A HIGH PERFORMANCE AND LOW COST HARDWARE ARCHITECTURE FOR H.264 TRANSFORM AND QUANTIZATION ALGORITHMS

Ozgur Tasdizen and Ilker Hamzaoglu

Faculty of Engineering and Natural Sciences, Sabanci University
34956, Orhanli, Tuzla, Istanbul, TURKEY
phone: + (90) 216 483-9577, fax: + (90) 216 483-9550, email: hamzaoglu@sabanciuniv.edu
web: www.sabanciuniv.edu/~hamzaoglu

ABSTRACT

In this paper, we present a high performance and low cost hardware architecture for real-time implementation of forward transform and quantization and inverse transform and quantization algorithms used in H.264 / MPEG4 Part 10 video coding standard. The hardware architecture is based on a reconfigurable datapath with only one multiplier. This hardware is designed to be used as part of a complete low power 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 81 MHz in a Xilinx Virtex II FPGA and it is verified to work at 210 MHz in a 0.18µ ASIC implementation. The FPGA and ASIC implementations can code 27 and 70 VGA frames (640x480) per second respectively.

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 International 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 Figure 1, two of these tools are the transform and quantization algorithms [1, 2, 3].

Even though most of the previous video coding standards, e.g. MPEG-1, H.261, MPEG-2, H.263 and MPEG-4, use the 8x8 Discrete Cosine Transform (DCT) to transform the residual data, H.264 uses a 4x4 integer transform for transforming residual data. The integer transform achieves very similar results to 8x8 DCT without any floating point operations. In addition, all the multiplication operations in the forward and inverse transform algorithms can be implemented in hardware with low cost binary shifters. Since the inverse transform in H.264 is defined by exact integer operations, inverse transform mismatches are avoided. Since a scaling factor is used in the quantization algorithm, a multiplier is needed for its implementation [3, 4, 5].

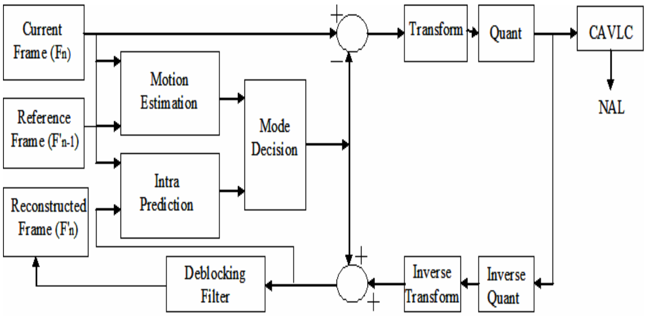

Figure 1 H.264 Encoder Block Diagram

In this paper, we present a high performance and low cost hardware architecture for real-time implementation of H.264 forward transform and quantization and inverse transform and quantization algorithms. The hardware architecture is based on a reconfigurable datapath with only one multiplier. This hardware is designed to be used as part of a complete low power 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 81 MHz in a Xilinx Virtex II FPGA and it is verified to work at 210 MHz in a 0.18µ ASIC implementation. The FPGA and ASIC implementations can code 27 and 70 VGA frames (640x480) per second respectively.

A hardware architecture only for real-time implementation of H.264 forward and inverse transform algorithms is presented in [6]. 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 16 adders and 16 internal register files in their datapath as opposed to 3 adders and 6 internal register files in the transform part of our datapath. Their datapath has an area of 6538 gates in TSMC 0.35µ technology. Our datapath, on the other hand, has an area of 2904 gates in AMS 0.35µ technology.

The rest of the paper is organized as follows. Section II presents a brief overview of transform and quantization algorithms used in H.264. Section III describes the proposed hardware architecture in detail. The implementation results are given in Section IV. Finally, Section V presents the conclusions.

2. OVERVIEW OF H.264 TRANSFORM AND QUANTIZATION ALGORITHMS

The basic transform coding process in H.264, shown in Figure 1, is similar to that of previous standards. The process includes a forward transform and quantization followed by zig-zag ordering and

