  Destination:       B4x4_10_out<12> (PAD)
  Source Clock:      clk rising

  Data Path: B4x4_10_B/PSAD_3/PSAD_3_12 to B4x4_10_out<12>
                            Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
    FDRE:C->Q             3   0.396   0.295  B4x4_10_B/PSAD_3/PSAD_3_12 (B4x4_10_B/PSAD_3/PSAD_3_12)
    OBUF:I->O                 2.144          B4x4_10_out_12_OBUF (B4x4_10_out<12>)
    ----------------------------------------
    Total                  2.835ns (2.540ns logic, 0.295ns route)
                              (89.6% logic, 10.4% route)

=========================================================================
CPU : 274.81 / 275.05 s | Elapsed : 275.00 / 275.00 s

-->

Total memory usage is 550700 kilobytes

Number of errors   :    0 (   0 filtered)
Number of warnings :    0 (   0 filtered)
Number of infos    :    0 (   0 filtered)
```

Second design results with 2 PPUs utilized:

```
Release 9.2i - xst J.36
Copyright (c) 1995-2007 Xilinx, Inc.  All rights reserved.
--> Parameter TMPDIR set to ./xst/projnav.tmp
CPU : 0.00 / 0.20 s | Elapsed : 0.00 / 0.00 s

--> Parameter xsthdpdir set to ./xst
CPU : 0.00 / 0.20 s | Elapsed : 0.00 / 0.00 s

--> Reading design: CU_2.prj

=========================================================================
*                Synthesis Options Summary                *
=========================================================================
```

---- Source Parameters
Input File Name : "CU_2.prj"
Input Format : mixed
Ignore Synthesis Constraint File : NO

---- Target Parameters
Output File Name : "CU_2"
Output Format : NGC
Target Device : xc5vlx330-2-ff1760

---- Source Options
Top Module Name : CU_2
Automatic FSM Extraction : YES
FSM Encoding Algorithm : Auto
Safe Implementation : No
FSM Style : lut
RAM Extraction : Yes
RAM Style : Auto
ROM Extraction : Yes
Mux Style : Auto
Decoder Extraction : YES
Priority Encoder Extraction : YES
Shift Register Extraction : YES
Logical Shifter Extraction : YES
XOR Collapsing : YES
ROM Style : Auto
Mux Extraction : YES
Resource Sharing : YES
Asynchronous To Synchronous : NO
Use DSP Block : auto
Automatic Register Balancing : No

---- Target Options
Add IO Buffers : YES
Global Maximum Fanout : 100000
Add Generic Clock Buffer(BUFG) : 32
Register Duplication : YES
Slice Packing : YES
Optimize Instantiated Primitives : NO
Use Clock Enable : Auto
Use Synchronous Set : Auto
Use Synchronous Reset : Auto
Pack IO Registers into IOBs : auto
Equivalent register Removal : YES

---- General Options
Optimization Goal : Speed
Optimization Effort : 1
Power Reduction : NO
Library Search Order : CU_2.lso
Keep Hierarchy : NO
RTL Output : Yes
Global Optimization : AllClockNets
Read Cores : YES

```
Write Timing Constraints        : NO
Cross Clock Analysis            : NO
Hierarchy Separator             : /
Bus Delimiter                   : <>
Case Specifier                  : maintain
Slice Utilization Ratio         : 100
BRAM Utilization Ratio          : 100
DSP48 Utilization Ratio         : 100
Verilog 2001                    : YES
Auto BRAM Packing               : NO
Slice Utilization Ratio Delta   : 5

=========================================================================


=========================================================================
*                     HDL Compilation                   *
=========================================================================
=========================================================================
*                  Design Hierarchy Analysis                    *
=========================================================================
Analyzing hierarchy for module <CU_2> in library <work>.

Analyzing hierarchy for module <main_data_path> in library <work>.

Analyzing hierarchy for module <four_by_four_block> in library <work>.

Analyzing hierarchy for module <four_stage_delay_line> in library <work>.

Analyzing hierarchy for module <eight_stage_delay_line> in library <work>.

Analyzing hierarchy for module <Row_Adder_0> in library <work>.

Analyzing hierarchy for module <Row_Adder_1_CSA> in library <work>.

Analyzing hierarchy for module <Row_Adder_2_CSA> in library <work>.

Analyzing hierarchy for module <Row_Adder_3_CSA> in library <work>.

Analyzing hierarchy for module <PE_AD> in library <work>.

Analyzing hierarchy for module <PE_AD_CSA> in library <work>.

Analyzing hierarchy for module <CSA_8> in library <work>.

Analyzing hierarchy for module <CSA_9> in library <work>.

Analyzing hierarchy for module <CSA_Full_Adder> in library <work>.

Unit <CU_2> synthesized.

=========================================================================

=========================================================================
```

```
*                  Advanced HDL Synthesis                  *
==========================================================================

Loading device for application Rf_Device from file '5vlx330.nph' in environment C:\Xilinx92i.

==========================================================================
Advanced HDL Synthesis Report

Macro Statistics
# ROMs                          : 2
 16x15-bit ROM                  : 2
# Adders/Subtractors            : 882
 10-bit adder                   : 32
 11-bit adder carry in          : 32
 12-bit adder carry in          : 32
 13-bit adder carry in          : 32
 14-bit adder                   : 32
 15-bit adder                   : 8
 16-bit adder                   : 8
 17-bit adder                   : 2
 8-bit adder                    : 128
 9-bit adder                    : 576
# Registers                     : 8183
 Flip-Flops                     : 8183
# Comparators                   : 41
 13-bit comparator less         : 16
 14-bit comparator less         : 16
 15-bit comparator less         : 4
 16-bit comparator less         : 4
 17-bit comparator less         : 1
# Xors                          : 5312
 1-bit xor3                     : 4800
 8-bit xor2                     : 512

==========================================================================

==========================================================================
*                  Low Level Synthesis                     *
==========================================================================

Optimizing unit <CU_2> ...

Optimizing unit <four_stage_delay_line> ...

Optimizing unit <eight_stage_delay_line> ...

Optimizing unit <PE_AD_CSA> ...

Optimizing unit <Row_Adder_0> ...

Optimizing unit <Row_Adder_1_CSA> ...

Optimizing unit <Row_Adder_2_CSA> ...
```

Optimizing unit <Row_Adder_3_CSA> ...

Optimizing unit <four_by_four_block> ...

Optimizing unit <main_data_path> ...

Mapping all equations...
Building and optimizing final netlist ...
Found area constraint ratio of 100 (+ 5) on block CU_2, actual ratio is 13.

Final Macro Processing ...

Processing Unit <CU_2> :
Unit <CU_2> processed.

===
Final Register Report

Macro Statistics
Registers : 5219
Flip-Flops : 5219
Shift Registers : 268
3-bit shift register : 208
7-bit shift register : 60

===

===
* Partition Report *
===

Partition Implementation Status

No Partitions were found in this design.

===
* Final Report *
===
Final Results
RTL Top Level Output File Name : CU_2.ngr
Top Level Output File Name : CU_2
Output Format : NGC
Optimization Goal : Speed
Keep Hierarchy : NO

Design Statistics
IOs : 857

Cell Usage :
BELS : 30644
GND : 1

```
#      LUT1                : 96
#      LUT2                : 4043
#      LUT3                : 1373
#      LUT4                : 4169
#      LUT5                : 832
#      LUT6                : 7674
#      MUXCY               : 6673
#      VCC                 : 1
#      XORCY               : 5782
# FlipFlops/Latches        : 5487
#      FDE                 : 268
#      FDRE                : 5219
# Shift Registers          : 268
#      SRLC32E             : 268
# Clock Buffers            : 2
#      BUFGP               : 2
# IO Buffers               : 855
#      IBUF                : 282
#      OBUF                : 573
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 5vlx330ff1760-2


Slice Logic Utilization:
 Number of Slice Registers:          5487  out of  207360     2%
 Number of Slice LUTs:              18455  out of  207360     8%
    Number used as Logic:           18187  out of  207360     8%
    Number used as Memory:            268  out of  54720     0%
      Number used as SRL:             268

Slice Logic Distribution:
 Number of Bit Slices used:         20163
   Number with an unused Flip Flop   14676  out of  20163    72%
   Number with an unused LUT:         1708  out of  20163     8%
   Number of fully used Bit Slices:   3779  out of  20163    18%

IO Utilization:
 Number of IOs:                   857
 Number of bonded IOBs:               857  out of   1200    71%

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:              2  out of     32     6%

---------------------------
Partition Resource Summary:
---------------------------

  No Partitions were found in this design.

---------------------------
```

```
=========================================================================
TIMING REPORT

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
      FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
      GENERATED AFTER PLACE-and-ROUTE.

Clock Information:
------------------
-----------------------------------+------------------------+-------+
Clock Signal                       | Clock buffer(FF name)  | Load  |
-----------------------------------+------------------------+-------+
clk                                | BUFGP                  | 5755  |
-----------------------------------+------------------------+-------+

Asynchronous Control Signals Information:
----------------------------------------
No asynchronous control signals found in this design

Timing Summary:
---------------
Speed Grade: -2

   Minimum period: 5.024ns (Maximum Frequency: 199.027MHz)
   Minimum input arrival time before clock: 6.407ns
   Maximum output required time after clock: 2.826ns
   Maximum combinational path delay: No path found

Timing Detail:
--------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default period analysis for Clock 'clk'
  Clock period: 5.024ns (frequency: 199.027MHz)
  Total number of paths / destination ports: 2347882 / 3579
-------------------------------------------------------------------------
Delay:               5.024ns (Levels of Logic = 16)
  Source:            PU_2/B4x4_00_B/PSAD_0/AD2/C_Registered_0 (FF)
  Destination:       PU_1/B4x4_00_B/PSAD_0/PSAD_0_8 (FF)
  Source Clock:      clk rising
  Destination Clock: clk rising

  Data Path: PU_2/B4x4_00_B/PSAD_0/AD2/C_Registered_0 to PU_1/B4x4_00_B/PSAD_0/PSAD_0_8
                            Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDRE:C->Q            18   0.396   0.491  PU_2/B4x4_00_B/PSAD_0/AD2/C_Registered_0
(PU_2/B4x4_00_B/PSAD_0/AD2/C_Registered_0)
     LUT2:I1->O            3   0.086   0.000  PU_1/B4x4_00_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_lut<0>
(PU_1/B4x4_00_B/PSAD_0/AD3/Nine_Bit_Sum<0>)
```

```
    MUXCY:S->O            1   0.305   0.000  PU_1/B4x4_00_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<0>
(PU_1/B4x4_00_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<0>)
    MUXCY:CI->O           1   0.023   0.000  PU_1/B4x4_00_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<1>
(PU_1/B4x4_00_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<1>)
    MUXCY:CI->O           1   0.023   0.000  PU_1/B4x4_00_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<2>
(PU_1/B4x4_00_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<2>)
    MUXCY:CI->O           1   0.023   0.000  PU_1/B4x4_00_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<3>
(PU_1/B4x4_00_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<3>)
    MUXCY:CI->O           1   0.023   0.000  PU_1/B4x4_00_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<4>
(PU_1/B4x4_00_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<4>)
    MUXCY:CI->O           1   0.023   0.000  PU_1/B4x4_00_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<5>
(PU_1/B4x4_00_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<5>)
    MUXCY:CI->O           1   0.023   0.000  PU_1/B4x4_00_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<6>
(PU_1/B4x4_00_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<6>)
    MUXCY:CI->O           9   0.222   0.938  PU_1/B4x4_00_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<7>
(PU_1/B4x4_00_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<7>)
    LUT6:I0->O            1   0.086   0.487  PU_1/B4x4_00_B/PSAD_0/AD3_Out<5>1
(PU_1/B4x4_00_B/PSAD_0/AD3_Out<5>)
    LUT2:I0->O            1   0.086   0.000  PU_1/B4x4_00_B/PSAD_0/Madd_Nine_Bit_Sum_2_lut<5>
(PU_1/B4x4_00_B/PSAD_0/N50)
    MUXCY:S->O            1   0.305   0.000  PU_1/B4x4_00_B/PSAD_0/Madd_Nine_Bit_Sum_2_cy<5>
(PU_1/B4x4_00_B/PSAD_0/Madd_Nine_Bit_Sum_2_cy<5>)
    XORCY:CI->O           3   0.300   0.496  PU_1/B4x4_00_B/PSAD_0/Madd_Nine_Bit_Sum_2_xor<6>
(PU_1/B4x4_00_B/PSAD_0/Nine_Bit_Sum_2<6>)
    LUT6:I4->O            1   0.086   0.000  PU_1/B4x4_00_B/PSAD_0/Madd_PSAD_0_add0000_Madd_lut<7>
(PU_1/B4x4_00_B/PSAD_0/N59)
    MUXCY:S->O            1   0.305   0.000  PU_1/B4x4_00_B/PSAD_0/Madd_PSAD_0_add0000_Madd_cy<7>
(PU_1/B4x4_00_B/PSAD_0/Madd_PSAD_0_add0000_Madd_cy<7>)
    XORCY:CI->O           1   0.300   0.000  PU_1/B4x4_00_B/PSAD_0/Madd_PSAD_0_add0000_Madd_xor<8>
(PU_1/B4x4_00_B/PSAD_0/PSAD_0_add0000<8>)
    FDRE:D                   -0.022          PU_1/B4x4_00_B/PSAD_0/PSAD_0_8
    ----------------------------------------
    Total                    5.024ns (2.613ns logic, 2.411ns route)
                             (52.0% logic, 48.0% route)

