Novel VLSI architecture of motion estimation and compensation for H.264 standard

Xiang Li

Follow this and additional works at: https://scholarworks.rit.edu/theses

Recommended Citation
Novel VLSI architecture of Motion Estimation and Compensation for H.264 standard

by

Xiang Li

A Thesis Submitted in Partial Fulfillment Of the Requirements for the Degree of MASTER OF SCIENCE in Electrical Engineering

Approved by:

Kenneth W. Hsu
Dr. Kenneth W. Hsu

Pratapa Reddy
Dr. Pratapa Reddy

Dorin Patru
Dr. Dorin Patru

Robert Bowman
Dr. Robert J. Bowman (Department Head)

Department of Electrical Engineering
College of Engineering
Rochester Institute of Technology
Rochester, New York
August 2004
RELEASE PERMISSION FORM

Rochester Institute of Technology

Novel VLSI architecture
of Motion Estimation and Compensation
for H.264 standard

I, Xiang Li, hereby grant permission to any individual or organization to reproduce this thesis in whole or in part for non-commercial and non-profit purposes only.

Xiang Li

Xiang Li

08/31/04

Date
Abstract

This thesis presents a high performance novel VLSI architecture of a H.264 motion estimator, which can be used as a building block for real-time H.264 video compression. Full-search block matching algorithm was used in this design. Pipeline structure was developed for variable block size processing units to work in parallel. The speed at 125MHz is good for real time motion estimation with 25/sec frame rate and 640x480 resolutions. The processing speed is also independent of the threshold level of Sum of Absolute Difference (SAD), which is used to determine the size of the macro block. The architecture is implemented with Register Transfer Level VHDL codes then synthesized with Synopsys Design Compiler, using TSMC 0.25um technology. The synthesized Application Specific Integrated Circuits (ASIC’s) has an area of 664um x 664um.
# TABLE OF CONTENTS

Table of Contents......................................................................................... i

Glossary........................................................................................................ iv

CHAPTER 1: Introduction................................................................................. 1

1.1 H.264 features......................................................................................... 1
1.2 Block-matching Algorithm.................................................................... 2
1.3 Description of global VLSI Block Diagram........................................... 4

CHAPTER 2: Literature review....................................................................... 5

2.1 Origin of H.264....................................................................................... 5
2.2 H.264 Codec........................................................................................ 5
2.3 Motion Estimation and Compensation................................................... 8
2.4 Existing popular ME/MC Algorithms.................................................... 11

CHAPTER 3: Dataflow and VLSI Architecture Design................................. 13

3.1 Dataflow Diagram for 16x16, 8x8, and 4x4 ME.................................. 13
3.2 VLSI Architecture for 16x16, 8x8, and 4x4 ME.................................. 16
3.3 Dataflow Diagram for Fractional ME................................................... 18
3.4 VLSI Architecture for Fractional ME.................................................. 20

CHAPTER 4: Behavior VHDL design........................................................... 21

CHAPTER 5: Register Transfer Level VHDL design.................................... 22

5.1 16x16 processing block......................................................................... 22
5.2 8x8, 4x4 processing blocks................................................................... 35
5.3 Fractional MV processing block............................................................ 37

CHAPTER 6: Simulation Results and Analysis.............................................. 42

6.1: Simulation (Gymnast).......................................................................... 42
6.2: Simulation (Artist with big movement)................................................. 45

CHAPTER 7: Synthesis of RTL VHDL codes................................................. 49

7.1: Constraints for Synthesis.................................................................... 49
7.2: Area Report........................................................................................ 49
7.3: Timing Report....................................................................................... 50
<table>
<thead>
<tr>
<th>APPENDIX A: Synthesized Circuits</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A-1 : AGU</td>
<td>57</td>
</tr>
<tr>
<td>A-2 : Comparator</td>
<td>58</td>
</tr>
<tr>
<td>A-3 : Controller</td>
<td>59</td>
</tr>
<tr>
<td>A-4 : Interconnection</td>
<td>60</td>
</tr>
<tr>
<td>A-5 : PE</td>
<td>61</td>
</tr>
<tr>
<td>A-6 : SPU</td>
<td>62</td>
</tr>
<tr>
<td>A-7 : Memory previous frame</td>
<td>63</td>
</tr>
<tr>
<td>A-8 : Memory current frame</td>
<td>63</td>
</tr>
<tr>
<td>A-9 : Mux between memory and interconnection</td>
<td>63</td>
</tr>
<tr>
<td>A-10 : Transfer Unit</td>
<td>64</td>
</tr>
<tr>
<td>A-11 : Bridging unit</td>
<td>65</td>
</tr>
<tr>
<td>A-12 : ME top level</td>
<td>66</td>
</tr>
<tr>
<td>A-13 : 8x8 AGU</td>
<td>67</td>
</tr>
<tr>
<td>A-14 : 8x8 Interconnection</td>
<td>68</td>
</tr>
<tr>
<td>A-15 : 8x8 Controller</td>
<td>69</td>
</tr>
<tr>
<td>A-16 : 4x4 AGU</td>
<td>70</td>
</tr>
<tr>
<td>A-17 : 4x4 Interconnection</td>
<td>71</td>
</tr>
<tr>
<td>A-18 : 4x4 Controller</td>
<td>72</td>
</tr>
<tr>
<td>A-19 : IP1</td>
<td>73</td>
</tr>
<tr>
<td>A-20 : IP2</td>
<td>74</td>
</tr>
<tr>
<td>A-21 : Frac_AGU</td>
<td>75</td>
</tr>
<tr>
<td>A-22 : Frac_Controller</td>
<td>76</td>
</tr>
<tr>
<td>A-23 : Frac_Comparator</td>
<td>77</td>
</tr>
<tr>
<td>A-24 : Frac_PE</td>
<td>78</td>
</tr>
<tr>
<td>A-25 : Frac_SPU</td>
<td>79</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>APPENDIX B: Source Codes</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>B-1: Behavior VHDL code</td>
<td>90</td>
</tr>
<tr>
<td>B-2: Register Transfer Level Code</td>
<td>100</td>
</tr>
<tr>
<td>B-2.1 : AGU</td>
<td>100</td>
</tr>
<tr>
<td>B-2.2 : Comparator</td>
<td>102</td>
</tr>
<tr>
<td>B-2.3 : Controller</td>
<td>104</td>
</tr>
<tr>
<td>B-2.4 : Interconnection</td>
<td>107</td>
</tr>
<tr>
<td>B-2.5 : PE</td>
<td>109</td>
</tr>
<tr>
<td>B-2.6 : SPU</td>
<td>110</td>
</tr>
<tr>
<td>B-2.7 : Memory previous frame</td>
<td>112</td>
</tr>
<tr>
<td>Page</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>B-2.8</td>
<td>Memory current frame</td>
</tr>
<tr>
<td>B-2.9</td>
<td>Mux between memory and interconnection</td>
</tr>
<tr>
<td>B-2.10</td>
<td>Transfer Unit</td>
</tr>
<tr>
<td>B-2.11</td>
<td>Bridging unit</td>
</tr>
<tr>
<td>B-2.12</td>
<td>ME top level</td>
</tr>
<tr>
<td>B-2.13</td>
<td>8x8 AGU</td>
</tr>
<tr>
<td>B-2.14</td>
<td>8x8 Interconnection</td>
</tr>
<tr>
<td>B-2.15</td>
<td>8x8 Controller</td>
</tr>
<tr>
<td>B-2.16</td>
<td>4x4 AGU</td>
</tr>
<tr>
<td>B-2.17</td>
<td>4x4 Interconnection</td>
</tr>
<tr>
<td>B-2.18</td>
<td>4x4 Controller</td>
</tr>
<tr>
<td>B-2.19</td>
<td>IP1</td>
</tr>
<tr>
<td>B-2.20</td>
<td>IP2</td>
</tr>
<tr>
<td>B-2.21</td>
<td>Frac_AGU</td>
</tr>
<tr>
<td>B-2.22</td>
<td>Frac_Controller</td>
</tr>
<tr>
<td>B-2.23</td>
<td>Frac_Comparator</td>
</tr>
<tr>
<td>B-2.24</td>
<td>Frac_PE</td>
</tr>
<tr>
<td>B-2.25</td>
<td>Frac_SPU</td>
</tr>
<tr>
<td>B-2.26</td>
<td>Frac_Mem_current</td>
</tr>
<tr>
<td>B-2.27</td>
<td>Frac_mem_Previous</td>
</tr>
<tr>
<td>B-2.28</td>
<td>Frac_Toplevel</td>
</tr>
</tbody>
</table>

113 114 115 117 118 123 126 127 131 133 135 138 138 139 141 143 144 145 145 146 147
Glossary

AGU
Address Generation Unit, Unit that generates sequence of memory address in a specific way.

ASIC
Application Specific Integrated Circuit.

Behavior VHDL code
A technique used in describing a circuit functionality at a high level. Generally, the architecture can not be inferred from the description.

VHDL
Very large Hardware Description Language

MC
Motion Compensation, a technique to substract a current frame from its previous frame which has already been motion estimated, in order to obtain the residual.

ME
Motion Estimation, a technique to predict the current frame from previous frame by calculating the motion vectors and residuals.

MV
Motion Vector, used to describe the displacement of two matching blocks in neighborhood frames.

RTL VHDL code
Registered Transfer Level VHDL code, a technique used in describing a circuit at state machine and data path level. Generally, the architecture can be easily inferred from the description.

SAD
Sum of Absolute Difference, between two macro blocks with same size.

SPU
Start Point Generator.
CHAPTER 1: Introduction

1.1 H.264 features in this design.

The latest H.264 standard provides adaptive and powerful coding schemes, including tree structure motion compensation and quarter-pixel motion vector. These features will provide smaller residual by more accurate motion vectors, but use more hardware.

It is not easy to implement H.264 codec as a real-time system due to its high requirement of memory bandwidth and intensive computation [1]. Variable block size, and quarter-pixel motion vectors, being the key features of H.264 standard, demands substantial computational complexity. Most existent fast motion estimation algorithms are not suitable [2] for H.264 having variable block sizes.

In this thesis, a novel VLSI architecture is proposed and implemented; following highly desirable features are achieved

1. Pipeline data processing allows the variable block size computation to work as fast as the traditional 16x16 block size motion estimation.

2. Full-search algorithm is implemented which is also the optimal solution in block matching.

3. Sequential inputs from memory to reduce the memory bandwidth and pin count.

4. 8x8, 4x4 sized block matching requires only local memory access, by using very small on-chip memories.

5. Adaptive design to balance the computation complexity and compression ratio for specific applications.
1.2 Block-matching Algorithm.

There is a significant amount of frame-to-frame redundancy existing in full motion video sequence. Usually the scenes in successive frames are highly correlated. Motion estimation/compensation is the inter-frame coding that reduces the redundancy information and achieves high data compression ratio.

Motion estimation is in most cases bases on a search scheme which tries to find the best matching position of a macro-block (MB) of the current frame which a block of same size within a predetermined or adaptive search range in the previous frame. The position offset between these 2 matching blocks is called motion vector (MV). The size of MB is 16x16 pixels in previous compression standards, and variable from 16x16 to 4x4 in H.264.

![Figure 1-1 Block Matching Algorithm](image-url)
The key to determine the best motion vector is the SAD (Sum of Absolute Differences), cf. eq. (1.1)

\[ SAD_{\text{min}} = \min_{i=0}^{N} \min_{j=0}^{N} \sum_{i} \sum_{j} |A(x+i, y+j) - B(x+i+m, y+j+n)| \] .......................... eq 1.1

\( A(x+i, y+j) \) is the pixel of macro block from current frame, \( B(x+i+m, y+j+n) \) is the pixel of a candidate matching block from previous frame, with a candidate MV (m,n), while m and n are the searching range. \( N \) is the size of macro block.

In this design, the search range is from -7 to 8, which means \( 16 \times 16 = 256 \) candidate motion vectors in search area. Size of macro block can vary from \( 16 \times 16 \), \( 8 \times 8 \) to \( 4 \times 4 \), the size of search area will then vary from \( 32 \times 32 \), \( 24 \times 24 \), to \( 16 \times 16 \).

The motion vector (m,n) can also be presented as eq 1.2.

\[ MV = 16 \ast m + n \] .................................................................................. eq 1.2

The goal of block matching is to find the matching block with the smallest SAD, thus will also yield the smallest residual. Since all the residual and motion vectors will be transmitted to entropy encoder, choosing a large partition size (e.g. \( 16 \times 16 \)) means that a small number of bits are required to signal the choice of motion vector(s) and the type of partition; however, the motion compensated residual may contain a significant amount of energy in frame areas with high detail; choosing a small partition size (e.g. \( 8 \times 8 \), \( 4 \times 4 \)) may give a lower-energy residual after motion compensation but requires a larger number of bits to signal the motion vectors and choice of partition(s).

In general, a large partition size is appropriate for homogeneous areas of the frame and a small partition size may be beneficial for detailed areas. As a direct reflection the residual, Min SAD is used to determine if smaller block size processing is needed. If the min SAD of a larger block surpasses the threshold, smaller blocks will be processed.
1.3 Description of global VLSI Block Diagram

A global block diagram of the architecture is shown in Fig. 3.1. It basically consists 4-pipelined blocks that process 16x16, 8x8, 4x4 and quarter-pixel motion vectors, with handshake and data-transform elements between them.

![Global Block Diagram](image)

Every processing block in this architecture has roughly the same processing time in order to achieve maximum pipeline efficiency. Only local access from small size memories are needed except of the 16x16 processing block. Thus, drastically decreases the memory bandwidth and pin count, which also decreases the power consumption.

For real time processing with 640x480 resolution video at 25/sec frame rate,

\[ 640 \times 480 \times 25 \times 16 = 122 \text{M clock cycles are needed per sec. (for every pixel, 16 clock cycles are needed in this design)} \]

After synthesis, a clock frequency at 125MHz is achieved, which is fast enough for real time processing.
CHAPTER 2: Literature review

2.1 Origin of H.264

Broadcast television and home entertainment are being revolutionized by the invention of digital TV and DVD-video. These applications and many more were made possible by the standardization of video compression technology. The next standard in the MPEG series, MPEG4, is enabling a new generation of internet-based video applications while the ITU-T H.263 standard for video compression is now widely used in videoconferencing systems.

MPEG4 and H.263 are standards that are based on video compression ("video coding") technology from circa. 1995. The groups responsible for these standards, the Motion Picture Experts Group and the Video Coding Experts Group (MPEG and VCEG) are in the final stages of developing a new standard that promises to significantly outperform MPEG4 and H.263, providing better compression of video images together with a range of features supporting high-quality, low-bit-rate streaming video. The new standard, "Advanced Video Coding" (AVC) is also known by its old working title, H.26L and by its ITU document number, H.264. [3]

2.2 H.264 Codec [11]

In common with earlier standards (such as MPEG1, MPEG2 and MPEG4), the H.264 draft standard does not explicitly define a CODEC (enCODer / DECoder pair). Rather, the standard defines the syntax of an encoded video bit-stream together with the method of decoding this bit-stream. In practice, however, a compliant encoder and decoder are likely to include the functional elements shown in Figure 2-1 and Figure 2-2. There is scope for considerable variation in the structure of the CODEC. The basic functional elements (prediction, transform, quantization, entropy encoding) are little different from previous standards (MPEG1, MPEG2, MPEG4, H.261, H.263); the important changes in H.264 occur in the details of each functional element. The Encoder (Figure 2-1) includes
two dataflow paths, a “forward” path (left to right) and a “reconstruction” path (right to left). The dataflow path in the Decoder (Figure 2-2) is shown from right to left to illustrate the similarities between Encoder and Decoder.

![Figure 2-1 AVC Encoder [4]](image1)

![Figure 2-2 AVC Decoder [4]](image2)

2.2.1 Encoder (forward path)
An input frame Fn is presented for encoding. The frame is processed in units of a macroblock (corresponding to 16x16 pixels in the original image). Each macroblock is encoded. In either case, a prediction macroblock P is formed based on a reconstructed frame. P is formed by motion-compensated prediction from one or more reference frame(s). In the Figures, the reference frame is shown as the previous encoded frame F'n-1; however, the prediction for each macroblock may be formed from one or two past or future frames (in time order) that have already been encoded and reconstructed. The prediction P is subtracted from the current macroblock to produce a residual or difference macroblock Dn. This is transformed (using a block transform) and quantized to give X, a
set of quantized transform coefficients. These coefficients are re-ordered and entropy encoded. The entropy encoded coefficients, together with side information required to decode the macroblock (such as the macroblock prediction mode, quantizer step size, motion vector information describing how the macroblock was motion-compensated, etc) form the compressed bitstream. This is passed to a Network Abstraction Layer (NAL) for transmission or storage. [5]

2.2.2 Encoder (reconstruction path)

The quantized macroblock coefficients X are decoded in order to reconstruct a frame for encoding of further macroblocks. The coefficients X are re-scaled (Q-1) and inverse transformed (T-1) to produce a difference macroblock Dn'. This is not identical to the original difference macroblock Dn; the quantization process introduces losses and so Dn' is a distorted version of Dn.

The prediction macroblock P is added to Dn' to create a reconstructed macroblock uF'n (a distorted version of the original macroblock). A filter is applied to reduce the effects of blocking distortion and reconstructed reference frame is created from a series of macroblocks F'n.

2.2.3 Decoder

The decoder receives a compressed bitstream from the NAL. The data elements are entropy decoded and reordered to produce a set of quantized coefficients X. These are rescaled and inverse transformed to give Dn' (this identical to the Dn' shown in the Encoder). Using the header information decoded from the bitstream, the decoder creates a prediction macroblock P, identical to the original prediction P formed in the encoder. P is added to Dn' to produce uF'n which this is filtered to create the decoded macroblock F'n.

It should be clear from the Figures and from the discussion above that the purpose of the reconstruction path in the encoder is to ensure that both encoder and decoder use identical reference frames to create the prediction P. If this is not the case, then the predictions P in encoder and decoder will not be identical, leading to an increasing error or "drift" between the encoder and decoder. [5]
2.3 Motion estimation and compensation

2.3.1 Introduction

The AVC CODEC uses block-based motion compensation, the same principle adopted by every major coding standard since H.261. Important differences from earlier standards include the support for a range of block sizes (down to 4x4) and fine sub-pixel motion vectors (1/4 pixel).

2.3.2 Tree structured motion compensation

AVC supports motion compensation block sizes ranging from 16x16 to 4x4 luminance samples with many options between the two. The luminance component of each macroblock (16x16 samples) may be split up in 4 ways as shown in Figure 3-1: 16x16, 16x8, 8x16 or 8x8. Each of the sub-divided regions is a macroblock partition. If the 8x8 mode is chosen, each of the four 8x8 macroblock partitions within the macroblock may be split in a further 4 ways as shown in Figure 2-2: 8x8, 8x4, 4x8 or 4x4 (known as macroblock sub-partitions). These partitions and sub-partitions give rise to a large number of possible combinations within each macroblock. This method of partitioning macroblocks into motion compensated sub-blocks of varying size is known as tree structured motion compensation.

![Figure 2-3 Macro block partitions: 16x16, 8x16, 16x8, 8x8 [5]](image)
A separate motion vector is required for each partition or sub-partition. Each motion vector must be coded and transmitted; in addition, the choice of partition(s) must be encoded in the compressed bit-stream. Choosing a large partition size (e.g. 16x16, 16x8, 8x16) means that a small number of bits are required to signal the choice of motion vector(s) and the type of partition; however, the motion compensated residual may contain a significant amount of energy in frame areas with high detail. Choosing a small partition size (e.g. 8x4, 4x4, etc.) may give a lower-energy residual after motion compensation but requires a larger number of bits to signal the motion vectors and choice of partition(s). The choice of partition size therefore has a significant impact on compression performance. In general, a large partition size is appropriate for homogeneous areas of the frame and a small partition size may be beneficial for detailed areas. [6]

In our design, 8x4 and 16x8 blocks are not used in order to reduce the complexity.

Example: Figure 2-5 shows a residual frame (without motion compensation). The AVC reference encoder selects the “best” partition size for each part of the frame, i.e. the partition size that minimizes the coded residual and motion vectors. The macro block partitions chosen for each area are shown superimposed on the residual frame. In areas where there is little change between the frames (residual appears grey), a 16x16 partition is chosen; in areas of detailed motion (residual appears black or white), smaller partitions are more efficient. [6]
2.3.3 Sub-pixel motion vectors

Each partition in an inter-coded macro block is predicted from an area of the same size in a reference picture. The offset between the two areas (the motion vector) has 1/4-pixel resolution (for the luma component). Figure 3-1 gives an example. A 4x4 sub-partition in the current frame (a) is to be predicted from a neighboring region of the reference picture. If the horizontal and vertical components of the motion vector are integers (b), the relevant samples in the reference block actually exist (grey dots). If one or both vector components are fractional values (c), the prediction samples (grey dots) are generated by interpolation between adjacent samples in the reference frame (white dots).
Sub-pixel motion compensation can provide significantly better compression performance than integer-pixel compensation, at the expense of increased complexity. Quarter-pixel accuracy outperforms half-pixel accuracy.

2.4 Existing popular ME/MC Algorithms

Motion estimation has proven to be effective to exploit the temporal redundancy of video sequences and is therefore a central part of the ISO/IEC MPEG-1, MPEG-2, MPEG-4 and the H.263 and H.264 video compression standards. These video compression schemes are based on a block-based hybrid-coding concept, which was extended within the MPEG-4 standardization effort to support arbitrarily-shaped video objects.

Motion estimation algorithms have attracted much attention in research and industry because of these reasons:

1. It is the computational most demanding algorithm of a video encoder (about 60%-80% of the total computation time) [7], which limits the performance of the encoder in terms of encoding speed.
2. The motion estimation algorithm has a high impact on the visual performance of an encoder for a given bit rate.
3. Finally, the method to extract motion vectors from the video material is not standardized, thus being open to competition.
Full-search block-matching algorithm is the most hardware friendly algorithm. [8] However, there do have some hardware based fast search algorithm, but none of them supports variable block size [9].

Four example fast search algorithms are briefly described below:

1. 2D logarithmic searches
2DLOG is a fast search algorithm based on minimum distortion, in which the distortion metric is only calculated for sparse sampling of the full-search area. The step size of the search area is reduced by n/2 with every search step.

2. Three Step Search
TSS uses a similar structure as 2DLOG but with the use of SAD. TSS is one of the most popular fast motion estimation algorithms requiring a fixed number of 25 search steps and is often used as reference.

NTSS adds additional checking points around zero MV of the first step of TSS, it is more robust and yield less errors than TSS. However for larger MV NTSS will take more computation power.

4. Diamond search
DS is a search algorithm that has a diamond shape-searching pattern, there is advanced DS search that has different size of diamond shape searching areas [11]. But there is no hardware architecture for this algorithm that also supports variable block size.
CHAPTER 3: Dataflow and VLSI Architecture Design

The 16x16 dataflow is shown in Table 3-1, after some modification, 8x8 and 4x4 dataflow can be shown in Table 3-2 and Table 3-3. These are all based on full search block matching Algorithm.

3.1 Dataflow Diagram for 16x16, 8x8, and 4x4 Motion Estimation

In our design, the first processing block size is 16x16, and the tracking range is -8 - +7 pixels. Following data flow is presented to solve first line of 16 possible matching blocks with 16 Processing Elements [12]. Current frame data will be shift through all the Processing Elements and the previous frame data will be broadcasted.

The tables below assume the current block (a) starts from (0,0), and the search area (b) in previous frame also starts from (0,0). In actual design the start points may vary depending on the location of the block.