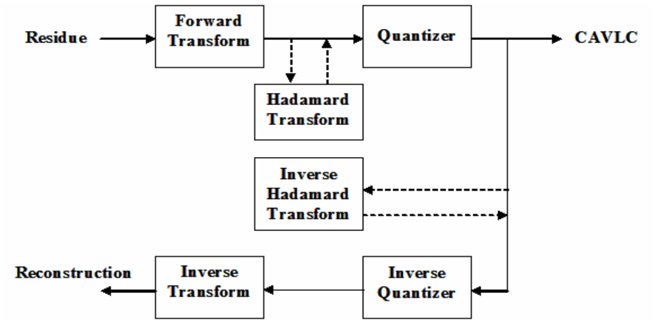

Figure 2 Block Diagram of Transform and Quant Algorithms

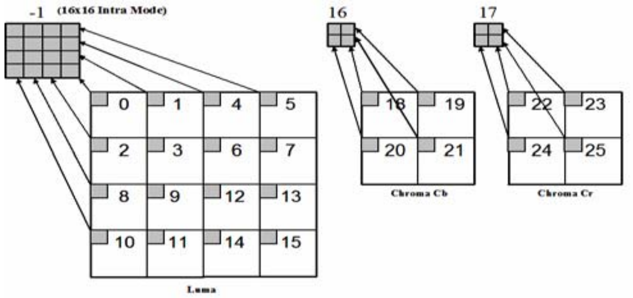

Figure 3 Processing Order of Blocks in a Macroblock

entropy coding. The transform coded residual data is also reconstructed. The reconstruction process includes an inverse quantization and inverse transform followed by motion compensation. The reconstructed data before deblocking filter is used for intra prediction in current frame, and the reconstructed data after deblocking filter is used for motion estimation in future frames.

A more detailed flow of the transform and quantization algorithms is presented in Figure 2. The input to the forward transform algorithm is a 4x4 block of residual data obtained by subtracting the prediction from the original image data. The transform and quantization algorithms process the blocks in a macroblock as explained in the following sections, and send the resulting data to entropy coding and reconstruction process in the order shown in Figure 3.

2.1 Transform Algorithm Overview

H.264 transform algorithm uses four different transform matrices shown in Figure 4; 4x4 forward integer, 4x4 hadamard, 2x2 hadamard, and 4x4 inverse integer [3, 4, 5]. Since 4x4 and 2x2 hadamard transform matrices are symmetric, inverse hadamard transform matrices are same as forward hadamard transform matrices.

In the transform coding process, 4x4 integer transform is applied to all the blocks independent of their prediction type and mode. As shown in Figure 3, 4x4 block -1 is formed by the transformed DC coefficients of 4x4 luminance blocks for the macroblocks that are coded in 16x16 Intra mode, and 2x2 blocks 16 and 17 are formed by the transformed DC coefficients of 4x4 chrominance blocks for all the macroblocks. After the 4x4 integer transform, 4x4 hadamard transform is applied to block -1 and 2x2 hadamard transform is applied to blocks 16 and 17.

In the reconstruction process, 4x4 inverse hadamard transform is applied to block -1, and 2x2 inverse hadamard transform is applied to blocks 16 and 17. After the inverse hadamard transforms, 4x4 inverse integer transform is applied to all the blocks independent of their prediction type and mode.

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} x0 & x1 & x2 & x3 \\ x4 & x5 & x6 & x7 \\ x8 & x9 & 10 & x11 \\ x12 & x13 & x14 & x15 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix}$$

(a)

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} z0 & z1 & z2 & z3 \\ z4 & z5 & z6 & z7 \\ z8 & z9 & z10 & z11 \\ z12 & z13 & z14 & z15 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}$$

(b)

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} z0 & z1 \\ z2 & z3 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

(c)

$$\begin{bmatrix} 1 & 1 & 1 & 1/2 \\ 1 & 1/2 & -1 & -1 \\ 1 & -1/2 & -1 & 1 \\ 1 & -1 & 1 & -1/2 \end{bmatrix} \begin{bmatrix} y0 & y1 & y2 & y3 \\ y4 & y5 & y6 & y7 \\ y8 & y9 & y10 & y11 \\ y12 & y13 & y14 & y15 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1/2 & -1/2 & -1 \\ 1 & -1 & -1 & 1 \\ 1/2 & -1 & 1 & -1/2 \end{bmatrix}$$

(d)

Figure 4 Matrices used in H.264 Transform Algorithm
a) 4x4 Forward Integer Transform, b) 4x4 Hadamard Transform, c) 2x2 Hadamard Transform, d) 4x4 Inverse Integer Transform

2.2 Quantization Algorithm Overview

A quantization parameter (QP), calculated by the rate control algorithm, is used for determining the quantization step size of transform coefficients in H.264 [3, 4, 5]. There are 52 quantization parameter values. These values are arranged so that an increase of 1 in quantization parameter means an increase of quantization step size by approximately 12%. An increase of quantization step size by approximately 12% means roughly a reduction of bit rate by approximately 12%.

