An Efficient Hardware Architecture for H.264 Intra Prediction Algorithm

Esra Sahin and Ilker Hamzaoglu
Faculty of Engineering and Natural Sciences, Sabanci University, 34956, Istanbul, TURKEY
Email: hamzaoglu@sabanciuniv.edu

Abstract

In this paper, we present an efficient hardware architecture for real-time implementation of intra prediction algorithm used in H.264 / MPEG4 Part 10 video coding standard. The hardware design is based on a novel organization of the intra prediction equations. This hardware is designed to be used as part of a complete H.264 video coding system for portable applications. The proposed architecture is implemented in Verilog HDL. The Verilog RTL code is verified to work at 90 MHz in a Xilinx Virtex II FPGA. The FPGA implementation can process 27 VGA frames (640x480) per second.

1. Introduction

Video compression systems are used in many commercial products, from consumer electronic devices such as digital camcorders, cellular phones to video teleconferencing systems. These applications make the video compression hardware devices an inevitable part of many commercial products. To improve the performance of the existing applications and to enable the applicability of video compression to new real-time applications, recently, a new international standard for video compression is developed. This new standard, offering significantly better video compression efficiency than previous video compression standards, is developed with the collobaration of ITU and ISO standardization organizations. Hence it is called with two different names, H.264 and MPEG4 Part 10.

The video compression efficiency achieved in H.264 standard is not a result of any single feature but rather a combination of a number of encoding tools. As it is shown in the top-level block diagram of an H.264 encoder in Fig. 1, one of these tools is the intra prediction algorithm used in the baseline profile of H.264 standard [1, 2, 3]. Intra prediction algorithm generates a prediction for a Macroblock (MB) based on spatial redundancy. H.264 intra prediction algorithm achieves better coding results than the intra prediction algorithms used in the previous video compression standards. However, this coding gain comes with an increase in encoding complexity which makes it an exciting challenge to have a real-time implementation of H.264 intra prediction algorithm.

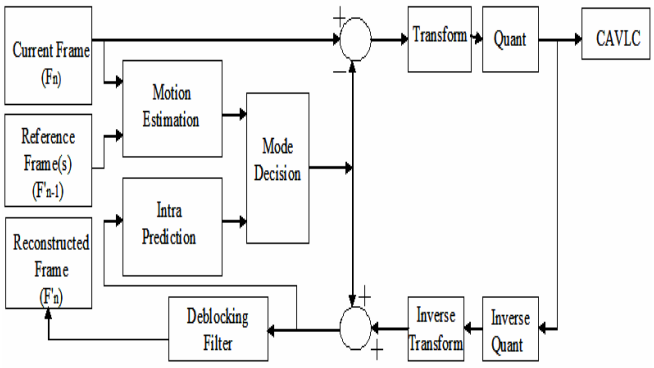

Figure 1. H.264 Encoder Block Diagram

In this paper, we present an efficient hardware architecture for real-time implementation of intra prediction algorithm used in H.264 / MPEG4 Part 10 video coding standard. The hardware design is based on a novel organization of the intra prediction equations. This hardware is designed to be used as part of a complete H.264 video coding system for portable applications. The proposed architecture is implemented in Verilog HDL. The Verilog RTL code is verified to work at 90 MHz in a Xilinx Virtex II FPGA. The FPGA implementation can process 27 VGA frames (640x480) per second.

A hardware architecture for real-time implementation of H.264 intra prediction algorithm is presented in [4, 5]. This hardware achieves higher performance than our hardware design at the expense of a much higher hardware cost. Our hardware design is a more cost-effective solution for portable applications. They use four reconfigurable datapaths, which include 12 adders, 16 multiplexers, 4 shifters and 4 clippers, in their design. They use additional adders and multiplexers for preprocessing in 16x16 plane mode and 8x8 plane mode. On the other hand, we use three reconfigurable datapaths, which include 6 adders, 12 multiplexers, 6 shifters and 2 clippers, in our design. We don't use any aditional hardware resources for 16x16 plane mode and 8x8 plane mode.

The rest of the paper is organized as follows. Section 2 explains the H.264 intra prediction algorithm. Section 3 describes the proposed architecture in detail. The implementation results are given in Section 4. Finally, Section 5 presents the conclusions.