Table 3-1 shows the dataflow for 16x16 block matching. “a” and “b” will be providing to the PE array for calculation.
<table>
<thead>
<tr>
<th>Data Sequence</th>
<th>PE0</th>
<th>PE1</th>
<th>PE14</th>
<th>PE15</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a(0,0)</td>
<td>b(0,0)</td>
<td>a(0,0)-b(0,0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a(0,1)</td>
<td>b(0,1)</td>
<td>a(0,1)-b(0,1)</td>
<td>a(0,0)-b(0,1)</td>
<td></td>
</tr>
<tr>
<td>a(0,2)</td>
<td>b(0,2)</td>
<td>a(0,2)-b(0,2)</td>
<td>a(0,1)-b(0,2)</td>
<td></td>
</tr>
<tr>
<td>a(0,14)</td>
<td>b(0,14)</td>
<td>a(0,14)-b(0,14)</td>
<td>a(0,13)-b(0,14)</td>
<td>a(0,0)-b(0,14)</td>
</tr>
<tr>
<td>a(0,15)</td>
<td>b(0,15)</td>
<td>a(0,15)-b(0,15)</td>
<td>a(0,14)-b(0,15)</td>
<td>a(0,1)-b(0,15)</td>
</tr>
<tr>
<td>a(1,0)</td>
<td>b(1,0)</td>
<td>b(0,16)</td>
<td>a(1,0)-b(1,0)</td>
<td>a(0,15)-b(0,16)</td>
</tr>
<tr>
<td>a(1,1)</td>
<td>b(1,1)</td>
<td>b(0,17)</td>
<td>a(1,1)-b(1,1)</td>
<td>a(1,0)-b(1,1)</td>
</tr>
<tr>
<td>a(1,2)</td>
<td>b(1,2)</td>
<td>b(0,18)</td>
<td>a(1,2)-b(1,2)</td>
<td>a(1,1)-b(1,2)</td>
</tr>
<tr>
<td>a(1,14)</td>
<td>b(1,14)</td>
<td>b(0,30)</td>
<td>a(1,14)-b(1,14)</td>
<td>a(1,13)-b(1,14)</td>
</tr>
<tr>
<td>a(1,15)</td>
<td>b(1,15)</td>
<td>b(0,31)</td>
<td>a(1,15)-b(1,15)</td>
<td>a(1,14)-b(1,15)</td>
</tr>
<tr>
<td>a(2,0)</td>
<td>b(2,0)</td>
<td>b(1,16)</td>
<td>a(2,0)-b(2,0)</td>
<td>a(1,15)-b(1,16)</td>
</tr>
<tr>
<td>a(2,1)</td>
<td>b(2,1)</td>
<td>b(1,17)</td>
<td>a(2,1)-b(2,1)</td>
<td>a(2,0)-b(2,1)</td>
</tr>
<tr>
<td>a(15,0)</td>
<td>b(15,0)</td>
<td>b(14,16)</td>
<td>a(15,0)-b(15,0)</td>
<td>a(14,15)-b(14,16)</td>
</tr>
<tr>
<td>a(15,1)</td>
<td>b(15,1)</td>
<td>b(14,17)</td>
<td>a(15,1)-b(15,1)</td>
<td>a(15,0)-b(15,1)</td>
</tr>
<tr>
<td>a(15,15)</td>
<td>b(15,15)</td>
<td>b(14,31)</td>
<td>a(15,14)-b(15,15)</td>
<td>a(15,13)-b(15,15)</td>
</tr>
<tr>
<td>b(15,16)</td>
<td>b(15,16)</td>
<td>b(15,16)</td>
<td>a(15,15)-b(15,16)</td>
<td>a(15,14)-b(15,16)</td>
</tr>
<tr>
<td>b(15,17)</td>
<td>b(15,17)</td>
<td>b(15,17)</td>
<td>a(15,16)-b(15,17)</td>
<td>a(15,15)-b(15,17)</td>
</tr>
<tr>
<td>b(15,29)</td>
<td>b(15,29)</td>
<td>b(15,29)</td>
<td>a(15,28)-b(15,29)</td>
<td>a(15,14)-b(15,29)</td>
</tr>
<tr>
<td>b(15,30)</td>
<td>b(15,30)</td>
<td>b(15,30)</td>
<td>a(15,30)-b(15,30)</td>
<td>a(15,15)-b(15,30)</td>
</tr>
<tr>
<td>b(15,31)</td>
<td>b(15,31)</td>
<td>b(15,31)</td>
<td>a(15,31)-b(15,31)</td>
<td>a(15,16)-b(15,31)</td>
</tr>
</tbody>
</table>

Table 3-1 Dataflow for 16x16 block size, assuming both start point is (0,0)
Similarly for 8x8 and 4x4 the data flows are shown in following tables.

<table>
<thead>
<tr>
<th>Cycle Time</th>
<th>Data Sequence c</th>
<th>p</th>
<th>p1</th>
<th>p2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a(0,0)</td>
<td>b(0,0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>a(0,1)</td>
<td>b(0,1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>a(0,2)</td>
<td>b(0,2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>a(0,6)</td>
<td>b(0,6)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>a(0,7)</td>
<td>b(0,7)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8+0</td>
<td>a(1,0)</td>
<td>b(1,0)</td>
<td>b(0,8)</td>
<td></td>
</tr>
<tr>
<td>8+1</td>
<td>a(1,1)</td>
<td>b(1,1)</td>
<td>b(0,9)</td>
<td></td>
</tr>
<tr>
<td>8+2</td>
<td>a(1,2)</td>
<td>b(1,2)</td>
<td>b(0,10)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8+6</td>
<td>a(1,6)</td>
<td>b(1,6)</td>
<td>b(0,14)</td>
<td></td>
</tr>
<tr>
<td>8+7</td>
<td>a(1,7)</td>
<td>b(1,7)</td>
<td>b(0,15)</td>
<td></td>
</tr>
<tr>
<td>2*8+0</td>
<td>a(2,0)</td>
<td>b(2,0)</td>
<td>b(1,8)</td>
<td>b(0,16)</td>
</tr>
<tr>
<td>2*8+1</td>
<td>a(2,1)</td>
<td>b(2,1)</td>
<td>b(1,9)</td>
<td>b(0,17)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7*8+0</td>
<td>a(7,0)</td>
<td>b(7,0)</td>
<td>b(6,8)</td>
<td>b(5,16)</td>
</tr>
<tr>
<td>7*8+1</td>
<td>a(7,1)</td>
<td>b(7,1)</td>
<td>b(6,9)</td>
<td>b(5,17)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7*8+7</td>
<td>a(7,7)</td>
<td>b(7,7)</td>
<td>b(6,15)</td>
<td>b(5,23)</td>
</tr>
<tr>
<td>8*8+0</td>
<td>b(7,8)</td>
<td>b(6,16)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8*8+1</td>
<td>b(7,9)</td>
<td>b(6,17)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8*8+5</td>
<td>b(7,13)</td>
<td>b(6,21)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8*8+6</td>
<td>b(7,14)</td>
<td>b(6,22)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8*8+7</td>
<td>b(7,15)</td>
<td>b(6,23)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>b(7,16)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>b(7,17)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>b(7,23)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3-2 Dataflow for 8x8 block size, assuming both start point are (0,0)
<table>
<thead>
<tr>
<th>Cycle Time</th>
<th>Data Sequence</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a(0,0)</td>
<td>b(0,0)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>a(0,1)</td>
<td>b(0,1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>a(0,2)</td>
<td>b(0,2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>a(0,3)</td>
<td>b(0,3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4+0</td>
<td>a(1,0)</td>
<td>b(1,0)</td>
<td>b(0,4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4+1</td>
<td>a(1,1)</td>
<td>b(1,1)</td>
<td>b(0,5)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4+2</td>
<td>a(1,2)</td>
<td>b(1,2)</td>
<td>b(0,6)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4+3</td>
<td>a(1,3)</td>
<td>b(1,3)</td>
<td>b(0,7)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2*4+0</td>
<td>a(2,0)</td>
<td>b(2,0)</td>
<td>b(1,4)</td>
<td>b(0,8)</td>
<td></td>
</tr>
<tr>
<td>2*4+1</td>
<td>a(2,1)</td>
<td>b(2,1)</td>
<td>b(1,5)</td>
<td>b(0,9)</td>
<td></td>
</tr>
<tr>
<td>2*4+2</td>
<td>a(2,2)</td>
<td>b(2,2)</td>
<td>b(1,6)</td>
<td>b(0,10)</td>
<td></td>
</tr>
<tr>
<td>2*4+3</td>
<td>a(2,3)</td>
<td>b(2,3)</td>
<td>b(1,7)</td>
<td>b(0,11)</td>
<td></td>
</tr>
<tr>
<td>3*4+0</td>
<td>a(3,0)</td>
<td>b(3,0)</td>
<td>b(2,4)</td>
<td>b(1,8)</td>
<td>b(0,12)</td>
</tr>
<tr>
<td>3*4+1</td>
<td>a(3,1)</td>
<td>b(3,1)</td>
<td>b(2,5)</td>
<td>b(1,9)</td>
<td>b(0,13)</td>
</tr>
<tr>
<td>3*4+2</td>
<td>a(3,2)</td>
<td>b(3,2)</td>
<td>b(2,6)</td>
<td>b(1,10)</td>
<td>b(0,14)</td>
</tr>
<tr>
<td>3*4+3</td>
<td>a(3,3)</td>
<td>b(3,3)</td>
<td>b(2,7)</td>
<td>b(1,11)</td>
<td>b(0,15)</td>
</tr>
<tr>
<td>4*4+0</td>
<td></td>
<td>b(3,4)</td>
<td>b(2,8)</td>
<td>b(1,12)</td>
<td>b(0,16)</td>
</tr>
<tr>
<td>4*4+1</td>
<td></td>
<td>b(3,5)</td>
<td>b(2,9)</td>
<td>b(1,13)</td>
<td>b(0,17)</td>
</tr>
<tr>
<td>4*4+2</td>
<td></td>
<td>b(3,6)</td>
<td>b(2,10)</td>
<td>b(1,14)</td>
<td>b(0,18)</td>
</tr>
<tr>
<td>4*4+3</td>
<td></td>
<td>b(3,7)</td>
<td>b(2,11)</td>
<td>b(1,15)</td>
<td>b(0,19)</td>
</tr>
<tr>
<td>5*4+0</td>
<td></td>
<td>b(3,8)</td>
<td>b(2,12)</td>
<td>b(1,16)</td>
<td></td>
</tr>
<tr>
<td>5*4+1</td>
<td></td>
<td>b(3,9)</td>
<td>b(2,13)</td>
<td>b(1,17)</td>
<td></td>
</tr>
<tr>
<td>5*4+2</td>
<td></td>
<td>b(3,10)</td>
<td>b(2,14)</td>
<td>b(1,18)</td>
<td></td>
</tr>
<tr>
<td>5*4+3</td>
<td></td>
<td>b(3,11)</td>
<td>b(2,15)</td>
<td>b(1,19)</td>
<td></td>
</tr>
<tr>
<td>6*4+0</td>
<td></td>
<td>b(3,12)</td>
<td>b(2,16)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6*4+1</td>
<td></td>
<td>b(3,13)</td>
<td>b(2,17)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6*4+2</td>
<td></td>
<td>b(3,14)</td>
<td>b(2,18)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6*4+3</td>
<td></td>
<td>b(3,15)</td>
<td>b(2,19)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7*4+0</td>
<td></td>
<td>b(3,16)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7*4+1</td>
<td></td>
<td>b(3,17)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7*4+2</td>
<td></td>
<td>b(3,18)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7*4+3</td>
<td></td>
<td>b(3,19)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3-3 Dataflow for 4x4 block size, assuming both start points are (0,0)

3.2 VLSI Architectures for 16x16, 8x8, and 4x4 Motion Estimation

From the Dataflow tables VLSI architecture was developed. The following figure shows the architecture of a 16x16 processing block.
8x8 and 4x4 processing blocks can use the same architecture, after changing the Interconnection, Controller and Address generator a little.

Between 16x16, 8x8, 4x4, and fractional MV blocks there are bridging units that coordinate and transfer data between the 4 processing blocks.
The bridging units can be shown in the figure below.

![Bridging Architecture Diagram](image)

Figure 3-2 Bridging Architecture between 2 Different block size stages

The bridging units between 8x8 and 4x4, 4x4 and fractional MV. have the similar architectures.

The design details of all these blocks will be discussed in Register Transfer Level code chapter.

3.3 Dataflow Diagram for Fractional Motion Estimation

The fractional motion estimation dataflow includes interpolations. The following Table shows the dataflow in fractional motion estimation. Here start point of (0,0) is assumed.
<table>
<thead>
<tr>
<th>Input</th>
<th>IP1</th>
<th>IP2</th>
<th>PE0</th>
<th>PE1</th>
</tr>
</thead>
<tbody>
<tr>
<td>B(0,0), B(1,0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$B_0^0 (0,0)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$B_0^1 (0,0)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$B_0^2 (0,0)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B(0,1), B(1,1)</td>
<td>$B_0^3 (0,0)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$B_0^0 (0,1)$</td>
<td>$B_0^0 (0,0)$</td>
<td>$B_0^2 (0,0)$</td>
<td>$B_0^2 (0,0)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$A(0,0)-B_0^0 (0,0)$</td>
<td></td>
<td>$A(0,0)-B_0^0 (0,0)$</td>
</tr>
<tr>
<td></td>
<td>$B_0^1 (0,1)$</td>
<td>$B_0^1 (0,0)$</td>
<td>$B_0^1 (0,0)$</td>
<td>$B_0^1 (0,0)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$A(0,0)-B_0^1 (0,0)$</td>
<td></td>
<td>$A(0,0)-B_0^1 (0,0)$</td>
</tr>
<tr>
<td></td>
<td>$B_0^2 (0,1)$</td>
<td>$B_0^2 (0,0)$</td>
<td>$B_0^2 (0,0)$</td>
<td>$B_0^2 (0,0)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$A(0,0)-B_0^2 (0,0)$</td>
<td></td>
<td>$A(0,0)-B_0^2 (0,0)$</td>
</tr>
<tr>
<td>B(0,2), B(1,2)</td>
<td>$B_0^3 (0,1)$</td>
<td>$B_0^3 (0,0)$</td>
<td>$B_0^3 (0,0)$</td>
<td>$B_0^3 (0,0)$</td>
</tr>
<tr>
<td></td>
<td>$B_0^0 (0,2)$</td>
<td>$B_0^0 (0,1)$</td>
<td>$B_0^0 (0,1)$</td>
<td>$B_0^0 (0,1)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$A(0,0)-B_0^0 (0,0)$</td>
<td></td>
<td>$A(0,0)-B_0^0 (0,0)$</td>
</tr>
<tr>
<td></td>
<td>$B_0^1 (0,2)$</td>
<td>$B_0^1 (0,1)$</td>
<td>$B_0^1 (0,1)$</td>
<td>$B_0^1 (0,1)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$A(0,0)-B_0^1 (0,0)$</td>
<td></td>
<td>$A(0,0)-B_0^1 (0,0)$</td>
</tr>
<tr>
<td></td>
<td>$B_0^2 (0,2)$</td>
<td>$B_0^2 (0,1)$</td>
<td>$B_0^2 (0,1)$</td>
<td>$B_0^2 (0,1)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$A(0,0)-B_0^2 (0,0)$</td>
<td></td>
<td>$A(0,0)-B_0^2 (0,0)$</td>
</tr>
<tr>
<td>B(0,16), B(1,16)</td>
<td>$B_0^0 (0,16)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$B_0^1 (0,16)$</td>
<td>$B_0^1 (0,15)$</td>
<td>$B_0^1 (0,15)$</td>
<td>$B_0^1 (0,15)$</td>
</tr>
<tr>
<td></td>
<td>$B_0^2 (0,16)$</td>
<td>$B_0^2 (0,15)$</td>
<td>$B_0^2 (0,15)$</td>
<td>$B_0^2 (0,15)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$A(0,15)-B_0^0 (0,15)$</td>
<td></td>
<td>$A(0,15)-B_0^0 (0,15)$</td>
</tr>
<tr>
<td></td>
<td>$B_0^3 (0,16)$</td>
<td>$B_0^3 (0,15)$</td>
<td>$B_0^3 (0,15)$</td>
<td>$B_0^3 (0,15)$</td>
</tr>
<tr>
<td></td>
<td>$B_0^4 (0,15)$</td>
<td>$B_0^4 (0,15)$</td>
<td>$B_0^4 (0,15)$</td>
<td>$B_0^4 (0,15)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$A(0,15)-B_0^1 (0,15)$</td>
<td></td>
<td>$A(0,15)-B_0^1 (0,15)$</td>
</tr>
<tr>
<td></td>
<td>$B_0^5 (0,15)$</td>
<td>$B_0^5 (0,15)$</td>
<td>$B_0^5 (0,15)$</td>
<td>$B_0^5 (0,15)$</td>
</tr>
<tr>
<td></td>
<td>$B_0^6 (0,15)$</td>
<td>$B_0^6 (0,15)$</td>
<td>$B_0^6 (0,15)$</td>
<td>$B_0^6 (0,15)$</td>
</tr>
</tbody>
</table>

Table 3-4 Dataflow diagram for Fractional Motion Estimation [12]
IP1 is the functional block that interpolates between the pixel rows, and IP2 is the functional block to interpolate between pixel columns. There are only 4 PEs but with 4 latches inside each PE they can process 16 blocks at the same time.

There are total 64 searches for quarter-pixel precision, namely the combination of -1, -0.75, -0.5, -0.25, 0, 0.25, 0.5, and 0.75.

3.4 The VLSI architecture of Fractional ME

The VLSI architecture of Fractional Pixel motion estimation [1] can be shown in figure 3-3.

![Figure 3-3 VLSI Architecture for Fractional ME](image)

IP1 and IP2 interpolate the entire search area, and the pixel data are processed in PE array and comparator.
CHAPTER 4: Behavior VHDL code

High-level behavior code was written for algorithm verification, also, it provides the "known good" motion vectors to test the Register Transfer Level codes. As show in the following code, 2 neighborhood frames were used to calculate the motion vectors, separated processes were used to calculate the 16x16, 8x8, 4x4, and fractional motion vectors. Compensations were also performed during each process. The motion vectors are directed into *.pgm files that can be access by text editor.

Behavior code is just for verification purpose, it cannot be synthesized. For VLSI hardware implementations, Register Transfer Level codes are needed to be developed, which will be covered in next chapter.

The source code can be found in Appendix B – 1.

The following segment of behavior code contains the key loops for 16x16 motion estimation. Variable m and n represent the number of the block, in this case, the frame size is 256x256, so there are 16 blocks in both row and column. Variable p and q represent the row and column number of the possible matching blocks, in this case, we have a search area of 32x32, so there are 256 possible matching blocks. Variable i and j represent the locations of pixels in current possible matching block.

```
for m in 1 to 16 loop
  for n in 1 to 16 loop
    for p in 0 to 15 loop
      for q in 0 to 15 loop
        for i in ((m-1)*16+p-8) to ((m-1)*16+p+7) loop
          for j in ((n-1)*16+q-8) to ((n-1)*16+q+7) loop
            k := (i+8-p); l := (j+8-q);
            if (i>0 and j>0 and i<256 and j<256) then
              if (prev_frame(i,j)>cur_frame(k,l)) then
                ad := prev_frame(i,j)-cur_frame(k,l);
              else ad := cur_frame(k,l)-prev_frame(i,j);
              end if;
              temp_sad_array(p,q) := temp_sad_array(p,q) + ad;
            else temp_sad_array(p,q) := 65535;
            end if;
          end loop;
        end loop;
      end loop;
    end loop;
  end loop;
end loop;
```
5.1 16X16 Processing block

5.1.1 AGU

AGU’s purpose is to generate serial of memory address and necessary control signal. As shown below, inputs are from controller, start point generator (SPU) and bridging units.

Input ports:
Start_c_row, Start_c_col, Start_p_row, start_p_col are from SPU.

Reset and Start are from Controller.

Start_Transfer and ME_to_sub_ready are from Bridging Units.

Output ports:
Add_c_row, add_c_col, add_p_row, add_p_col, add_p1_row and add_p1_col are connected to memory address lines.

Done_block is a feedback to Controller, when the address generations of a 16x16 block and its search area are done.

Start_cmp, Reset_cmp are control signals for Comparator.

Z_cp, Z_p1 are the signals to control the muxes which are used to determined either the actually pixel data or a “00000000” will go to the Interconnection.
The source code can be found in Appendix B – 2.1. Synthesis result can be found in 11 – 1.

5.1.2 Comparator

Comparator is the unit that collects the SADs from each PE and then output the smallest SAD and the motion vector related to the SAD. An 8-bit logic vector represents the motion vector because the search area contains 256 possible matching blocks.

Input ports:

PE0, PE1, …… PE15 are the inputs of SADs from the 16 PEs.

SAD_threshold is an input that specify whether the block need further 8x8 processing or not. If the smallest SAD is bigger than the threshold SAD, the block will be processing in 8x8 mode, if less than the threshold SAD, the block will no be processed in 8x8 mode.
Reset_cmp and Start_cmp are control signals from AGU, to start the comparison or reset the comparator.

Output ports:

SAD, min_SAD_out are the SAD outputs, while min_SAD_out is a test signal that updates at every round of “table 3-1” instead of the done_block in AGU.

Mv_out, and mv_out_sub are the motion vector outputs. Mv_out is the one that doesn't need 8x8 processing, while mv_out_sub will need further 8x8 processing.

![Comparator Diagram](image)

Figure 5-2 Comparator
FSM is used to design the comparator. Idle and initial states are used to control the comparator, while compare and output_mv states are used to compare SADs and generate motion vector outputs.

The source code can be found in Appendix B – 2.2. Synthesis result can be found in Appendix A – 2.
5.1.3 Controller

Controller are the toplevel control unit which is dedicated to coordinate and generate control signals for AGU, SPU, and bridging units, also receive feedback signals from those items.

Input ports:

Reset and start are the top-level control input.

ME_to_ME_sub_ready is the feedback signal from bridging units when the data transfer between 16x16 and 8x8 blocks are done.

Output ports:

Reset_AGU, Enable_SPU, Start_AGU are the control signals for AGU and SPU.

Done_FRAME is top-level output when the prediction for the entire frame is done.

Block_num_col, Block_num_row, position are the information for SPU to generator start point for AGU. It shows the location of the current block in the frame.
FSM is used to design the controller. To get the location of each 16x16 block, 9 output states were used to represent the 9 situations where block is: upper left corner, upper edge, upper right corner, left edge, center, right edge, bottom left corner, bottom edge, bottom right corner.

The source code can be found in Appendix B – 2.3. Synthesis result can be found in Appendix A 3.

5.1.4 Interconnection

Interconnection is in between the memory and the PEs. Under controlled of AGU, Interconnection unit distribute the pixel data to PEs.

Input ports:

c, p, p1 are pixel data inputs.
Reset_dff_bit, reset_dff_pixel are the dataflow control inputs.
Output ports:
PE0c, PE0p ....... PE15c, PE15p are the outputs that connect to the 16 Pes, they are all the coordinated current frame pixel data and previous frame pixel data.

![Figure 5-4 Interconnection](image)

The Interconnection is basically made by series of Dffs and Multiplexers. Reset_dff_bit is to reset the Mux, and the Reset_dff_pixel is to reset the D ports of the Dffs.

The source code can be found in Appendix B – 2.4. Synthesis result can be found in Appendix A – 4.

5.1.5 PE (Processing Element)

PE (Processing Element) is the unit that calculates the SAD by performing absolute difference calculation and adding them.

Input ports:
A, b are the inputs of pixel data, one is from current frame, another is from previous frame.
Reset is the control signal from AGU, which will set the SAD output 0.

Output ports:
SAD, Sum of Absolute Difference is the output to Comparator.

![Diagram of Processing Element (PE)](image)

**Figure 5-5 PE (Processing Element)**

PE conducts Equation 1.1. It consist a comparator, a substractor, an adder and an accumulator.

The source code can be found in Appendix B – 2.5. Synthesis result can be found in Appendix A – 5.

5.1.6 SPU (Start Point Generator)

SPU is the unit that generates the address of start point for current block and the according search area, which works as a part of Address Generator. In this design, it is separated from Address Generator for a simpler code for the Address Generator.

Input Ports:

Block_num_col, Block_num_row, position are the inputs from Address generator, they tell the SPU which block it is.

Enable is a control input from Controller.

Output ports:
Start_c_row, Start_c_col, start_p_row, start_p_col are the address of start points for current block and search area. These outputs connect to the AGU so AGU can generate sequence of addresses.

![Start Point Generator diagram](image)

Figure 5-6 SPU (Start Point Generator)

The source code can be found in Appendix B – 2.6. Synthesis result can be found in Appendix A – 6.

5.1.7 Memory previous frame

Memory for previous frame, it reads data from image file first.

Input ports:
Add_p_row, add_p_col, add_p1_row, add_p1_col are the addresses line inputs.

Output ports:
Out_p1, out_p are the pixel data outputs.
The source code can be found in Appendix B – 2.7. Synthesis result can be found in Appendix A – 7.

5.1.8 Memory current frame

Memory for current frame, it reads data from image file first.

Input ports:
Add_c_row, add_c_col are the addresses line inputs.

Output ports:
Out_c is the pixel data outputs.
The source code can be found in Appendix B – 2.8. Synthesis result can be found in Appendix A – 8.

5.1.9 Mux between Memory and Interconnection

This Mux provides control over the data inputs of Interconnection. It determines whether the actually pixel data goes into interconnection or “00000000”.

Input ports:

Z_cp, Z_p1 are the control inputs from AGU.
In_c, in_p, in_p1 are the pixel data inputs from memories.

Output ports:

Out_c, out_p, out_p1 are the pixel data outputs to interconnection, they can either be the actual pixel datas or “00000000”.

Figure 5-8 Memory Current Frame
5.1.10 Transfer unit

This is the unit that transfer current block and search area data from 16x16 memories to 8x8 processing block’s memories, it generates the memory addresses for memories on both stages, also generates some control signals and feedback for another bridging unit.

Input ports:

Start_transfer is the control signal from ME_to_ME_sub bridging unit.

S_c_row, s_c_col, s_p_row, s_p_col are the start point of current block and search area, come from SPU.