Quantization of AC coefficients is done by using the following equation: $|Zij| = (|Wij|.MF + f) >> qbits$, $sign(Zij) = sign(Wij)$

Wij is the result of forward transformation. MF is a scaling factor. f is a parameter used to avoid rounding errors and it depends on prediction type of the block and QP. qbits is a variable depending on QP.

Inverse quantization of AC coefficients is done by using the following equation: $W'ij = Zij.Vij.2^{floor(QP/6)}$

Zij is the result of forward quantization. Vij are rescaling factors. Quantization of DC coefficients is done similarly.

3. PROPOSED HARDWARE ARCHITECTURE

The proposed hardware architecture includes an input register file, a reconfigurable datapath and its control unit, internal register files and an output register file. The reconfigurable datapath and the register files are shown in Figure 5. The reconfigurable datapath is designed for implementing both forward and inverse transform and quant algorithms. Even though only one multiplier is used in the reconfigurable datapath, the proposed hardware performs forward transform, hadamard transform, quant, inverse hadamard transform, inverse quant and inverse transform operations for a macroblock, in the worst case, in 2500 clock cycles. The worst-case occurs for the macroblocks that are coded in 16x16 Intra mode. Therefore, the proposed high performance and low cost hardware can process 30 VGA frames per second at 90 MHz.

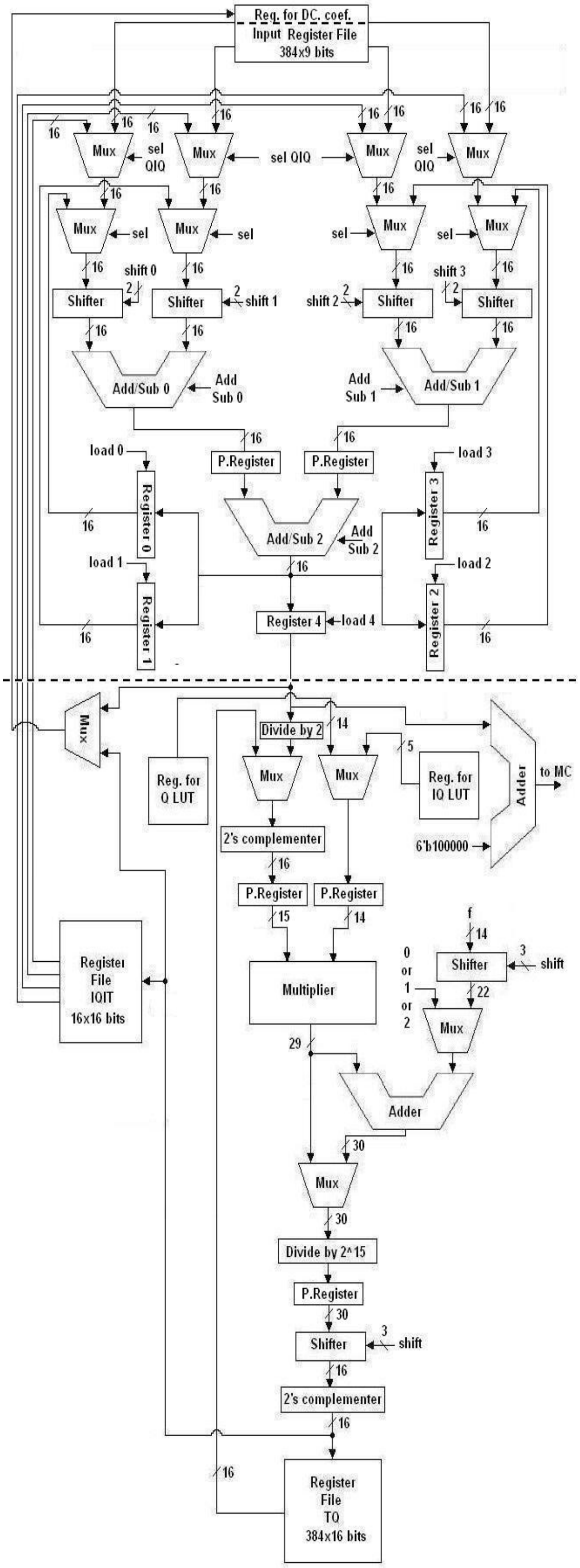

Figure 5 Proposed Reconfigurable Datapath

384x9 bit input register file stores residual data for a macroblock that will be transform coded including both luminance and chrominance blocks.