978-3-9810801-2-4/DATE07 © 2007 EDAA

2. Overview of H.264 Intra Prediction Algorithm

Intra prediction algorithm predicts the pixels in a MB using the pixels in the available neighboring blocks. For the luma component of a MB, a 16x16 predicted luma block is formed by performing intra predictions for each 4x4 luma block in the MB and by performing intra prediction for the 16x16 MB. There are nine prediction modes for each 4x4 luma block and four prediction modes for a 16x16 luma block. A mode decision algorithm is then used to compare the 4x4 and 16x16 predictions and select the best luma prediction mode for the MB. 4x4 prediction modes are generally selected for highly textured regions while 16x16 prediction modes are selected for flat regions.

There are nine 4x4 luma prediction modes designed in a directional manner. A 4x4 luma block consisting of the pixels a to p is shown in Fig. 2. The pixels A to M belong to the neighboring blocks and are assumed to be already encoded and reconstructed and are therefore available in the encoder and decoder to generate a prediction for the current MB. Each 4x4 luma prediction mode generates 16 predicted pixel values using some or all of the neighboring pixels A to M as shown in Fig. 3. The arrows indicate the direction of prediction in each mode. The predicted pixels are calculated by a weighted average of the neighboring pixels A-M for each mode except Vertical, Horizontal and DC modes.

The prediction equations used in 4x4 Diagonal Down-Left prediction mode are shown in Fig. 4 where [y,x] denotes the position of the pixel in a 4x4 block (the top left, top right, bottom left, and bottom right positions of a 4x4 block are denoted as [0, 0], [0, 3], [3, 0], and [3, 3], respectively) and pred[y,x] is the prediction for the pixel in the position [y,x].

M	A	B	C	D	E	F	G	H
I	a	b	c	d				
J	e	f	g	h				
K	i	j	k	l				
L	m	n	o	p				

Figure 2. A 4x4 Luma Block and Neighboring Pixels

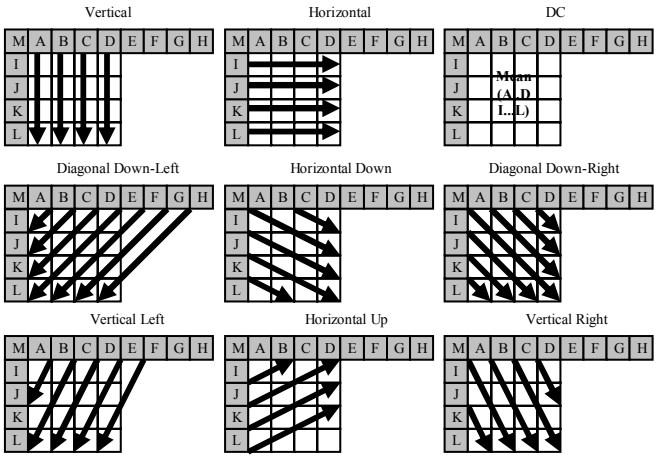

Figure 3. 4x4 Luma Prediction Modes

pred[0, 0] = A + 2B + C + 2 >> 2
pred[0, 1] = B + 2C + D + 2 >> 2
pred[0, 2] = C + 2D + E + 2 >> 2
pred[0, 3] = D + 2E + F + 2 >> 2
pred[1, 0] = B + 2C + D + 2 >> 2
pred[1, 1] = C + 2D + E + 2 >> 2
pred[1, 2] = D + 2E + F + 2 >> 2
pred[1, 3] = E + 2F + G + 2 >> 2
pred[2, 0] = C + 2D + E + 2 >> 2
pred[2, 1] = D + 2E + F + 2 >> 2
pred[2, 2] = E + 2F + G + 2 >> 2
pred[2, 3] = F + 2G + H + 2 >> 2
pred[3, 0] = D + 2E + F + 2 >> 2
pred[3, 1] = E + 2F + G + 2 >> 2
pred[3, 2] = F + 2G + H + 2 >> 2
pred[3, 3] = G + 3H + 2 >> 2