==========================================================================
Timing constraint: Default OFFSET IN BEFORE for Clock 'clk'
  Total number of paths / destination ports: 24529950 / 14622
--------------------------------------------------------------------------
Offset:              6.407ns (Levels of Logic = 18)
  Source:            bus_Select<2> (PAD)
  Destination:       PU_2/B4x4_10_B/PSAD_0/PSAD_0_8 (FF)
  Destination Clock: clk rising

  Data Path: bus_Select<2> to PU_2/B4x4_10_B/PSAD_0/PSAD_0_8
                            Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O          4071   0.694   0.844  bus_Select_2_IBUF (bus_Select_2_IBUF)
     LUT4:I0->O          353   0.086   0.645  PU_2/Mrom_bus_Select_rom0000121
(PU_1/Mrom_bus_Select_rom000011)
    LUT4:I2->O            8   0.086   0.000  PU_1/B4x4_10_B/PSAD_0/AD4/Madd_Nine_Bit_Sum_lut<0>
(PU_1/B4x4_10_B/PSAD_0/AD4/Nine_Bit_Sum<0>)
```

```
    MUXCY:S->O            1   0.305   0.000  PU_2/B4x4_10_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<0>
(PU_2/B4x4_10_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<0>)
    MUXCY:CI->O           1   0.023   0.000  PU_2/B4x4_10_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<1>
(PU_2/B4x4_10_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<1>)
    MUXCY:CI->O           1   0.023   0.000  PU_2/B4x4_10_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<2>
(PU_2/B4x4_10_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<2>)
    MUXCY:CI->O           1   0.023   0.000  PU_2/B4x4_10_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<3>
(PU_2/B4x4_10_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<3>)
    MUXCY:CI->O           1   0.023   0.000  PU_2/B4x4_10_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<4>
(PU_2/B4x4_10_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<4>)
    MUXCY:CI->O           1   0.023   0.000  PU_2/B4x4_10_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<5>
(PU_2/B4x4_10_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<5>)
    MUXCY:CI->O           1   0.023   0.000  PU_2/B4x4_10_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<6>
(PU_2/B4x4_10_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<6>)
    MUXCY:CI->O           9   0.222   0.938  PU_2/B4x4_10_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<7>
(PU_2/B4x4_10_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<7>)
    LUT6:I0->O            1   0.086   0.487  PU_2/B4x4_10_B/PSAD_0/AD3_Out<5>1
(PU_2/B4x4_10_B/PSAD_0/AD3_Out<5>)
    LUT2:I0->O            1   0.086   0.000  PU_2/B4x4_10_B/PSAD_0/Madd_Nine_Bit_Sum_2_lut<5>
(PU_2/B4x4_10_B/PSAD_0/N50)
    MUXCY:S->O            1   0.305   0.000  PU_2/B4x4_10_B/PSAD_0/Madd_Nine_Bit_Sum_2_cy<5>
(PU_2/B4x4_10_B/PSAD_0/Madd_Nine_Bit_Sum_2_cy<5>)
    XORCY:CI->O           3   0.300   0.496  PU_2/B4x4_10_B/PSAD_0/Madd_Nine_Bit_Sum_2_xor<6>
(PU_2/B4x4_10_B/PSAD_0/Nine_Bit_Sum_2<6>)
    LUT6:I4->O            1   0.086   0.000  PU_2/B4x4_10_B/PSAD_0/Madd_PSAD_0_add0000_Madd_lut<7>
(PU_2/B4x4_10_B/PSAD_0/N59)
    MUXCY:S->O            1   0.305   0.000  PU_2/B4x4_10_B/PSAD_0/Madd_PSAD_0_add0000_Madd_cy<7>
(PU_2/B4x4_10_B/PSAD_0/Madd_PSAD_0_add0000_Madd_cy<7>)
    XORCY:CI->O           1   0.300   0.000  PU_2/B4x4_10_B/PSAD_0/Madd_PSAD_0_add0000_Madd_xor<8>
(PU_2/B4x4_10_B/PSAD_0/PSAD_0_add0000<8>)
    FDRE:D                -0.022          PU_2/B4x4_10_B/PSAD_0/PSAD_0_8
    ----------------------------------------
    Total                 6.407ns (2.997ns logic, 3.409ns route)
                             (46.8% logic, 53.2% route)

=========================================================================
Timing constraint: Default OFFSET OUT AFTER for Clock 'clk'
  Total number of paths / destination ports: 573 / 573
-------------------------------------------------------------------------
Offset:              2.826ns (Levels of Logic = 1)
  Source:            B4x8_00_out_13 (FF)
  Destination:       B4x8_00_out<13> (PAD)
  Source Clock:      clk rising

  Data Path: B4x8_00_out_13 to B4x8_00_out<13>
                          Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
    FDRE:C->Q             1   0.396   0.286  B4x8_00_out_13 (B4x8_00_out_13)
    OBUF:I->O                 2.144          B4x8_00_out_13_OBUF (B4x8_00_out<13>)
    ----------------------------------------
    Total                 2.826ns (2.540ns logic, 0.286ns route)
                             (89.9% logic, 10.1% route)
```

```
================================================================================
CPU : 502.05 / 502.28 s | Elapsed : 502.00 / 502.00 s

-->

Total memory usage is 662572 kilobytes

Number of errors   :    0 (   0 filtered)
Number of warnings :    0 (   0 filtered)
Number of infos    : 1920 (   0 filtered)
```

Third design results with 4 PPUs utilized:

```
Release 9.2i - xst J.36
Copyright (c) 1995-2007 Xilinx, Inc.  All rights reserved.
--> Parameter TMPDIR set to ./xst/projnav.tmp
CPU : 0.00 / 0.89 s | Elapsed : 0.00 / 1.00 s

--> Parameter xsthdpdir set to ./xst
CPU : 0.00 / 0.91 s | Elapsed : 0.00 / 1.00 s

--> Reading design: CU_4.prj

=========================================================================
*                Synthesis Options Summary                *
=========================================================================
---- Source Parameters
Input File Name                    : "CU_4.prj"
Input Format                       : mixed
Ignore Synthesis Constraint File   : NO

---- Target Parameters
Output File Name                   : "CU_4"
Output Format                      : NGC
Target Device                      : xc5vlx330-2-ff1760

---- Source Options
Top Module Name                    : CU_4
Automatic FSM Extraction           : YES
FSM Encoding Algorithm             : Auto
Safe Implementation                : No
FSM Style                          : lut
RAM Extraction                     : Yes
RAM Style                          : Auto
ROM Extraction                     : Yes
Mux Style                          : Auto
Decoder Extraction                 : YES
Priority Encoder Extraction        : YES
Shift Register Extraction          : YES
Logical Shifter Extraction         : YES
XOR Collapsing                     : YES
ROM Style                          : Auto
Mux Extraction                     : YES
Resource Sharing                   : YES
Asynchronous To Synchronous        : NO
Use DSP Block                      : auto
Automatic Register Balancing       : No

---- Target Options
Add IO Buffers                     : YES
Global Maximum Fanout              : 100000
Add Generic Clock Buffer(BUFG)     : 32
Register Duplication               : YES
Slice Packing                      : YES
```

```
Optimize Instantiated Primitives   : NO
Use Clock Enable                   : Auto
Use Synchronous Set                : Auto
Use Synchronous Reset              : Auto
Pack IO Registers into IOBs        : auto
Equivalent register Removal        : YES

---- General Options
Optimization Goal                  : Speed
Optimization Effort                : 1
Power Reduction                    : NO
Library Search Order               : CU_4.lso
Keep Hierarchy                     : NO
RTL Output                         : Yes
Global Optimization                : AllClockNets
Read Cores                         : YES
Write Timing Constraints           : NO
Cross Clock Analysis               : NO
Hierarchy Separator                : /
Bus Delimiter                      : <>
Case Specifier                     : maintain
Slice Utilization Ratio            : 100
BRAM Utilization Ratio             : 100
DSP48 Utilization Ratio            : 100
Verilog 2001                       : YES
Auto BRAM Packing                  : NO
Slice Utilization Ratio Delta      : 5

=========================================================================


=========================================================================
*                     HDL Compilation                   *
=========================================================================

=========================================================================
*                Design Hierarchy Analysis              *
=========================================================================
Analyzing hierarchy for module <CU_4> in library <work>.

Analyzing hierarchy for module <main_data_path> in library <work>.

Analyzing hierarchy for module <four_by_four_block> in library <work>.

Analyzing hierarchy for module <four_stage_delay_line> in library <work>.

Analyzing hierarchy for module <eight_stage_delay_line> in library <work>.

Analyzing hierarchy for module <Row_Adder_0> in library <work>.

Analyzing hierarchy for module <Row_Adder_1_CSA> in library <work>.

Analyzing hierarchy for module <Row_Adder_2_CSA> in library <work>.
```

Analyzing hierarchy for module <Row_Adder_3_CSA> in library <work>.

Analyzing hierarchy for module <PE_AD> in library <work>.

Analyzing hierarchy for module <PE_AD_CSA> in library <work>.

Analyzing hierarchy for module <CSA_8> in library <work>.

Analyzing hierarchy for module <CSA_9> in library <work>.

Analyzing hierarchy for module <CSA_Full_Adder> in library <work>.

Unit <CU_4> synthesized.

```
=========================================================================
*                Advanced HDL Synthesis                 *
=========================================================================
```

Loading device for application Rf_Device from file '5vlx330.nph' in environment C:\Xilinx92i.

```
=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
# ROMs                          : 4
 16x15-bit ROM                  : 4
# Adders/Subtractors            : 1764
 10-bit adder                   : 64
 11-bit adder carry in          : 64
 12-bit adder carry in          : 64
 13-bit adder carry in          : 64
 14-bit adder                   : 64
 15-bit adder                   : 16
 16-bit adder                   : 16
 17-bit adder                   : 4
 8-bit adder                    : 256
 9-bit adder                    : 1152
# Registers                     : 16939
 Flip-Flops                     : 16939
# Comparators                   : 123
 13-bit comparator less         : 48
 14-bit comparator less         : 48
 15-bit comparator less         : 12
 16-bit comparator less         : 12
 17-bit comparator less         : 3
# Xors                          : 10624
 1-bit xor3                     : 9600
 8-bit xor2                     : 1024

=========================================================================

=========================================================================
*                Low Level Synthesis                    *
=========================================================================
```

```
Optimizing unit <CU_4> ...

Optimizing unit <four_stage_delay_line> ...

Optimizing unit <eight_stage_delay_line> ...

Optimizing unit <PE_AD_CSA> ...

Optimizing unit <Row_Adder_0> ...

Optimizing unit <Row_Adder_1_CSA> ...

Optimizing unit <Row_Adder_2_CSA> ...

Optimizing unit <Row_Adder_3_CSA> ...

Optimizing unit <four_by_four_block> ...

Optimizing unit <main_data_path> ...

Mapping all equations...
Building and optimizing final netlist ...

Final Macro Processing ...

Processing Unit <CU_4> :
Unit <CU_4> processed.

=========================================================================
Final Register Report

Macro Statistics
# Registers                          : 9105
 Flip-Flops                         : 9105
# Shift Registers                     : 536
 3-bit shift register                : 416
 7-bit shift register                : 120

=========================================================================

=========================================================================
*                     Partition Report                       *
=========================================================================

Partition Implementation Status
-------------------------------

  No Partitions were found in this design.