Output ports:

C_row, c_col, p_row, p_col, sub_c_row, sub_c_col, sub_p_row, sub_p_col are the addresses output for the memories on both stages.

Read_write, me_sub_add_select1, me_sub_add_Select2 are the control signals for the memories in next stage.
ME_add_select is the control signal for the memories in previous stage.

![Data Transfer Unit Diagram]

Figure 5-10 Data Transfer Unit

The source code can be found in Appendix B – 2.10. Synthesis result can be found in Appendix A – 10.

5.1.11 Bridge unit

This is the unit that coordinates the handshake signals between 2 stages and controls the transfer unit. With the transfer unit, these 2 pipeline the stages together.

Input ports:

Reset is the control input from top level.

Mv, SAD_in are the motion estimation results from last stage.
SAD_threshold determines whether the result need further processing or not. It is a top-level input.

ME_sub_ready and ME_ready are the 2 “ready” handshake signals from both stages.

Data_ready is the feedback signal from transfer unit.

Output ports:

Need_process is the signal that indicates whether the results need further processing or not.

Start_data_transfer is the control signal for transfer unit.

ME_to_ME_sub_ready is the signal that says the data transfer between 2 stages is done.

Figure 5-11 Bridging Unit

The source code can be found in Appendix B – 2.11. Synthesis result can be found in Appendix A- 11.
5.1.12 ME top level

This is the Structure VHDL code that combines all the units into a 16x16 processing block.
The source code can be found in Appendix B– 2.12. Synthesis result can be found in Appendix A – 12.

5.2 The 8x8, 4x4 Processing blocks

They are similar as the 16x16-processing unit. Only the interconnection, Controller and AGU have some significant changes.

The 8x8 memories contain the current block data, which is a 16x16 frame, and the search area data, which is a 32x32 frame.

The 4x4 memories contain the current sub block data, which is an 8x8 frame, and the search area data, which is a 24x24 frame.

The 8x8 and 4x4 dataflow should follow the Table 3-2 and 3-3.

5.2.1 The 8x8 AGU

The difference between this AGU and the 16x16 AGU discussed in 1.1 is that this AGU generates data according to Table 3-2, following the c, p, p1 and p2 sequence.

It has an extra p2 output, along with an extra control output Z_p2.

The source code can be found in Appendix B – 2.13. Synthesis result can be found in Appendix A – 13.

5.2.2 The 8x8 Interconnection
8x8 interconnection has one extra p2 input comparing to 16x16 interconnection, it also has different dataflow inside to select the output data between p, p1 and p2. Others are same as the 16x16 interconnection.

The source code can be found in Appendix B – 2.14. Synthesis result can be found in Appendix A – 14.

5.2.3 The 8x8 Controller

Same inputs and outputs as the 16x16 controller. The major difference is that instead of 9 output states, here only 4 output states exist because only 4 sub blocks existing in the 16x16 current block.

The source code can be found in Appendix B– 2.15. Synthesis result can be found in Appendix A – 15.

5.2.4 The 4x4 AGU

It has two extra output p3 and p4 compare to 8x8 AGU, in order to generate data sequence like Table 3-3.

The source code can be found in Appendix B – 2.16. Synthesis result can be found in Appendix A – 16.

5.2.5 The 4x4 Interconnection

Interconnection has extra inputs like p3 and p4, also different dataflow inside to select the output data between p, p1, p2, p3, and p4.

The source code can be found in Appendix B– 2.17. Synthesis result can be found in Appendix A – 17.

5.2.6 The 4x4 Controller

Same as 8x8 Controller.
Overall, it can be seen that 16x16, 8x8, 4x4 has very similar architecture, they also have almost same processing times for every block, that makes the 3 very good candidates for pipeline processing. With Transfer unit and Bridging unit in between neighborhood stages, the pipeline is achieved.

The source code can be found in Appendix B – 2.18. Synthesis result can be found in Appendix A – 18.

5.3 Fractional Motion Estimation Processing block

This processing block is a lot different from the former 16x16, 8x8, and 4x4 processing blocks. First the search area is only +1/-1 pixel wider than the current block, second the search area need to be interpolated before the search starts. Following is the RTL design for the functional blocks in the fractional motion estimation-processing block.

The interpolation detail is shown in the following example:

Neighborhood pixels needed to be interpolated: A(0,0), A(0,1).

\[
\begin{align*}
A_0^0(0,0) &= A(0,0) \\
A_0^1(0,0) &= 0.75*A(0,0) + 0.25*A(0,1) \\
A_30^2(0,0) &= 0.25*A(0,0) + 0.75*A(0,1)
\end{align*}
\]

5.3.1 Frac_IP1

Interpolation unit one.

Input ports:

P, P1 are the pixel data input from the frame that is needed to be interpolated.

Start_IP1 is the control signal that starts interpolation after every 4-clock cycles.

Output ports:

Quarter_out is the interpolated (row direction) pixel data output.

The source code can be found in Appendix B– 2.19. synthesis result can be found in Appendix A – 19.
5.3.2 Frac_IP2

Interpolation unit 2.

Input ports:
P, P1 are the pixel data input from IP1 (after delays as shown in Figure 3-3).

Output ports:
Out_p1, Out_p2, Out_p3, and Out_p4 are the interpolated pixel data outputs.

The source code can be found in Appendix B – 2.20. Synthesis result can be found in Appendix A – 20.

5.3.3 Frac_AGU

Fractional AGU is different with the variable block size AGUs because it is dealing with a different search area, which had already been interpolated, its also has to be synchronized with the IP1 and IP2.

Input ports:
Start_p_row, Start_p_col are the start point inputs from Frac_SPU.
Start_AGU is the control signal from Frac_Controller.

Output ports:
Reset_PE, reset_latch, output_PE, z_c, start_cmp, start_IP1 are the control signals sending to PE, Latches, Mux, comparator and IP1.
Add_p_row,add_p_col, add_p1_row,add_p1_col,add_c_row,add_c_col are the address outputs.
Done_block_sub is the feedback signal to Frac_controller when the address generation for 1 search area is done.

The source code can be found in Appendix B – 2.21. Synthesis result can be found in Appendix A – 21.

5.3.4 Frac_Controller
Missing Page
5.3.6 Frac_Pe

Frac_Pe's only difference with regular PE is its 4 latches to store the SADs, so 1 PE can handle 4 SADs instead of 1.

Input ports:
C, P are the pixel input from memory and interpolation unit 2.
Out is the control input to let the PE pumping out the 4 SADs stored in the latches.

Output ports:
SAD4 is the serial SAD output.

The source code can be found in Appendix B – 2.24. Synthesis result can be found in Appendix A – 24.

5.3.7 Frac_SPU

Unit that generates start point for Frac_AGU.

Input ports:

Position is the input that let SPU knows which block it is, therefore SPU can generate the start point address.
Enable is the control input from Frac_controller.

Output ports:

Start_p_row, and start_p_col are the start point outputs, since there is only 1 block in Frac_mem_c there is no need for a current block start point.

The source code can be found in Appendix B – 2.25. Synthesis result can be found in Appendix A – 25.
5.3.8 Frac_Memory of current frame

Memory of current frame in Fractional ME are just a 16x16 block.

The source code can be found in Section VIII – 2.26.

5.3.9 Frac_Memory of previous frame

Memory of Previous frame in Fractional ME are just a 18x18 block, which is 1 pixel wider than current frame at every edge.

The source code can be found in Section VIII – 2.27.

5.3.10 Frac_ME Toplevel

Structure VHDL code of the top level Fractional ME.

The source code can be found in Appendix B – 2.28.
CHAPTER 6: Simulation result and analysis

6.1 Simulation (Gymnast)

The following frames have been used in the simulation. The images show the two continuous frames of a gymnast in the 28th Olympic games at Athen, Greece.

Current Frame:

![Figure 6-1 Current frame]

Previous Frame:

![Figure 6-2 Previous frame]
After running the VHDL codes using the given 2 frames, 16x16 MV array, 8x8 MV array, 4x4 MV array, and fractional MV array can all be attained as follow:

### 16x16 MV array:

<table>
<thead>
<tr>
<th>189</th>
<th>178</th>
<th>176</th>
<th>181</th>
<th>176</th>
<th>191</th>
<th>191</th>
<th>186</th>
<th>176</th>
<th>176</th>
<th>176</th>
<th>185</th>
<th>187</th>
<th>176</th>
<th>176</th>
<th>184</th>
<th>180</th>
</tr>
</thead>
<tbody>
<tr>
<td>137</td>
<td>136</td>
<td>135</td>
<td>133</td>
<td>136</td>
<td>137</td>
<td>136</td>
<td>136</td>
<td>137</td>
<td>137</td>
<td>136</td>
<td>142</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>169</td>
<td>168</td>
<td>184</td>
<td>184</td>
<td>184</td>
<td>200</td>
<td>168</td>
<td>184</td>
<td>184</td>
<td>183</td>
<td>183</td>
<td>180</td>
<td>184</td>
<td>186</td>
<td>166</td>
<td>166</td>
<td>166</td>
</tr>
<tr>
<td>185</td>
<td>195</td>
<td>200</td>
<td>184</td>
<td>184</td>
<td>184</td>
<td>184</td>
<td>184</td>
<td>183</td>
<td>183</td>
<td>183</td>
<td>180</td>
<td>200</td>
<td>184</td>
<td>184</td>
<td>184</td>
<td>184</td>
</tr>
<tr>
<td>185</td>
<td>195</td>
<td>200</td>
<td>184</td>
<td>184</td>
<td>184</td>
<td>184</td>
<td>184</td>
<td>183</td>
<td>183</td>
<td>183</td>
<td>180</td>
<td>200</td>
<td>184</td>
<td>184</td>
<td>184</td>
<td>184</td>
</tr>
<tr>
<td>185</td>
<td>195</td>
<td>200</td>
<td>184</td>
<td>184</td>
<td>184</td>
<td>184</td>
<td>184</td>
<td>183</td>
<td>183</td>
<td>183</td>
<td>180</td>
<td>200</td>
<td>184</td>
<td>184</td>
<td>184</td>
<td>184</td>
</tr>
<tr>
<td>185</td>
<td>195</td>
<td>200</td>
<td>184</td>
<td>184</td>
<td>184</td>
<td>184</td>
<td>184</td>
<td>183</td>
<td>183</td>
<td>183</td>
<td>180</td>
<td>200</td>
<td>184</td>
<td>184</td>
<td>184</td>
<td>184</td>
</tr>
<tr>
<td>185</td>
<td>195</td>
<td>200</td>
<td>184</td>
<td>184</td>
<td>184</td>
<td>184</td>
<td>184</td>
<td>183</td>
<td>183</td>
<td>183</td>
<td>180</td>
<td>200</td>
<td>184</td>
<td>184</td>
<td>184</td>
<td>184</td>
</tr>
<tr>
<td>157</td>
<td>157</td>
<td>155</td>
<td>153</td>
<td>131</td>
<td>144</td>
<td>146</td>
<td>128</td>
<td>112</td>
<td>136</td>
<td>136</td>
<td>199</td>
<td>172</td>
<td>206</td>
<td>248</td>
<td></td>
<td></td>
</tr>
<tr>
<td>157</td>
<td>157</td>
<td>155</td>
<td>153</td>
<td>131</td>
<td>144</td>
<td>146</td>
<td>128</td>
<td>112</td>
<td>136</td>
<td>136</td>
<td>199</td>
<td>172</td>
<td>206</td>
<td>248</td>
<td></td>
<td></td>
</tr>
<tr>
<td>95</td>
<td>80</td>
<td>82</td>
<td>95</td>
<td>80</td>
<td>95</td>
<td>80</td>
<td>82</td>
<td>95</td>
<td>80</td>
<td>82</td>
<td>95</td>
<td>80</td>
<td>82</td>
<td>95</td>
<td>80</td>
<td></td>
</tr>
</tbody>
</table>

### 8x8 MV array:

<table>
<thead>
<tr>
<th>153</th>
<th>145</th>
<th>144</th>
<th>144</th>
<th>144</th>
<th>144</th>
<th>144</th>
<th>144</th>
<th>144</th>
<th>144</th>
<th>144</th>
<th>144</th>
<th>144</th>
<th>144</th>
<th>144</th>
<th>144</th>
<th>144</th>
<th>144</th>
</tr>
</thead>
<tbody>
<tr>
<td>189</td>
<td>191</td>
<td>183</td>
<td>179</td>
<td>184</td>
<td>178</td>
<td>189</td>
<td>181</td>
<td>181</td>
<td>191</td>
<td>184</td>
<td>191</td>
<td>190</td>
<td>182</td>
<td>176</td>
<td>176</td>
<td>197</td>
<td>179</td>
</tr>
<tr>
<td>157</td>
<td>193</td>
<td>136</td>
<td>130</td>
<td>128</td>
<td>134</td>
<td>136</td>
<td>135</td>
<td>135</td>
<td>136</td>
<td>136</td>
<td>134</td>
<td>136</td>
<td>137</td>
<td>136</td>
<td>136</td>
<td>134</td>
<td>133</td>
</tr>
<tr>
<td>217</td>
<td>183</td>
<td>176</td>
<td>183</td>
<td>184</td>
<td>180</td>
<td>53</td>
<td>216</td>
<td>151</td>
<td>244</td>
<td>171</td>
<td>120</td>
<td>168</td>
<td>194</td>
<td>161</td>
<td>205</td>
<td>56</td>
<td>152</td>
</tr>
<tr>
<td>233</td>
<td>167</td>
<td>184</td>
<td>183</td>
<td>152</td>
<td>183</td>
<td>183</td>
<td>183</td>
<td>183</td>
<td>183</td>
<td>183</td>
<td>183</td>
<td>183</td>
<td>183</td>
<td>183</td>
<td>183</td>
<td>183</td>
<td>183</td>
</tr>
<tr>
<td>113</td>
<td>113</td>
<td>113</td>
<td>113</td>
<td>113</td>
<td>113</td>
<td>113</td>
<td>113</td>
<td>113</td>
<td>113</td>
<td>113</td>
<td>113</td>
<td>113</td>
<td>113</td>
<td>113</td>
<td>113</td>
<td>113</td>
<td>113</td>
</tr>
<tr>
<td>169</td>
<td>191</td>
<td>206</td>
<td>174</td>
<td>175</td>
<td>191</td>
<td>223</td>
<td>255</td>
<td>31</td>
<td>239</td>
<td>225</td>
<td>240</td>
<td>234</td>
<td>240</td>
<td>246</td>
<td>246</td>
<td>246</td>
<td>246</td>
</tr>
<tr>
<td>233</td>
<td>240</td>
<td>146</td>
<td>168</td>
<td>142</td>
<td>133</td>
<td>175</td>
<td>138</td>
<td>9</td>
<td>67</td>
<td>107</td>
<td>200</td>
<td>104</td>
<td>120</td>
<td>56</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>159</td>
<td>192</td>
<td>130</td>
<td>140</td>
<td>138</td>
<td>222</td>
<td>208</td>
<td>252</td>
<td>248</td>
<td>254</td>
<td>245</td>
<td>244</td>
<td>244</td>
<td>244</td>
<td>244</td>
<td>244</td>
<td>244</td>
<td>244</td>
</tr>
<tr>
<td>137</td>
<td>139</td>
<td>130</td>
<td>128</td>
<td>129</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>95</td>
<td>91</td>
<td>80</td>
<td>82</td>
<td>95</td>
<td>80</td>
<td>95</td>
<td>80</td>
<td>82</td>
<td>95</td>
<td>80</td>
<td>82</td>
<td>95</td>
<td>80</td>
<td>82</td>
<td>95</td>
<td>80</td>
<td>82</td>
</tr>
</tbody>
</table>

### 4x4 MV array is included in the attachment CD.
Quarter-pixel MV array is included in the attachment CD.

The simulation results from Behavior VHDL code and RTL VHDL code match perfectly.

The following figure show the residual frames. Figure 6-3 shows the residual frame without any ME; figure 6-4 shows the residual frame with 16x16 ME; figure 6-5 shows the residual frame with 8x8 ME; and figure 6-6 shows the residual frame with 4x4 ME.

The results are significant. The residual from more precise compensation are smaller, Figure 6-6 contains less energy than Figure 6-5, and Figure 6-5 contains less energy than Figure 6-4. Hence, Figure 6-6 will result in smallest bit stream after Discrete Cosine Transform.

![Image of a page from a document with table and text]
Residual Frame without motion estimation.

Figure 6-3 residual frame without ME

Residual Frame after 16x16 motion compensation:

Figure 6-4 16x16 residual frame
Residual Frame after 8x8 motion compensation:

Figure 6-5 8x8 residual frame

Residual Frame after 4x4 motion compensation:

Figure 6-6 4x4 residual frame

6.2 Simulation (Artist with big movement)

These 2 frames have relative big differences, we will see how the residual is when there is major movement. Figure 6-7 and 6-8 are two continuous frames. The residuals of different block size are given in Figure 6-9, 6-10, 6-11 and 6-12. The residual with the 4x4 ME has the least energy level.
Figure 6-10 Residual frame with 16x16 ME

Figure 6-11 Residual frame with 8x8 ME

Figure 6-12 Residual frame with 4x4 ME
The results are significant. The residual from more precise compensation contain less energy.
CHAPTER 7: Synthesis of the RTL VHDL codes

7.1 Constraints for Synthesis

The Register Transfer Level VHDL codes have been synthesized by Synopsys Design Compiler. Blank memory block has been used to avoid the long synthesis time. The following constraints were applied to the design. The worst operation condition is 1.62 Volt, which is 10% lower than normal voltage level, and 125 degree centigrade. The clock period is set at 7ns, anything lower than 7ns will cause timing violation. The input and output ports delays are also set. TSMC 0.25 um library is linked to this design.

reset_design
create_clock -period 7 -name my_clk [get_ports me_clk]
set_dont_touch_network [get_clocks my_clk]
set_input_delay 1 -max -clock my_clk [get_ports me_reset]
set_input_delay 1 -max -clock my_clk [get_ports me_start]
set_output_delay 1 -max -clock my_clk [all_outputs]
set_input_delay 0.2 -min -clock my_clk [all_inputs]
set_output_delay 0.3 -min -clock my_clk [all_outputs]
set_operating_conditions -max slow_125_1.62
set all_in_ex_clk [remove_from_collection [all_inputs] [get_ports me_clk]]
set_driving_cell -lib_cell fdeflal -pin Q $all_in_ex_clk
set_max_cap [expr [load_of ssc_core_slow/and2al/A]*5]
set_max_capacitance $max_cap $all_in_ex_clk
set_load [expr 3 * $max_cap] [all_outputs]

7.2 Area report

The Area after synthesis is 141231um². As the area report shows

The following logs are the Area report.

******************************************************
Report : area
Design : MEnoMEM
Version: 2001.08-SP1
Date : Tue May 18 11:21:22 2004
******************************************************

Library(s) Used:

ssc_core_slow (File: /home/xxl7341/CHIP_2002.05/libs/core_slow.db)
Number of ports: 262  
Number of nets: 22244  
Number of cells: 232198  
Number of references: 29  

Combinational area: 289784.578125  
Noncombinational area: 151447.332031  
Net Interconnect area: undefined (Wire load has zero net area)  

Total cell area: 441231.906250  
Total area: undefined  

## 7.3 Timing report

High mapping effort and DesignWare_foundation library were used in order to meet the timing requirements.

Following log is the default timing report, which can be attained by using "report_timing" command. The critical path is shown in the log, along with the delays across cells. As we can see the slack is positive, it means the timing is met for setup time.

```
**********************************************************************
Report: timing
   -Path full
   -delay max
   -max_paths 1
Design : MEnoMEM
Version: 2001.08-SP1
Date : Tue May 18 11:19:19 2004
**********************************************************************

Operating Conditions: slow_125_1.62  Library: ssc_core_slow
Wire Load Model Mode: enclosed

Startpoint: delay5/delay_out_reg
   (rising edge-triggered flip-flop clocked by my_clk)
   (rising edge-triggered flip-flop clocked by my_clk)
Path Group: my_clk
Path Type: max

Des/Clust/Port   Wire Load Model   Library
-----------------------------------------------
MEnoMEM          20KGATES          ssc_core_slow
mux_mem_inter    5KGATES           ssc_core_slow
Interconnection  5KGATES           ssc_core_slow
PE_3_DW01_cmp2_8_0 5KGATES         ssc_core_slow
PE_3_DW01_add_16_1 5KGATES         ssc_core_slow
PE_3            5KGATES           ssc_core_slow

Point    Incr
Path
```
clock my_clk (rise edge) 0.00
0.00
clock network delay (ideal) 0.00
0.00
delay5/delay_out_reg/CLK (fdmfla15) 0.00
0.00 r
delay5/delay_out_reg/Q (fdmfla15) 0.55
0.55 f
delay5/delay_out (delay_clock_1) 0.00
0.55 f
mux/z_cp (mux_mem_inter) 0.00
0.55 f
mux/U20/Y (buf1a15) 0.17
0.71 f
mux/U21/Y (inv1a15) 0.12
0.83 r
mux/U19/Y (or2c6) 0.10
0.93 f
mux/U34/Y (inv1a9) 0.19
1.12 r
mux/out_p[3] (mux_mem_inter) 0.00
1.12 r
Interconnection_map/p[3] (Interconnection) 0.00
1.12 r
Interconnection_map/U70/Y (buf1a27) 0.26
1.38 r
Interconnection_map/U69/Y (mx2a9) 0.31
1.69 r
Interconnection_map/PE15p[3] (Interconnection) 0.00
1.69 r
pe_16/b[3] (PE_3) 0.00
1.69 r
pe_16/gt_45/A[3] (PE_3_DW01_cmp2_8_0) 0.00
1.69 r
pe_16/gt_45/U21/Y (inv1a3) 0.10
1.79 f
pe_16/gt_45/U41/Y (or2c6) 0.09
1.88 r
pe_16/gt_45/U28/Y (aol6a) 0.11
1.99 f
pe_16/gt_45/U29/Y (aol6a) 0.16
2.14 r
pe_16/gt_45/U13/Y (aol6a) 0.11
2.26 f
pe_16/gt_45/LT_LE (PE_3_DW01_cmp2_8_0) 0.00
2.26 f
pe_16/U40/Y (buf1a15) 0.19
2.45 f
pe_16/U68/Y (mx2d6) 0.20
2.65 f
pe_16/add_50/A[0] (PE_3_DW01_add_16_1) 0.00
2.65 f
pe_16/add_50/U15/Y (or2c6) 0.09
2.74 r

51
pe_16/add_50/U16/Y (invla6)  0.05
2.79 f
pe_16/add_50/U31/CO (fa1a2)  0.33
3.13 f
pe_16/add_50/U32/CO (fa1a2)  0.36
3.49 f
pe_16/add_50/U33/CO (fa1a2)  0.36
3.85 f
pe_16/add_50/U34/CO (fa1a2)  0.36
4.22 f
pe_16/add_50/U35/CO (fa1a2)  0.37
4.59 f
pe_16/add_50/U8/CO (fa1a3)  0.37
4.96 f
pe_16/add_50/U7/CO (fa1a3)  0.37
5.33 f
pe_16/add_50/U17/Y (and2a6)  0.20
5.52 f
pe_16/add_50/U13/Y (or2b3)  0.30
5.83 r
pe_16/add_50/U14/Y (inv1a2)  0.09
5.91 f
pe_16/add_50/U25/Y (xor2a1)  0.30
6.22 r
pe_16/add_50/SUM[11] (PE_3_DW01_add_16_1)  0.00
6.22 r
pe_16/U47/Y (and2b6)  0.18
6.40 r
pe_16/U46/Y (inv1a6)  0.04
6.44 f
pe_16/SAD_reg[11]/D0 (fdmf1c9)  0.00
6.44 f
... data arrival time
6.44
clock my_clk (rise edge)  7.00
7.00
clock network delay (ideal)  0.00
7.00
clock uncertainty  -0.25
6.75
pe_16/SAD_reg[11]/CLK (fdmf1c9)  0.00
6.75 r
library setup time  -0.30
6.45
data required time
6.45

---------------------------------------------------------------------
data required time
6.45
data arrival time
-6.44

---------------------------------------------------------------------
slack (MET)
7.4 Violation report

Following report log is the constraint report, which reports violations of constraints during synthesis. As we can see in following log, the constraints are all met, with no violation. There were some hold time violation at first, “fix_timing -min” command was used to fix the problems.

*==================================
Report : constraint
   -all_violators
Design : MEnoMEM
Version: 2001.08-SP1
Date : Tue May 18 11:22:50 2004
*==================================

This design has no violated constraints.

The synthesis was successful, the required speed was achieved and the core area is fairly small. There is no violation.
H.264 motion estimation has been successfully implemented as ASICs, which meet the real-time speed requirement (125MHz), when 16x16, 8x8 and 4x4 blocks are being processed in parallel. The 16x16 and fractional ME 's basic concept is from paper [12], however novel dataflow and architectures for 8x8 and 4x4 block size processing were developed and a pipeline global diagram was implemented.