Figure 4. Prediction Equations for 4x4 Diagonal Down-Left Mode

Table 1. Availability of 4x4 Luma Prediction Modes

Availability of Neighboring 4x4 Luma Blocks	Available 4x4 Luma Prediction Modes
None available	DC
Left available, Top not available	Horizontal, DC, Horizontal-Up
Top available, Left not available	Vertical Right, DC, Vertical Left, Diagonal Down-Left
Both available	All Modes

DC mode is always used regardless of the availability of the neighboring pixels. However, it is adopted based on which neighboring pixels A-M are available. If pixels E, F, G and H have not yet been encoded and reconstructed, the value of pixel D is copied to these positions and they are marked as available for DC mode. The other prediction modes can only be used if all of the required neighboring pixels are available [2, 3]. Available 4x4 luma prediction modes for a 4x4 luma block depending on the availability of the neighboring 4x4 luma blocks are given in Table 1.

There are four 16x16 luma prediction modes designed in a directional manner. Each 16x16 luma prediction mode generates 256 predicted pixel values using some or all of the upper (H) and left-hand (V) neighboring pixels as shown in Fig. 5. Vertical, Horizontal and DC modes are similar to 4x4 luma prediction modes. Plane mode is an approximation of bilinear transform with only integer arithmetic. The prediction equations used in 16x16 Plane mode are shown in Fig. 6 where [y,x] denotes the position of the pixel in a MB (the top left, top right, bottom left, and bottom right positions of a MB are denoted as [0,0], [0,15], [15,0], and [15,15], respectively), *p* represents the neighboring pixel values and Clip1 is to clip the result between 0 and 255.

DC mode is always used regardless of the availability of the neighboring pixels. However, it is adopted based on which neighboring pixels are available. The other prediction modes can only be used if all of the required neighboring pixels are available [2, 3]. Available 16x16 luma prediction modes for a MB depending on the availability of the neighboring MBs are given in Table 2.

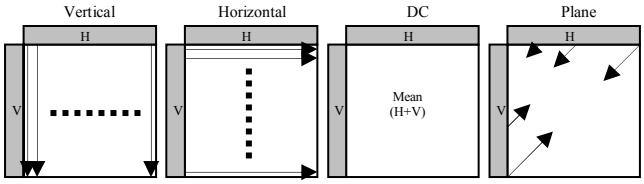

Figure 5. 16x16 Luma Prediction Modes

$$pred[y,x] = Clip1\ [((a + b * (x - 3) + c * (y - 3) + 16) >> 5]$$

$$a = 16 * (p[-1,15] + p[15,-1])$$
$$b = (5 * H + 32) >> 6$$
$$c = (5 * V + 32) >> 6$$

$$H = \sum (x'+1) * (p[-1,8 + x'] + p[-1, 6- x']),\ x' = 0, 1, 2, 3, 4, 5, 6, 7$$
$$V = \sum (y'+1) * (p[8 + y',-1] + p[6- y',-1]),\ y' = 0, 1, 2, 3, 4, 5, 6, 7$$

Figure 6. Prediction Equations for 16x16 Plane Mode

Table 2. Availability of 16x16 Luma Prediction Modes

Availability of Neighboring MBs	Available 16x16 Luma Prediction Modes
None available	DC
Left available, Top not available	Horizontal, DC
Top available, Left not available	Vertical, DC
Both available	All Modes

For the chroma components of a MB, a predicted 8x8 chroma block is formed for each 8x8 chroma component by performing intra prediction for the MB. There are four 8x8 chroma prediction modes which are similar to 16x16 luma prediction modes. A mode decision algorithm is used to compare the 8x8 predictions and select the best chroma prediction mode for each chroma component of the MB. Both chroma components of a MB always use the same prediction mode.

3. Proposed Hardware Architecture

The proposed hardware architecture for intra prediction is shown in Fig. 7. The proposed hardware generates the predicted pixels for both luma and chroma components of a MB using available prediction modes. In the proposed hardware, there are two parts operating in parallel in order to perform intra prediction faster.

The upper part is used for generating the predicted pixels for the luma component of a MB using available 16x16 luma prediction modes and for generating the predicted pixels for the chroma components of a MB using available 8x8 chroma prediction modes. The size of register files that are used for the current MB and the prediction buffer is 384x8, because they are used for storing both luma and chroma components of the current and predicted MB respectively.