-------------------------------

=========================================================================
*                     Final Report                           *
```

```
=========================================================================
Final Results
RTL Top Level Output File Name     : CU_4.ngr
Top Level Output File Name         : CU_4
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 889

Cell Usage :
# BELS                             : 62448
#      GND                         : 1
#      LUT1                        : 192
#      LUT2                        : 8113
#      LUT3                        : 3319
#      LUT4                        : 8691
#      LUT5                        : 1648
#      LUT6                        : 15276
#      MUXCY                       : 13643
#      VCC                         : 1
#      XORCY                       : 11564
# FlipFlops/Latches                : 9641
#      FDE                         : 536
#      FDRE                        : 9105
# Shift Registers                  : 536
#      SRLC32E                     : 536
# Clock Buffers                    : 2
#      BUFGP                       : 2
# IO Buffers                       : 887
#      IBUF                        : 314
#      OBUF                        : 573
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 5vlx330ff1760-2

Slice Logic Utilization:
 Number of Slice Registers:          9641  out of  207360     4%
 Number of Slice LUTs:              37775  out of  207360    18%
    Number used as Logic:           37239  out of  207360    17%
    Number used as Memory:            536  out of  54720     0%
      Number used as SRL:             536

Slice Logic Distribution:
 Number of Bit Slices used:         39997
   Number with an unused Flip Flop   30356  out of  39997    75%
   Number with an unused LUT:         2222  out of  39997     5%
   Number of fully used Bit Slices:   7419  out of  39997    18%

IO Utilization:
```

```
 Number of IOs:                   889
 Number of bonded IOBs:           889  out of   1200    74%

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:          2  out of     32     6%

---------------------------
Partition Resource Summary:
---------------------------

  No Partitions were found in this design.

---------------------------


=========================================================================
TIMING REPORT

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
      FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
      GENERATED AFTER PLACE-and-ROUTE.

Clock Information:
------------------
-----------------------------------+------------------------+-------+
Clock Signal                       | Clock buffer(FF name)  | Load  |
-----------------------------------+------------------------+-------+
clk                                | BUFGP                  | 10177 |
-----------------------------------+------------------------+-------+

Asynchronous Control Signals Information:
----------------------------------------
No asynchronous control signals found in this design

Timing Summary:
---------------
Speed Grade: -2

   Minimum period: 5.041ns (Maximum Frequency: 198.356MHz)
   Minimum input arrival time before clock: 6.430ns
   Maximum output required time after clock: 2.826ns
   Maximum combinational path delay: No path found

Timing Detail:
--------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default period analysis for Clock 'clk'
  Clock period: 5.041ns (frequency: 198.356MHz)
  Total number of paths / destination ports: 4729122 / 7731
-------------------------------------------------------------------------
Delay:               5.041ns (Levels of Logic = 16)
  Source:            PU_4/B4x4_03_B/PSAD_0/AD2/C_Registered_0 (FF)
```

```
  Destination:       PU_3/B4x4_03_B/PSAD_0/PSAD_0_8 (FF)
  Source Clock:      clk rising
  Destination Clock: clk rising

  Data Path: PU_4/B4x4_03_B/PSAD_0/AD2/C_Registered_0 to PU_3/B4x4_03_B/PSAD_0/PSAD_0_8
                            Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDRE:C->Q            22   0.396   0.508  PU_4/B4x4_03_B/PSAD_0/AD2/C_Registered_0
(PU_4/B4x4_03_B/PSAD_0/AD2/C_Registered_0)
     LUT2:I1->O            5   0.086   0.000  PU_3/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_lut<0>
(PU_2/B4x4_03_B/PSAD_0/AD4/Nine_Bit_Sum<0>)
     MUXCY:S->O            1   0.305   0.000  PU_3/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<0>
(PU_3/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<0>)
     MUXCY:CI->O           1   0.023   0.000  PU_3/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<1>
(PU_3/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<1>)
     MUXCY:CI->O           1   0.023   0.000  PU_3/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<2>
(PU_3/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<2>)
     MUXCY:CI->O           1   0.023   0.000  PU_3/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<3>
(PU_3/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<3>)
     MUXCY:CI->O           1   0.023   0.000  PU_3/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<4>
(PU_3/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<4>)
     MUXCY:CI->O           1   0.023   0.000  PU_3/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<5>
(PU_3/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<5>)
     MUXCY:CI->O           1   0.023   0.000  PU_3/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<6>
(PU_3/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<6>)
     MUXCY:CI->O           9   0.222   0.938  PU_3/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<7>
(PU_3/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<7>)
     LUT6:I0->O            1   0.086   0.487  PU_3/B4x4_03_B/PSAD_0/AD3_Out<5>1
(PU_3/B4x4_03_B/PSAD_0/AD3_Out<5>)
     LUT2:I0->O            1   0.086   0.000  PU_3/B4x4_03_B/PSAD_0/Madd_Nine_Bit_Sum_2_lut<5>
(PU_3/B4x4_03_B/PSAD_0/N50)
     MUXCY:S->O            1   0.305   0.000  PU_3/B4x4_03_B/PSAD_0/Madd_Nine_Bit_Sum_2_cy<5>
(PU_3/B4x4_03_B/PSAD_0/Madd_Nine_Bit_Sum_2_cy<5>)
     XORCY:CI->O           3   0.300   0.496  PU_3/B4x4_03_B/PSAD_0/Madd_Nine_Bit_Sum_2_xor<6>
(PU_3/B4x4_03_B/PSAD_0/Nine_Bit_Sum_2<6>)
     LUT6:I4->O            1   0.086   0.000  PU_3/B4x4_03_B/PSAD_0/Madd_PSAD_0_add0000_Madd_lut<7>
(PU_3/B4x4_03_B/PSAD_0/N59)
     MUXCY:S->O            1   0.305   0.000  PU_3/B4x4_03_B/PSAD_0/Madd_PSAD_0_add0000_Madd_cy<7>
(PU_3/B4x4_03_B/PSAD_0/Madd_PSAD_0_add0000_Madd_cy<7>)
     XORCY:CI->O           1   0.300   0.000  PU_3/B4x4_03_B/PSAD_0/Madd_PSAD_0_add0000_Madd_xor<8>
(PU_3/B4x4_03_B/PSAD_0/PSAD_0_add0000<8>)
     FDRE:D                    -0.022          PU_3/B4x4_03_B/PSAD_0/PSAD_0_8
    ----------------------------------------
    Total                      5.041ns (2.613ns logic, 2.428ns route)
                               (51.8% logic, 48.2% route)

=========================================================================
Timing constraint: Default OFFSET IN BEFORE for Clock 'clk'
  Total number of paths / destination ports: 49047704 / 24672
-------------------------------------------------------------------------
Offset:              6.430ns (Levels of Logic = 18)
  Source:            bus_Select<2> (PAD)
  Destination:       PU_4/B4x4_10_B/PSAD_0/PSAD_0_8 (FF)
```

```
  Destination Clock: clk rising

  Data Path: bus_Select<2> to PU_4/B4x4_10_B/PSAD_0/PSAD_0_8
                            Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O          8071   0.694   0.844  bus_Select_2_IBUF (bus_Select_2_IBUF)
     LUT4:I0->O          691   0.086   0.668  PU_4/Mrom_bus_Select_rom0000121
(PU_1/Mrom_bus_Select_rom000011)
     LUT4:I2->O           19   0.086   0.000  PU_4/B4x4_10_B/PSAD_0/AD4/Madd_Nine_Bit_Sum_lut<0>
(PU_1/B4x4_11_B/PSAD_0/AD3/Nine_Bit_Sum<0>)
     MUXCY:S->O            1   0.305   0.000  PU_1/B4x4_11_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<0>
(PU_1/B4x4_11_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<0>)
     MUXCY:CI->O           1   0.023   0.000  PU_1/B4x4_11_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<1>
(PU_1/B4x4_11_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<1>)
     MUXCY:CI->O           1   0.023   0.000  PU_1/B4x4_11_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<2>
(PU_1/B4x4_11_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<2>)
     MUXCY:CI->O           1   0.023   0.000  PU_1/B4x4_11_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<3>
(PU_1/B4x4_11_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<3>)
     MUXCY:CI->O           1   0.023   0.000  PU_1/B4x4_11_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<4>
(PU_1/B4x4_11_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<4>)
     MUXCY:CI->O           1   0.023   0.000  PU_1/B4x4_11_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<5>
(PU_1/B4x4_11_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<5>)
     MUXCY:CI->O           1   0.023   0.000  PU_1/B4x4_11_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<6>
(PU_1/B4x4_11_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<6>)
     MUXCY:CI->O           9   0.222   0.938  PU_1/B4x4_11_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<7>
(PU_1/B4x4_11_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<7>)
     LUT6:I0->O            1   0.086   0.487  PU_1/B4x4_11_B/PSAD_0/AD3_Out<5>1
(PU_1/B4x4_11_B/PSAD_0/AD3_Out<5>)
     LUT2:I0->O            1   0.086   0.000  PU_1/B4x4_11_B/PSAD_0/Madd_Nine_Bit_Sum_2_lut<5>
(PU_1/B4x4_11_B/PSAD_0/N50)
     MUXCY:S->O            1   0.305   0.000  PU_1/B4x4_11_B/PSAD_0/Madd_Nine_Bit_Sum_2_cy<5>
(PU_1/B4x4_11_B/PSAD_0/Madd_Nine_Bit_Sum_2_cy<5>)
     XORCY:CI->O           3   0.300   0.496  PU_1/B4x4_11_B/PSAD_0/Madd_Nine_Bit_Sum_2_xor<6>
(PU_1/B4x4_11_B/PSAD_0/Nine_Bit_Sum_2<6>)
     LUT6:I4->O            1   0.086   0.000  PU_1/B4x4_11_B/PSAD_0/Madd_PSAD_0_add0000_Madd_lut<7>
(PU_1/B4x4_11_B/PSAD_0/N59)
     MUXCY:S->O            1   0.305   0.000  PU_1/B4x4_11_B/PSAD_0/Madd_PSAD_0_add0000_Madd_cy<7>
(PU_1/B4x4_11_B/PSAD_0/Madd_PSAD_0_add0000_Madd_cy<7>)
     XORCY:CI->O           1   0.300   0.000  PU_1/B4x4_11_B/PSAD_0/Madd_PSAD_0_add0000_Madd_xor<8>
(PU_1/B4x4_11_B/PSAD_0/PSAD_0_add0000<8>)
     FDRE:D                    -0.022          PU_1/B4x4_11_B/PSAD_0/PSAD_0_8
    ----------------------------------------
    Total                      6.430ns (2.997ns logic, 3.433ns route)
                               (46.6% logic, 53.4% route)

=========================================================================
Timing constraint: Default OFFSET OUT AFTER for Clock 'clk'
  Total number of paths / destination ports: 573 / 573
-------------------------------------------------------------------------
Offset:              2.826ns (Levels of Logic = 1)
  Source:            B4x8_00_out_13 (FF)
  Destination:       B4x8_00_out<13> (PAD)
  Source Clock:      clk rising
```

```
  Data Path: B4x8_00_out_13 to B4x8_00_out<13>
                            Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDRE:C->Q             1   0.396   0.286  B4x8_00_out_13 (B4x8_00_out_13)
     OBUF:I->O                 2.144          B4x8_00_out_13_OBUF (B4x8_00_out<13>)
    ----------------------------------------
    Total                    2.826ns (2.540ns logic, 0.286ns route)
                               (89.9% logic, 10.1% route)

========================================================================
CPU : 1113.08 / 1114.03 s | Elapsed : 1113.00 / 1114.00 s

-->

Total memory usage is 820524 kilobytes

Number of errors   :    0 (   0 filtered)
Number of warnings :    0 (   0 filtered)
Number of infos    : 2176 (   0 filtered)
```

Fourth design results with 8 PPUs utilized:

```
Release 9.2i - xst J.36
Copyright (c) 1995-2007 Xilinx, Inc.  All rights reserved.
--> Parameter TMPDIR set to ./xst/projnav.tmp
CPU : 0.00 / 1.44 s | Elapsed : 0.00 / 2.00 s

--> Parameter xsthdpdir set to ./xst
CPU : 0.00 / 1.44 s | Elapsed : 0.00 / 2.00 s

--> Reading design: CU_8.prj
=========================================================================
*                Synthesis Options Summary                *
=========================================================================
---- Source Parameters
Input File Name                    : "CU_8.prj"
Input Format                       : mixed
Ignore Synthesis Constraint File   : NO

---- Target Parameters
Output File Name                   : "CU_8"
Output Format                      : NGC
Target Device                      : xc5vlx330-2-ff1760