It shows that full search block-matching algorithm is very hardware friendly because of the fixed searching sequence. The processing speed is fixed at 16 clock cycles per pixel. Also, this algorithm only requires a few memory ports which greatly reduces the power consumption.

In the simulation, the results are significant, the smaller the block size, the less the energy level in residual frame.

In this design two static frames were used for prediction. A streaming interface is needed to process actual video sequence. At 125MHz this design can handle 640X480 resolution video at 25/sec frame rate. The area of the synthesized chip is 664um x 664um.
REFERENCE


ACKNOWLEDGMENTS:

I would like to thank Dr. Hsu for the tremendous amount of assistance he provided me with. I would also like to thank Dr. Reddy and Dr. Patru for their valuable feedbacks. Dr. Lukowiak for his help with VHDL. And Suriyadi Gowanan for his helps me to gather the references.
APPENDIX A: Synthesis Circuits

1 AGU
2 Comparator
4 Interconnection
7 Memory Previous Frame
Used blank block with I/O ports.

8 **Memory Previous Frame**

Used blank block with I/O ports.

9 **Mux between Memory and Interconnection**

Used regular Mux from library.

10 **Transfer Unit**
12 ME_top level
14 8x8 Interconnection
15 8x8 Controller
18 4x4 Controller
21 Fractional AGU
23 Fractional Comparator
24 Fractional PE
25 Fractional SPU
12.1 Behavior VHDL code

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_textio.all;
use ieee.std_logic_unsigned.all;
use std.textio.all;

entity beh_me is
  generic (
    max_row : integer := 257;
    max_col : integer := 257
  );
end beh_me;

architecture beh of beh_me is
  type frame is array (-8 to 262, -8 to 262) of integer;  -- defined frame
  shared variable prev_frame, cur_frame, residual_frame1,
    residual_frame2, residual_frame3 : frame;
  type inter_frame is array (-1 to 1024, -1 to 1024) of integer;  -- defined frame
  shared variable inter_prev_frame : inter_frame;
  type mv_array is array(1 to 16, 1 to 16) of integer range 0 to 65535;  -- defined mv array
  shared variable mv_array_col, mv_array_row, mv, sad_out, mode_sub : mv_array;
  type mv_array_sub is array(1 to 32, 1 to 32) of integer range 0 to 65535;  -- defined mv array
  shared variable mv_array_col_sub, mv_array_row_sub,
    mv_sub, sad_out_sub, test_mode_sub, mode_sub_sub : mv_array_sub;
  type mv_array_sub_sub is array(1 to 64, 1 to 64) of integer range 0 to 65535;  -- defined mv array
  shared variable mv_array_col_sub_sub, mv_array_row_sub_sub,
    mv_sub_sub, sad_out_sub_sub, test_mode_sub_sub, mode_sub_sub_sub : mv_array_sub_sub;
  type mv_array_frac is array(0 to 64, 0 to 64) of integer range 0 to 65535;  -- defined mv array
  shared variable mv_frac : mv_array_frac;

  shared variable h1 : line;
  shared variable h2 : line;
  shared variable h3 : line;
  shared variable h4 : line;
  shared variable h5 : line;
  shared variable h6 : line;
  shared variable h7 : line;
  shared variable h8 : line;

  signal read_ok1, read_ok2 : std_logic;
  signal operation_ok1, operation_ok2, operation_ok3, interok, frac_ok : std_logic;
  signal write_ok : std_logic;