The lower part is used for generating the predicted pixels for each 4x4 block in the luma component of a MB using available 4x4 luma prediction modes. The lower part is more computationally demanding and it is the bottleneck in the

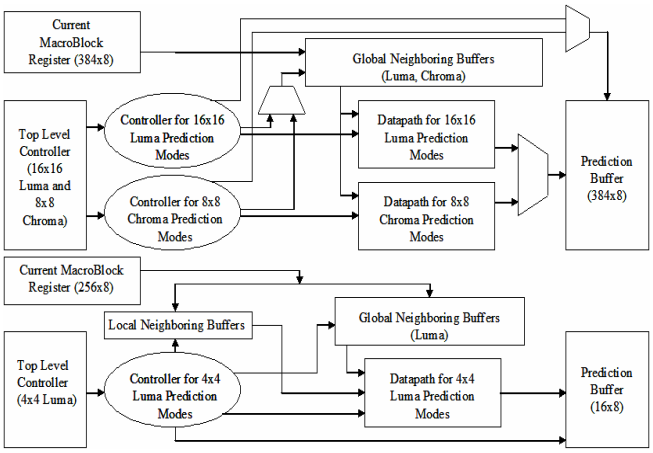

Figure 7. Intra Prediction Hardware

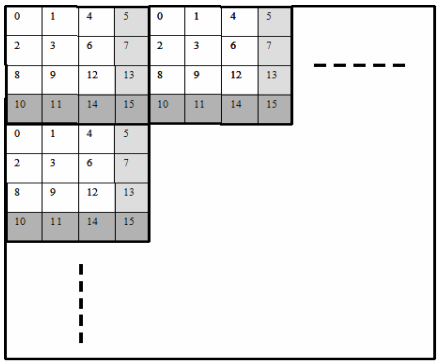

Figure 8. 16x16 and 4x4 Luma Blocks in a Frame

intra prediction hardware. The size of the current MB register file is 256x8, because it is used for storing only luma components of the current MB. The size of the prediction buffer is 16x8 since it is used for storing the predicted pixels for a 4x4 luma block.

Two local neighboring buffers, local vertical register file and local horizontal register file, are used to store the neighboring pixels in the previously coded and reconstructed neighboring 4x4 luma blocks in the current MB. After a 4x4 luma block in the current MB is coded and reconstructed, the neighboring pixels in this block are stored in the corresponding local register files.

Local vertical register file is used to store the neighboring pixels d, h, l, and p in the left-hand previously coded and reconstructed neighboring 4x4 luma blocks in the current MB. Local horizontal register file is used to store the neighboring pixels m, n, o, and p in the upper previously coded and reconstructed 4x4 luma blocks in the current MB. The proposed hardware uses this data to determine the neighboring pixels in the left-hand and upper previously coded neighboring 4x4 luma blocks in the current MB.

Six global neighboring buffers, three global vertical neighboring buffers and three global horizontal neighboring buffers, are used to store the neighboring pixels in the

previously coded and reconstructed neighboring MBs of the current MB. The 16x16 luma components of the MBs in a frame and the 4x4 luma blocks in them are shown in Fig. 8.

Global luma vertical register file is used to store the neighboring pixels d, h, l, and p in the 4x4 luma blocks 5, 7, 13 and 15 of the previously coded MB. The proposed hardware uses this data to determine the neighboring pixels in the left-hand previously coded neighboring MB of the 4x4 luma blocks 0, 2, 8, and 10 in the current MB. Global Cb vertical register file and global Cr vertical register file are used for the chroma (Cb and Cr) components of the MBs.

Global luma horizontal register file is used to store the neighboring pixels m, n, o, and p in the luma blocks 10, 11, 14, and 15 of the previously coded MBs in the previously coded MB row of the frame. The proposed hardware uses this data to determine the neighboring pixels in the upper previously coded neighboring MB of the 4x4 luma blocks 0, 1, 4, and 5 in the current MB. Global Cb horizontal register file and global Cr horizontal register file are used for the chroma (Cb and Cr) components of the MBs.

Instead of using one large external SRAM, we have used 8 internal register files to store the neighboring reconstructed pixels in order to reduce power consumption. The power consumption is reduced by accessing a small register file for storing and reading a reconstructed pixel instead of accessing a large external SRAM. In addition, we have disabled the register files when they are not accessed in order to reduce power consumption.