---- Source Options
Top Module Name                    : CU_8
Automatic FSM Extraction           : YES
FSM Encoding Algorithm             : Auto
Safe Implementation                : No
FSM Style                          : lut
RAM Extraction                     : Yes
RAM Style                          : Auto
ROM Extraction                     : Yes
Mux Style                          : Auto
Decoder Extraction                 : YES
Priority Encoder Extraction        : YES
Shift Register Extraction          : YES
Logical Shifter Extraction         : YES
XOR Collapsing                     : YES
ROM Style                          : Auto
Mux Extraction                     : YES
Resource Sharing                   : YES
Asynchronous To Synchronous        : NO
Use DSP Block                      : auto
Automatic Register Balancing       : No

---- Target Options
Add IO Buffers                     : YES
Global Maximum Fanout              : 100000
Add Generic Clock Buffer(BUFG)     : 32
Register Duplication               : YES
Slice Packing                      : YES
Optimize Instantiated Primitives   : NO
```

```
Use Clock Enable                : Auto
Use Synchronous Set             : Auto
Use Synchronous Reset           : Auto
Pack IO Registers into IOBs     : auto
Equivalent register Removal     : YES

---- General Options
Optimization Goal               : Speed
Optimization Effort             : 1
Power Reduction                 : NO
Library Search Order            : CU_8.lso
Keep Hierarchy                  : NO
RTL Output                      : Yes
Global Optimization             : AllClockNets
Read Cores                      : YES
Write Timing Constraints        : NO
Cross Clock Analysis            : NO
Hierarchy Separator             : /
Bus Delimiter                   : <>
Case Specifier                  : maintain
Slice Utilization Ratio         : 100
BRAM Utilization Ratio          : 100
DSP48 Utilization Ratio         : 100
Verilog 2001                    : YES
Auto BRAM Packing               : NO
Slice Utilization Ratio Delta   : 5

=========================================================================


=========================================================================
*                     HDL Compilation                   *
=========================================================================

=========================================================================
*                Design Hierarchy Analysis              *
=========================================================================
Analyzing hierarchy for module <CU_8> in library <work>.

Analyzing hierarchy for module <main_data_path> in library <work>.

Analyzing hierarchy for module <four_by_four_block> in library <work>.

Analyzing hierarchy for module <four_stage_delay_line> in library <work>.

Analyzing hierarchy for module <eight_stage_delay_line> in library <work>.

Analyzing hierarchy for module <Row_Adder_0> in library <work>.

Analyzing hierarchy for module <Row_Adder_1_CSA> in library <work>.

Analyzing hierarchy for module <Row_Adder_2_CSA> in library <work>.

Analyzing hierarchy for module <Row_Adder_3_CSA> in library <work>.
```

Analyzing hierarchy for module <PE_AD> in library <work>.

Analyzing hierarchy for module <PE_AD_CSA> in library <work>.

Analyzing hierarchy for module <CSA_8> in library <work>.

Analyzing hierarchy for module <CSA_9> in library <work>.

Analyzing hierarchy for module <CSA_Full_Adder> in library <work>.

Unit <CU_8> synthesized.

===

===
* Advanced HDL Synthesis *
===

Loading device for application Rf_Device from file '5vlx330.nph' in environment C:\Xilinx92i.

===
Advanced HDL Synthesis Report

Macro Statistics
ROMs : 8
 16x15-bit ROM : 8
Adders/Subtractors : 3528
 10-bit adder : 128
 11-bit adder carry in : 128
 12-bit adder carry in : 128
 13-bit adder carry in : 128
 14-bit adder : 128
 15-bit adder : 32
 16-bit adder : 32
 17-bit adder : 8
 8-bit adder : 512
 9-bit adder : 2304
Registers : 34451
 Flip-Flops : 34451
Comparators : 287
 13-bit comparator less : 112
 14-bit comparator less : 112
 15-bit comparator less : 28
 16-bit comparator less : 28
 17-bit comparator less : 7
Xors : 21248
 1-bit xor3 : 19200
 8-bit xor2 : 2048

===

===
* Low Level Synthesis *

```
=========================================================================

Optimizing unit <CU_8> ...

Optimizing unit <four_stage_delay_line> ...

Optimizing unit <eight_stage_delay_line> ...

Optimizing unit <PE_AD_CSA> ...

Optimizing unit <Row_Adder_0> ...

Optimizing unit <Row_Adder_1_CSA> ...

Optimizing unit <Row_Adder_2_CSA> ...

Optimizing unit <Row_Adder_3_CSA> ...

Optimizing unit <four_by_four_block> ...

Optimizing unit <main_data_path> ...

Mapping all equations...
Building and optimizing final netlist ...

Final Macro Processing ...

Processing Unit <CU_8> :
Unit <CU_8> processed.

=========================================================================
Final Register Report

Macro Statistics
# Registers                          : 16895
 Flip-Flops                         : 16895
# Shift Registers                     : 1072
 3-bit shift register               : 832
 7-bit shift register               : 240

=========================================================================

=========================================================================
*                  Partition Report                       *
=========================================================================

Partition Implementation Status
-------------------------------

  No Partitions were found in this design.

-------------------------------

=========================================================================
```

```
*                      Final Report                      *
============================================================================
Final Results
RTL Top Level Output File Name     : CU_8.ngr
Top Level Output File Name         : CU_8
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 953

Cell Usage :
# BELS                             : 126069
#      GND                         : 1
#      LUT1                        : 384
#      LUT2                        : 16254
#      LUT3                        : 7211
#      LUT4                        : 17771
#      LUT5                        : 3280
#      LUT6                        : 30456
#      MUXCY                       : 27583
#      VCC                         : 1
#      XORCY                       : 23128
# FlipFlops/Latches                : 17967
#      FDE                         : 1089
#      FDRE                        : 16878
# Shift Registers                  : 1072
#      SRLC32E                     : 1072
# Clock Buffers                    : 2
#      BUFG                        : 1
#      BUFGP                       : 1
# IO Buffers                       : 952
#      IBUF                        : 379
#      OBUF                        : 573
============================================================================

Device utilization summary:
---------------------------

Selected Device : 5vlx330ff1760-2


Slice Logic Utilization:
 Number of Slice Registers:         17967  out of  207360     8%
 Number of Slice LUTs:              76428  out of  207360    36%
    Number used as Logic:           75356  out of  207360    36%
    Number used as Memory:           1072  out of  54720     1%
       Number used as SRL:           1072

Slice Logic Distribution:
 Number of Bit Slices used:         79690
   Number with an unused Flip Flop   61723  out of  79690    77%
   Number with an unused LUT:         3262  out of  79690     4%
```

```
   Number of fully used Bit Slices:  14705  out of  79690    18%

IO Utilization:
 Number of IOs:                 953
 Number of bonded IOBs:             953  out of   1200    79%

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:            2  out of     32     6%

---------------------------
Partition Resource Summary:
---------------------------

  No Partitions were found in this design.

---------------------------


=========================================================================
TIMING REPORT

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
      FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
      GENERATED AFTER PLACE-and-ROUTE.

Clock Information:
------------------
-----------------------------------+------------------------+-------+
Clock Signal                       | Clock buffer(FF name)  | Load  |
-----------------------------------+------------------------+-------+
clk                                | BUFGP                  | 19039 |
-----------------------------------+------------------------+-------+

Asynchronous Control Signals Information:
----------------------------------------
No asynchronous control signals found in this design

Timing Summary:
---------------
Speed Grade: -2

   Minimum period: 5.041ns (Maximum Frequency: 198.356MHz)
   Minimum input arrival time before clock: 6.430ns
   Maximum output required time after clock: 2.826ns
   Maximum combinational path delay: No path found

Timing Detail:
--------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default period analysis for Clock 'clk'
  Clock period: 5.041ns (frequency: 198.356MHz)
  Total number of paths / destination ports: 9491602 / 16035
```

```
---------------------------------------------------------------------------
Delay:               5.041ns (Levels of Logic = 16)
  Source:            PU_8/B4x4_03_B/PSAD_0/AD2/C_Registered_0 (FF)
  Destination:       PU_7/B4x4_03_B/PSAD_0/PSAD_0_8 (FF)
  Source Clock:      clk rising
  Destination Clock: clk rising

  Data Path: PU_8/B4x4_03_B/PSAD_0/AD2/C_Registered_0 to PU_7/B4x4_03_B/PSAD_0/PSAD_0_8
                            Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDRE:C->Q            22   0.396   0.508  PU_8/B4x4_03_B/PSAD_0/AD2/C_Registered_0
(PU_8/B4x4_03_B/PSAD_0/AD2/C_Registered_0)
     LUT2:I1->O            5   0.086   0.000  PU_7/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_lut<0>
(PU_6/B4x4_03_B/PSAD_0/AD4/Nine_Bit_Sum<0>)
     MUXCY:S->O            1   0.305   0.000  PU_7/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<0>
(PU_7/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<0>)
     MUXCY:CI->O           1   0.023   0.000  PU_7/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<1>
(PU_7/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<1>)
     MUXCY:CI->O           1   0.023   0.000  PU_7/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<2>
(PU_7/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<2>)
     MUXCY:CI->O           1   0.023   0.000  PU_7/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<3>
(PU_7/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<3>)
     MUXCY:CI->O           1   0.023   0.000  PU_7/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<4>
(PU_7/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<4>)
     MUXCY:CI->O           1   0.023   0.000  PU_7/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<5>
(PU_7/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<5>)
     MUXCY:CI->O           1   0.023   0.000  PU_7/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<6>
(PU_7/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<6>)
     MUXCY:CI->O           9   0.222   0.938  PU_7/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<7>
(PU_7/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<7>)
     LUT6:I0->O            1   0.086   0.487  PU_7/B4x4_03_B/PSAD_0/AD3_Out<5>1
(PU_7/B4x4_03_B/PSAD_0/AD3_Out<5>)
     LUT2:I0->O            1   0.086   0.000  PU_7/B4x4_03_B/PSAD_0/Madd_Nine_Bit_Sum_2_lut<5>
(PU_7/B4x4_03_B/PSAD_0/N50)
     MUXCY:S->O            1   0.305   0.000  PU_7/B4x4_03_B/PSAD_0/Madd_Nine_Bit_Sum_2_cy<5>
(PU_7/B4x4_03_B/PSAD_0/Madd_Nine_Bit_Sum_2_cy<5>)
     XORCY:CI->O           3   0.300   0.496  PU_7/B4x4_03_B/PSAD_0/Madd_Nine_Bit_Sum_2_xor<6>
(PU_7/B4x4_03_B/PSAD_0/Nine_Bit_Sum_2<6>)
     LUT6:I4->O            1   0.086   0.000  PU_7/B4x4_03_B/PSAD_0/Madd_PSAD_0_add0000_Madd_lut<7>
(PU_7/B4x4_03_B/PSAD_0/N59)
     MUXCY:S->O            1   0.305   0.000  PU_7/B4x4_03_B/PSAD_0/Madd_PSAD_0_add0000_Madd_cy<7>
(PU_7/B4x4_03_B/PSAD_0/Madd_PSAD_0_add0000_Madd_cy<7>)
     XORCY:CI->O           1   0.300   0.000  PU_7/B4x4_03_B/PSAD_0/Madd_PSAD_0_add0000_Madd_xor<8>
(PU_7/B4x4_03_B/PSAD_0/PSAD_0_add0000<8>)
     FDRE:D                   -0.022          PU_7/B4x4_03_B/PSAD_0/PSAD_0_8
    ----------------------------------------
    Total                      5.041ns (2.613ns logic, 2.428ns route)
                               (51.8% logic, 48.2% route)

=========================================================================
Timing constraint: Default OFFSET IN BEFORE for Clock 'clk'
  Total number of paths / destination ports: 98066460 / 27931
---------------------------------------------------------------------------
```

```
Offset:             6.430ns (Levels of Logic = 18)
  Source:           bus_Select<2> (PAD)
  Destination:      PU_8/B4x4_10_B/PSAD_0/PSAD_0_8 (FF)
  Destination Clock: clk rising