begin
  READ: process
    variable ll1, ll2 : line;
    variable pix1, pix2 : integer;
```
variable row1, row2 : integer;
variable col1, col2 : integer;
variable status1, status2 : boolean;

file fin2 : text open read_mode is
"/home/xxl7341/ME_whole/pgm/shan1.pgm";
file fin1 : text open read_mode is
"/home/xxl7341/ME_whole/pgm/shan2.pgm";
begin
begin

readline(fin1,h1);
readline(fin1,h2);
readline(fin1,h3);
readline(fin1,h4);
readline(fin1,h5);
readline(fin1,h6);
readline(fin1,h7);
readline(fin1,h8);
row1 := 0;
row2 := 0;
coll := 0;
col2 := 0;

while not endfile(fin1) loop

readline(fin1,h1);
status1 := true;
while status1 = true loop
read(h1, pix1, status1);
if (status1 = true and row1 < 256) then
    prev_frame(row1, col1) := pix1;
    if col1 = 255 then
        col1 := 0;
        row1 := row1 + 1;
    else
        col1 := col1 + 1;
    end if;
end if;
end loop;
read_ok1 <= '1';
while not endfile(fin2) loop

readline(fin2,h1);
status2 := true;
while status2 = true loop
read(h1, pix2, status2);
if (status2 = true and row2 < 256) then
    cur_frame(row2, col2) := pix2;
    if col2 = 255 then
        col2 := 0;
        row2 := row2 + 1;
    else
        col2 := col2 + 1;
    end if;
end if;
end loop;
read_ok2 <= '1';
wait;
end process READING;
end process R_4>

interpolate: process
begin
wait until (read_ok1 = '1' and read_ok2 = '1');
for m in 0 to 254 loop
for n in 0 to 253 loop
inter_prev_frame(m*4,n*4) := prev_frame(m,n);
inter_prev_frame(m*4+1,n*4) := (3*prev_frame(m,n))/4 + prev_frame(m+1,n)/4;
inter_prev_frame(m*4+2,n*4) := (prev_frame(m,n))/2 + prev_frame(m+1,n)/2;
inter_prev_frame(m*4+3,n*4) := (prev_frame(m,n))/4 + (3*prev_frame(m+1,n))/4;
inter_prev_frame(m*4+4,n*4) := prev_frame(m+1,n);
end loop;
end loop;

for i in 0 to 254 loop
for j in 0 to 1020 loop
inter_prev_frame(j,i*4+1) := (3*inter_prev_frame(j,i*4))/4 +
inter_prev_frame(j,i*4+4)/4;
inter_prev_frame(j,i*4+2) := (inter_prev_frame(j,i*4))/2 +
inter_prev_frame(j,i*4+4)/2;
inter_prev_frame(j,i*4+3) := (inter_prev_frame(j,i*4))/4 + (3*inter_prev_frame(j,i*4+4))/4;
end loop;
end loop;

interok <= '1';
wait;
end process;

ME : process — 16x16 ME

variable sad_row, sad_col, min_sad, ad, m, n, p, q, i, j, k, l :
integer;
type valid_array is array(0 to 15, 0 to 15) of integer;
type sad_array is array(0 to 15, 0 to 15) of integer;

variable valid_sad_array : valid_array;
variable temp_sad_array : sad_array;

begin
wait until (read_ok1 = '1' and read_ok2 = '1') ;
for m in 1 to 16 loop
for n in 1 to 16 loop

    min_sad := 65535;
sad_row := 1;
sad_col := 1;

for p in 0 to 15 loop
for q in 0 to 15 loop

    valid_sad_array(p,q) := 1;
temp_sad_array(p,q) := 0;

    for i in ((m-1)*16+p-8) to ((m-1)*16+p+7) loop
for j in ((n-1)*16+q-8) to ((n-1)*16+q+7) loop

k := (i+8-p); l := (j+8-q);
if (i>0 and j>0 and i<256 and j<256) then
    if (prev_frame(i,j)>cur_frame(k,l)) then
        ad :=
        prev_frame(i,j)-cur_frame(k,l);
    else ad := cur_frame(k,l)-prev_frame(i,j);
    end if;
temp_sad_array(p,q) := temp_sad_array(p,q) + ad;
else temp_sad_array(p,q) := 65535;
end if;
end loop;
end loop;

end loop;
end loop;

end process;
end loop;
if (temp_sad_array(p,q)<min_sad) then
    min_sad := temp_sad_array(p,q);
    sad_row := p;
    sad_col := q;
end if;
for r in 0 to 15 loop
    for s in 0 to 15 loop
        if (cur_frame(((m-1)*16+r),((n-1)*16+s)) > prev_frame(((m-1)*16+r-8+sad_row),((n-1)*16+s-8+sad_col)) then
            residual_frame(((m-1)*16+r),((n-1)*16+s)) := cur_frame(((m-1)*16+r),((n-1)*16+s)) - prev_frame(((m-1)*16+r-8+sad_row),((n-1)*16+s-8+sad_col));
        else
            residual_frame(((m-1)*16+r),((n-1)*16+s)) := prev_frame(((m-1)*16+r-8+sad_row),((n-1)*16+s-8+sad_col))- cur_frame(((m-1)*16+r),((n-1)*16+s));
        end if;
        end loop;
        end loop;
        mv_array_row(m,n) := sad_row;
        mv_array_col(m,n) := sad_col;
        mv(m,n) := sad_row*16 + sad_col;
        sad_out(m,n) := min_sad;
        end loop;
    end loop;
operation_ok <= '1';
wait;
end process ME;

ME_sub: process — 8x8 ME
variable sad_row, sad_col, min_sad, ad, m, n, p, q, i, j, k, l, e, f:
    integer;
    type valid_array_sub is array(0 to 15, 0 to 15) of integer;
    type sad_array_sub is array(0 to 15, 0 to 15) of integer;
variable valid_sad_array_sub : valid_array_sub;
variable temp_sad_array_sub : sad_array_sub;
begin
    wait until (operation_ok = '1') ;
    for m in 1 to 16 loop
        for n in 1 to 16 loop
            if (sad_out(m,n) > 0) then
                for e in 1 to 2 loop
                    for f in 1 to 2 loop
                        min_sad := 65535;
                        sad_row := 1;
                        sad_col := 1;
                        for p in 0 to 15 loop
                            for q in 0 to 15 loop
                                valid_sad_array_sub(p,q) := 1;
                                temp_sad_array_sub(p,q) := 0;
                                for i in ((m-1)*16+(e-1)*8+p-8) to ((m-1)*16+(e-1)*8+p-1) loop
                                    for j in ((n-1)*16+(f-1)*8+q-8) to ((n-1)*16+(f-1)*8+q-1) loop
                                        if (temp_sad_array(p,q)<min_sad) then
                                            min_sad := temp_sad_array(p,q);
                                            sad_row := p;
                                            sad_col := q;
                                        end if;
                                    end loop;
                                end loop;
                            end loop;
                        end loop;
                    end loop;
                end loop;
            end if;
        end loop;
    end loop;
end process ME;
k := (i+8,p); l := (j+8,q);
if (i>0 and j>0 and i<256 and j<256) then
if (prev_frame(i,j)>cur_frame(i,j)) then
temp_sad_array_sub(p,q) + ad;
else temp_sad_array_sub(p,q) := 65535;
end if;
end loop;

if (temp_sad_array_sub(p,q)<min_sad) then
min_sad := temp_sad_array_sub(p,q);
sad_row := p;
sad_col := q;
end if;
end loop;
end loop;
end loop;
variable temp_sad_array_sub_sub : sad_array_sub_sub;

begin
wait until (operation_ok2 = '1') ;
for m in 1 to 32 loop
for n in 1 to 32 loop

if (test_mode_sub(m,n)=1 and sad_out_sub(m,n)>0) then

for e in 1 to 2 loop
for f in 1 to 2 loop

min_sad := 65535;
sad_row := 1;
sad_col := 1;

for p in 0 to 15 loop
for q in 0 to 15 loop

valid_sad_array_sub_sub(p,q) := 1;
temp_sad_array_sub_sub(p,q) := 0;

for i in ((m-l)*8+(e-l)*4+p-8) to ((m-l)*8+(e-l)*4+p-5) loop
for j in ((n-l)*8+(f-l)*4+q-8) to ((n-l)*8+(f-l)*4+q-5) loop

if (i+8-p; j+8-q);
if (i>0 and j>0 and i<256 and j<256) then
ad = prev_frame(i,j)-cur_frame(k,l);
else ad = cur_frame(k,l)-prev_frame(i,j);
end if;
temp_sad_array_sub_sub(p,q) := temp_sad_array_sub_sub(p,q) + ad;
else temp_sad_array_sub_sub(p,q) := 65535;
end if;
end loop;
end loop;

if (temp_sad_array_sub_sub(p,q)<min_sad) then
min_sad := temp_sad_array_sub_sub(p,q);
sad_row := p;
sad_col := q;
end if;
end loop;
end loop;
end loop;
for r in 0 to 4 loop
for s in 0 to 4 loop

k := (i+8-p); l := (j+8-q);
if (i>0 and j>0 and i<256 and j<256) then
ad = prev_frame(i,j)-cur_frame(k,l);
else ad = cur_frame(k,l)-prev_frame(i,j);
end if;
temp_sad_array_sub_sub(p,q) := temp_sad_array_sub_sub(p,q) + ad;
else temp_sad_array_sub_sub(p,q) := 65535;
end if;
end loop;
end loop;
end loop;

end if;
end loop;
end loop;
end loop;
if cur_frame(((2*m-3+e)*4+r),(2*n-3+f)*4+s)) >
prev_frame(((2*m-3+e)*4+r),(2*n-3+f)*4+s))
then
residual_frame3(((2*m-3+e)*4+r),(2*n-3+f)*4+s)) :=
cur_frame(((2*m-3+e)*4+r),(2*n-3+f)*4+s)) -
prev_frame(((2*m-3+e)*4+r),(2*n-3+f)*4+s)) +
residual_frame3(((2*m-3+e)*4+r),(2*n-3+f)*4+s)) -
cur_frame(((2*m-3+e)*4+r),(2*n-3+f)*4+s));
end if;
end loop;
end loop;
end loop;
mode_sub_sub(m,n) := 1;
test_mode_sub_sub(2*m-2+e,2*n-2+f) := 1;
mv_array_row_sub_sub(2*m-2+e,2*n-2+f) := sad_row;
mv_array_col_sub_sub(2*m-2+e,2*n-2+f) := sad_col;
mv_sub_sub(2*m-2+e,2*n-2+f) := sad_row*16 + sad_col;
sad_out_sub_sub(2*m-2+e,2*n-2+f) := min_sad;
end loop;
end loop;
else

for e in 1 to 2 loop
for f in 1 to 2 loop
mode_sub_sub(m,n) := 0;
test_mode_sub_sub(2*m-2+e,2*n-2+f) := 0;
mv_array_row_sub_sub(2*m-2+e,2*n-2+f) := mv_array_row_sub(m,n);
mv_array_col_sub_sub(2*m-2+e,2*n-2+f) := mv_array_col_sub(m,n);
mv_sub_sub(2*m-2+e,2*n-2+f) := mv_sub(m,n);
sad_out_sub_sub(2*m-2+e,2*n-2+f) := sad_out_sub(m,n);
end loop;
end loop;
end if;
end loop;
end loop;
operation_ok3<='1';
wait;
end process ME_sub_sub;

frac : process -quarter pixel
variable sad_row, sad_col, min_sad, ad, m, n, p, q, i, j, k, l, e, f :
integer;
type valid_array_frac is array(0 to 15, 0 to 15) of integer;
type sad_array_frac is array(0 to 15, 0 to 15) of integer;

variable valid_sad_array frac : valid_array frac;
variable temp_sad_array frac : sad_array frac;
begin
wait until (interok = '1' and operation_ok3 = '1');
for m_sub in 1 to 16 loop
for n_sub in 1 to 16 loop
if mode_sub(m_sub,n_sub) = 0 then -- 16x16 frac
  min_sad := 65535;
sad_row := 1;
sad_col := 1;
for p in 0 to 7 loop
  for q in 0 to 7 loop
    valid_sad_array_frac(p,q) := 1;
temp_sad_array_frac(p,q) := 0;

    for x in 1 to 16 loop
      for y in 1 to 16 loop
        i := ((m_sub-1)*16+mv_array_row(m_sub,n_sub)-8)*4+4+(x-i)*4+p;
j := ((n_sub-1)*16+mv_array_col(m_sub,n_sub)-8)*4+4+(y-j)*4+q;
k := (i+4-p)/4;
l := (j+4-q)/4;
if (i>0 and j>0 and i<1021 and j<1021) then
  if (inter_prev_frame(i,j)>cur_frame(k,l)) then
    ad := inter_prev_frame(i,j)-cur_frame(k,l);
  else ad := cur_frame(k,l)-inter_prev_frame(i,j);
  end if;
  temp_sad_array_frac(p,q) := temp_sad_array_frac(p,q) + ad;
else temp_sad_array_frac(p,q) := 65535;
end if;
end loop;
end loop;
if (temp_sad_array_frac(p,q)<min_sad) then
  min_sad := temp_sad_array_frac(p,q);
sad_row := p;
sad_col := q;
end if;

end if;
end loop;
end process ME_sub_sub;
else
for m_sub in m_sub*2-1 to m_sub*2 loop
for n_sub in n_sub*2-1 to n_sub*2 loop
if mode_sub(m_sub,n_sub) = 0 then --8x8 frac
  min_sad := 65535;
  sad_row := 1;
  sad_col := 1;
  for p in 0 to 7 loop
    for q in 0 to 7 loop
      valid_sad_array_frac(p,q) := 1;
      temp_sad_array_frac(p,q) := 0;
    end loop;
    for x in 1 to 8 loop
      for y in 1 to 8 loop
        i := ((m_sub-1)*8+mv_array_row(m_sub,n_sub)-8)*4-4+(x-1)*4+p;
        j := ((n_sub-1)*8+mv_anay_col(m_sub,n_sub)-8)*4-4+(y-1)*4+q;
        if (inter_prev_frame(i,j)>cur_frame(k,l)) then
          ad := inter_prev_frame(i,j)-cur_frame(k,l);
        else
          ad := cur_frame(k,l)-inter_prev_frame(i,j);
        end if;
        temp_sad_array_frac(p,q) := temp_sad_array_frac(p,q) + ad;
        end if;
      end loop;
    end loop;
    if (temp_sad_array_frac(p,q)<min_sad) then
      min_sad := temp_sad_array_frac(p,q);
      sad_row := p;
      sad_col := q;
    end if;
  end loop;
end loop;
l_for i in 0 to 1 loop
  for j in 0 to 1 loop
    mv_frac((m_sub-1)*2+i,(n_sub-1)*2+j):=sad_row*8+sad_col;
  end loop;
end loop;
else
for m_sub in m_sub*2-1 to m_sub*2 loop
for n_sub in n_sub*2-1 to n_sub*2 loop
  -- 4x4 frac
  min_sad := 65535;
  sad_row := 1;
  sad_col := 1;
  for p in 0 to 7 loop
    for q in 0 to 7 loop
end loop;
valid_sad_array_frac(p,q) := 1;
 temp_sad_array_frac(p,q) := 0;

 for x in 1 to 4 loop
   for y in 1 to 4 loop
     i := ((m-1)*4+mv_array_row(m,n)-8)*4+(x-1)*4+p;
     j := ((n-1)*4+mv_array_col(m,n)-8)*4+(y-1)*4+q;
     k := (i+4-p)/4;
     l := (j+4-q)/4;
     if (i>0 and j>0 and i<1021 and j<1021) then
       if (inter_prev_frame(i,j)>cur_frame(k,l)) then
         ad := inter_prev_frame(i,j)-cur_frame(k,l);
       else ad := cur_frame(k,l)-inter_prev_frame(i,j);
       end if;
       temp_sad_array_frac(p,q) := temp_sad_array_frac(p,q) + ad;
     else temp_sad_array_frac(p,q) := 65535;
     end if;
   end loop;
 end loop;

 if (temp_sad_array_frac(p,q)<min_sad) then
   min_sad := temp_sad_array_frac(p,q);
   sad_row := p;
   sad_col := q;
 end if;

 end loop;
 end loop;

 mv_frac(m,n) := sad_row*8+ sad_col;

 end loop;

 end loop;

 frac_ok <= '1';
 wait;
 end process;

WRITING : process

 file f_image_out1 : text open write_mode is "pgm/mv_sub.pgm";
 file f_image_out2 : text open write_mode is "pgm/mode_sub.pgm";
 file f_image_out3 : text open write_mode is "pgm/mv.pgm";
 file f_image_out4 : text open write_mode is "pgm/mv_sub.pgm";
 file f_image_out5 : text open write_mode is "pgm/SAD_sub.pgm";
 file f_image_out6 : text open write_mode is "pgm/SAD.pgm";
 file f_image_out7 : text open write_mode is "pgm/mode_sub.pgm";
 file f_image_out8 : text open write_mode is "pgm/mv_frac.pgm";
 file f_image_out9 : text open write_mode is "pgm/res1.pgm";
 file f_image_out10 : text open write_mode is "pgm/res2.pgm";
 file f_image_out11 : text open write_mode is "pgm/res3.pgm";
 variable L_OUT1, L_OUT2, L_OUT3, L_OUT4, L_OUT5, L_OUT6, L_OUT7, L_OUT8, L_OUT9, L_OUT10, L_OUT11 : LINE;
 variable CHAR1, CHAR2, CHAR3, CHAR4, CHAR5, CHAR6, CHAR7, CHAR8 : character := "1" ;

 begin
  wait until frac_ok = '1' ;
  writeln(f_image_out9,h1);

89
for x in 1 to 256 loop
    for y in 1 to 256 loop
        write (L_OUT9, residual_frame1(x,y));
        write(L_OUT9, CHAR1);
    end loop;
end loop;
write(f_image_out9, L_OUT9);
for x in 1 to 256 loop
    for y in 1 to 256 loop
        write (L_OUT10, residual_frame2(x,y));
        write(L_OUT10, CHAR1);
    end loop;
end loop;
write(f_image_out10, L_OUT10);
for x in 1 to 256 loop
    for y in 1 to 256 loop
        write (L_OUT11, residual_frame3(x,y));
        write(L_OUT11, CHAR1);
    end loop;
end loop;
write(f_image_out11, L_OUT11);
for x in 1 to 64 loop
    for y in 1 to 64 loop
        write (L_OUT1, mv_sub_sub(x,y));
        write(L_OUT1, CHAR1);
    end loop;
end loop;
write(f_image_out1, L_OUT1);
for x in 1 to 32 loop
    for y in 1 to 32 loop
        write(L_OUT2, mode_sub_sub(x,y));
        write(L_OUT2, CHAR2);
    end loop;
end loop;
write(f_image_out2, L_OUT2);
for x in 1 to 16 loop
    for y in 1 to 16 loop
        write(L_OUT3, mv(x,y));
        write(L_OUT3, CHAR3);
    end loop;
end loop;
write(f_image_out3, L_OUT3);
for x in 1 to 32 loop
    for y in 1 to 32 loop
        write(L_OUT4, mv_sub(x,y));
        write(L_OUT4, CHAR4);
    end loop;
end loop;
write(f_image_out4, L_OUT4);
for x in 1 to 32 loop
    for y in 1 to 32 loop
        write(L_OUT5, sad_out_sub(x,y));
        write(L_OUT5, CHAR5);
    end loop;
end loop;
write(f_image_out5, L_OUT5);
for x in 1 to 16 loop
  for y in 1 to 16 loop
    write(L_OUT6, sad_out(x,y));
    write(L_OUT6, CHAR6);
  end loop;
end loop;
write(____OUT6, L_OUT6);
end loop;

for x in 1 to 16 loop
  for y in 1 to 16 loop
    write(L_OUT7, mode_sub(x,y));
    write(L_OUT7, CHAR7);
  end loop;
end loop;
write(____OUT7, L_OUT7);
end loop;
for x in 0 to 63 loop
  for y in 0 to 63 loop
    write(L_OUT8, mv_frac(x,y));
    write(L_OUT8, CHAR8);
  end loop;
end loop;
write(f_image_out8, L_OUT8);
end loop;
writeline(f_image_out7, L_OUT7);
end loop;
write_ok<=T;
wait;
end process WRITING;
end architecture beh;

12.2 Register Transfer Level Codes

8.2.1 AGU

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity AGU is
  port(
    start_c_row, start_c_col,
    start_p_row, start_p_col : in std_logic_vector(7 downto 0);
    reset, clk, start_transfer,
    ME_to_ME_sub_ready : in std_logic;
    done_block, start_cmp : out std_logic;
    add_c_row, add_c_col, add_p_row, add_p_col,
    add_pl_row, add_pl_col : out std_logic_vector(7 downto 0);
    Z_cp, Z_pl : out std_logic;
    reset_inter_pixel, reset_inter_bit,
    reset_cmp, reset_PE : out std_logic
  );
end AGU;

Architecture RTL_AGU of AGU is

  type state_type is (idle, output, done_16, finish);
  shared variable s_c_row, s_c_col,
    s_p_row, s_p_col, c_row, c_col, p_row, p_col, pl_row, pl_col: integer;
  shared variable i_cp, i_p1, l_cp, l_p1, line: integer;
  shared variable done : bit := '0';

91
signal state : state_type;
begin
get_state: process(clk, reset, start)
begin
if reset = '1' then state <= idle;
elsif (clk'event and clk = '1') then
  case state is
  when idle =>
    add_c_row <= "00000000";
    add_c_col <= "00000000";
    add_p_row <= "00000000";
    add_p_col <= "00000000";
    add_pl_row <= "00000000";
    add_pl_col <= "00000000";
    done_block <= '0';
    reset_PE <= '1';
    reset_cmp <= '1';
    start_cmp <= '0';
    Z_cp <= '1';
    Z_pl <= '1';
    reset_inter_pixel <= '1';
    reset_inter_bit <= '1';
    done <= '0';
    line := 0;
    s_c_row := conv_integer(start_c_row);
    s_c_col := conv_integer(start_c_col);
    s_p_row := conv_integer(start_p_row);
    s_p_col := conv_integer(start_p_col);
    i_cp := 0; i_pl := 0;
    l_cp := 0; l_pl := 0;
    if start = '1' then done_block <= '0';
    state <= output;
    else
    state <= idle;
  end if;

  when output => -- output address
    start_cmp <= '0';
    reset_PE <= '0';
    reset_cmp <= '0';
    reset_inter_pixel <= '0';
    reset_inter_bit <= '0';
    if (done = '1') then state <= finish; done_block <= '1';
    else state <= output;
  end if;

  if (l_cp < 16) then
    Z_cp <= '0';
    c_row := s_c_row + l_cp;
    c_col := s_c_col + i_cp;
    p_row := s_p_row + l_cp;
    p_col := s_p_col + i_cp;
    add_c_row <= conv_std_logic_vector(c_row, 8);
    add_c_col <= conv_std_logic_vector(c_col, 8);
    add_p_row <= conv_std_logic_vector(p_row, 8);
    add_p_col <= conv_std_logic_vector(p_col, 8);
    else Z_cp <= '1';
    add_c_row <= "00000000";
  end if;
end if;
add_c_col <= "00000000";
add_p_row <= "00000000";
add_p_col <= "00000000";
end if;

if (l_pl > 0) then
  Z_pl <= '0';
  pl_row := l_pl + s_p_row - 1;
  pl_col := s_p_col + 16 + i_pl;
  add_pl_row <= conv_std_logic_vector(pl_row,8);
  add_pl_col <= conv_std_logic_vector(pl_col,8);
else
  Z_pl <= T;
  add_pl_row <= "00000000";
  add_pl_col <= "00000000";
end if;

if (i_cp=15 and l_cp=16 and line=15) then
  done := '1';
  start_cmp <= '1';
elsif (i_cp=15 and l_cp=16) then
  i_cp := 0; i_p1 := 0; l_cp := l_cp + 1;
  l_pl := l_pl + 1; reset_inter_bit <= '1';
  start_cmp <= '0'; reset_PE <= '0';
else i_cp := i_cp + 1; i_p1 := i_p1 + 1;
  start_cmp <= '0'; reset_PE <= '0';
end if;
end if;

when done_16 =>
  start_cmp <= '1';
  reset_PE <= '1';
  reset_inter_bit <= '1';
  state <= output;

when finish =>
  reset_PE <= '1';
  if (start_transfer = '1' or ME_to_ME_sub_ready = '1') then
    done_block <= '0';
  end if;
  end case;
end if;

end process;

end architecture RTL_AGU;

8.2.2 Comparator

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity comparator is
  port(mv_ready : out std_logic;
   PE0, PE1, PE2, PE3, PE4, PE5, PE6, PE7, PE8,
   PE9, PE10, PE11, PE12, PE13,
   PE14, PE15 : in std_logic_vector(15 downto 0);
   SAD_threshold : in std_logic_vector(15 downto 0);
   reset_cmp, start_cmp : in std_logic;
   clk: in std_logic;
   SAD, min_SAD_out : out std_logic_vector(15 downto 0);
   mv_out,mv_out_sub : out std_logic_vector(7 downto 0)
  );
end comparator;

Architecture comparator of comparator is
type state_type is (idle, compare, outputmv, initial);
signal state : state_type;

begin

process(clk, reset_cmp, start_cmp)
variable pe_count, mv : integer range 0 to 256;
variable min_SAD : integer range 0 to 65535;
begin
if reset_cmp = '1' then state <= initial;
end if;
if start_cmp = '1' then state <= compare; end if;
if clk'event and clk = '1' then

case state is

when idle =>
if (start_cmp = '1') then state <= compare;
else
state <= idle;
end if;

when compare =>
if (conv_integer(PE0) < min_SAD) then
min_SAD := conv_integer(PE0);
mv := pe_count;
end if;
if (conv_integer(PE1) < min_SAD) then
min_SAD := conv_integer(PE1);
mv := pe_count + 1;
end if;
if (conv_integer(PE2) < min_SAD) then
min_SAD := conv_integer(PE2);
mv := pe_count + 2;
end if;
if (conv_integer(PE3) < min_SAD) then
min_SAD := conv_integer(PE3);
mv := pe_count + 3;
end if;
if (conv_integer(PE4) < min_SAD) then
min_SAD := conv_integer(PE4);
mv := pe_count + 4;
end if;
if (conv_integer(PE5) < min_SAD) then
min_SAD := conv_integer(PE5);
mv := pe_count + 5;
end if;
if (conv_integer(PE6) < min_SAD) then
min_SAD := conv_integer(PE6);
mv := pe_count + 6;
end if;
if (conv_integer(PE7) < min_SAD) then
min_SAD := conv_integer(PE7);
mv := pe_count + 7;
end if;
if (conv_integer(PE8) < min_SAD) then
min_SAD := conv_integer(PE8);
mv := pe_count + 8;
end if;

end case;
end process;
end
if (conv_integer(PE9) < min_SAD)  
min_SAD := conv_integer(PE9);  
mv := pe_count+9;  
end if;

if (conv_integer(PE10) < min_SAD)  
min_SAD := conv_integer(PE10);  
mv := pe_count+10;  
end if;

if (conv_integer(PE11) < min_SAD)  
min_SAD := conv_integer(PE11);  
mv := pe_count+11;  
end if;

if (conv_integer(PE12) < min_SAD)  
min_SAD := conv_integer(PE12);  
mv := pe_count+12;  
end if;

if (conv_integer(PE13) < min_SAD)  
min_SAD := conv_integer(PE13);  
mv := pe_count+13;  
end if;

if (conv_integer(PE14) < min_SAD)  
min_SAD := conv_integer(PE14);  
mv := pe_count+14;  
end if;

if (conv_integer(PE15) < min_SAD)  
min_SAD := conv_integer(PE15);  
mv := pe_count+15;  
end if;

min_SAD_out <= conv_std_logic_vector(min_SAD,16); --test signal  
SAD <= conv_std_logic_vector(min_SAD,16);  
if pe_count = 240 then state <= outputmv;  
else state <= idle; pe_count := pe_count + 16;  
end if;

when outputmv =>  
if min_SAD < conv_integer(SAD_threshold) then  
mv_out <= conv_std_logic_vector(mv,8);  
else mv_out_sub <= conv_std_logic_vector(mv,8);  
end if;  
SAD <= conv_std_logic_vector(min_SAD,16);  
state <= initial; mv_ready <= '1';

when initial =>  

mv_ready <= '0';  
min_SAD := 65535;  
SAD <= "1111111111111111";  
pe_count := 0;  
if(start_cmp = '1') then state <= compare;  
else state <= idle;  
end if;
end if;

end case;
end process;
end architecture comparator;

8.2.3 Controller
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity Controller is
port(
    reset, start : in std_logic;
    ME_to_ME_sub_ready : in std_logic;
    clk : in std_logic;
    reset_AGU, enable_SPU, start_AGU : out std_logic;
    done_frame : out std_logic;
    Block_num_col, Block_num_row : out std_logic_vector(3 downto 0);
    position : out std_logic_vector(3 downto 0)
);
end entity Controller;

Architecture FSM_Controller of Controller is

type state_type is (idle, startup, output1, output2, output3, output4, output5, output6, output7, output8, output9, finish);
signal state : state_type;

begin
process(clk, reset)

variable col, row : integer range 0 to 15;
variable block_count : integer range 0 to 256;
variable sub_block_count : integer range 0 to 14;
begin

if reset = '1' then
    state <= idle;
elsif clk'event and clk = '1' then

case state is

when idle =>
    col := 0;
    row := 0;
    block_count := 0;
    sub_block_count := 0;
    if start = '1' then
        reset_AGU <= '1';
        start_AGU <= '1';
        done_frame <= '1';
        state <= startup;
    else
        state <= idle;
    end if;

when startup =>
    reset_AGU <= '0';
    enable_SPU <= '0';
    done_frame <= '0';
    Block_num_col <= "0000";
    Block_num_row <= "0000";
    state <= output1;

when output1 =>
    enable_SPU <= '1';
    Block_num_col <= conv_std_logic_vector(col,4);
    Block_num_row <= conv_std_logic_vector(row,4);
    position <= "0000";

    if (ME_to_ME_sub_ready = '1') then
        block_count := block_count+1;
        col := col + 1;
        state <=
output2; reset_AGU <= '1';
else state <= output1;
end if;

when output2 =>
reset_AGU <= '0';
Block_num_col <= conv_std_logic_vector(col,4);
Block_num_row <= conv_std_logic_vector(row,4);
position <= "0001";
if (ME_to_ME_sub_ready = '1') then
block_count := block_count + 1; reset_AGU <= '1';
col := col + 1;
end if;
if (block_count = 15) then
state <= output3;
else state <= output2;
end if;

when output3 =>
reset_AGU <= '0';
Block_num_col <= conv_std_logic_vector(col,4);
Block_num_row <= conv_std_logic_vector(row,4);
position <= "0010";
if (ME_to_ME_sub_ready = '1') then
block_count := block_count + 1; reset_AGU <= '1';
col := 0; row := row + 1;
state <= output4;
else state <= output3;
end if;

when output4 =>
reset_AGU <= '0';
Block_num_col <= conv_std_logic_vector(col,4);
Block_num_row <= conv_std_logic_vector(row,4);
position <= "0011";
if (ME_to_ME_sub_ready = '1') then
block_count := block_count + 1; reset_AGU <= '1';
col := col + 1;
state <= output5;
else state <= output4;
end if;

when output5 =>
reset_AGU <= '0';
Block_num_col <= conv_std_logic_vector(col,4);
Block_num_row <= conv_std_logic_vector(row,4);
position <= "0100";

if (ME_to_ME_sub_ready = '1' and sub_block_count = 13) then