3.1 Proposed Hardware for 4x4 Luma Prediction Modes

After a careful analysis of the equations used in 4x4 luma prediction modes, it is observed that there are common parts in the equations and some of the equations are identical. The intra prediction equations are organized for exploiting these observations to reduce both the number of memory accesses and computation time required for generating the predicted pixels. The organized prediction equations for Diagonal Down-Left, Diagonal Down-Right, Vertical Right and Vertical Left 4x4 luma prediction modes are shown in Fig. 9. As it can be seen from the figure, (A + B), (B + C), (C + D), (D + E), (E + F), (F + G), (G + H), (J + K), (I + J), (M + I) and (M + A) are common in two or more equations, and some of the prediction equations (e.g. [(A + B) + (B + C) + 2] >> 2) are identical.

The proposed hardware first calculates the results of the common parts in all the 4x4 luma prediction modes and stores them in temporary registers. It, then, calculates the results of the prediction equations using the values stored in these temporary registers. If both the left and top neighboring blocks of a 4x4 luma block are available, 12 common parts are calculated in the preprocessing step and this takes 8 clock cycles. The neighboring buffers are only accessed during this preprocessing. Therefore, they are disabled after the preprocessing for reducing power consumption.

The proposed hardware calculates the results of the identical prediction equations only once and stores them in temporary registers. It, then, determines the results of identical prediction equations by reading the values stored in these temporary registers, instead of calculating the same equations again.

The proposed datapath for generating predicted pixels for a 4x4 luma block using all 4x4 luma prediction modes is shown in Fig. 10. Level0 (L0) registers are used to store the results of the common parts in the equations of all the 4x4 luma prediction modes. Level1 (L1) registers are used to store the results of the identical prediction equations used in all the 4x4 luma prediction modes. If both the left and top neighboring blocks of a 4x4 luma block are available, it takes 165 clock cycles to generate the predicted pixels for that 4x4 block using available 4x4 luma prediction modes.

pred[0, 0] = [(A + B) + (B + C) + 2] >> 2
pred[0, 1] = pred[1, 0] = [(C + D) + (B + C) + 2] >> 2
pred[0, 2] = pred[1, 1] = pred[2, 0] = [(C + D) + (D + E) + 2] >> 2
pred[0, 3] = pred[1, 2] = pred[2, 1] = [(E + F) + (D + E) + 2] >> 2
pred[3, 0] = [(E + F) + (D + E) + 2] >> 2
pred[1, 3] = pred[2, 2] = pred[3, 1] = [(E + F) + (F + G) + 2] >> 2
pred[2, 3] = pred[3, 2] = [(G + H) + (F + G) + 2] >> 2
pred[3, 3] = [(G + H) + (H +H) + 2] >> 2

(a) 4x4 Diagonal Down-Left Prediction Mode

pred[0, 2] = pred[1, 3] = [(A + B) + (B + C) + 2] >> 2
pred[0, 3] = [(C + D) + (B + C) + 2] >> 2
pred[3, 0] = [(J + K) + (K+ L) + 2] >> 2
pred[2, 0] = pred[3, 1] = [(J + K) + (I + J) + 2] >> 2
pred[1, 0] = pred[2, 1] = pred[3, 2] = [(M + I) + (I + J) + 2] >> 2
pred[0, 0] = pred[1, 1] = pred[2, 2] = pred[3, 3] = [(M + I) + (M + A) + 2] >> 2
pred[0, 1] = pred[1, 2] = pred[2, 3] = [(A + B) + (M + A) + 2] >> 2

(b) 4x4 Diagonal Down-Right Prediction Mode

pred[3, 0] = [(I + J) + (J + K) + 2] >> 2
pred[2, 0] = [(I + J) + (M + I) + 2] >> 2
pred[1, 0] = pred[3, 1] = [(M + A) + (M + I) + 2] >> 2
pred[1, 1] = pred[3, 2] = [(M + A) + (A + B) + 2] >> 2
pred[1, 2] = pred[3, 3] = [(B + C) + (A + B) + 2] >> 2
pred[1, 3] = [(B + C) + (C + D) + 2] >> 2
pred[0, 1] = pred[2, 1] = [(A + B) + 1] >> 1
pred[0, 3] = [(C + D) + 1] >> 1
pred[0, 0] = pred[2, 1] = [(M + A) + 1] >> 1
pred[0, 2] = pred[2, 3] = [(B + C) + 1] >> 1