  Data Path: bus_Select<2> to PU_8/B4x4_10_B/PSAD_0/PSAD_0_8
                              Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O          16047   0.694   0.844  bus_Select_2_IBUF (bus_Select_2_IBUF)
     LUT4:I0->O          1379   0.086   0.668  PU_8/Mrom_bus_Select_rom0000121
(PU_1/Mrom_bus_Select_rom000011)
     LUT4:I2->O            23   0.086   0.000  PU_8/B4x4_12_B/PSAD_0/AD4/Madd_Nine_Bit_Sum_lut<0>
(PU_4/B4x4_13_B/PSAD_0/AD4/Nine_Bit_Sum<0>)
     MUXCY:S->O             1   0.305   0.000  PU_5/B4x4_13_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<0>
(PU_5/B4x4_13_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<0>)
     MUXCY:CI->O            1   0.023   0.000  PU_5/B4x4_13_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<1>
(PU_5/B4x4_13_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<1>)
     MUXCY:CI->O            1   0.023   0.000  PU_5/B4x4_13_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<2>
(PU_5/B4x4_13_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<2>)
     MUXCY:CI->O            1   0.023   0.000  PU_5/B4x4_13_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<3>
(PU_5/B4x4_13_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<3>)
     MUXCY:CI->O            1   0.023   0.000  PU_5/B4x4_13_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<4>
(PU_5/B4x4_13_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<4>)
     MUXCY:CI->O            1   0.023   0.000  PU_5/B4x4_13_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<5>
(PU_5/B4x4_13_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<5>)
     MUXCY:CI->O            1   0.023   0.000  PU_5/B4x4_13_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<6>
(PU_5/B4x4_13_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<6>)
     MUXCY:CI->O            9   0.222   0.938  PU_5/B4x4_13_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<7>
(PU_5/B4x4_13_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<7>)
     LUT6:I0->O             1   0.086   0.487  PU_5/B4x4_13_B/PSAD_0/AD3_Out<5>1
(PU_5/B4x4_13_B/PSAD_0/AD3_Out<5>)
     LUT2:I0->O             1   0.086   0.000  PU_5/B4x4_13_B/PSAD_0/Madd_Nine_Bit_Sum_2_lut<5>
(PU_5/B4x4_13_B/PSAD_0/N50)
     MUXCY:S->O             1   0.305   0.000  PU_5/B4x4_13_B/PSAD_0/Madd_Nine_Bit_Sum_2_cy<5>
(PU_5/B4x4_13_B/PSAD_0/Madd_Nine_Bit_Sum_2_cy<5>)
     XORCY:CI->O            3   0.300   0.496  PU_5/B4x4_13_B/PSAD_0/Madd_Nine_Bit_Sum_2_xor<6>
(PU_5/B4x4_13_B/PSAD_0/Nine_Bit_Sum_2<6>)
     LUT6:I4->O             1   0.086   0.000  PU_5/B4x4_13_B/PSAD_0/Madd_PSAD_0_add0000_Madd_lut<7>
(PU_5/B4x4_13_B/PSAD_0/N59)
     MUXCY:S->O             1   0.305   0.000  PU_5/B4x4_13_B/PSAD_0/Madd_PSAD_0_add0000_Madd_cy<7>
(PU_5/B4x4_13_B/PSAD_0/Madd_PSAD_0_add0000_Madd_cy<7>)
     XORCY:CI->O            1   0.300   0.000  PU_5/B4x4_13_B/PSAD_0/Madd_PSAD_0_add0000_Madd_xor<8>
(PU_5/B4x4_13_B/PSAD_0/PSAD_0_add0000<8>)
     FDRE:D                -0.022          PU_5/B4x4_13_B/PSAD_0/PSAD_0_8
    ----------------------------------------
    Total                  6.430ns (2.997ns logic, 3.433ns route)
                              (46.6% logic, 53.4% route)

==========================================================================
Timing constraint: Default OFFSET OUT AFTER for Clock 'clk'
  Total number of paths / destination ports: 573 / 573
--------------------------------------------------------------------------
Offset:             2.826ns (Levels of Logic = 1)
```

```
  Source:          B4x8_00_out_13 (FF)
  Destination:     B4x8_00_out<13> (PAD)
  Source Clock:    clk rising

  Data Path: B4x8_00_out_13 to B4x8_00_out<13>
                          Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDRE:C->Q             1   0.396   0.286  B4x8_00_out_13 (B4x8_00_out_13)
     OBUF:I->O                 2.144          B4x8_00_out_13_OBUF (B4x8_00_out<13>)
    ----------------------------------------
    Total                    2.826ns (2.540ns logic, 0.286ns route)
                             (89.9% logic, 10.1% route)

=========================================================================
CPU : 2240.63 / 2242.13 s | Elapsed : 2241.00 / 2243.00 s

-->

Total memory usage is 1119724 kilobytes

Number of errors   :    0 (   0 filtered)
Number of warnings :    0 (   0 filtered)
Number of infos    : 2688 (   0 filtered)
```

Fifth design results with 16 PPUs utilized:

```
Release 9.2i - xst J.36
Copyright (c) 1995-2007 Xilinx, Inc.  All rights reserved.
--> Parameter TMPDIR set to ./xst/projnav.tmp
CPU : 0.00 / 0.36 s | Elapsed : 0.00 / 0.00 s

--> Parameter xsthdpdir set to ./xst
CPU : 0.00 / 0.36 s | Elapsed : 0.00 / 0.00 s

--> Reading design: CU_16.prj
=========================================================================
*                Synthesis Options Summary                *
=========================================================================
---- Source Parameters
Input File Name                    : "CU_16.prj"
Input Format                       : mixed
Ignore Synthesis Constraint File   : NO

---- Target Parameters
Output File Name                   : "CU_16"
Output Format                      : NGC
Target Device                      : xc5vlx330-2-ff1760

---- Source Options
Top Module Name                    : CU_16
Automatic FSM Extraction           : YES
FSM Encoding Algorithm             : Auto
Safe Implementation                : No
FSM Style                          : lut
RAM Extraction                     : Yes
RAM Style                          : Auto
ROM Extraction                     : Yes
Mux Style                          : Auto
Decoder Extraction                 : YES
Priority Encoder Extraction        : YES
Shift Register Extraction          : YES
Logical Shifter Extraction         : YES
XOR Collapsing                     : YES
ROM Style                          : Auto
Mux Extraction                     : YES
Resource Sharing                   : YES
Asynchronous To Synchronous        : NO
Use DSP Block                      : auto
Automatic Register Balancing       : No

---- Target Options
Add IO Buffers                     : YES
Global Maximum Fanout              : 100000
Add Generic Clock Buffer(BUFG)     : 32
Register Duplication               : YES
Slice Packing                      : YES
Optimize Instantiated Primitives   : NO
```

```
Use Clock Enable                : Auto
Use Synchronous Set             : Auto
Use Synchronous Reset           : Auto
Pack IO Registers into IOBs     : auto
Equivalent register Removal     : YES

---- General Options
Optimization Goal               : Speed
Optimization Effort             : 1
Power Reduction                 : NO
Library Search Order            : CU_16.lso
Keep Hierarchy                  : NO
RTL Output                      : Yes
Global Optimization             : AllClockNets
Read Cores                      : YES
Write Timing Constraints        : NO
Cross Clock Analysis            : NO
Hierarchy Separator             : /
Bus Delimiter                   : <>
Case Specifier                  : maintain
Slice Utilization Ratio         : 100
BRAM Utilization Ratio          : 100
DSP48 Utilization Ratio         : 100
Verilog 2001                    : YES
Auto BRAM Packing               : NO
Slice Utilization Ratio Delta   : 5

=========================================================================


=========================================================================
*                     HDL Compilation                   *
=========================================================================

=========================================================================
*                Design Hierarchy Analysis              *
=========================================================================
Analyzing hierarchy for module <CU_16> in library <work>.

Analyzing hierarchy for module <main_data_path> in library <work>.

Analyzing hierarchy for module <four_by_four_block> in library <work>.

Analyzing hierarchy for module <four_stage_delay_line> in library <work>.

Analyzing hierarchy for module <eight_stage_delay_line> in library <work>.

Analyzing hierarchy for module <Row_Adder_0> in library <work>.

Analyzing hierarchy for module <Row_Adder_1_CSA> in library <work>.

Analyzing hierarchy for module <Row_Adder_2_CSA> in library <work>.

Analyzing hierarchy for module <Row_Adder_3_CSA> in library <work>.
```

Analyzing hierarchy for module <PE_AD> in library <work>.

Analyzing hierarchy for module <PE_AD_CSA> in library <work>.

Analyzing hierarchy for module <CSA_8> in library <work>.

Analyzing hierarchy for module <CSA_9> in library <work>.

Analyzing hierarchy for module <CSA_Full_Adder> in library <work>.

Unit <CU_16> synthesized.

==========

==========
* Advanced HDL Synthesis *
==========

Loading device for application Rf_Device from file '5vlx330.nph' in environment C:\Xilinx92i.

==========
Advanced HDL Synthesis Report

Macro Statistics
ROMs : 16
 16x15-bit ROM : 16
Adders/Subtractors : 7056
 10-bit adder : 256
 11-bit adder carry in : 256
 12-bit adder carry in : 256
 13-bit adder carry in : 256
 14-bit adder : 256
 15-bit adder : 64
 16-bit adder : 64
 17-bit adder : 16
 8-bit adder : 1024
 9-bit adder : 4608
Registers : 69475
 Flip-Flops : 69475
Comparators : 615
 13-bit comparator less : 240
 14-bit comparator less : 240
 15-bit comparator less : 60
 16-bit comparator less : 60
 17-bit comparator less : 15
Xors : 42496
 1-bit xor3 : 38400
 8-bit xor2 : 4096

==========

==========
* Low Level Synthesis *

```
=========================================================================

Optimizing unit <CU_16> ...

Optimizing unit <four_stage_delay_line> ...

Optimizing unit <eight_stage_delay_line> ...

Optimizing unit <PE_AD_CSA> ...

Optimizing unit <Row_Adder_0> ...

Optimizing unit <Row_Adder_1_CSA> ...

Optimizing unit <Row_Adder_2_CSA> ...

Optimizing unit <Row_Adder_3_CSA> ...

Optimizing unit <four_by_four_block> ...

Optimizing unit <main_data_path> ...

Mapping all equations...
Building and optimizing final netlist ...

Final Macro Processing ...

Processing Unit <CU_16> :
Unit <CU_16> processed.

=========================================================================
Final Register Report

Macro Statistics
# Registers                          : 32415
 Flip-Flops                         : 32415
# Shift Registers                     : 2144
 3-bit shift register                : 1664
 7-bit shift register                : 480

=========================================================================

=========================================================================
*                  Partition Report                       *
=========================================================================

Partition Implementation Status
-------------------------------

  No Partitions were found in this design.

-------------------------------

=========================================================================
```

```
*                      Final Report                      *
==========================================================================
Final Results
RTL Top Level Output File Name     : CU_16.ngr
Top Level Output File Name         : CU_16
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 1081

Cell Usage :
# BELS                             : 253228
#      GND                         : 1
#      LUT1                        : 768
#      LUT2                        : 32534
#      LUT3                        : 14995
#      LUT4                        : 35850
#      LUT5                        : 6544
#      LUT6                        : 60816
#      MUXCY                       : 55463
#      VCC                         : 1
#      XORCY                       : 46256
# FlipFlops/Latches                : 34559
#      FDE                         : 2161
#      FDRE                        : 32398
# Shift Registers                  : 2144
#      SRLC32E                     : 2144
# Clock Buffers                    : 2
#      BUFG                        : 1
#      BUFGP                       : 1
# IO Buffers                       : 1080
#      IBUF                        : 507
#      OBUF                        : 573
==========================================================================

Device utilization summary:
---------------------------

Selected Device : 5vlx330ff1760-2


Slice Logic Utilization:
 Number of Slice Registers:         34559  out of  207360    16%
 Number of Slice LUTs:              153651  out of  207360    74%
    Number used as Logic:           151507  out of  207360    73%
    Number used as Memory:            2144  out of  54720     3%
       Number used as SRL:          2144

Slice Logic Distribution:
 Number of Bit Slices used:         158955
   Number with an unused Flip Flop   124396  out of  158955    78%
   Number with an unused LUT:         5304  out of  158955     3%
```

```
   Number of fully used Bit Slices:  29255  out of  158955    18%

IO Utilization:
 Number of IOs:                 1081
 Number of bonded IOBs:         1081  out of   1200    90%

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:         2  out of     32     6%

---------------------------
Partition Resource Summary:
---------------------------

  No Partitions were found in this design.

---------------------------


=========================================================================
TIMING REPORT

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
      FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
      GENERATED AFTER PLACE-and-ROUTE.