block_count := block_count + 1; reset_AGU <= '1'; state <= output6;
col := col + 1;
elsif (ME_to_ME_sub_ready = '1') then
block_count := block_count + 1; reset_AGU <= '1'; state <= output5;
sub_block_count := sub_block_count + 1; col := col + 1;
else state <= output5;
end if;

when output6 =>
reset_AGU <= '0';
Block_num_col <= conv_std_logic_vector(col,4);
Block_num_row <= conv_std_logic_vector(row,4);
position <= "0101";
if (ME_to_ME_sub_ready = '1' and block_count < 239) then
block_count := block_count + 1; sub_block_count := 0; reset_AGU <=
col := 0; row := row + 1; state <= output4;
elsif (ME_to_ME_sub_ready = '1') then
    block_count := block_count + 1; reset_AGU <= '1';
    col := 0; row := row + 1;
    state <= output7;
else state <= output6;
end if;

when output7 =>
    reset_AGU <= '0';
    Block_num_col <= conv_std_logic_vector(col,4);
    Block_num_row <= conv_std_logic_vector(row,4);
    position <= "0110";
    if (ME_to_ME_sub_ready = '1') then
        block_count := block_count + 1; col := col + 1; reset_AGU <= '1';
        state <= output8;
    else state <= output7;
    end if;

when output8 =>
    reset_AGU <= '0';
    Block_num_col <= conv_std_logic_vector(col,4);
    Block_num_row <= conv_std_logic_vector(row,4);
    position <= "0111";
    if (ME_to_ME_sub_ready = '1') then
        block_count := block_count + 1; col := col + 1; reset_AGU <= '1';
    end if;
    if (block_count = 255) then
        state <= output9;
    else state <= output8;
    end if;

when output9 =>
    reset_AGU <= '0';
    Block_num_col <= conv_std_logic_vector(col,4);
    Block_num_row <= conv_std_logic_vector(row,4);
    position <= "1000";
    if (ME_to_ME_sub_ready = '1') then
        block_count := block_count + 1; reset_AGU <= '1';
        state := finish; done_frame <= '1';
    else state <= output9;
    end if;

when finish =>
    state <= idle;
    done_frame <= '1';
end case;

end if;
end process;
end architecture FSM_Controller;

8.2.4 Interconnection

library ieee;
use ieee.std_logic_1164.all;
entity Interconnection is
  port (  
c, p, pi : in std_logic_vector(7 downto 0);
  reset_diff_bit, reset_diff_pixel : in std_logic;
  clk : in std_logic;
  PE0c, PE0p, PE1c, PE1p, PE2c, PE2p, PE3c, PE3p, PE4c, PE4p, PE5c, PE5p, PE6c, PE6p, PE7c, PE7p, PE8c, PE8p, PE9c, PE9p, PE10c, PE10p, PE11c, PE11p, PE12c, PE12p, PE13c, PE13p, PE14c, PE14p, PE15c, PE15p : inout std_logic_vector(7 downto 0)
  );
end Interconnection;

Architecture RTL_Interconnection of Interconnection is

  type Dff_anay is array(0 to 14) of bit;
  signal Dffbit : Dff_anay;
begin
  PE0c <= c;
  P_output : process(p, pi, Dffbit)
  begin
    PE0p <= p;
    if (Dffbit(0) = '0') then
      PE1p <= p; else PE1p <= p;
    end if;
    if (Dffbit(1) = '0') then
      PE2p <= p; else PE2p <= p;
    end if;
    if (Dffbit(2) = '0') then
      PE3p <= p; else PE3p <= p;
    end if;
    if (Dffbit(3) = '0') then
      PE4p <= p; else PE4p <= p;
    end if;
    if (Dffbit(4) = '0') then
      PE5p <= p; else PE5p <= p;
    end if;
    if (Dffbit(5) = '0') then
      PE6p <= p; else PE6p <= p;
    end if;
    if (Dffbit(6) = '0') then
      PE7p <= p; else PE7p <= p;
    end if;
    if (Dffbit(7) = '0') then
      PE8p <= p; else PE8p <= p;
    end if;
    if (Dffbit(8) = '0') then
      PE9p <= p; else PE9p <= p;
    end if;
    if (Dffbit(9) = '0') then
      PE10p <= p; else PE10p <= p;
    end if;
    if (Dffbit(10) = '0') then
      PE11p <= p; else PE11p <= p;
    end if;
    if (Dffbit(11) = '0') then
      PE12p <= p; else PE12p <= p;
    end if;
    if (Dffbit(12) = '0') then
      PE13p <= p; else PE13p <= p;
    end if;
    if (Dffbit(13) = '0') then
      PE14p <= p; else PE14p <= p;
    end if;
    if (Dffbit(14) = '0') then
  end if;

  end if;
end process P_output;
end Architecture;
PE15p <= p1; else PE15p <= p;
end if;
end process;
c_output_process: process(clk)
begin
if (clk'event and clk = '1') then
  if (reset_diff_bit = '1') then
    Dffbit(1) <= '0'; Dffbit(2) <= '0'; Dffbit(3) <= '0'; Dffbit(4) <=
    Dffbit(7) <= '0'; Dffbit(8) <= '0'; Dffbit(9) <= '0'; Dffbit(10) <=
    Dffbit(13) <= '0'; Dffbit(14) <= '0'; Dffbit(0) <= '0';
else
    Dffbit(14) <= Dffbit(13); Dffbit(13) <= Dffbit(12); Dffbit(12) <=
    Dffbit(9); Dffbit(8) <= Dffbit(7); Dffbit(7) <=
    Dffbit(4); Dffbit(3) <= Dffbit(2); Dffbit(2) <=
  end if;
if (reset_diff_pixel = '1') then
  PE1c <= (others => '0'); PE2c <= (others => '0');
  PE3c <= (others => '0'); PE4c <= (others => '0'); PE5c <=
  PE6c <= (others => '0'); PE7c <= (others => '0'); PE8c <=
  PE9c <= (others => '0'); PE10c <= (others => '0'); PE11c <=
  PE12c <= (others => '0'); PE13c <= (others => '0'); PE14c <=
  PE15c <= (others => '0');
else
  PE15c <= PE14c; PE14c <= PE13c; PE13c <= PE12c; PE12c <=
  PE9c <= PE8c; PE8c <= PE7c; PE7c <= PE6c; PE6c <= PE5c;
  PE2c <= PE1c; PE1c <= PE0c;
end if;
end if;
end process;
end architecture RTL_Interconnection;

8.2.5 PE (Processing Element)

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity PE is
port(
a, b : in std_logic_vector(7 downto 0);
reset : in std_logic;
Clk : in std_logic;
end process;
c_output_process: process(clk)
begin
if (clk'event and clk = '1') then
  if (reset_diff_bit = '1') then
    Dffbit(1) <= '0'; Dffbit(2) <= '0'; Dffbit(3) <= '0'; Dffbit(4) <=
    Dffbit(7) <= '0'; Dffbit(8) <= '0'; Dffbit(9) <= '0'; Dffbit(10) <=
    Dffbit(13) <= '0'; Dffbit(14) <= '0'; Dffbit(0) <= '0';
else
    Dffbit(14) <= Dffbit(13); Dffbit(13) <= Dffbit(12); Dffbit(12) <=
    Dffbit(9); Dffbit(8) <= Dffbit(7); Dffbit(7) <=
    Dffbit(4); Dffbit(3) <= Dffbit(2); Dffbit(2) <=
  end if;
if (reset_diff_pixel = '1') then
  PE1c <= (others => '0'); PE2c <= (others => '0');
  PE3c <= (others => '0'); PE4c <= (others => '0'); PE5c <=
  PE6c <= (others => '0'); PE7c <= (others => '0'); PE8c <=
  PE9c <= (others => '0'); PE10c <= (others => '0'); PE11c <=
  PE12c <= (others => '0'); PE13c <= (others => '0'); PE14c <=
  PE15c <= (others => '0');
else
  PE15c <= PE14c; PE14c <= PE13c; PE13c <= PE12c; PE12c <=
  PE9c <= PE8c; PE8c <= PE7c; PE7c <= PE6c; PE6c <= PE5c;
  PE2c <= PE1c; PE1c <= PE0c;
end if;
end if;
end process;
end architecture RTL_Interconnection;
SAD : inout std_logic_vector(15 downto 0)
);
end PE;

architecture str_PE of PE is

begin

process(clk)
variable c : integer range 0 to 255;
variable temp : integer range 0 to 65535;
begin
  if (clk'event and clk = '1') then
    if reset = '1' then
      SAD<= (others => '0');
    else
      if (conv_integer(a) > conv_integer(b)) then
        c := conv_integer(a) - conv_integer(b);
      else
        c := conv_integer(b) - conv_integer(a);
      end if;
      temp := c + conv_integer(SAD);
      SAD <= conv_std_logic_vector(temp,16);
    end if;
  end if;
end process;
end architecture str_PE;

8.2.6 SPU (Start Point Generator)

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity SPU is
  port(
    Block_num_col, Block_num_row : in std_logic_vector(3 downto 0);
    position : in std_logic_vector(3 downto 0);
    enable : in std_logic;
    start_c_row,start_c_col, start_p_row,start_p_col : out std_logic_vector(7 downto 0)
  );
end entity SPU;

architecture RTL_SPU of SPU is

begin

process(Block_num_col,Block_num_row, position, enable)
variable row, col : integer range 0 to 15;
variable c_row,c_col, p_row,p_col : integer range 0 to 255;
begin
  if enable = '1' then
    row := conv_integer(Block_num_row);
    col := conv_integer(Block_num_col);
    c_row := row*16;
    c_col := col*16;
    case position is
      when "0000" =>
        p_row :=0;
  end if;
end process;

end architecture RTL_SPU;
p_col := 0;
start_c_row <= conv_std_logic_vector(c_row,8);
start_c_col <= conv_std_logic_vector(c_col,8);
start_p_row <= conv_std_logic_vector(p_row,8);
start_p_col <= conv_std_logic_vector(p_col,8);

when "0001" =>
p_row := 0;
p_col := c_col - 8;
start_c_row <= conv_std_logic_vector(c_row,8);
start_c_col <= conv_std_logic_vector(c_col,8);
start_p_row <= conv_std_logic_vector(p_row,8);
start_p_col <= conv_std_logic_vector(p_col,8);

when "0010" =>
p_row := 0;
p_col := c_col -16;
start_c_row <= conv_std_logic_vector(c_row,8);
start_c_col <= conv_std_logic_vector(c_col,8);
start_p_row <= conv_std_logic_vector(p_row,8);
start_p_col <= conv_std_logic_vector(p_col,8);

when "0011" =>
p_row := c_row - 8;
p_col := c_col;
start_c_row <= conv_std_logic_vector(c_row,8);
start_c_col <= conv_std_logic_vector(c_col,8);
start_p_row <= conv_std_logic_vector(p_row,8);
start_p_col <= conv_std_logic_vector(p_col,8);

when "0100" =>
p_row := c_row - 8;
p_col := c_col - 8;
start_c_row <= conv_std_logic_vector(c_row,8);
start_c_col <= conv_std_logic_vector(c_col,8);
start_p_row <= conv_std_logic_vector(p_row,8);
start_p_col <= conv_std_logic_vector(p_col,8);

when "0101" =>
p_row := c_row - 8;
p_col := c_col - 16;
start_c_row <= conv_std_logic_vector(c_row,8);
start_c_col <= conv_std_logic_vector(c_col,8);
start_p_row <= conv_std_logic_vector(p_row,8);
start_p_col <= conv_std_logic_vector(p_col,8);

when "0110" =>
p_row := p_row - 16;
p_col := c_col;
start_c_row <= conv_std_logic_vector(c_row,8);
start_c_col <= conv_std_logic_vector(c_col,8);
start_p_row <= conv_std_logic_vector(p_row,8);
start_p_col <= conv_std_logic_vector(p_col,8);

when "0111" =>
p_row := c_row -16;
p_col := c_col - 8;
start_c_row <= conv_std_logic_vector(c_row,8);
start_c_col <= conv_std_logic_vector(c_col,8);
start_p_row <= conv_std_logic_vector(p_row,8);
start_p_col <= conv_std_logic_vector(p_col,8);

when "1000" =>
p_row := c_row - 16;
p_col := p_col - 16;
start_c_row <= conv_std_logic_vector(c_row,8);
start_c_col <= conv_std_logic_vector(c_col,8);
start_p_row <= conv_std_logic_vector(p_row,8);
start_p_col <= conv_std_logic_vector(p_col,8);
8.2.7 Previous frame memory

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_textio.all;
use STD.textio.all;

entity mem_previous_frame is
  port(
    add_p_row, add_p_col : in std_logic_vector(7 downto 0);
    add_pl_row, add_pl_col : in std_logic_vector(7 downto 0);
    out_pl : out std_logic_vector(7 downto 0);
    out_p : out std_logic_vector(7 downto 0);
    elk : in stdjogic);

end mem_previous_frame;

architecture Mem_p of mem_previous_frame is

  type frame is array (0 to 255, 0 to 255) of integer;
  shared variable prevjrame : frame;
  shared variable h1 : line;
  shared variable h2 : line;
  shared variable h3 : line;
  shared variable h4 : line;
  shared variable p,p1 : integer;

  signal read_ok1 : std_logic;

begin
  reading : process
    variable rowl : integer := 0;
    variable col1 : integer := 0;
    variable status1 : boolean;
    file fin1 : text open read_mode is "pgm/shan2.pgm";
    begin
      readline(fin1, h1);
      readline(fin1, h2);
      readline(fin1, h3);
      --readline(fin1, h4);
      rowl := 0;
      col1 := 0;

end process;
end architecture RTL_SPU;
while not endfile(finl) loop
  readline(finl, l11);
  status1 := true;
  while status1 = true loop
    read(l11, pixl, status1);
    if (status1 = true and row1 < 256) then
      prev_frame(row1, coll) := pix1;
      if coll = 255 then
        coll := 0;
        row1 := row1 + 1;
      else
        coll := coll + 1;
        end if;
    end if;
  end loop;
end loop;

read_okl <= '1';
wait;
end process;

process(clk)
begin
  if(read_okl = '1') then
    if clk'event and clk = '1' then
      p := prev_frame(conv_integer(add_p_row), conv_integer(add_p_col));
      out_p <= conv_std_logic_vector(p, 8);
      end if;
    end if;
  end if;
end process;

process(clk)
begin
  if(read_okl = '1') then
    if clk'event and clk = '1' then
      pl := prev_frame(conv_integer(add_pl_row), conv_integer(add_pl_col));
      out_pl <= conv_std_logic_vector(pl, 8);
    end if;
  end if;
end process;

end architecture Mem_p;

8.2.8 Current Frame Memory

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_textio.all;
use STD.textio.all;

entity mem_current_frame is
  port(
    add_c_row, add_c_col : in std_logic_vector(7 downto 0);
    out_c : out std_logic_vector(7 downto 0);
    clk : in std_logic
  );
end mem_current_frame;

Architecture Mem_c of mem_current_frame is
type frame is array (0 to 255, 0 to 255) of integer;
shared variable curr_frame : frame;
shared variable h1 : line;
shared variable h2 : line;
shared variable h3 : line;
shared variable h4, h5 : line;
shared variable c : integer;
signal read_ok1 : std_logic;

begin

reading : process
variable ll1 : line;
variable pixl : integer;
variable row1 : integer := 0;
variable col1 : integer := 0;
variable status1 : boolean;
file fin1 : text open read_mode is "pgm/shan1.pgm";

begin
    readline(fin1, h1);
    readline(fin1, h2);
    readline(fin1, h3);
    readline(fin1, h4);
    readline(fin1, h5);
    row1 := 0;
    col1 := 0;
    while not endfile(fin1)
        loop
            readline(fin1, ll1);
            status1 := true;
            while status1 = true loop
                read(ll1, pix1, status1);
                if (status1 = true and row1 < 256) then
                    curr_frame(row1, col1) := pix1;
                    if col1 = 255 then
                        col1 := 0;
                        row1 := row1 + 1;
                    else
                        col1 := col1 + 1;
                    end if;
                end if;
            end loop;
            read_ok1 <= '1';
        end loop;
    wait;
end process;

mem: process(clk)

begin
    if (read_ok1 = '1') then
        if clk'event and clk='1' then
            c := curr_frame(conv_integer(add_c_row), conv_integer(add_c_col));
            out_c <= conv_std_logic_vector(c, 8);
        end if;
    end if;
end process;

end architecture Mem_c;

8.2.9 Mux between memory and interconnection
library ieee;
use ieee.std_logic_1164.all;

entity mux_mem_inter is
port(
    z_cp, z_p1 : in std_logic;
    clk : in std_logic;
    in_c, in_p, in_p1 : in std_logic_vector(7 downto 0);
    out_c, out_p, out_p1 : out std_logic_vector(7 downto 0);
);
end entity mux_mem_inter;

Architecture RTL_mux of mux_mem_inter is
begin
process(z_cp,z_p1,in_c,in_p,in_p1,clk)
begin
    case z_cp is
    when '1' =>
        out_c <= "00000000";
        out_p <= "00000000";
    when '0' =>
        out_c <= in_c;
        out_p <= in_p;
    when others =>
        out_c <= in_c;
        out_p <= in_p;
    end case;

    case z_p1 is
    when '1' =>
        out_p1 <= "00000000";
    when '0' =>
        out_p1 <= in_p1;
    when others =>
        out_p1 <= in_p1;
    end case;
end process;
end architecture RTL_mux;

8.2.10 Transfer Unit

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity TransME is
port(
    clk : in std_logic;
    start_transfer : in std_logic;
    s_c_row, s_c_col, s_p_row,
    s_p_col : in std_logic_vector(7 downto 0);
    data_ready : out std_logic;
    c_row, c_col, p_row, p_col : out std_logic_vector(7 downto 0);
);
sub_c_row, sub_c_col : out std_logic_vector(3 downto 0);
sub_p_row, sub_p_col : out std_logic_vector(5 downto 0);
read_write, ME_add_select : out std_logic;
ME_sub_add_select1, ME_sub_add_select2 : out std_logic
);
end TransME;

Architecture TransME of TransME is

type state_type is (startup, transfer);
signal state: state_type;
begin
process(clk)
variable num_c_row, num_c_col, num_p_row, num_p_col,
      i_c, i_p, l_c, l_p, start_c_row, start_c_col, start_p_row, start_p_col : integer;
begin
if start_Transfer = '0' then state <= startup; end if;
if clk'event and clk = '1' then
  case state is
  when startup =>
    c_row <= "00000000";
    sub_c_row <= "0000";
    c_col <= "00000000";
    sub_c_col <= "0000";
    p_row <= "00000000";
    sub_p_row <= "00000000";
    p_col <= "00000000";
    sub_p_col <= "00000000";
    read_write <= '0';
    data_ready <= '0';
    ME_add_select <= '0';
    ME_sub_add_select1 <= '1';
    ME_sub_add_select2 <= '1';
    start_c_row := conv_integer(s_c_row);
    start_c_col := conv_integer(s_c_col);
    if conv_integer(s_p_row) = 0 then
      start_p_row := conv_integer(s_p_row)+1;
    elsif conv_integer(s_p_row) = 240 then
      start_p_row := conv_integer(s_p_row)-1;
    else
      start_p_row := conv_integer(s_p_row);
    end if;
    if conv_integer(s_p_col) = 0 then
      start_p_col := conv_integer(s_p_col)+1;
    elsif conv_integer(s_p_col) = 240 then
      start_p_col := conv_integer(s_p_col)-1;
    else
      start_p_col := conv_integer(s_p_col);
    end if;
    i_c := 0; i_p := 0; l_c := 0; l_p := 0;
    if start_transfer = '1' then state <= transfer;
    else state <= startup;
    end if;
  when transfer =>
    read_write <= '1';
    ME_add_select <= '1';
    ME_sub_add_select1 <= '0';
    ME_sub_add_select2 <= '0';
    if l_c < 16 then
      num_c_row := start_c_row + l_c;
      c_row <= conv_std_logic_vector(num_c_row,8);
      sub_c_row <= conv_std_logic_vector(l_c,4);
      num_c_col := start_c_col + i_c;
      c_col <= conv_std_logic_vector(num_c_col,8);
  end when;
  when others =>
    state <= startup;
  end case;
end if;
end process;
end TransME;
sub_c_col <= conv_std_logic_vector(i_c,4);

if i_c < 15 then
  i_c := i_c + 1;
else i_c := 0; l_c := l_c + 1;
end if;

end if;

if l_p < 34 then
  num_p_row := start_p_row + l_p-1;
  p_row <= conv_std_logic_vector(num_p_row,8);
  sub_p_row <= conv_std_logic_vector(l_p,6);
  num_p_col := start_p_col + l_p-1;
  p_col <= conv_std_logic_vector(num_p_col,8);
  sub_p_col <= conv_std_logic_vector(l_p,6);
  if i_p < 33 then
    i_p := i_p + 1;
  else i_p := 0; l_p := l_p + 1;
  end if;
  else data_ready <= '1'; state <= startup;
end if;
end if;

end case;

8.2.11 Bridging unit

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity ME_to_ME_sub is
port(clk, reset : in std_logic;
    mv : in std_logic_vector(7 downto 0);
    SAD_threshold : in std_logic_vector(15 downto 0);
    SAD_in : in std_logic_vector(15 downto 0);
    ME_sub_ready, ME_ready : in std_logic;
    data_ready : in std_logic;
    need_process : out std_logic;
    start_data_transfer : out std_logic;
    ME_to_ME_sub_ready : out std_logic
);
end ME_to_ME_sub;

architecture ME_to_ME_sub of ME_to_ME_sub is

begin
process(clk)

begin
if reset = '1' then state <= reset_state;
elsif clk'event and clk = '1' then
  case state is
  when reset_state =>
    ME_to_ME_sub_ready <= '0';
  start_data_transfer <= '0';

end if;
end if;
end process;

begin
process(clk)

begin
if reset = '1' then state <= reset_state;
elsif clk'event and clk = '1' then
  case state is
  when reset_state =>
    ME_to_ME_sub_ready <= '0';
  start_data_transfer <= '0';

end if;
end if;
end process;

8.2.11 Bridging unit

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity ME_to_ME_sub is
port(clk, reset : in std_logic;
    mv : in std_logic_vector(7 downto 0);
    SAD_threshold : in std_logic_vector(15 downto 0);
    SAD_in : in std_logic_vector(15 downto 0);
    ME_sub_ready, ME_ready : in std_logic;
    data_ready : in std_logic;
    need_process : out std_logic;
    start_data_transfer : out std_logic;
    ME_to_ME_sub_ready : out std_logic
);
end ME_to_ME_sub;

architecture ME_to_ME_sub of ME_to_ME_sub is

begin
process(clk)

begin
if reset = '1' then state <= reset_state;
elsif clk'event and clk = '1' then
  case state is
  when reset_state =>
    ME_to_ME_sub_ready <= '0';
  start_data_transfer <= '0';

end if;
end if;
end process;

begin
process(clk)

begin
if reset = '1' then state <= reset_state;
elsif clk'event and clk = '1' then
  case state is
  when reset_state =>
    ME_to_ME_sub_ready <= '0';
  start_data_transfer <= '0';

end if;
end if;
end process;

8.2.11 Bridging unit

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity ME_to_ME_sub is
port(clk, reset : in std_logic;
    mv : in std_logic_vector(7 downto 0);
    SAD_threshold : in std_logic_vector(15 downto 0);
    SAD_in : in std_logic_vector(15 downto 0);
    ME_sub_ready, ME_ready : in std_logic;
    data_ready : in std_logic;
    need_process : out std_logic;
    start_data_transfer : out std_logic;
    ME_to_ME_sub_ready : out std_logic
);
end ME_to_ME_sub;

architecture ME_to_ME_sub of ME_to_ME_sub is

begin
process(clk)

begin
if reset = '1' then state <= reset_state;
elsif clk'event and clk = '1' then
  case state is
  when reset_state =>
    ME_to_ME_sub_ready <= '0';
  start_data_transfer <= '0';

end if;
end if;
end process;

begin
process(clk)

begin
if reset = '1' then state <= reset_state;
elsif clk'event and clk = '1' then
  case state is
  when reset_state =>
    ME_to_ME_sub_ready <= '0';
  start_data_transfer <= '0';

end if;
end if;
end process;

8.2.11 Bridging unit

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity ME_to_ME_sub is
port(clk, reset : in std_logic;
    mv : in std_logic_vector(7 downto 0);
    SAD_threshold : in std_logic_vector(15 downto 0);
    SAD_in : in std_logic_vector(15 downto 0);
    ME_sub_ready, ME_ready : in std_logic;
    data_ready : in std_logic;
    need_process : out std_logic;
    start_data_transfer : out std_logic;
    ME_to_ME_sub_ready : out std_logic
);
end ME_to_ME_sub;

architecture ME_to_ME_sub of ME_to_ME_sub is

begin
process(clk)

begin
if reset = '1' then state <= reset_state;
elsif clk'event and clk = '1' then
  case state is
  when reset_state =>
    ME_to_ME_sub_ready <= '0';
  start_data_transfer <= '0';

end if;
end if;
end process;

begin
process(clk)

begin
if reset = '1' then state <= reset_state;
elsif clk'event and clk = '1' then
  case state is
  when reset_state =>
    ME_to_ME_sub_ready <= '0';
  start_data_transfer <= '0';

end if;
end if;
end process;

8.2.11 Bridging unit

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity ME_to_ME_sub is
port(clk, reset : in std_logic;
    mv : in std_logic_vector(7 downto 0);
    SAD_threshold : in std_logic_vector(15 downto 0);
    SAD_in : in std_logic_vector(15 downto 0);
    ME_sub_ready, ME_ready : in std_logic;
    data_ready : in std_logic;
    need_process : out std_logic;
    start_data_transfer : out std_logic;
    ME_to_ME_sub_ready : out std_logic
);
end ME_to_ME_sub;

architecture ME_to_ME_sub of ME_to_ME_sub is

begin
process(clk)

begin
if reset = '1' then state <= reset_state;
elsif clk'event and clk = '1' then
  case state is
  when reset_state =>
    ME_to_ME_sub_ready <= '0';
  start_data_transfer <= '0';

end if;
end if;
end process;

begin
process(clk)

begin
if reset = '1' then state <= reset_state;
elsif clk'event and clk = '1' then
  case state is
  when reset_state =>
    ME_to_ME_sub_ready <= '0';
  start_data_transfer <= '0';

end if;
end if;
end process;
if reset = '0' then state <= idle;
else state <= reset_state;
end if;

when idle =>
ME_to_ME_sub_ready <= '0';

if ME_ready='1' and ME_sub_ready='1' then
if conv_integer(SAD_in) < conv_integer(SAD_threshold) then state <=
not_process_state;
else state <= need_process_state;
end if;
end if;

when need_process_state =>
start_data_transfer <= '1';

need_process <= '1';
if data_ready = '1' then start_data_transfer <= '0'; state <= idle;
ME_to_ME_sub_ready <= '1';
else state <= need_process_state;
end if;

when not_process_state =>
start_data_transfer <= '0';
need_process <= '0';
ME_to_ME_sub_ready <= '1';
state <= reset_state;
end case;
end if;
end process;
end architecture ME_to_ME_sub;

8.2.12 ME Toplevel

library ieee;
use ieee.std_logic_1164.all;

entity ME is
port(me_reset,me_start, me_ME_sub_ready: in std_logic;
    me_clk : in std_logic;
    me_SAD_threshold : in std_logic_vector(15 downto 0);
    need_process : out std_logic;
    mv_out, mv_out_sub, in_c_in_p : out std_logic_vector(7 downto 0);
    sub_c_row, sub_c_col : out std_logic_vector(3 downto 0);
    sub_p_row, sub_p_col : out std_logic_Vector(3 downto 0);
    read_write, done, transfer_done : out std_logic;
    ME_sub_add_select1, ME_sub_add_select2 : out std_logic);
end ME;

Architecture str_ME of ME is

component delay_clock
port(clk : in std_logic;
    delay_in : in std_logic;
    delay_out : out std_logic);
end component;
component controller
port(reset, start : in std_logic;
    ME_to_ME_sub_ready : in std_logic;
    clk : in std_logic;
    reset_AGU, enable_SPU, start_AGU : out std_logic;
    done_frame : out std_logic;
    Block_num_col, Block_num_row : out std_logic_vector(3 downto 0);
    position : out std_logic_vector(3 downto 0)
);
end component;

component AGU
port(start_c_row,start_c_col, start_p_row, start_p_col : in
    std_logic_vector(7 downto 0);
    reset, start, elk, start_transfer, ME_to_ME_sub_ready :
in std_logic;
    done_block, start_cmp : out std_logic;
    add_c_row, add_c_col, add_p_row, add_p_col,