(c) 4x4 Vertical Right Prediction Mode

pred[1, 0] = [(A + B) + (B + C) + 2] >> 2
pred[1, 1] = pred[3, 0] = [(C + D) + (B + C) + 2] >> 2
pred[1, 2] = pred[3, 1] = [(C + D) + (D + E) + 2] >> 2
pred[1, 3] = pred[3, 2] = [(E + F) + (D + E) + 2] >> 2
pred[3, 3] = [(E + F) + (F + G) + 2] >> 2
pred[0, 0] = [(A + B) + 1] >> 1
pred[0, 1] = pred[2, 0] = [(B + C) + 1] >> 1
pred[0, 2] = pred[2, 1] = [(C + D) + 1] >> 1
pred[0, 3] = pred[2, 2] = [(D + E) + 1] >> 1
pred[2, 3] = [(E + F) + 1] >> 1

(d) 4x4 Vertical Left Prediction Mode

Figure 9. Organized Prediction Equations for 4x4 Luma Prediction Modes

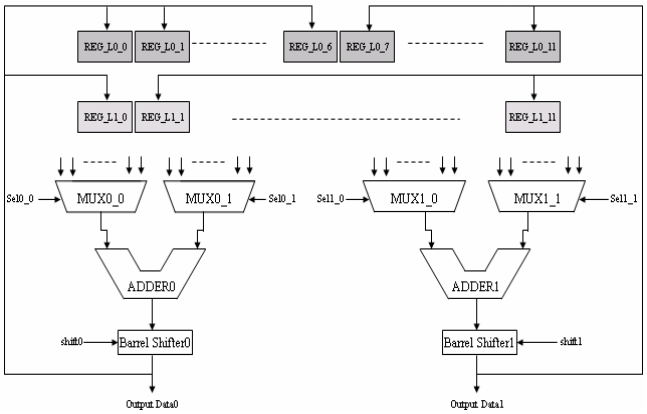

Figure 10. Datapath for 4x4 Luma Prediction Modes

Since the order of the equations used in a 4x4 luma prediction mode is not important for functional correctness, the equations are ordered to keep the inputs of the adders the same for as many consecutive clock cycles as possible. This avoids unnecessary switching activity and reduces the power consumption.

3.2 Proposed Hardware for 16x16 Luma Prediction Modes

After a careful analysis of the equations used in 16x16 luma prediction modes, it is observed that Vertical, Horizontal and DC mode equations can directly be implemented using adders and shifters, however the equations used in Plane mode can be organized to avoid using a multiplier and to reduce computation time required for generating the predicted pixels. The organized Plane mode prediction equations for block 0 in a MB are shown in Fig. 11. A similar organization of the Plane mode prediction equations is given in [4, 5]. However, our hardware design is different than their design and it is a more cost-effective solution for portable applications.

The proposed hardware first calculates the common parts C0, (C0 + b), (C0 + 2b), and (C0 + 3b) and stores them in temporary registers. It, then, generates the predicted pixels in the first row by using the values stored in these temporary registers. The proposed hardware, then, adds c to the values stored in the temporary registers and stores the resulting values in the same temporary registers. It, then, generates the predicted pixels in the second row by using the values stored in these temporary registers. The proposed hardware repeats this process until all the predicted pixels for the current MB are generated.

The proposed datapath for generating predicted pixels for a 16x16 luma block using all 16x16 luma prediction modes is shown in Fig. 12. REG0 - REG7 registers are used to store the results of the common parts in the equations. The neighboring reconstructed pixels stored in the neighboring buffers are given as inputs to the datapath. If both the left and top neighboring MBs of a 16x16 luma block are available, it takes 1127 clock cycles to generate the predicted

$$a = (p[-1,15] + p[15,-1]) << 4$$
$$b = [(H << 2) + (H + 32)] >> 6$$
$$c = [(V << 2) + (V + 32)] >> 6$$