Clock Information:
------------------
-----------------------------------+------------------------+-------+
Clock Signal                       | Clock buffer(FF name)  | Load  |
-----------------------------------+------------------------+-------+
clk                                | BUFGP                  | 36703 |
-----------------------------------+------------------------+-------+

Asynchronous Control Signals Information:
----------------------------------------
No asynchronous control signals found in this design

Timing Summary:
---------------
Speed Grade: -2

   Minimum period: 5.041ns (Maximum Frequency: 198.356MHz)
   Minimum input arrival time before clock: 6.430ns
   Maximum output required time after clock: 2.826ns
   Maximum combinational path delay: No path found

Timing Detail:
--------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default period analysis for Clock 'clk'
  Clock period: 5.041ns (frequency: 198.356MHz)
  Total number of paths / destination ports: 19016562 / 32643
```

```
---------------------------------------------------------------------------
Delay:               5.041ns (Levels of Logic = 16)
  Source:            PU_16/B4x4_03_B/PSAD_0/AD2/C_Registered_0 (FF)
  Destination:       PU_15/B4x4_03_B/PSAD_0/PSAD_0_8 (FF)
  Source Clock:      clk rising
  Destination Clock: clk rising

  Data Path: PU_16/B4x4_03_B/PSAD_0/AD2/C_Registered_0 to PU_15/B4x4_03_B/PSAD_0/PSAD_0_8
                            Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDRE:C->Q            22   0.396   0.508  PU_16/B4x4_03_B/PSAD_0/AD2/C_Registered_0
(PU_16/B4x4_03_B/PSAD_0/AD2/C_Registered_0)
     LUT2:I1->O            5   0.086   0.000  PU_15/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_lut<0>
(PU_14/B4x4_03_B/PSAD_0/AD4/Nine_Bit_Sum<0>)
     MUXCY:S->O            1   0.305   0.000  PU_15/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<0>
(PU_15/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<0>)
     MUXCY:CI->O           1   0.023   0.000  PU_15/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<1>
(PU_15/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<1>)
     MUXCY:CI->O           1   0.023   0.000  PU_15/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<2>
(PU_15/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<2>)
     MUXCY:CI->O           1   0.023   0.000  PU_15/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<3>
(PU_15/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<3>)
     MUXCY:CI->O           1   0.023   0.000  PU_15/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<4>
(PU_15/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<4>)
     MUXCY:CI->O           1   0.023   0.000  PU_15/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<5>
(PU_15/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<5>)
     MUXCY:CI->O           1   0.023   0.000  PU_15/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<6>
(PU_15/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<6>)
     MUXCY:CI->O           9   0.222   0.938  PU_15/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<7>
(PU_15/B4x4_03_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<7>)
     LUT6:I0->O            1   0.086   0.487  PU_15/B4x4_03_B/PSAD_0/AD3_Out<5>1
(PU_15/B4x4_03_B/PSAD_0/AD3_Out<5>)
     LUT2:I0->O            1   0.086   0.000  PU_15/B4x4_03_B/PSAD_0/Madd_Nine_Bit_Sum_2_lut<5>
(PU_15/B4x4_03_B/PSAD_0/N50)
     MUXCY:S->O            1   0.305   0.000  PU_15/B4x4_03_B/PSAD_0/Madd_Nine_Bit_Sum_2_cy<5>
(PU_15/B4x4_03_B/PSAD_0/Madd_Nine_Bit_Sum_2_cy<5>)
     XORCY:CI->O           3   0.300   0.496  PU_15/B4x4_03_B/PSAD_0/Madd_Nine_Bit_Sum_2_xor<6>
(PU_15/B4x4_03_B/PSAD_0/Nine_Bit_Sum_2<6>)
     LUT6:I4->O            1   0.086   0.000  PU_15/B4x4_03_B/PSAD_0/Madd_PSAD_0_add0000_Madd_lut<7>
(PU_15/B4x4_03_B/PSAD_0/N59)
     MUXCY:S->O            1   0.305   0.000  PU_15/B4x4_03_B/PSAD_0/Madd_PSAD_0_add0000_Madd_cy<7>
(PU_15/B4x4_03_B/PSAD_0/Madd_PSAD_0_add0000_Madd_cy<7>)
     XORCY:CI->O           1   0.300   0.000  PU_15/B4x4_03_B/PSAD_0/Madd_PSAD_0_add0000_Madd_xor<8>
(PU_15/B4x4_03_B/PSAD_0/PSAD_0_add0000<8>)
     FDRE:D                   -0.022          PU_15/B4x4_03_B/PSAD_0/PSAD_0_8
    ----------------------------------------
    Total                      5.041ns (2.613ns logic, 2.428ns route)
                               (51.8% logic, 48.2% route)

=========================================================================
Timing constraint: Default OFFSET IN BEFORE for Clock 'clk'
  Total number of paths / destination ports: 196121788 / 52539
---------------------------------------------------------------------------
```

```
Offset:             6.430ns (Levels of Logic = 18)
  Source:           bus_Select<2> (PAD)
  Destination:      PU_16/B4x4_10_B/PSAD_0/PSAD_0_8 (FF)
  Destination Clock: clk rising

  Data Path: bus_Select<2> to PU_16/B4x4_10_B/PSAD_0/PSAD_0_8
                              Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O           31999   0.694   0.844  bus_Select_2_IBUF (bus_Select_2_IBUF)
     LUT4:I0->O           2728   0.086   0.668  PU_9/Mrom_bus_Select_rom0000121
(PU_1/Mrom_bus_Select_rom000011)
     LUT4:I2->O             61   0.086   0.000  PU_16/B4x4_10_B/PSAD_0/AD4/Madd_Nine_Bit_Sum_lut<0>
(PU_10/B4x4_12_B/PSAD_0/AD2/Nine_Bit_Sum<0>)
     MUXCY:S->O              1   0.305   0.000  PU_13/B4x4_11_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<0>
(PU_13/B4x4_11_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<0>)
     MUXCY:CI->O             1   0.023   0.000  PU_13/B4x4_11_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<1>
(PU_13/B4x4_11_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<1>)
     MUXCY:CI->O             1   0.023   0.000  PU_13/B4x4_11_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<2>
(PU_13/B4x4_11_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<2>)
     MUXCY:CI->O             1   0.023   0.000  PU_13/B4x4_11_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<3>
(PU_13/B4x4_11_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<3>)
     MUXCY:CI->O             1   0.023   0.000  PU_13/B4x4_11_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<4>
(PU_13/B4x4_11_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<4>)
     MUXCY:CI->O             1   0.023   0.000  PU_13/B4x4_11_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<5>
(PU_13/B4x4_11_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<5>)
     MUXCY:CI->O             1   0.023   0.000  PU_13/B4x4_11_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<6>
(PU_13/B4x4_11_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<6>)
     MUXCY:CI->O             9   0.222   0.938  PU_13/B4x4_11_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<7>
(PU_13/B4x4_11_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<7>)
     LUT6:I0->O              1   0.086   0.487  PU_13/B4x4_11_B/PSAD_0/AD3_Out<5>1
(PU_13/B4x4_11_B/PSAD_0/AD3_Out<5>)
     LUT2:I0->O              1   0.086   0.000  PU_13/B4x4_11_B/PSAD_0/Madd_Nine_Bit_Sum_2_lut<5>
(PU_13/B4x4_11_B/PSAD_0/N50)
     MUXCY:S->O              1   0.305   0.000  PU_13/B4x4_11_B/PSAD_0/Madd_Nine_Bit_Sum_2_cy<5>
(PU_13/B4x4_11_B/PSAD_0/Madd_Nine_Bit_Sum_2_cy<5>)
     XORCY:CI->O             3   0.300   0.496  PU_13/B4x4_11_B/PSAD_0/Madd_Nine_Bit_Sum_2_xor<6>
(PU_13/B4x4_11_B/PSAD_0/Nine_Bit_Sum_2<6>)
     LUT6:I4->O              1   0.086   0.000  PU_13/B4x4_11_B/PSAD_0/Madd_PSAD_0_add0000_Madd_lut<7>
(PU_13/B4x4_11_B/PSAD_0/N59)
     MUXCY:S->O              1   0.305   0.000  PU_13/B4x4_11_B/PSAD_0/Madd_PSAD_0_add0000_Madd_cy<7>
(PU_13/B4x4_11_B/PSAD_0/Madd_PSAD_0_add0000_Madd_cy<7>)
     XORCY:CI->O             1   0.300   0.000  PU_13/B4x4_11_B/PSAD_0/Madd_PSAD_0_add0000_Madd_xor<8>
(PU_13/B4x4_11_B/PSAD_0/PSAD_0_add0000<8>)
     FDRE:D                -0.022          PU_13/B4x4_11_B/PSAD_0/PSAD_0_8
    ----------------------------------------
    Total                   6.430ns (2.997ns logic, 3.433ns route)
                              (46.6% logic, 53.4% route)

==========================================================================
Timing constraint: Default OFFSET OUT AFTER for Clock 'clk'
  Total number of paths / destination ports: 573 / 573
--------------------------------------------------------------------------
Offset:             2.826ns (Levels of Logic = 1)
```

```
  Source:          B4x8_00_out_13 (FF)
  Destination:     B4x8_00_out<13> (PAD)
  Source Clock:    clk rising

  Data Path: B4x8_00_out_13 to B4x8_00_out<13>
                          Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDRE:C->Q             1   0.396   0.286  B4x8_00_out_13 (B4x8_00_out_13)
     OBUF:I->O                 2.144          B4x8_00_out_13_OBUF (B4x8_00_out<13>)
    ----------------------------------------
    Total                    2.826ns (2.540ns logic, 0.286ns route)
                             (89.9% logic, 10.1% route)

=========================================================================
CPU : 9113.59 / 9114.02 s | Elapsed : 9114.00 / 9114.00 s

-->

Total memory usage is 1780844 kilobytes

Number of errors   :    0 (   0 filtered)
Number of warnings :    0 (   0 filtered)
Number of infos    : 3712 (   0 filtered)
```

Sixth design results for a 16 PPUs System with Double-Buffering:

```
Release 9.2.01i - xst J.37
Copyright (c) 1995-2007 Xilinx, Inc.  All rights reserved.
-->
Parameter TMPDIR set to ./xst/projnav.tmp
CPU : 0.00 / 0.25 s | Elapsed : 0.00 / 0.00 s

-->
Parameter xsthdpdir set to ./xst
CPU : 0.00 / 0.25 s | Elapsed : 0.00 / 0.00 s

-->
Reading design: System_16.prj

=========================================================================
*                Synthesis Options Summary                *
=========================================================================
---- Source Parameters
Input File Name                    : "System_16.prj"
Input Format                       : mixed
Ignore Synthesis Constraint File   : NO

---- Target Parameters
Output File Name                   : "System_16"
Output Format                      : NGC
Target Device                      : xc5vlx330-2-ff1760

---- Source Options
Top Module Name                    : System_16
Automatic FSM Extraction           : YES
FSM Encoding Algorithm             : Auto
Safe Implementation                : No
FSM Style                          : lut
RAM Extraction                     : Yes
RAM Style                          : Auto
ROM Extraction                     : Yes
Mux Style                          : Auto
Decoder Extraction                 : YES
Priority Encoder Extraction        : YES
Shift Register Extraction          : YES
Logical Shifter Extraction         : YES
XOR Collapsing                     : YES
ROM Style                          : Auto
Mux Extraction                     : YES
Resource Sharing                   : YES
Asynchronous To Synchronous        : NO
Use DSP Block                      : auto
Automatic Register Balancing       : No

---- Target Options
Add IO Buffers                     : YES
Global Maximum Fanout              : 100000
```

```
Add Generic Clock Buffer(BUFG)     : 32
Register Duplication               : YES
Slice Packing                      : YES
Optimize Instantiated Primitives   : NO
Use Clock Enable                   : Auto
Use Synchronous Set                : Auto
Use Synchronous Reset              : Auto
Pack IO Registers into IOBs        : auto
Equivalent register Removal        : YES

---- General Options
Optimization Goal                  : Speed
Optimization Effort                : 1
Power Reduction                    : NO
Library Search Order               : System_16.lso
Keep Hierarchy                     : NO
RTL Output                         : Yes
Global Optimization                : AllClockNets
Read Cores                         : YES
Write Timing Constraints           : NO
Cross Clock Analysis               : NO
Hierarchy Separator                : /
Bus Delimiter                      : <>
Case Specifier                     : maintain
Slice Utilization Ratio            : 100
BRAM Utilization Ratio             : 100
DSP48 Utilization Ratio            : 100
Verilog 2001                       : YES
Auto BRAM Packing                  : NO
Slice Utilization Ratio Delta      : 5

=========================================================================


=========================================================================
*                     HDL Compilation                   *
=========================================================================
=========================================================================
*                Design Hierarchy Analysis              *
=========================================================================
Analyzing hierarchy for module <System_16> in library <work>.