The part of the datapath above the dashed line performs transform and inverse transform operations. The registers, adders and shifters in this part of the datapath are shared by forward and inverse transform operations. When the hardware is used to perform forward transform, the control unit configures the datapath to perform the forward transform operations. When it is used to perform inverse transform, the control unit configures the datapath to perform the inverse transform operations.

The first row of multiplexers is used for selecting the proper inputs for transform operations. They select the data from input register file for forward transform operations and the data from IQIT register file for inverse transform operations. The second row of multiplexers is used for selecting the proper input data for the first and the second matrix multiplications. They select the data from the first row of multiplexers for the first matrix multiplication operations and the data from register 0, register 1, register 2, register 3 for the second matrix multiplication operations.

Shifters are one bit shifters used for shifting left (multiply by 2) for forward transform operations and for shifting right (divide by 2) for inverse transform operations.

Three adder/subtractors are used in the datapath to achieve high performance with low hardware cost. The first column of the result matrix for the matrix multiplication operations shown in Figure 4 (a) can be calculated using the following four equations:

$$[(x0+x4+x8+x12) + (x1+x5+x9+x13) + (x2+x6+x10+x14) + (x3+x7+x11+x15)]$$
$$[2*(x0+x4+x8+x12) + (x1+x5+x9+x13) - (x2+x6+x10+x14) - 2*(x3+x7+x11+x15)]$$
$$[(x0+x4+x8+x12) - (x1+x5+x9+x13) - (x2+x6+x10+x14) + (x3+x7+x11+x15)]$$
$$[(x0+x4+x8+x12) - 2*(x1+x5+x9+x13) + 2*(x2+x6+x10+x14) - (x3+x7+x11+x15)]$$

The four values (x0+x4+x8+x12), (x1+x5+x9+x13), (x2+x6+x10+x14) and (x3+x7+x11+x15) are the results of first matrix multiplication and they are used for calculating the first column of the result matrix containing the transform coefficients. Similarly, the equations for calculating the transform coefficients in each remaining column of the result matrix have four common values that are used to calculate the corresponding transform coefficient. Therefore, 16-bit registers register 0, register 1, register 2, and register 3 are used to store these four common values, i.e. the results of first matrix multiplications. This reduces both the number of cycles and the power consumption of both forward and inverse transform operations. The same method is used to implement the other matrix multiplication operations shown in Figure 4.

Since the order of some of the equations used to perform the matrix multiplications for 4x4 and 2x2 hadamard transforms are not important for functional correctness, we have used the order that gives the lowest power consumption. For example, the first column of the result matrix for the matrix multiplication operations for 4x4 hadamard transform shown in Figure 4 (b) can be calculated using the following four equations in the given order:

$$[(z0+z4+z8+z12) + (z1+z5+z9+z13) + (z2+z6+z10+z14) + (z3+z7+z11+z15)]$$
$$[(z0+z4+z8+z12) + (z1+z5+z9+z13) – ((z2+z6+z10+z14) + (z3+z7+z11+z15))]$$
$$[(z0+z4+z8+z12) - (z1+z5+z9+z13) – ((z2+z6+z10+z14) - (z3+z7+z11+z15))]$$
$$[(z0+z4+z8+z12) - (z1+z5+z9+z13) + (z2+z6+z10+z14) - (z3+z7+z11+z15)]$$

When the equations are calculated in the given order, both the operations (addition or subtraction) performed by adder/subtractor 0 and adder/subtractor 1 and their inputs stay the same in first and second cycles and in third and fourth cycles. Since their inputs and the operations they perform stay the same for two consecutive clock cycles, their outputs stay the same as well. This avoids un-

necessary switching activity resulting in lower power consumption for both forward and inverse hadamard transforms.
P. Registers are pipelining registers used to achieve 81 MHz clock frequency in a 2V8000ff1157 Xilinx Virtex II FPGA with speed grade 5. Register 4 stores the results of forward or inverse transform operations.