$$C0 = [a - (7 * b) - (7 * c) + 16]$$

$$pred[0, 0] = Clip1\ [(C0) >> 5]$$
$$pred[0, 1] = Clip1\ [(C0 + b) >> 5]$$
$$pred[0, 2] = Clip1\ [(C0 + 2b) >> 5]$$
$$pred[0, 3] = Clip1\ [(C0 + 3b) >> 5]$$

$$pred[1, 0] = Clip1\ [(C0 + c) >> 5]$$
$$pred[1, 1] = Clip1\ [((C0 + c) + b) >> 5]$$
$$pred[1, 2] = Clip1\ [((C0 + c) + 2b) >> 5]$$
$$pred[1, 3] = Clip1\ [((C0 + c) + 3b) >> 5]$$

$$pred[2, 0] = Clip1\ [(C0 + 2c) >> 5]$$
$$pred[2, 1] = Clip1\ [((C0 + 2c) + b) >> 5]$$
$$pred[2, 2] = Clip1\ [((C0 + 2c) + 2b) >> 5]$$
$$pred[2, 3] = Clip1\ [((C0 + 2c) + 3b) >> 5]$$

$$pred[3, 0] = Clip1\ [(C0 + 3c) >> 5]$$
$$pred[3, 1] = Clip1\ [((C0 + 3c) + b) >> 5]$$
$$pred[3, 2] = Clip1\ [((C0 + 3c) + 2b) >> 5]$$
$$pred[3, 3] = Clip1\ [((C0 + 3c) + 3b) >> 5]$$

Figure 11. Organized Prediction Equations for 16x16 Luma Plane Mode

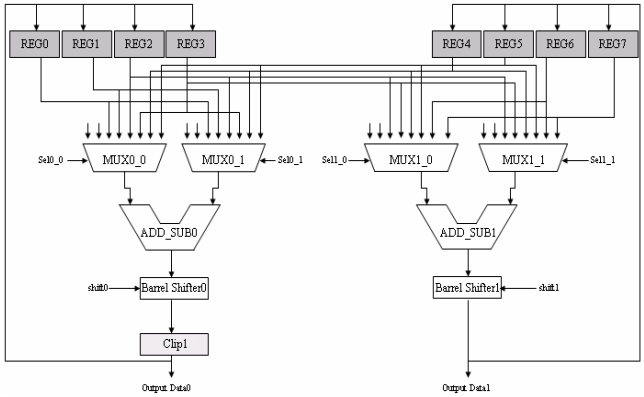

Figure 12. Datapath for 16x16 Luma Prediction Modes

pixels for that 16x16 luma block using available 16x16 luma prediction modes.

Plane mode is the most computationally demanding 16x16 luma prediction mode. Therefore, using two parallel adders and shifters in the proposed datapath is especially important for Plane mode. The predicted pixels for a 16x16 luma block are generated in 340 clock cycles using Plane mode.

3.3 Proposed Hardware for 8x8 Chroma Prediction Modes

Since the 8x8 chroma prediction modes are similar to 16x16 luma prediction modes, the proposed hardware for 8x8 chroma prediction modes is also similar to the proposed hardware for 16x16 luma prediction modes. If both the left and top neighboring MBs of an 8x8 chroma block are available, it takes 302 clock cycles to generate the predicted

pixels for that 8x8 chroma block using available 8x8 chroma prediction modes. Plane mode is also the most computationally demanding 8x8 chroma prediction mode. The predicted pixels for an 8x8 chroma block are generated in 95 clock cycles using Plane mode.

4. Implementation Results

The proposed architecture is implemented in Verilog HDL. The implementation is verified with RTL simulations using Mentor Graphics ModelSim SE. The Verilog RTL is then synthesized to a 2V8000ff1152 Xilinx Virtex II FPGA with speed grade 5 using Mentor Graphics Leonardo Spectrum. The resulting netlist is placed and routed to the same FPGA at 90 MHz under worst-case PVT conditions using Xilinx ISE Series 7.1i.

The Verilog RTL for proposed architecture is integrated into an H.264 intra frame coder hardware [6]. The H.264 intra frame coder hardware is integrated into the 2V8000ff1152 Xilinx Virtex II FPGA on the logic tile of the ARM Versatile / PB926EJ-S development board shown in Fig. 13 as a master of the AHB S bus and it is verified to work correctly on this development board.