    add_p1_row, add_p1_col : out std_logic_vector(7 downto 0);
    Z_cp, Z_p1 : out std_logic;
    reset_inter_pixel, reset_inter_bit, reset_cmp, reset_PE : out std_logic
);
end component;

component SPU
port(Block_num_col, Block_num_row : in std_logic_vector(3 downto 0);
    position : in std_logic_vector(3 downto 0);
    enable : in std_logic;
    start_c_row, start_c_col,
    start_p_row, start_p_col : out std_logic_vector(7 downto 0)
);
end component;

component PE
port(a,b : in std_logic_vector(7 downto 0);
    reset : in std_logic;
    Clk : in std_logic;
    SAD : inout std_logic_vector(15 downto 0)
);
end component;

component comparator
port(mv_ready : out std_logic;
    PE0, PE1, PE2, PE3, PE4, PE5, PE6, PE7, PE8, PE9, PE10,
    PE11, PE12, PE13, PE14, PE15 : in std_logic_vector(15 downto 0);
    reset_cmp, start_cmp : in std_logic;
    clk : in std_logic;
    SAD_threshold : in std_logic_vector(15 downto 0);
    SAD, min_SAD_out : out std_logic_vector(15 downto 0);
    mv_out, mv_out_sub : out std_logic_vector(7 downto 0)
);
end component;

component Interconnection
port(c, p, pl : in std_logic_vector(7 downto 0);
    reset_diff_bit, reset_diff_pixel : in std_logic;
    clk : in std_logic;
    PE0c, PE0p, PE1c, PE1p, PE2c, PE2p, PE3c, PE3p, PE4c, PE4p, PE5c,
    PE5p, PE6c, PE6p, PE7c, PE7p,
    PE8c, PE8p, PE9c, PE9p, PE10c, PE10p, PE11c, PE11p, PE12c, PE12p, PE13c,
    PE13p, PE14c, PE14p,
    PE15c, PE15p : inout std_logic_vector(7 downto 0)
);
end component;

component ME_mux21
port(mux_select : in std_logic;
    ME_add1, ME_add2 : in std_logic_vector(7 downto 0);
    ME_add : out std_logic_vector(7 downto 0)
);
end component;
component mem_current_frame
port(add_c_row, add_c_col : in std_logic_vector(7 downto 0);
    out_c : out std_logic_vector(7 downto 0);
    clk : in std_logic
);
end component;

component mux_mem_inter
port(z_cp, z_p1 : in std_logic;
    clk : in std_logic;
    in_c, in_p, in_p1 : in std_logic_vector(7 downto 0);
    out_c, out_p, out_p1 : out std_logic_vector(7 downto 0)
);
end component;

class Mem_previous_frame
port(add_p_row, add_p_col : in std_logic_vector(7 downto 0);
    add_p1_row, add_p1_col : in std_logic_vector(7 downto 0);
    out_p : out std_logic_vector(7 downto 0);
    out_p1 : out std_logic_vector(7 downto 0);
    clk : in std_logic
);
end class;

component ME_to_ME_sub
port(clk, reset : in std_logic;
    mv : in std_logic_vector(7 downto 0);
    SAD_threshold : in std_logic_vector(15 downto 0);
    SAD_in : in std_logic_vector(15 downto 0);
    ME_sub_ready, ME_ready : in std_logic;
    data_ready : in std_logic;
    need_process : out std_logic;
    start_data_transfer : out std_logic;
    ME_to_ME_sub_ready : out std_logic);
end component;

component TransME
port(clk, start_transfer : in std_logic;
    s_c_row, s_c_col, s_p_row, s_p_col : in std_logic_vector(7 downto 0);
    data_ready : out std_logic;
    c_row, c_col, p_row, p_col : out std_logic_vector(7 downto 0);
    sub_c_row, sub_c_col : out std_logic_vector(3 downto 0);
    sub_p_row, sub_p_col : out std_logic_vector(5 downto 0);
    read_write, ME_add_select : out std_logic;
    ME_sub_add_select1, ME_sub_add_select2 : out std_logic
);
end component;

signal me_z_cp, me_z_p1, me_ME_add_select, me_mv_ready : std_logic;
signal me_start_cmp, me_reset_AGU, me_start_AGU,
    me_done_block, me_need_process, me_read_write,
    me_reset_PE,
    me_delay_reset_pixel, me_delay_reset_bit,
    me_done_frame, me_ME_to_ME_sub_ready, me_start_transfer, me_data_ready
: std_logic;
signal me_ME_sub_add_select1, me_ME_sub_add_select2 : std_logic;

signal me_delay_reset_pixel, me_delay_reset_bit;
me_delay_reset_PE, me_delay_delay_start_cmp, me_delay_delay_reset_cmp, me_delay_x_cp, me_delay_x_p1 : std_logic;

signal me_position, me_block_num_col,
    me_block_num_row, me_sub_p_row, me_sub_p_col : std_logic_vector(3 downto 0);
signal me_sub_p_row, me_sub_p_col : std_logic_vector(5 downto 0);
signal me_PE0_c, me_PE0_p, me_PE1_c, me_PE1_p, me_PE2_c,
me_PE2_p, me_PE3_c, me_PE3_p, me_PE4_c, me_PE4_p,
me_PE8_p, me_PE9_c, me_PE9_p,
me_PE12_p, me_PE13_c, me_PE13_p,
me_PE15_p, me_mv, me_mv_out, me_mv_out_sub, me_c_col, me_c_row, me_p_row, me_p_col,
me_memo_out_c, me_memo_out_p, me_memo_out_pl, me_start_c_row,
me_start_c_col,
me_add_p_row,
me_add_p1_col, me_mux_out_c, me_mux_out_p;
std_logic_vector(7 downto 0);
signal me_SAD, me_min_SAD_out,
me_PE0_SAD, me_PE1_SAD, me_PE2_SAD, me_PE3_SAD, me_PE4_SAD, me_PE5_SAD, me_PE6_SAD,
me_PE7_SAD, me_PE8_SAD, me_PE9_SAD, me_PE10_SAD, me_PE11_SAD, me_PE12_SAD, me_PE13_SAD,
me_PE14_SAD, me_PE15_SAD : std_logic_vector(15 downto 0);

begin
ME_mux21crow:ME_mux21 port map(mux_select =>
me_ME_add_select, ME_addl => me_add_c_row, ME_add2 => me_c_row,
ME_mux21ccol:ME_mux21 port map(mux_select =>
me_ME_add_select, ME_addi => me_add_c_col, ME_add2 => me_c_col,
ME_mux21prow:ME_mux21 port map(mux_select =>
me_ME_add_select, ME_addl => me_add_p_row, ME_add2 => me_p_row,
ME_mux21pcol:ME_mux21 port map(mux_select =>
me_ME_add_select, ME_addl => me_add_p_col, ME_add2 => me_p_col,
TransME_map:TransME port map(clk
=> me_clk, start_transfer => me_start_transfer, s_c_row => me_start_c_row,
s_c_col
=> me_start_c_col, s_p_row => me_start_p_row, s_p_col => me_start_p_col, data_ready => me_data_ready,
c_row => me_c_row, c_col =>
me_c_col, p_row => me_p_row, p_col => me_p_col, sub_c_row => me_sub_c_row,
sub_c_col => me_sub_c_col, sub_p_row => me_sub_p_row, sub_p_col => me_sub_p_col, read_write => me_read_write,
ME_Add_select =>
me_ME_add_select, ME_sub_add select1 => me_ME_sub_add select1, ME_sub_add select2 => me_ME_sub_add select2);
ME_sub:ME_to_ME_sub port map(clk => me_clk, reset => me_reset, mv => me_mv,
SAD_threshold => me_SAD_threshold,
SAD_in => me_min_SAD_out, MB_sub_ready => me_ME_sub_ready, ME_ready
=> me_done_block, data_ready => me_data_ready,
need_process => me_need_process,
start_data_transfer => me_start_transfer, MB_to_ME_sub_ready =>
me_ME_to_ME_sub_ready);

controller port map(reset => me_reset, start => me_start,
MB_to_ME_sub_ready => me_ME_to_ME_sub_ready,

me_AGU, done_frame => me_done_frame, block_num_col => me_block_num_col,
block_num_row => me_block_num_row, position => me_position, clk => me_clk);

SPU_map: SPU port
map(block_num_col => me_block_num_col, block_num_row => me_block_num_row, position => me_position,
enable => me_enable_SPU, start_c_row => me_start_c_row, start_c_col => me_start_c_col,
map(clk=>me_clk,delay_in=>me_x_cp,delay_out=>me_delay_x_cp);
delay7:delay_clock port
map(clk=>me_clk,delay_in=>me_x_p1,delay_out=>me_delay_x_p1);

-- PE_1: PE port
map(reset=>me_delay_reset_PE,clk=>me_clk,SAD=>me_PE0_SAD,a=>me_PE0_c,b=>me_PE0_p);
PE_2: PE port
map(reset=>me_delay_reset_PE,clk=>me_clk,SAD=>me_PE1_SAD,a=>me_PE1_c,b=>me_PE1_p);
PE_3: PE port
map(reset=>me_delay_reset_PE,clk=>me_clk,SAD=>me_PE2_SAD,a=>me_PE2_c,b=>me_PE2_p);
PE_4: PE port
map(reset=>me_delay_reset_PE,clk=>me_clk,SAD=>me_PE3_SAD,a=>me_PE3_c,b=>me_PE3_p);
PE_5: PE port
map(reset=>me_delay_reset_PE,clk=>me_clk,SAD=>me_PE4_SAD,a=>me_PE4_c,b=>me_PE4_p);
PE_6: PE port
map(reset=>me_delay_reset_PE,clk=>me_clk,SAD=>me_PE5_SAD,a=>me_PE5_c,b=>me_PE5_p);
PE_7: PE port
map(reset=>me_delay_reset_PE,clk=>me_clk,SAD=>me_PE6_SAD,a=>me_PE6_c,b=>me_PE6_p);
PE_8: PE port
map(reset=>me_delay_reset_PE,clk=>me_clk,SAD=>me_PE7_SAD,a=>me_PE7_c,b=>me_PE7_p);
PE_9: PE port
map(reset=>me_delay_reset_PE,clk=>me_clk,SAD=>me_PE8_SAD,a=>me_PE8_c,b=>me_PE8_p);
PE_10: PE port
map(reset=>me_delay_reset_PE,clk=>me_clk,SAD=>me_PE9_SAD,a=>me_PE9_c,b=>me_PE9_p);
PE_11: PE port
map(reset=>me_delay_reset_PE,clk=>me_clk,SAD=>me_PE10_SAD,a=>me_PE10_c,b=>me_PE10_p);
PE_12: PE port
map(reset=>me_delay_reset_PE,clk=>me_clk,SAD=>me_PE11_SAD,a=>me_PE11_c,b=>me_PE11_p);
PE_13: PE port
map(reset=>me_delay_reset_PE,clk=>me_clk,SAD=>me_PE12_SAD,a=>me_PE12_c,b=>me_PE12_p);
PE_14: PE port
map(reset=>me_delay_reset_PE,clk=>me_clk,SAD=>me_PE13_SAD,a=>me_PE13_c,b=>me_PE13_p);
PE_15: PE port
map(reset=>me_delay_reset_PE,clk=>me_clk,SAD=>me_PE14_SAD,a=>me_PE14_c,b=>me_PE14_p);
PE_16: PE port
map(reset=>me_delay_reset_PE,clk=>me_clk,SAD=>me_PE15_SAD,a=>me_PE15_c,b=>me_PE15_p);

need_process <= me_need_process;
mv_out <= me_mv_out;
mv_out_sub <= me_mv_out_sub;
sub_c_col <= me_sub_c_col; sub_c_row <= me_sub_c_row; sub_p_row <=
me_sub_p_row; sub_p_col <= me_sub_p_col;
read_write <= me_read_write;
done<= me_done_frame;
MB_sub_add_select1<= me_MB_sub_add_select1;
MB_sub_add_select2<= me_MB_sub_add_select2;
in_p<= me_mem_out_c;
in_p<= me_mem_out_p;
transfer_done <= me_data_ready;
end architecture str_ME;

8.2.13 8x8 AGU

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

despery AGU_sub is

port (  
start_c_row, start_c_col : in std_logic_vector(3 downto 0);
start_p_row, start_p_col : in std_logic_vector(5 downto 0);
reset, start,clk : in std_logic;
done_block_sub, start_cmp : out std_logic;
add_c_row,add_c_col : out std_logic_vector(3 downto 0);

add_p_row, add_p_col, add_p1_row, add_p1_col, add_p2_row, add_p2_col :
std_logic_vector(5 downto 0);
Z_cp, Z_pl, Z_p2 : out std_logic;
reset_inter_pixel, reset_inter_bit, reset_cmp, reset_PE : out std_logic
end AGU_sub;

Architecture RTL_AGU_sub of AGU_sub is

type state_type is (idle, output, done_16, finish);
shared variable s_c_row, s_c_col,
s_p_row, s_p_col, c_row, c_col, p_row, p_col, pl_row, pl_col, p2_row, p2_col : integer;
shared variable i_cp, i_p1, l_cp, l_p1, i_p2, l_p2, line : integer;
shared variable done : bit := '0';
signal state : state_type;
begin
get_state: process(clk, reset, start)
begin
if reset = '1' then done_block_sub <= '0'; state <= idle;
elsif (clk'event and clk = '1') then
  case state is
    when idle =>
      add_c_row <= "0000";
      add_c_col <= "0000";
      add_p_row <= "000000";
      add_p_col <= "000000";
      add_p1_row <= "000000";
      add_p1_col <= "000000";
      add_p2_row <= "000000";
      add_p2_col <= "000000";
      done_block_sub <= '0';
      reset_PE <= '1';
      reset_cmp <= '1';
      start_cmp <= '0';
      Z_cp <= '1';
      Z_p1 <= '1';
      Z_p2 <= '1';
      reset_inter_pixel <= '1';
      reset_inter_bit <= '1';
      done := '0';
      line := 0;
      s_c_row := conv_integer(start_c_row);
      s_c_col := conv_integer(start_c_col);
      s_p_row := conv_integer(start_p_row) + 1;
      s_p_col := conv_integer(start_p_col) + 1;
      i_cp := 0; i_p1 := 0; l_cp := 0; l_p1 := 0; i_p2 := 0; l_p2 := 0;
      if start = '1' then state <= output;
      else
        state <= idle;
      end if;
    when output => -- output address
      start_cmp <= '0';
      reset_PE <= '0';
      reset_cmp <= '0';
      reset_inter_pixel <= '0';
      reset_inter_bit <= '0';
  end when;
end case;
end get_state;
end AGU_sub;
if (done = '1') then state <= finish;
else state <= output;

if (l_cp < 8) then
  Z_cp <= '0';
c_row := s_c_row + l_cp;
c_col := s_c_col + i_cp;
p_row := s_p_row + l_cp;
p_col := s_p_col + i_cp;
  add_c_row <= std_logic_vector(c_row,4);
  add_c_col <= std_logic_vector(c_col,4);
  add_p_row <= std_logic_vector(p_row,6);
  add_p_col <= std_logic_vector(p_col,6);
else Z_cp <= '1';
  add_c_row <= "0000";
  add_c_col <= "0000";
  add_p_row <= "000000";
  add_p_col <= "000000";
end if;

if (l_pl > 0 and l_pl < 9) then
  Z_pl <= '0';
  pl_row := l_pl + s_p_row - 1;
  pl_col := s_p_col + 8 + i_pl;
  add_pl_row <= std_logic_vector(pl_row,6);
  add_pl_col <= std_logic_vector(pl_col,6);
else Z_pl <= '1';
  add_pl_row <= "000000";
  add_pl_col <= "000000";
end if;

if (i_cp=7 and l_cp=9 and line = 15) then done := '1';start_cmp <= '1';
elsif (i_cp=7 and l_cp=9) then
  i_cp := 0; l_cp := 0; i_pl := 0; l_pl := 0;i_p2:=0;i_p2:=0;
line := line + 1;
  s_p_row := s_p_row + 1;state <= done_16;
elsif (i_cp=7 ) then
  i_cp := 0; i_pl := 0; l Cp := l_cp + 1; l_pl := l_p1 + 1;i_p2 :=0;
  start_cmp <= '0'; reset_PE <= '0';
else i_cp := i_cp + 1; i_pl := i_pl + 1; i_p2 := i_p2 + 1;start_cmp <= '0';reset_PE <= '0';
end if;
end if;

when done_16 =>
  start_cmp <= '1';
  reset_PE <= '1';
  reset_inter_bit <= '1';
  state <= output;

when finish =>
  done_block_sub <= '1';
  reset_PE <= '1';
end if;
end process;
end architecture RTL_AGU_sub;

8.2.14 8x8 Controller

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity Controller_sub is
port(
    reset, start : in std_logic;
    ME_sub_tc_ME_sub_sub_ready : in std_logic;
    elk : in std_logic;
    reset_AGU, enable_SPU, start_AGU : out std_logic;
    done_block, ME_sub_ready : out std_logic;
    position : out std_logic_vector(1 downto 0);
);
end entity Controller_sub;

Architecture FSM_Controller_sub of Controller_sub is

type state_type is (idle, startup, output1, output2, output3, output4, finish);
signal state : state_type;
begin

process(clk,reset)

variable block_count : integer range 0 to 4;
--variable sub_block_count : integer range 0 to 14;
begin
if reset = '1' then
    state <= idle;
elsif clk'event and clk = '1' then
    case state is
    when idle =>
        block_count := 0;
        done_block <= '1';
        ME_sub_ready <= '1';
        if start = '1' then
            reset_AGU <= '1';
            start_AGU <= '1';
            state <= startup;
        else
            state <= idle;
        end if;

    when startup =>
        reset_AGU <= '0'; enable_SPU <= '0';
        done_block <= '0'; position <= "00"; ME_sub_ready <= '0';
        state <= output1;

    when output1 =>
        enable_SPU <= '1';
        position <= "00";

end case;
end if;
end process;
end architecture RTL_AGU_sub;
if (ME_sub_to_ME_sub_sub_ready = '1') then
block_count := block_count + 1; state <= output2; reset_AGU <= '1';
else state <= output1;
end if;

when output2 =>
reset_AGU <= '0';
position <= "01";

if (ME_sub_to_ME_sub_sub_ready = '1') then
block_count := block_count + 1; reset_AGU <= '1';
state <= output3;
else state <= output2;
end if;

when output3 =>
reset_AGU <= '0';
position <= "10";

if (ME_sub_to_ME_sub_sub_ready = '1') then
block_count := block_count + 1; reset_AGU <= '1';
state <= output4;
else state <= output3;
end if;

when output4 =>
reset_AGU <= '0';
position <= "11";

if (ME_sub_to_ME_sub_sub_ready = '1') then
block_count := block_count + 1; reset_AGU <= '1';
state <= finish; done_block <= '1';
else state <= output4;
end if;

when finish =>
state <= idle;
done_block <= '1';
end case;
end if;
end process;
end architecture FSM_Controller_sub;

8.2.14 8x8 Interconnection

library ieee;
use ieee.std_logic_1164.all;
entity Interconnection_sub is
port (c, p, pl, p2 : in std_logic_vector(7 downto 0);
reset_dff_bit, reset_dff_pixel : in std_logic;
clk : in std_logic;

PE0c, PE0p, PE1c, PE1p, PE2c, PE2p, PE3c, PE3p, PE4c, PE4p, PE5c,
PE5p, PE6c, PE6p, PE7c, PE7p, PE8c, PE8p,
PE9c, PE9p, PE10c, PE10p, PE11c, PE11p, PE12c, PE12p, PE13c, PE13p,
PE14c, PE14p, PE15c,

PE15p : inout std_logic_vector(7 downto 0)
);
end Interconnection_sub;
Architecture RTL_Interconnection_sub of Interconnection_sub is

signal
Dffbit10,Dffbit20,Dffbit11,Dffbit21,Dffbit12,Dffbit22,Dffbit13,Dffbit23,Dffbit14,Dffbit24,
Dffbit15,Dffbit25,Dffbit16,Dffbit26 : bit;

begin
process(c,p)
begin
PE0c <= c;
PE0p <= p;
end process;

P_output1 :process(p1,Dffbit10)
begin
begin
case Dffbit10 is
when '0' =>
PE1p <= p1;
when '1' =>
PE1p <= p;
end case;
end process;

P_output2 :process(p1,Dffbit11)
begin
begin
case Dffbit11 is
when '0' =>
PE2p <= p1;
when '1' =>
PE2p <= p;
end case;
end process;

P_output3 :process(p1,Dffbit12)
begin
begin
case Dffbit12 is
when '0' =>
PE3p <= p1;
when '1' =>
PE3p <= p;
end case;
end process;

P_output4 :process(p1,Dffbit13)
begin
begin
case Dffbit13 is
when '0' =>
PE4p <= p1;
when '1' =>
PE4p <= p;
end case;
end process;

P_output5 :process(p1,Dffbit14)
begin
begin
case Dffbit14 is
when '0' =>
PE5p <= p1;
when '1' =>
PE5p <= p;
end case;
end process;

P_output6 :process(p1,Dffbit15)
begin
begin
case Dffbit15 is
when '0' =>
PE6p <= p1;
end process;
P_output7 : process(p, p1, Dffbit16)
begin
    case Dffbit16 is
        when '0' =>
            PE7p <= p1;
        when '1' =>
            PE7p <= p;
    end case;
end process;

P_output8 : process(p1, p2, Dffbit20)
begin
    PE8p <= p1;
    case Dffbit20 is
        when '0' =>
            PE9p <= p2;
        when '1' =>
            PE9p <= p1;
    end case;
end process;

P_output9 : process(p1, p2, Dffbit21)
begin
    case Dffbit21 is
        when '0' =>
            PE10p <= p2;
        when '1' =>
            PE10p <= p1;
    end case;
end process;

P_output10 : process(p1, p2, Dffbit22)
begin
    case Dffbit22 is
        when '0' =>
            PE11p <= p2;
        when '1' =>
            PE11p <= p1;
    end case;
end process;

P_output11 : process(p1, p2, Dffbit23)
begin
    case Dffbit23 is
        when '0' =>
            PE12p <= p2;
        when '1' =>
            PE12p <= p1;
    end case;
end process;

P_output12 : process(p1, p2, Dffbit24)
begin
    case Dffbit24 is
        when '0' =>
            PE13p <= p2;
        when '1' =>
            PE13p <= p1;
    end case;
end process;

P_output13 : process(p1, p2, Dffbit25)
begin
    case Dffbit25 is

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

begin
  if (clk\'event and clk = '1') then
    if (reset_dff_bit = '1') then
      Dffbit1 <= '0'; Dffbit12 <= '0'; Dffbit13 <= '0'; Dffbit14 <= '0';
      Dffbit20 <= '0'; Dffbit21 <= '0'; Dffbit22 <= '0'; Dffbit23 <= '0';
      Dffbit16 <= '0'; Dffbit15 <= '0'; Dffbit14 <= '0';
      else
        Dffbit1 <= Dffbit12; Dffbit12 <= Dffbit11; Dffbit11 <= Dffbit10;
        Dffbit10 <= Dffbit22;
        Dffbit20 <= '1';
        end if;
        if (reset_dff_pixel = '1') then
          PE1c <= (others => '0'); PE2c <= (others => '0');
          PE3c <= (others => '0'); PE4c <= (others => '0');
          PE5c <= (others => '0'); PE6c <= (others => '0');
          PE7c <= (others => '0'); PE8c <= (others => '0');
          PE9c <= (others => '0'); PE10c <= (others => '0');
          PE11c <= (others => '0'); PE12c <= (others => '0');
          PE13c <= (others => '0'); PE14c <= (others => '0');
          PE15c <= (others => '0');
          PE16c <= PE14c; PE14c <= PE13c; PE13c <= PE12c; PE12c <= PE11c;
          PE9c <= PE8c; PE8c <= PE7c; PE7c <= PE6c; PE6c <= PE5c; PE5c <= PE4c;
          PE2c <= PE1c; PE1c <= PE0c;
        end if;
      end if;
  end if;
end architecture RTL_Interconnection_sub;

8.2.15 4x4 AGU

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

begin

end process;
end process;

P_outputl4 : process(p1,p2,Dffbit26)
begin
  case Dffbit26 is
    when '0' =>
      PE14p <= p2;
    when '1' =>
      PE14p <= p1;
  end case;
end process;
c_output_process: process(clk)
begin

end process;

end architecture RTL_Interconnection_sub;
use ieee.std_logic_arith.all;

entity AGU_sub_sub is
port (  
  start_c_row, start_c_col : in std_logic_vector(2 downto 0);
  start_p_row, start_p_col : in std_logic_vector(4 downto 0);
  reset, start_clk : in std_logic;
  done_block_sub, start_cmp : out std_logic;
  add_c_row, add_c_col : out std_logic_vector(2 downto 0);
  add_p_row, add_p_col, add_p1_col, add_p2_row, add_p2_col,
  add_p3_row, add_p3_col, add_p4_row, add_p4_col : out std_logic_vector(4
downto 0);
  Z_cp, Z_p1,Z_p2,Z_p3,Z_p4 : out std_logic;
  reset_inter_pixel, reset_inter_bit, reset_cmp, reset_PE : out std_logic
);  
end AGU_sub_sub;

Architecture RTL_AGU_sub_sub of AGU_sub_sub is

type state_type is (idle, output, done_16, finish);
shared variable s_c_row, s_c_col,
  s_p_row, s_p_col, c_row, c_col, p_row, p_col, p1_row, p1_col, p2_row, p2_col,
  line : integer;
shared variable done : bit := '0';

signal state : state_type;

begin
get_state: process(clk, reset, start)
begin
if reset = '1' then state <= idle; done_block_sub <= '0';
elsif (clk'event and clk = '1') then
  case state is
  when idle =>
    add_c_row <= "000";
    add_c_col <= "000";
    add_p_row <= "00000";
    add_p_col <= "00000";
    add_p1_row <= "00000";
    add_p1_col <= "00000";
    add_p2_row <= "00000";
    add_p2_col <= "00000";
    add_p3_row <= "00000";
    add_p3_col <= "00000";
    add_p4_row <= "00000";
    add_p4_col <= "00000";
    done_block_sub <= '0';
    reset_PE <= '1';
    reset_cmp <= '1';
    start_cmp <= '0';
    Z_cp <= '1';
    Z_p1 <= '1';
    Z_p2 <= '1';
    Z_p3 <= '1';

end if;
end process get_state;

122
Z_p4 <= '1';
reset_inter_pixel <= '1';
reset_inter_bit <= '1';
done := '0';
line := 0;
s_c_row := conv_integer(start_c_row);
s_c_col := conv_integer(start_c_col);
s_p_row := conv_integer(start_p_row)+1;
s_p_col := conv_integer(start_p_col)+1;

if start = '1' then state <= output;
else state <= idle;
end if;

when output => -- output address
start_cmp <= '0';
reset_PE <= '0';
reset_cmp <= '0';
reset_inter_pixel <= '0';
reset_inter_bit <= '0';
if (done = '1') then state <= finish;
else state <= output;
end if;

if (l_cp < 4) then
Z_cp <= '0';
c_row := s_c_row + l_cp;
c_col := s_c_col + i_cp;
p_row := s_p_row + l_cp;
p_col := s_p_col + i_cp;
add_c_row <= conv_std_logic_vector(c_row,3);
add_c_col <= conv_std_logic_vector(c_col,3);
add_p_row <= conv_std_logic_vector(p_row,5);
add_p_col <= conv_std_logic_vector(p_col,5);
else Z_cp <= '1';
add_c_row <= "000";
add_c_col <= "000";
add_p_row <= "00000";
add_p_col <= "00000";
end if;

if (l_p1 > 0 and l_p1 < 5) then
Z_p1 <= '0';
p1_row := l_p1 + s_p_row - 1;
p1_col := s_p_col + 4 + i_p1;
add_p1_row <= conv_std_logic_vector(p1_row,5);
add_p1_col <= conv_std_logic_vector(p1_col,5);
else Z_p1 <= '1';
add_p1_row <= "00000";
add_p1_col <= "00000";
end if;

if (l_p2 > 1 and l_p2 < 6) then
Z_p2 <= '0';
p2_row := l_p2 + s_p_row - 2;
p2_col := s_p_col + 8 + i_p2;
add_p2_row <= conv_std_logic_vector(p2_row,5);
add_p2_col <= conv_std_logic_vector(p2_col,5);
else Z_p2 <= '1';
add_p2_row <= "00000";
add_p2_col <= "00000";
end if;

if (l_p3 > 2 and l_p3 < 7) then
Z_p3 <= '0';
p3_row := l_p3 + s_p_row - 3;
p3_col := s_p_col + 12 + i_p3;
add_p3_row <= conv_std_logic_vector(p3_row,5);
add_p3_col <= conv_std_logic_vector(p3_col,5);
else Z_p3 <= '1';
    add_p3_row <= "00000";
    add_p3_col <= "00000";
end if;

if(l_p4 > 3 and l_p4 < 8) then
    Z_p4 <= '0';
p4_row := l_p4 + s_p_row - 4;
p4_col := s_p_col + 16 + i_p4;
add_p4_row <= conv_std_logic_vector(p4_row,5);
add_p4_col <= conv_std_logic_vector(p4_col,5);
else Z_p4 <= '1';
    add_p4_row <= "00000";
    add_p4_col <= "00000";
end if;

if(i_cp=3 and l_cp=7 and line = 15) then done := '1';start_cmp <= '1';
elsif(i_cp=3 and l_cp=7) then
0:i_p2:=0;l_p2:=0;i_p3:=0;l_p3:=0;i_p4:=0;l_p4:=0;line := line + 1;
s_p_row := s_p_row + 1;state <= done_16;
elif(ihcp=3) then
    i_cp := 0; i_p1 := 0; Lp1 :=
    s_p_row := s_p_row + 1; state <= done_16;
elif(ihcp=3) then
    i_cp := 0; i_p1 := 0; Lp1 := Lp1 + 1; i_p2 := 0;
end if;

when done_16 =>
    start_cmp <= '1';
    reset_PE <= '0';
    reset_inter_bit <= '1';
    state <= output;

when finish =>
    done_block_sub <= '1';
    reset_PE <= '1';
end if;
end case;
end if;
end process;

end architecture RTL_AGU_sub_sub;

8.2.16 4x4 Controller

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity Controller_sub_sub is
port(
    reset, start, ME_4_to_frac_ready : in std_logic;
    done_block_sub : in std_logic;
    clk : in std_logic;
    reset_AGU, enable_SPU, start_AGU : out std_logic;
    done_block : out std_logic;
    position : out std_logic_vector(1 downto 0)
);

end entity Controller_sub_sub;

Architecture FSM_Controller_sub_sub of Controller_sub_sub is

type state_type is (idle, startup, output1, output2, output3, output4, finish);
signal state : state_type;

begin

process(clk,reset)

variable block_count : integer range 0 to 4;
variable sub_block_count : integer range 0 to 14;

begin

if reset = '1' then
    state <= idle;
elsif clk'event and clk = '1' then

case state is

when idle =>
    block_count := 0;
    done_block <= '1';
    if start = '1' then
        reset_AGU <= '1';
        start_AGU <= '1';
        state <= startup;
        done_block <= '0';
    else
        state <= idle;
    end if;

when startup =>
    reset_AGU <= '0'; enable_SPU <= '0';
    done_block <= '0'; position <= "00";
    state <= output1;

when output1 =>
    enable_SPU <= '1';
    position <= "00";
    if (ME_4_to_frac_ready = '1') then
        block_count := block_count + 1; state <= output2; reset_AGU <= '1';
    else
        state <= output1;
    end if;

when output2 =>
    reset_AGU <= '0';
    position <= "01";
    if (ME_4_to_frac_ready = '1') then
        block_count := block_count + 1; reset_AGU <= '1';
        state <= output3;