The part of the datapath below the dashed line performs forward and inverse quantization operations. The registers, adders, shifters and the multiplier in this part of the datapath are shared by forward and inverse quant operations. When the hardware is used to perform forward quantization, the control unit configures the datapath to perform the forward quant operations. When it is used to perform inverse quantization, the control unit configures the datapath to perform the inverse quant operations.
Register 4 contains the input data for the quantization and inverse quantization operations. P. Registers are pipelining registers used to achieve 81 MHz clock frequency in a 2V8000ff1157 Xilinx Virtex II FPGA with speed grade 5.
The multiplier used in the datapath is a 15x14 unsigned multiplier. Two multiplexers are used for selecting the proper inputs for the multiplier. One of the multiplexers is used to select either a transformed or inverse transformed value coming from register 4 or a quantized value coming from the output register file TQ. The other multiplexer is used to select either a value from quant lookup table or a value from inverse quant lookup table.
The adder at the output of the multiplier and the shifter at one of the inputs of the adder are used to avoid rounding errors that can happen during scaling and rescaling operations.
The 3-bit shifter at the output of the multiplier is used to perform scaling and rescaling operations depending on the value of qbits parameter. The result of the shift operation is converted into two's complement form and stored in the output register file TQ.

The transform and quant operations are executed in a pipelined manner. After a transform coefficient is computed, in the next cycle, this coefficient is quantized in the quant part of the datapath and a new transform coefficient is computed in the transform part of the datapath. Since only one multiplier is used in the datapath, quant and inverse quant operations cannot be pipelined. After all the transform coefficients in a block are quantized, inverse quantization starts followed by inverse transform.

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 2V8000ff1157 Xilinx Virtex II FPGA with speed grade 5 using Mentor Graphics Leonardo Spectrum [7]. The resulting netlist is placed and routed to the same FPGA using Xilinx ISE Series 5.2i.
The FPGA implementation including input and output register files as well is placed and routed at 81 MHz under worst-case PVT conditions. Since, in the worst-case, it takes 2500 clock cycles to process a MB, the FPGA implementation can code 27 VGA frames (640x480) per second. The FPGA implementation is verified to work in a Xilinx Virtex II FPGA on an Arm Versatile Platform development board.
The FPGA implementation including input and output register files as well used the following FPGA resources; 4054 Function Generators, 2027 CLB Slices, 1 Block Multiplier, and 583 Dffs /Latches, i.e. 4.35% of Function Generators, 4.35% of CLB Slices, 0.60% of Block Multipliers, and 0.61% of Dffs /Latches. The FPGA implementation excluding input and output register files used the following FPGA resources; 2497 Function Generators, 1249 CLB Slices, 1 Block Multiplier, and 581 Dffs /Latches, i.e. 2.68% of Function Generators, 2.68% of CLB Slices, 0.60% of Block Multipliers, and 0.61% of Dffs /Latches.
The Verilog RTL is also synthesized to Virtual Silicon UMC 0.18µ standard-cell library using Synopsys Design Compiler. The synthesis results are presented in Table 1. The netlist excluding input and output register files has an area of 23K gates. The netlist is verified to work at 210 MHz under worst-case PVT conditions with post synthesis simulations. This 0.18µ ASIC implementation can code 70 VGA frames (640x480) per second.

Table 1

	Critical Path Delay [ns]	Area [Gate Count]
Transform part of the Datapath	2.77	1978
Datapath	4.78	12773
Datapath + Control Unit	4.8	23162
Datapath + Control Unit + Input Register File + Output Register File TQ	4.8	130505

5. CONCLUSIONS

In this paper, we presented a high performance and low cost hardware architecture for real-time implementation of H.264 forward transform and quantization and inverse transform and quantization algorithms. The hardware architecture is based on a reconfigurable datapath with only one multiplier. This hardware is designed to be used as part of a complete low power 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 81 MHz in a Xilinx Virtex II FPGA and it is verified to work at 210 MHz in a 0.18µ ASIC implementation. The FPGA and ASIC implementations can code 27 and 70 VGA frames (640x480) per second respectively.

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. 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] Joint Video Team (JVT) of ITU-T VCEG and ISO/IEC MPEG, *Joint Model (JM) Reference Software Version 9.2*, http://bs.hhi.de/suehring/

[5] H. Malvar, A. Hallapuro, M. Karczewicz. and L. Kerofsky, "*Low-Complexity Transform and Quantization in H.264 / AVC*", IEEE Trans. on Circuits and Systems for Video Technology, vol. 13, no. 7, pp. 598–603, July 2003.

[6] T. C. Wang, Y. W. Huang, H. C. Fang, and L. G. Chen, "*Parallel 4x4 2D Transform and Inverse Transform Architecture for MPEG-4 AVC / H.264*", Proc. of IEEE ISCAS, 2003

[7] Xilinx Inc., *Virtex-II™ Platform FPGAs: Complete Data Sheet DS031*, http://www.xilinx.com, March 2004