As shown in Table 3, the number of clock cycles it takes to perform intra prediction for a MB using 4x4 luma prediction modes depends on the availability of the neighboring MBs. In a VGA frame (40x30 = 1200 MBs), there is only 1 MB (MB0) which has no available neighboring MBs, there are 39 MBs (the first row of MBs except MB0) which have only left neighboring MBs available, there are 29 MBs (the first column of MBs except MB0) which have only top neighboring MBs available, and there are 1131 MBs (remaining MBs) which have both left and top neighboring MBs available.

In addition to the number clock cycles given in Table 3, 16 clock cycles are required for loading the neighboring reconstructed pixels to the corresponding neighboring buffers for each 4x4 luma block. Therefore, generating the predicted pixels for a VGA frame using 4x4 luma prediction modes takes 1910 + (1980 x 39) + (1797 x 29) + (2640 x 1131) + (16x16x1200) = 3424283 clock cycles.

Since, in the proposed hardware, there are two parts operating in parallel and the lower part which is used for generating the predicted pixels for a MB using 4x4 luma prediction modes is the bottleneck, the FPGA implementation can process a VGA frame in 37.6 msec (3424283 clock cycles per VGA frame x 11 ns clock cycle = 37.6 msec). Therefore, it can process 1000/37.6 = 27 VGA frames per second.

The FPGA implementation including input, output and internal RAMs and register files uses the following FPGA resources; 2002 Function Generators, 1001 CLB Slices, and 518 DFFs, i.e. %2.15 of Function Generators, %2.15 of CLB Slices, and %0.54 of DFFs.

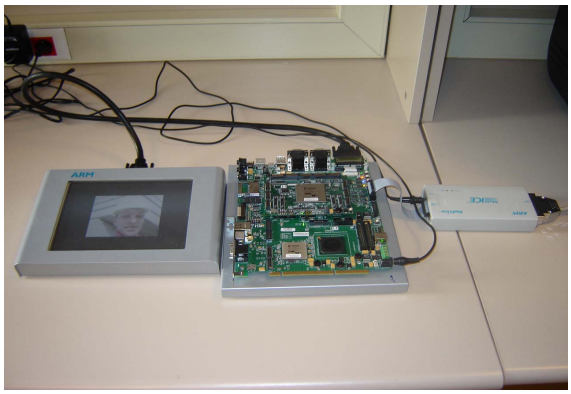

Figure 13. Arm Versatile / PB926EJ-S Development Board

Table 3. Number of Clock Cycles for Performing Intra Prediction for a MB Using 4x4 Luma Prediction Modes

Availability of Neighboring MBs	Clock Cycles/MB
None available	1910
Left available, Top not available	1980
Top available, Left not available	1797
Both available	2640

5. Conclusion

In this paper, we presented an efficient hardware architecture for real-time implementation of intra prediction algorithm used in H.264 / MPEG4 Part 10 video coding standard. The hardware design is based on a novel organization of the intra prediction equations. The proposed architecture is implemented in Verilog HDL. The Verilog RTL code is verified to work at 90 MHz in a Xilinx Virtex II FPGA. The FPGA implementation can process 27 VGA frames per second.

6. References

[1] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC Video Coding Standard", IEEE Trans. on Circuits and Systems for Video Technology, vol. 13, no. 7, pp. 560–576, July 2003.

[2] I. G. Richardson, H.264 and MPEG-4 Video Compression, Wiley, 2003.

[3] Joint Video Team (JVT) of ITU-T VCEG and ISO/IEC MPEG, Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification, ITU-T Rec. H.264 and ISO/IEC 14496-10 AVC, May 2003.

[4] Y. Huang, B. Hsieh, T. Chen, and L. Chen, "Hardware Architecture Design for H.264/AVC Intra Frame Coder", Proc. of IEEE ISCAS, pp. 269-272, April 2004.

[5] Y. Huang, B. Hsieh, T. Chen, and L. Chen, "Analysis, Fast Algorithm, and VLSI Architecture Design for H.264/AVC Intra Frame Coder", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 15, No. 3, March 2005.

[6] Esra Sahin, "An Efficient H.264 Intra Frame Coder Hardware Design", MS Thesis, Sabanci University, August 2006.