Analyzing hierarchy for module <CU_16> in library <work>.

Analyzing hierarchy for module <dual_buffer> in library <work>.

Analyzing hierarchy for module <main_data_path> in library <work>.

Analyzing hierarchy for module <on_chip_mem> in library <work>.

Analyzing hierarchy for module <four_by_four_block> in library <work>.

Analyzing hierarchy for module <four_stage_delay_line> in library <work>.
```

Analyzing hierarchy for module <eight_stage_delay_line> in library <work>.

Analyzing hierarchy for module <spblockram_66> in library <work> with parameters.
PIXEL_ROWS = "00000000000000000000000001000010"
RAM_ADDRESS_WIDTH = "00000000000000000000000000000111"
RAM_DATA_WIDTH = "00000000000000000000000010000000"
RAM_SIZE = "00000000000000000000000001000010"

Analyzing hierarchy for module <spblockram_30> in library <work> with parameters.
PIXEL_ROWS = "00000000000000000000000000011110"
RAM_ADDRESS_WIDTH = "00000000000000000000000000000101"
RAM_DATA_WIDTH = "00000000000000000000000010000000"
RAM_SIZE = "00000000000000000000000000011110"

Analyzing hierarchy for module <Row_Adder_0> in library <work>.

Analyzing hierarchy for module <Row_Adder_1_CSA> in library <work>.

Analyzing hierarchy for module <Row_Adder_2_CSA> in library <work>.

Analyzing hierarchy for module <Row_Adder_3_CSA> in library <work>.

Analyzing hierarchy for module <PE_AD> in library <work>.

Analyzing hierarchy for module <PE_AD_CSA> in library <work>.

Analyzing hierarchy for module <CSA_8> in library <work>.

Analyzing hierarchy for module <CSA_9> in library <work>.

Analyzing hierarchy for module <CSA_Full_Adder> in library <work>.

Unit <System_16> synthesized.

===

===
* Advanced HDL Synthesis *
===

Loading device for application Rf_Device from file '5vlx330.nph' in environment /usr/local/Xilinx92i.
INFO:Xst:2691 - Unit <spblockram_66> : The RAM <Mram_ram> will be implemented as a BLOCK RAM, absorbing the following register(s): <read_a>.

ram_type	Block		
Port A			
aspect ratio	66-word x 128-bit		
mode	write-first		
clkA	connected to signal <clk>	rise	
weA	connected to signal <we>	high	
addrA	connected to signal <a>		
diA	connected to signal <di>		
doA	connected to signal <do>		

```
    ----------------------------------------------------------------------
    | optimisation       | speed                         |        |
    ----------------------------------------------------------------------
INFO:Xst:2691 - Unit <spblockram_30> : The RAM <Mram_ram> will be implemented as a BLOCK RAM, absorbing the following register(s): <read_a>.
    ----------------------------------------------------------------------
    | ram_type           | Block                         |        |
    ----------------------------------------------------------------------
    | Port A                                                              |
    |     aspect ratio   | 30-word x 128-bit               |        |
    |     mode           | write-first                     |        |
    |     clkA           | connected to signal <clk>       | rise   |
    |     weA            | connected to signal <we>        | high   |
    |     addrA          | connected to signal <a>         |        |
    |     diA            | connected to signal <di>        |        |
    |     doA            | connected to signal <do>        |        |
    ----------------------------------------------------------------------
    | optimisation       | speed                         |        |
    ----------------------------------------------------------------------

=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
# RAMs                                : 8
 30x128-bit single-port block RAM    : 4
 66x128-bit single-port block RAM    : 4
# ROMs                                : 16
 16x15-bit ROM                       : 16
# Adders/Subtractors                  : 7065
 10-bit adder                        : 256
 11-bit adder carry in               : 256
 12-bit adder carry in               : 256
 13-bit adder carry in               : 256
 14-bit adder                        : 256
 15-bit adder                        : 64
 16-bit adder                        : 64
 17-bit adder                        : 16
 4-bit adder                         : 2
 4-bit adder carry out               : 2
 6-bit adder                         : 2
 6-bit adder carry out               : 2
 7-bit adder                         : 1
 8-bit adder                         : 1024
 9-bit adder                         : 4608
# Registers                           : 69999
 Flip-Flops                          : 69999
# Comparators                         : 625
 13-bit comparator less              : 240
 14-bit comparator less              : 240
 15-bit comparator less              : 60
 16-bit comparator less              : 60
 17-bit comparator less              : 15
 6-bit comparator greater            : 2
```

```
 6-bit comparator less                 : 4
 7-bit comparator less                 : 4
# Xors                                 : 42497
 1-bit xor2                            : 1
 1-bit xor3                            : 38400
 8-bit xor2                            : 4096

=========================================================================

=========================================================================
*                     Low Level Synthesis                               *
=========================================================================

Optimizing unit <System_16> ...

Optimizing unit <four_stage_delay_line> ...

Optimizing unit <eight_stage_delay_line> ...

Optimizing unit <PE_AD_CSA> ...

Optimizing unit <Row_Adder_0> ...

Optimizing unit <dual_buffer> ...

Optimizing unit <Row_Adder_1_CSA> ...

Optimizing unit <Row_Adder_2_CSA> ...

Optimizing unit <Row_Adder_3_CSA> ...

Optimizing unit <four_by_four_block> ...

Optimizing unit <main_data_path> ...

Optimizing unit <CU_16> ...

Mapping all equations...
Building and optimizing final netlist ...
Area constraint is met for block <System_16>, final ratio is 91.

Final Macro Processing ...

Processing Unit <System_16> :
Unit <System_16> processed.

=========================================================================
Final Register Report

Macro Statistics
# Registers                           : 33092
 Flip-Flops                           : 33092
# Shift Registers                     : 2144
 3-bit shift register                 : 1664
```

```
 7-bit shift register                : 480

=========================================================================

=========================================================================
*                     Partition Report                        *
=========================================================================

Partition Implementation Status
-------------------------------

  No Partitions were found in this design.

-------------------------------

=========================================================================
*                     Final Report                            *
=========================================================================
Final Results
RTL Top Level Output File Name     : System_16.ngr
Top Level Output File Name         : System_16
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 848

Cell Usage :
# BELS                             : 254559
#      GND                         : 1
#      LUT1                        : 768
#      LUT2                        : 32341
#      LUT3                        : 15531
#      LUT4                        : 37531
#      LUT5                        : 6895
#      LUT6                        : 59772
#      MUXCY                       : 55463
#      VCC                         : 1
#      XORCY                       : 46256
# FlipFlops/Latches                : 35236
#      FDE                         : 2161
#      FDR                         : 655
#      FDRE                        : 32393
#      FDS                         : 27
# RAMS                             : 32
#      RAMB36_EXP                  : 32
# Shift Registers                  : 2144
#      SRLC16E                     : 2144
# Clock Buffers                    : 2
#      BUFG                        : 1
#      BUFGP                       : 1
# IO Buffers                       : 847
#      IBUF                        : 274
```

```
#      OBUF                   : 573
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 5vlx330ff1760-2


Slice Logic Utilization:
 Number of Slice Registers:         35236  out of  207360    16%
 Number of Slice LUTs:              154982  out of  207360    74%
    Number used as Logic:           152838  out of  207360    73%
    Number used as Memory:            2144  out of  54720     3%
      Number used as SRL:           2144

Slice Logic Distribution:
 Number of Bit Slices used:         160461
   Number with an unused Flip Flop   125225  out of  160461    78%
   Number with an unused LUT:         5479  out of  160461     3%
   Number of fully used Bit Slices:  29757  out of  160461    18%

IO Utilization:
 Number of IOs:                  848
 Number of bonded IOBs:              848  out of   1200    70%

Specific Feature Utilization:
 Number of Block RAM/FIFO:           32  out of    288    11%
    Number using Block RAM only:     32
 Number of BUFG/BUFGCTRLs:            2  out of     32     6%

---------------------------
Partition Resource Summary:
---------------------------

  No Partitions were found in this design.

---------------------------


=========================================================================
TIMING REPORT

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
      FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
      GENERATED AFTER PLACE-and-ROUTE.

Clock Information:
------------------
-----------------------------------+------------------------+-------+
Clock Signal                       | Clock buffer(FF name)  | Load  |
-----------------------------------+------------------------+-------+
clk                                | BUFGP                  | 37412 |
-----------------------------------+------------------------+-------+
```

```
Asynchronous Control Signals Information:
----------------------------------------
No asynchronous control signals found in this design

Timing Summary:
---------------
Speed Grade: -2

   Minimum period: 5.218ns (Maximum Frequency: 191.661MHz)
   Minimum input arrival time before clock: 6.605ns
   Maximum output required time after clock: 2.826ns
   Maximum combinational path delay: No path found

Timing Detail:
--------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default period analysis for Clock 'clk'
  Clock period: 5.218ns (frequency: 191.661MHz)
  Total number of paths / destination ports: 84955940 / 38019
-------------------------------------------------------------------------
Delay:               5.218ns (Levels of Logic = 16)
  Source:            Data_Path/PU_12/B4x4_33_B/PSAD_0/AD4/C_Registered_0 (FF)
  Destination:       Data_Path/PU_13/B4x4_33_B/PSAD_0/PSAD_0_8 (FF)
  Source Clock:      clk rising
  Destination Clock: clk rising

  Data Path: Data_Path/PU_12/B4x4_33_B/PSAD_0/AD4/C_Registered_0 to
Data_Path/PU_13/B4x4_33_B/PSAD_0/PSAD_0_8
                           Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDRE:C->Q            19   0.396   0.684  Data_Path/PU_12/B4x4_33_B/PSAD_0/AD4/C_Registered_0
(Data_Path/PU_12/B4x4_33_B/PSAD_0/AD4/C_Registered_0)
     LUT4:I1->O           21   0.086   0.000
Data_Path/PU_13/B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_lut<0>
(Data_Path/PU_12/B4x4_33_B/PSAD_0/AD4/Nine_Bit_Sum<0>)
     MUXCY:S->O            1   0.305   0.000
Data_Path/PU_13/B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<0>
(Data_Path/PU_13/B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<0>)
     MUXCY:CI->O           1   0.023   0.000
Data_Path/PU_13/B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<1>
(Data_Path/PU_13/B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<1>)
     MUXCY:CI->O           1   0.023   0.000
Data_Path/PU_13/B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<2>
(Data_Path/PU_13/B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<2>)
     MUXCY:CI->O           1   0.023   0.000
Data_Path/PU_13/B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<3>
(Data_Path/PU_13/B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<3>)
     MUXCY:CI->O           1   0.023   0.000
Data_Path/PU_13/B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<4>
(Data_Path/PU_13/B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<4>)
```

```
    MUXCY:CI->O           1   0.023   0.000
Data_Path/PU_13/B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<5>
(Data_Path/PU_13/B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<5>)
    MUXCY:CI->O           1   0.023   0.000
Data_Path/PU_13/B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<6>
(Data_Path/PU_13/B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<6>)
    MUXCY:CI->O           9   0.222   0.938
Data_Path/PU_13/B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<7>
(Data_Path/PU_13/B4x4_33_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<7>)
    LUT6:I0->O            1   0.086   0.487  Data_Path/PU_13/B4x4_33_B/PSAD_0/AD3_Out<5>1
(Data_Path/PU_13/B4x4_33_B/PSAD_0/AD3_Out<5>)
    LUT2:I0->O            1   0.086   0.000  Data_Path/PU_13/B4x4_33_B/PSAD_0/Madd_Nine_Bit_Sum_2_lut<5>
(Data_Path/PU_13/B4x4_33_B/PSAD_0/N50)
    MUXCY:S->O            1   0.305   0.000  Data_Path/PU_13/B4x4_33_B/PSAD_0/Madd_Nine_Bit_Sum_2_cy<5>
(Data_Path/PU_13/B4x4_33_B/PSAD_0/Madd_Nine_Bit_Sum_2_cy<5>)
    XORCY:CI->O           3   0.300   0.496
Data_Path/PU_13/B4x4_33_B/PSAD_0/Madd_Nine_Bit_Sum_2_xor<6>
(Data_Path/PU_13/B4x4_33_B/PSAD_0/Nine_Bit_Sum_2<6>)
    LUT6:I4->O            1   0.086   0.000
Data_Path/PU_13/B4x4_33_B/PSAD_0/Madd_PSAD_0_add0000_Madd_lut<7>
(Data_Path/PU_13/B4x4_33_B/PSAD_0/N59)
    MUXCY:S->O            1   0.305   0.000
Data_Path/PU_13/B4x4_33_B/PSAD_0/Madd_PSAD_0_add0000_Madd_cy<7>
(Data_Path/PU_13/B4x4_33_B/PSAD_0/Madd_PSAD_0_add0000_Madd_cy<7>)
    XORCY:CI->O           1   0.300   0.000
Data_Path/PU_13/B4x4_33_B/PSAD_0/Madd_PSAD_0_add0000_Madd_xor<8>
(Data_Path/PU_13/B4x4_33_B/PSAD_0/PSAD_0_add0000<8>)
    FDRE:D                -0.022          Data_Path/PU_13/B4x4_33_B/PSAD_0/PSAD_0_8
    ----------------------------------------
    Total                 5.218ns (2.613ns logic, 2.604ns route)
                          (50.1% logic, 49.9% route)