    else
        state <= output2;
    end if;

when output3 =>
    reset_AGU <= '0';
    position <= "10";
    if (ME_4_to_frac_ready = '1') then
        block_count := block_count + 1; reset_AGU <= '1';
        state <= output4;
    else
        state <= output3;
end case;

end process(clk,reset);
end if;
when output4 =>
  reset_AGU <= '0';
  position <= "11";
  if (ME_4_to_frac_ready = '1') then
    block_count := block_count + 1; reset_AGU <= '1';
    state <= finish; done_block <= '1';
  else state <= output4;
  end if;
when finish =>
  state <= idle;
  done_block <= '1';
end case;
end process;
end architecture FSM_Controller_sub_sub;

8.2.17 4x4 Interconnection

library ieee;
use ieee.std_logic_1164.all;
-- Interconnection network for full search Motion Estimation
-- coded by Xiang Li, 05/07/04

entity Interconnection_sub_sub is
  port (c, p, p1, p2, p3, p4 : in std_logic_vector(7 downto 0);
        reset_dff_bit, reset_dff_pixel : in std_logic;
        clk : in std_logic;
        PEOc, PE0p, PE1c, PE1p, PE2c, PE2p, PE3c, PE3p, PE4c, PE4p, PE5c,
        PE5p, PE6c, PE6p, PE7c, PE7p, PE8c, PE8p,
        PE9c, PE9p, PE10c, PE10p, PE11c, PE11p, PE12c, PE12p, PE13c, PE13p,
        PE14c, PE14p, PE15c,
        PE15p : inout std_logic_vector(7 downto 0)
    );
end Interconnection_sub_sub;

Architecture RTL_Interconnection_sub_sub of Interconnection_sub_sub is

signal Dffbit10, Dffbit11, Dffbit12, Dffbit20, Dffbit21, Dffbit30, Dffbit31,
Dffbit32, Dffbit40, Dffbit41, Dffbit42 : bit;

begin
PE0c <= c;
P_output1 : process(p, p1, Dffbit10)
  begin
    PE0p <= p;
    case Dffbit10 is
    when '0' =>
      PE1p <= p1;
    when '1' =>
      PE1p <= p;
    end case;
  end process;

126
P_output2 : process(p, pl, Dffbit11) begin

    case Dffbit11 is
      when '0' =>
        PE2p <= p1;
      when '1' =>
        PE2p <= p;
    end case;
end process;

P_output3 : process(p, pl, Dffbit12) begin

    case Dffbit12 is
      when '0' =>
        PE3p <= p1;
      when '1' =>
        PE3p <= p;
    end case;
end process;

P_output4 : process(p1, p2, Dffbit20) begin

    PE4p <= p1;
    case Dffbit20 is
      when '0' =>
        PE5p <= p2;
      when '1' =>
        PE5p <= p1;
    end case;
end process;

P_output5 : process(p1, p2, Dffbit21) begin

    case Dffbit21 is
      when '0' =>
        PE6p <= p2;
      when '1' =>
        PE6p <= p1;
    end case;
end process;

P_output6 : process(p1, p2, Dffbit22) begin

    case Dffbit22 is
      when '0' =>
        PE7p <= p2;
      when '1' =>
        PE7p <= p1;
    end case;
end process;

P_output7 : process(p2, p3, Dffbit30) begin

    PE8p <= p2;
    case Dffbit30 is
      when '0' =>
        PE9p <= p3;
      when '1' =>
        PE9p <= p2;
    end case;
end process;

P_output8 : process(p2, p3, Dffbit31) begin

    case Dffbit31 is
      when '0' =>
        PE10p <= p3;
      when '1' =>
        PE10p <= p2;
    end case;
end process;
end process;

P_output9 :process(p2,p3,Dffbit32)
begin
  case Dffbit32 is
    when '0' =>
      PE11p <= p3;
    when '1' =>
      PE11p <= p2;
  end case;
end process;

P_output10 :process(p3,p4,Dffbit40)
begin
  if reset_dff_bit = '1') then
    Dffbit10 <= '0';
    Dffbit11 <= '0';
    Dffbit12 <= '0';
    Dffbit20 <= '0';
    Dffbit21 <= '0';
    Dffbit22 <= '0';
    Dffbit40 <= '0';
  else
    Dffbit10 <= Dffbit11;
    Dffbit11 <= Dffbit10;
    Dffbit20 <= Dffbit21;
    Dffbit21 <= Dffbit22;
    Dffbit22 <= Dffbit23;
  end if;
end process;

P_output11 :process(p3,p4,Dffbit41)
begin
  if reset_dff_bit = '1') then
    Dffbit10 <= '0';
    Dffbit11 <= '0';
    Dffbit12 <= '0';
    Dffbit20 <= '0';
    Dffbit21 <= '0';
    Dffbit22 <= '0';
    Dffbit40 <= '0';
  else
    Dffbit10 <= Dffbit11;
    Dffbit11 <= Dffbit10;
    Dffbit20 <= Dffbit21;
    Dffbit21 <= Dffbit22;
    Dffbit22 <= Dffbit23;
  end if;
end process;

P_output12 :process(p3,p4,Dffbit42)
begin
  if reset_dff_bit = '1') then
    Dffbit10 <= '0';
    Dffbit11 <= '0';
    Dffbit12 <= '0';
    Dffbit20 <= '0';
    Dffbit21 <= '0';
    Dffbit22 <= '0';
    Dffbit40 <= '0';
  else
    Dffbit10 <= Dffbit11;
    Dffbit11 <= Dffbit10;
    Dffbit20 <= Dffbit21;
    Dffbit21 <= Dffbit22;
    Dffbit22 <= Dffbit23;
  end if;
end process;

c_output_process: 
  process(clk)
begin
  if (clk'event and clk = '1') then
    if (reset_diff_bit = '1') then
      Dffbit10 <= '0';
      Dffbit11 <= '0';
      Dffbit12 <= '0';
      Dffbit20 <= '0';
      Dffbit21 <= '0';
      Dffbit22 <= '0';
      Dffbit40 <= '0';
    else
      Dffbit10 <= Dffbit11;
      Dffbit11 <= Dffbit10;
      Dffbit20 <= Dffbit21;
      Dffbit21 <= Dffbit22;
      Dffbit22 <= Dffbit23;
    end if;
  end if;
end process;
end process;
end architecture RTL_Interconnection_sub_sub;

8.2.18 Frac_IP1

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity frac_IP1 is
port(p, p1 : in std_logic_vector(7 downto 0);
     clk, start_IP1 : in std_logic;
     quarter_p : out std_logic_vector(7 downto 0));
end;

architecture frac_IP1 of frac_IP1 is

shared variable n, u, v, x, y : integer;

begin

process(clk, start_IP1)

variable z : integer;

begin
if clk'event and clk = '1' then
if start_IP1 = '1' then
  n := 0;
  x := conv_integer(p);
  y := conv_integer(p1);
  u := x/4;
  v := y/4;
  end if;
  z := x + n*(v - u);
  quarter_p <= conv_std_logic_vector(z, 8);
  n := n + 1;
  end if;
end process;

end architecture frac_IP1;

8.2.19 Frac_IP2


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity frac_IP2 is
port(p,pl : in std_logic_vector(7 downto 0);
    clk : in std_logic;
    out_p1,out_p2,out_p3,out_p4 : out std_logic_vector(7 downto 0));
end frac_IP2;

architecture frac_IP2 of frac_IP2 is
begin
process(p,pl)
variable x,y,z,u,v,p2,p4 : integer,
begin
x := conv_integer(p);
y := conv_integer(pl);
u := x/2;
v := y/2;
p2 := 3*x/4+y/4;
p4 := x/4+3*y/4;
out_pl <= p;
out_p2 <= conv_std_logic_vector(p2,8);
out_p3 <= conv_std_logic_vector(z,8);
out_p4 <= conv_std_logic_vector(p4,8);
end process;
end architecture frac_IP2;

8.2.20 Frac_AGU

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity frac_AGU is
port(
    mode : in std_logic_vector(1 downto 0);
    start_p_row,start_p_col : in std_logic;
    clk,start_AGU,reset_AGU : in std_logic;
    reset_PE,reset_latch,
    output_PE,Z_c,start_cmp,reset_cmp,done_block,start_ip1 : out
std_logic;
    add_p_row,add_p_col,add_p1_row,add_p1_col : out std_logic_vector(4
downto 0);
    add_c_row,add_c_col : out std_logic_vector(3 downto 0)
);
end frac_AGU;

architecture frac_AGU of frac_AGU is
begin
process(clk)

variable p_row, c_row, p_col, c_col, pl_row, pl_col : integer range 0 to 16;
variable i_p, l_p, s_p_row, s_p_col : integer range 0 to 16;
variable clk_count : integer range 0 to 3;
variable size : integer range 0 to 16;

begin

if clk'event and clk = '1' then

if reset_AGU = '1' then
reset_PE <= '1'; reset_latch <= '1';
output_PE <= '0'; done_block <= '0';
clk_count := 3; Z_c <= '0'; reset_cmp <= '1';
size := 4*(conv_integer(mode) + 1);
state <= idle;
end if;

when idle =>
add_p_row <= "00000";
add_p_col <= "00000";
add_pl_row <= "00000";
add_c_row <= "0000";
add_pl_col <= "0000";
s_p_row := conv_integer(start_p_row); s_p_col :=
conv_integer(start_p_col);
l_p := 0; i_p := 0; l_c <= '0'; output_PE <= '0';
done_block <= '0';
start_cmp <= '0'; reset_latch <= '1';
reset_PE <= '1'; clk_count := 3;
if start_AGU = '1' then state <= output;
else state <= idle;
end if;

when output =>
output_PE <= '0'; reset_cmp <= '0';
reset_PE <= '0';
if (l_p=0) then
Z_c <= '0'; else Z_c <= '1';
end if;
p_row := s_p_row + l_p; p_col := s_p_col + i_p; pl_row := p_row + 1;
pl_col := p_col;
c_row := l_p; c_col := i_p;

if (clk_count = 2) then
add_p_row <= conv_std_logic_vector(p_row,5);
add_p_col <= conv_std_logic_vector(p_col,5);
add_pl_row <= conv_std_logic_vector(pl_row,5);
add_pl_col <= conv_std_logic_vector(pl_col,5);
add_c_row <= conv_std_logic_vector(c_row,4);
add_c_col <= conv_std_logic_vector(c_col,4);
clk_count := clk_count - 1;
start_ipl := '1';
elsif (clk_count > 0) then
add_p_row <= conv_std_logic_vector(p_row,5);
add_p_col <= conv_std_logic_vector(p_col,5);
add_pl_row <= conv_std_logic_vector(pl_row,5);
add_pl_col <= conv_std_logic_vector(pl_col,5);
add_c_row <= conv_std_logic_vector(c_row,4);
add_c_col <= conv_std_logic_vector(c_col,4);
clk_count := clk_count - 1; start_ipl <= '0';
else
add_p_row <= conv_std_logic_vector(p_row,5);
add_p_col <= conv_std_logic_vector(p_col,5);
add_pl_row <= conv_std_logic_vector(pl_row,5);
add_pl_col <= conv_std_logic_vector(pl_col,5);

end if;

end if;
end if;
end if;
end if;
add_pl_col <= conv_std_logic_vector(pl_col,5);
add_c_row <= conv_std_logic_vector(c_row,4);
add_c_col <= conv_std_logic_vector(c_col,4);
start_ip1 <= '0';
if (i_p = size and l_p = size) then state <= finish;
elseif (i_p = size) then
  i_p := 0; l_p := l_p + 1; state <= output;
else i_p := i_p + 1; state <= output;
end if;
clk_count := 3;
end if;

when finish =>
if clk_count > 0 then output_PE <= '1'; start_cmp <= '1';
  clk_count := clk_count - 1; state <= finish;
else state <= idle; done_block <= '1';
end if;
end case;
end if;
end process;
end architecture frac_AGU;

8.2.21 Frac_Controller

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity frac_Controller is
port(
  reset, start : in std_logic;
  done_block_sub : in std_logic;
  clk : in std_logic;
  reset_AGU, enable_SPU, start_AGU : out std_logic;
  done_block : out std_logic;
  position : out std_logic_vector(1 downto 0)
);
end entity frac_Controller;

Architecture frac_Controller of frac_Controller is

type state_type is (idle, startup, output1, output2, output3, output4, finish);
signal state : state_type;

begin

process(clk,reset)

variable block_count : integer range 0 to 4;
--variable sub_block_count : integer range 0 to 14;
begin

if reset = '1' then
  state <= idle;
elseif clk'event and clk = '1' then
  case state is
    when idle =>
      block_count := 0;
  end case;
end if;
end process;
end architecture frac_Controller;
done_block <= '1';
if start = '1' then
    reset_AGU <= '1';
    state <= startup;
    done_block <= '0';
else
    state <= idle;
end if;

when startup =>
    reset_AGU <= '0';
    enable_SPU <= '0';
    done_block <= '0';
    position <= "00";
    state <= output1;

when output1 =>
    start_AGU <= '1';
    enable_SPU <= '1';
    position <= "00";
    if (done_block_sub = '1') then
        block_count := block_count+1; start_AGU <= '0';
        state <= output2;
        reset_AGU <= '1';
    else
        state <= output1;
    end if;

when output2 =>
    reset_AGU <= '0';
    start_AGU <= '1';
    position <= "01";
    if (done_block_sub = '1') then
        block_count := block_count + 1; start_AGU <= '0';
        reset_AGU <= '1';
        state <= output3;
    else
        state <= output2;
    end if;

when output3 =>
    start_AGU <= '1';
    reset_AGU <= '0';
    position <= "10";
    if (done_block_sub = '1') then
        block_count := block_count + 1; start_AGU <= '0';
        reset_AGU <= '1';
        state <= output4;
    else
        state <= output3;
    end if;

when output4 =>
    start_AGU <= '1';
    reset_AGU <= '0';
    position <= "11";
    if (done_block_sub = '1') then
        block_count := block_count + 1; start_AGU <= '0';
        reset_AGU <= '1';
        state <= finish; done_block <= '1';
    else
        state <= output4;
    end if;

when finish =>
    state <= idle;
    done_block <= '1';
end case;
end if;
end process;
end architecture frac_Controller;

8.2.22 Frac_comparator

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity frac_comp is
  port(
    SAD1, SAD2, SAD3, SAD4 : in std_logic_vector(15 downto 0);
    clk, start_cmp, reset_cmp : in std_logic;
    mv : out std_logic_vector(5 downto 0)
  );
end frac_comp;

architecture frac_comp of frac_comp is

  type state_type is (idle, compare, output);
  signal state : state_type;

  begin
    process(clk)
    begin
      if clk'event and clk = '1' then
        if reset_cmp = '1' then
          min_SAD := "1111111111111111";
          m := 0;
          n := 0;
          state := idle;
          end if;

        case state is
          when idle =>
            n := 0;
            if start_cmp = '1' then state := compare;
            else state := idle;
            end if;

          when compare =>
            if (conv_integer(SAD1) < conv_integer(min_SAD)) then min_SAD := SAD1;
            mv_calc := m*16+1; end if;

            if (conv_integer(SAD2) < conv_integer(min_SAD)) then min_SAD := SAD2;
            mv_calc := m*16+2; end if;

            if (conv_integer(SAD3) < conv_integer(min_SAD)) then min_SAD := SAD3;
            mv_calc := m*16+3; end if;

            if (conv_integer(SAD4) < conv_integer(min_SAD)) then min_SAD := SAD4;
            mv_calc := m*16+4; end if;

            n := n + 4;

            if (n=16 and m=3) then state := output;
            elsif (n=16) then m := m + 1; state := idle;
        end case;
    end process;

end architecture frac_comp;
else state <= compare;
end if;

when output =>
  mv <= conv_std_logic_vector(mv_calc,6);
  min_SAD := "1111111111111111";
  m := 0; n:= 0;
  state <= idle;
end case;
end if;
end process;
end architecture frac_comp;

8.2.23 Frac_PE

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity frac_PE is
  port(
    c, p : in std_logic_vector(7 downto 0);
    reset, clk, output : in std_logic;
    SAD4 : inout std_logic_vector(15 downto 0)
  );
end frac_PE;

architecture frac_PE of frac_PE is
  signal SAD3, SAD2, SAD1 : std_logic_vector(15 downto 0);

begin
  process(clk)
  variable q, in_c, in_p, sad_temp: integer;

  begin

    if clk'event and clk = '1' then

      if reset = '1' then

        SAD1 <= "0000000000000000";
        SAD2 <= "0000000000000000";
        SAD3 <= "0000000000000000";
        SAD4 <= "0000000000000000";

      else

        if output = '1' then
          q := 0;
        else
          in_c := conv_integer(c); in_p := conv_integer(p); sad_temp :=
          conv_integer(SAD4);
          if (in_c > in_p) then q := in_c - in_p; else q := in_p - in_c; end if;
          end if;
          sad_temp := sad_temp + q;
          SAD4 <= SAD3; SAD3 <= SAD2; SAD2 <= SAD1; SAD1 <=
          conv_std_logic_vector(sad_temp,16);
        end if;

      end if;

    end if;

  end if;

end process;
end architecture frac_PE;
8.2.24 Frac_SPU

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

discipline frac_SPU is
port(
    --
    Block_num_col, Block_num_row : in std_logic_vector(3 downto 0);
    position : in std_logic_vector(1 downto 0);
    enable : in std_logic;
    start_p_row, start_p_col : out std_logic
);
end entity frac_SPU;

architecture frac_SPU of frac_SPU is
begin
process( position, enable)
variable p_row, p_col : integer range 0 to 8;
begin
if enable = '1' then
    case position is
    when "00" =>
        start_p_row <= '0';
        start_p_col <= '0';
    when "01" =>
        start_p_row <= '0';
        start_p_col <= '1';
    when "10" =>
        start_p_row <= '1';
        start_p_col <= '0';
    when "11" =>
        start_p_row <= '1';
        start_p_col <= '1';
    when others =>
        start_p_row <= '0';
        start_p_col <= '0';
end case;
else
    start_p_row <= '0';
    start_p_col <= '0';
end if;
end process;
end architecture frac_SPU;

8.2.25 Frac_Memory of Current Frame
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity mem_frac_c is
  port(
    add_c_row, add_c_col : in std_logic_vector(3 downto 0);
    in_c : in std_logic_vector(7 downto 0);
    read_write : in std_logic;
    out_c : out std_logic_vector(7 downto 0);
    clk : in std_logic
  );
end mem_frac_c;

Architecture mem_frac_c_of_mem_frac_c is

  type block_type is array (0 to 15, 0 to 15) of integer,
  shared variable block_sub : block_type;

begin
mem: process(clk)
  variable c : integer;
  begin
    if elk'event and clk='l' then
      case read_write is
        when '0' =>
          c := block_sub(conv_integer(add_c_row),conv_integer(add_c_col));
          out_c <= conv_std_logic_vector(c,8);
        when '1' =>
          c := conv_integer(in_c);
          block_sub(conv_integer(add_c_row),conv_integer(add_c_col)) := c;
        when others =>
      end case;
    end if;
  end process;
end architecture mem_frac_c;

8.2.26 Frac Memory Previous Frame

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity mem_frac_p is
  port(
    add_p_row, add_p_col,add_pl_row,add_pl_col : in std_logic_vector(4
downto 0);
    in_p : in std_logic_vector(7 downto 0);
    read_write : in std_logic;
    out_p, out_pl : out std_logic_vector(7 downto 0);
    clk : in std_logic
  );
end mem_frac_p;

Architecture mem_frac_p of mem_frac_p is

type block_type is array (0 to 17, 0 to 17) of integer;
shared variable block_sub : block_type;

begin

mem_process(clk)
variable p,p1,p2 : integer;
begin
if clk'event and clk='1' then
  case read_write is
  when '0' =>
    p := block_sub(conv_integer(add_p_row),conv_integer(add_p_col));
    p1 := block_sub(conv_integer(add_p1_row),conv_integer(add_p1_col));
    out_p <= conv_std_logic_vector(p,8);
    out_p1 <= conv_std_logic_vector(p1,8);
  when '1' =>
    p := conv_integer(in_p);
    block_sub(conv_integer(add_p_row),conv_integer(add_p_col)) := p;
  when others =>
    p := block_sub(conv_integer(add_p_row),conv_integer(add_p_col));
    p1 := block_sub(conv_integer(add_p1_row),conv_integer(add_p1_col));
    out_p <= conv_std_logic_vector(p,8);
    out_p1 <= conv_std_logic_vector(p1,8);
  end case;
end if;
end process;
end architecture mem_frac_p;

8.2.27 Frac_TopLevel

library ieee;
use ieee.std_logic_1164.all;

entity ME_frac is
  port(me_reset,me_start : in std_logic;
       me_clk : in std_logic;
       in_c_row,in_c_col : in std_logic_vector(3 downto 0);
       in_p_row,in_p_col : in std_logic_vector(4 downto 0);
       read_write,frac_add_select1,frac_add_select2 : in std_logic;
       frac_trans_c,frac_trans_p : in std_logic_vector(7 downto 0);
       mode : in std_logic_vector(1 downto 0);
       mv : out std_logic_vector(5 downto 0);
       done : out std_logic);
end ME_frac;

Architecture ME_frac of ME_frac is

component ME_sub_c_mux31
  port(max_select : in std_logic;
       ME_add1, ME_add2 : in std_logic_vector(3 downto 0);
       ME_sub : out std_logic_vector(3 downto 0));
end component;
component ME_sub_sub_muxP
port(mux_select : in std_logic;
    ME_add1, ME_add2 : in std_logic_vector(4 downto 0);
    ME_add : out std_logic_vector(4 downto 0));
end component;

component delay_clock_sub_3
port(clk : in std_logic;
    delay_in : in std_logic_vector(3 downto 0);
    delay_out : out std_logic_vector(3 downto 0));
end component;

component delay_clock_sub_4
port(clk : in std_logic;
    delay_in : in std_logic_vector(4 downto 0);
    delay_out : out std_logic_vector(4 downto 0));
end component;

component frac_delay_pixel
port(delay_in : in std_logic_vector(7 downto 0);
    clk,reset : in std_logic;
    delay_out : out std_logic_vector(7 downto 0));
end component;

component frac_IP1
port(p, p1 : in std_logic_vector(7 downto 0);
    clk, start_ip1 : in std_logic;
    quarter_p : out std_logic_vector(7 downto 0));
end component;

component frac_IP2
port(p, p1 : in std_logic_vector(7 downto 0);
    clk : in std_logic;
    out_p1, out_p2, out_p3, out_p4 : out std_logic_vector(7 downto 0));
end component;

component delay_clock_sub
port(clk : in std_logic;
    delay_in : in std_logic;
    delay_out : out std_logic);
end component;

component frac_controller
port(reset, start : in std_logic;
    done_block_sub : in std_logic;
    clk : in std_logic;
    reset_AGU, enable_SPU, start_AGU : out std_logic;
    done_block : out std_logic;
    position : out std_logic_vector(1 downto 0));
end component;

component frac_AGU
port(mode : in std_logic_vector(1 downto 0);
    start_p_row, start_p_col : in std_logic;
    reset_AGU, start_AGU, clk : in std_logic;
    done_block, start_cmp, output_PE, Z_c, reset_latch, start_ip1 : out std_logic;
    add_c_row, add_c_col : out std_logic_vector(3 downto 0);
    add_p_row, add_p_col,
add_p1_row, add_p1_col : out std_logic_vector(4 downto 0);
reset_cmp, reset_PE : out std_logic
);

end component;

component frac_SPU
port(
    position : in std_logic_vector(1 downto 0);
    enable : in std_logic;
    start_p_row, start_p_col : out std_logic
);
end component;

component frac_PE
port(c, p : in std_logic_vector(7 downto 0);
    reset, output : in std_logic;
    Clk : in std_logic;
    SAD4 : inout std_logic_vector(15 downto 0)
);
end component;

component frac_comp
port(SAD1, SAD2, SAD3, SAD4 : in std_logic_vector(15 downto 0);
    reset_cmp, start_cmp : in std_logic;
    clk : in std_logic;
    mv : out std_logic_vector(5 downto 0)
);
end component;

component mem_frac_c
port(add_c_row, add_c_col : in std_logic_vector(3 downto 0);
    in_c : in std_logic_vector(7 downto 0);
    read_write : in std_logic;
    out_c : out std_logic_vector(7 downto 0);
    clk : in std_logic
);
end component;

component mux_frac
port(z_c : in std_logic;
    clk : in std_logic;
    in_c : in std_logic_vector(7 downto 0);
    out_c : out std_logic_vector(7 downto 0)
);
end component;

component mem_frac_p
port(add_p_row, add_p_col : in std_logic_vector(4 downto 0);
    add_p1_row, add_p1_col : in std_logic_vector(4 downto 0);
    in_p : in std_logic_vector(7 downto 0);
    read_write : in std_logic;
    out_p : out std_logic_vector(7 downto 0);
    out_p1 : out std_logic_vector(7 downto 0);
    clk : in std_logic
);
end component;

signal me_z_c, me_read_write_c, me_read_write_p, me_start_p_row, me_start_p_col, me_read_write : std_logic;
signal me_start_cmp, me_reset_AGU, me_start_AGU,
    me_done_block_sub, me_start_ip1,
    me_reset_cmp, me_enable_SPU, me_reset_PE,
    me_read_write_block, me_read_write_area,
    me_done_block, me_reset_latch, me_delay_reset_latch, me_output_PE :
    std_logic;
--
signal
me_delay_reset_PE, me_delay_start_cmp, me_delay_reset_cmp, me_delay_z_c : std_logic;

signal me_position : std_logic_vector(1 downto 0);

signal me_add_c_row, me_add_c_col, out_c_row, out_c_col, delay_in_c_row, delay_in_c_col : std_logic_vector(3 downto 0);

signal me_add_p_row, me_add_p_col, me_add_p1_row, me_add_p1_col, out_p_row, out_p_col, delay_in_p_row, delay_in_p_col : std_logic_vector(4 downto 0);

signal me_mv_out : std_logic_vector(5 downto 0);

signal me_PE0_c, me_PE0_p, me_PE1_c, me_PE1_p, me_PE2_c, me_PE2_p, me_PE3_c, me_PE3_p,
me_mem_out_c, me_mem_out_p,
me_ip1_out, me_ip2_in1, me_ip2_in2, me_latch_2, me_latch_3, me_latch_4 :
std_logic_vector(7 downto 0);

signal me_PE0_SAD, me_PE1_SAD, me_PE2_SAD, me_PE3_SAD : std_logic_vector(15 downto 0);

begin
cont: frac_controller port map(reset => me_reset, start => me_start,
done_block_sub => me_done_block_sub,
reset_AGU => me_reset_AGU, enable_SPU => me_enable_SPU,
start_AGU => me_start_AGU, done_block => me_done_block,
position => me_position, clk => me_clk);

SPU_map: frac_SPU port map(position => me_position,
enable => me_enable_SPU,
start_p_row => me_start_p_row, start_p_col => me_start_p_col);

AGU_map: frac_AGU port map(start_p_row => me_start_p_row, output_PE =>
me_output_PE, start_ip1 => me_start_ip1,
start_p_col => me_start_p_col, reset_AGU => me_reset_AGU, start_AGU => me_start_AGU, clk => me_clk,
done_block => me_done_block_sub, start_cmp => me_start_cmp, Z_c => me_Z_c,
reset_cmp => me_reset_cmp, mode => mode, reset_latch => me_reset_latch,
add_c_row => me_add_c_row, add_c_col => me_add_c_col, add_p_row => me_add_p_row, reset_PE => me_reset_PE,
add_p_col => me_add_p_col, add_p1_row => me_add_p1_row, add_p1_col => me_add_p1_col
);

cmp: frac_comp port
map(reset_cmp => me_delay_reset_cmp, start_cmp => me_delay_start_cmp, clk => me_clk,
mv => me_mv_out, SAD1 => me_PE0_SAD,
SAD2 => me_PE1_SAD, SAD3 => me_PE2_SAD, SAD4 => me_PE3_SAD);

mux: mux_frac
port map(Z_c => me_delay_z_c,
in_c => me_mem_out_c,
out_c => me_mux_out_c,
clk => me_clk);
map(reset=>me_delay_reset_PE,clk=>me_clk,SAD4=>me_PE0_SAD,c=>me_mux_out_c,p=>me_PE0_p,output=>me_output_PE);
pe_2 : frac_PE port
map(reset=>me_delay_reset_PE,clk=>me_clk,SAD4=>me_PE1_SAD,c=>me_mux_out_c,p=>me_PE1_p,output=>me_output_PE);
pe_3 : frac_PE port
map(reset=>me_delay_reset_PE,clk=>me_clk,SAD4=>me_PE2_SAD,c=>me_mux_out_c,p=>me_PE2_p,output=>me_output_PE);
pe_4 : frac_PE port
map(reset=>me_delay_reset_PE,clk=>me_clk,SAD4=>me_PE3_SAD,c=>me_mux_out_c,p=>me_PE3_p,output=>me_output_PE);
mv <= me_mv_out;done<=me_done_block;
end architecture ME_frac;