==========================================================================
Timing constraint: Default OFFSET IN BEFORE for Clock 'clk'
  Total number of paths / destination ports: 128506318 / 50180
--------------------------------------------------------------------------
Offset:              6.605ns (Levels of Logic = 18)
  Source:            bus_Select<3> (PAD)
  Destination:       Data_Path/PU_1/B4x4_32_B/PSAD_0/PSAD_0_8 (FF)
  Destination Clock: clk rising

  Data Path: bus_Select<3> to Data_Path/PU_1/B4x4_32_B/PSAD_0/PSAD_0_8
                        Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
    IBUF:I->O         32505   0.694   0.844  bus_Select_3_IBUF (bus_Select_3_IBUF)
    LUT4:I0->O         2290   0.086   0.844  Data_Path/PU_9/Mrom_bus_Select_rom000041
(Data_Path/PU_1/Mrom_bus_Select_rom00003)
    LUT4:I0->O           15   0.086   0.000
Data_Path/PU_1/B4x4_30_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_lut<0>
(Data_Path/PU_1/B4x4_30_B/PSAD_0/AD3/Nine_Bit_Sum<0>)
    MUXCY:S->O            1   0.305   0.000
Data_Path/PU_1/B4x4_30_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<0>
(Data_Path/PU_1/B4x4_30_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<0>)
```

```
    MUXCY:CI->O           1   0.023   0.000
Data_Path/PU_1/B4x4_30_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<1>
(Data_Path/PU_1/B4x4_30_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<1>)
    MUXCY:CI->O           1   0.023   0.000
Data_Path/PU_1/B4x4_30_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<2>
(Data_Path/PU_1/B4x4_30_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<2>)
    MUXCY:CI->O           1   0.023   0.000
Data_Path/PU_1/B4x4_30_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<3>
(Data_Path/PU_1/B4x4_30_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<3>)
    MUXCY:CI->O           1   0.023   0.000
Data_Path/PU_1/B4x4_30_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<4>
(Data_Path/PU_1/B4x4_30_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<4>)
    MUXCY:CI->O           1   0.023   0.000
Data_Path/PU_1/B4x4_30_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<5>
(Data_Path/PU_1/B4x4_30_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<5>)
    MUXCY:CI->O           1   0.023   0.000
Data_Path/PU_1/B4x4_30_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<6>
(Data_Path/PU_1/B4x4_30_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<6>)
    MUXCY:CI->O           9   0.222   0.938
Data_Path/PU_1/B4x4_30_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<7>
(Data_Path/PU_1/B4x4_30_B/PSAD_0/AD3/Madd_Nine_Bit_Sum_cy<7>)
    LUT6:I0->O            1   0.086   0.487  Data_Path/PU_1/B4x4_30_B/PSAD_0/AD3_Out<5>1
(Data_Path/PU_1/B4x4_30_B/PSAD_0/AD3_Out<5>)
    LUT2:I0->O            1   0.086   0.000  Data_Path/PU_1/B4x4_30_B/PSAD_0/Madd_Nine_Bit_Sum_2_lut<5>
(Data_Path/PU_1/B4x4_30_B/PSAD_0/N50)
    MUXCY:S->O            1   0.305   0.000  Data_Path/PU_1/B4x4_30_B/PSAD_0/Madd_Nine_Bit_Sum_2_cy<5>
(Data_Path/PU_1/B4x4_30_B/PSAD_0/Madd_Nine_Bit_Sum_2_cy<5>)
    XORCY:CI->O           3   0.300   0.496  Data_Path/PU_1/B4x4_30_B/PSAD_0/Madd_Nine_Bit_Sum_2_xor<6>
(Data_Path/PU_1/B4x4_30_B/PSAD_0/Nine_Bit_Sum_2<6>)
    LUT6:I4->O            1   0.086   0.000
Data_Path/PU_1/B4x4_30_B/PSAD_0/Madd_PSAD_0_add0000_Madd_lut<7>
(Data_Path/PU_1/B4x4_30_B/PSAD_0/N59)
    MUXCY:S->O            1   0.305   0.000
Data_Path/PU_1/B4x4_30_B/PSAD_0/Madd_PSAD_0_add0000_Madd_cy<7>
(Data_Path/PU_1/B4x4_30_B/PSAD_0/Madd_PSAD_0_add0000_Madd_cy<7>)
    XORCY:CI->O           1   0.300   0.000
Data_Path/PU_1/B4x4_30_B/PSAD_0/Madd_PSAD_0_add0000_Madd_xor<8>
(Data_Path/PU_1/B4x4_30_B/PSAD_0/PSAD_0_add0000<8>)
    FDRE:D                -0.022          Data_Path/PU_1/B4x4_30_B/PSAD_0/PSAD_0_8
    ----------------------------------------
    Total                 6.605ns (2.997ns logic, 3.608ns route)
                          (45.4% logic, 54.6% route)

==========================================================================
Timing constraint: Default OFFSET OUT AFTER for Clock 'clk'
  Total number of paths / destination ports: 573 / 573
--------------------------------------------------------------------------
Offset:              2.826ns (Levels of Logic = 1)
  Source:            Data_Path/B4x8_00_out_13 (FF)
  Destination:       B4x8_00_out<13> (PAD)
  Source Clock:      clk rising

  Data Path: Data_Path/B4x8_00_out_13 to B4x8_00_out<13>
                          Gate     Net
```

```
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDRE:C->Q             1   0.396   0.286  Data_Path/B4x8_00_out_13 (Data_Path/B4x8_00_out_13)
     OBUF:I->O                 2.144          B4x8_00_out_13_OBUF (B4x8_00_out<13>)
    ----------------------------------------
    Total                     2.826ns (2.540ns logic, 0.286ns route)
                              (89.9% logic, 10.1% route)

==========================================================================
CPU : 25644.38 / 25644.66 s | Elapsed : 25935.00 / 25935.00 s

-->


Total memory usage is 2602688 kilobytes

Number of errors   :    0 (   0 filtered)
Number of warnings :    0 (   0 filtered)
Number of infos    : 3714 (   0 filtered)
```

BIBLIOGRAPHY

[1] Internation Telecommunication Union (ITU), H.264 Advanced Video Coding Standard: http://www.itu.int/rec/T-REC-H.264/e

[2] Cliff Wootton, "A Practical Guide to Video and Audio Compression From Sprockets and Rasters to Macro Blocks", *Elsevier Inc.*, 2003.

[3] Mohammed E. Al-Mualla, C. Nishan Canagarajah, and David R. Bull, "Video Coding for Mobile Communications Efficiency, Complexity, and Resilience", *Elsevier Science*, 2002.

[4] Yu-Wen Huang, Tu-Chih Wang, Bing-Yu Hsieh, Liang-Gee Chen, "Hardware Architecture Design for Variable Block Size Motion Estimation in MPEG-4 AVC/JVT/ITU-T H.264," *Proceedings of the 2003 International Symposium on Circuits and Systems*, Vol. 2, pp. 25-28, May 2003.

[5] S. Yap and J. V. McCanny, "A VLSI Architecture for Variable Block Size Video Motion Estimation," *IEEE Transactions on Circuits and Systems II*, Vol. 51, No. 7, pp. 384-389, July 2004.

[6] M. Kim, I. Hwang, and S. Chae, "A Fast VLSI Architecture for Full-Search Variable Block Size Motion Estimation in MPEG-4 AVC/H.264," *Proceedings of the 2005 conference on Asia South Pacific design automation*, pp. 631-634, 2005.

[7] Yang Song, Zhenyu Liu, Satoshi Goto, Takeshi Ikenaga, "Scalable VLSI Architecture for Variable Block Size Integer Motion Estimation in H.264/AVC," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. E89-A, No. 4, pp. 979-988, April 2006.

[8] Yang Song, Zhenyu Liu, Takeshi Ikenaga, Satoshi Goto, "VLSI Architecture for Variable Block Size Motion Estimation in H.264/AVC with Low Cost Memory Organization," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. E89-A, No. 12, pp. 3594-3601, December 2006.

[9] Zhenyu Liu, Yang Song, Takeshi Ikenaga, Satoshi Goto, "A Fine-Grain Scalable and Low Memory Cost Variable Block Size Motion Estimation Architecture for H.264/AVC," *IEICE Transactions on Electronics*, Vol. E89-C, No. 12, pp. 1928-1936, December 2006.

[10] Tung-Chien Chen, Shao-Yi Chien, Yu-Wen Huang, Chen-Han Tsai, Ching-Yeh Chen, To-Wei Chen, Liang-Gee Chen, "Analysis and Architecture Design of an HDTV720p 30 Frames/s H.264/AVC Encoder," IEEE Transactions on Circuits and Systems for Video Technology, Vol. 16, No. 6, pp. 673-688, June 2006.

[11] 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.

[12] W. Chung, "Implementing the H.264/AVC Video Coding Standards on FPGAs," *Xilinx Broadcast Solution Guide*, pp. 18-21, September 2005.

[13] "Faraday H.264 Baseline Video Encoder & Decoder IPs: FTMCP210/FTMCP220," *Faraday Technology Corporation Product Documentation*, 2005.

[14] "H.264 Motion Estimation Engine (DO-DI-H264-ME)," *Xilinx Corporation Product Documentation*, October 2007.

[15] S. Lopez, F. Tobajas, A. Villar, V. de Armas, J.F. Lopez, R. Sarmiento, "Low Cost Efficient Architecture for H.264 Motion Estimation," *IEEE International Symposium on Circuits and Systems*, Vol. 1, pp. 412-415, May 2005.

[16] S. Yalcin, H.F. Ates, I. Hamzaoglu, "A High Performance Hardware Architecture for an SAD Reuse Based Hierarchical Motion Estimation Algorithm for H.264 Video Coding," *Proceedings of the 2005 International Conference on Field Programmable Logic and Applications*, pp. 509-514, August 2005.

[17] Giovanni Agnoli, "Developer Connection – Quick Time Pixel Formats", http://developer.apple.com/quicktime/icefloe/dispatch020.html, *Apple Computer Inc.*, 2005.

[18] Peter Kuhn, "Algorithms, Cimplexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation", *Kluwer Academic Publishers*, 1999.

[19] Jian-Wen Chen, Chao-Yang Kao, and Youn-Long Lin, "Introduction to H.264 Advanced Video Coding," *Asia and South Pacific Conference on Design Automation*, Page(s):6 pp. Jan 24-27 2006.

[20] "H.264/MPEG-4 AVC Video Compression Tutorial", *LSI Logic Inc.*

[21] "V5 Product Table", Xilinx Corporation : http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex5/v5product_table.pdf

[22] "H.264 Motion Estimation Engine v1.0", *Xilinx LogiCore Product Specification*, DS648 April 23, 2008: http://www.xilinx.com/support/documentation/ip_documentation/h264_mee_ds648.pdf

[23] Peter M. Nyasulu, J. Knight, "Verilog Manual" : http://web.doe.carleton.ca/~pavan/courses/resources-elec3500/peter_verilog.pdf

[24] Michael John Sebastian Smith, "Application Specific Integrated Circuits", *Addison Wesley*, 1997.

[25] Miles J. Murdocca, Vincent P. Heuring, "Principles of Computer Architecture", *Prentice-Hall Inc.*, 2000.

[26] Charles H. Roth Jr., "Digital Systems Design Using VHDL", *PWS Publishing*, 1998.

[27] Stephen Brown, Zvonko Vranesic, "Fundamentals of Digital Logic with VHDL Design", *McGraw-Hill Higher Education*, 2000.