ECE 511
Digital ASIC Design

Project: JPEG2000 Hardware Compression

December 18, 2004

Team Members:
Rami Zewail
Nan Ying
Philip Marshall
Sean Kozicki

Declaration of Content

The design elements of this project and report are entirely the original work of the authors and have not been submitted for credit in any other course except as follows:

MicroBlaze processor VHDL [8]
LMB Memory Controller VHDL [9]
OPB Extended Memory Controller VHDL [10]
UART Lite VHDL [11]
OPB Ethernet VHDL [12]
OPB Bus with Arbiter VHDL [13]
Microprocessor Debug Module (MDM) [14]
Block RAM (BRAM) VHDL [15]
Local Memory Bus VHDL [16]
LibXil Net [18]
LibXil Memory File System [18]
Non-OPB Bus ZBT RAM controller VHDL[25]
DCM [19]
MiniUart VHDL [20]
Xess Board SDRAM and VGA Drivers [29]

Rami Zewail	Date	Nan Ying	Date
Philip Marshall	Date	Sean Kozicki	Date

Abstract

The new still compression image standard, JPEG2000, has emerged with a number of significant features that would allow it to be used efficiently over a wide variety of images. The scalability of the new standard is intended to allow trading off between compression rates and quality of images. Due to multi-resolution nature of wavelet transforms, they have been adopted by the JPEG2000 standard as the transform of choice.

In project, we present an implementation for a reconfigurable fully scalable Integer Wavelet Transform (IWT) unit that satisfies the specifications of the JPEG2000 standard. The implementation is based on the lifting scheme, which is the most computation efficient implementation of the discrete wavelet transform. Integer wavelet transforms, also known as reversible DWT, have recently received special interest due to a number of significant properties in terms of computational efficiency and storage requirements.

The Lifting scheme-based Integer Wavelet Transform (IWT) unit was implemented in Field Programmable Gate Arrays (FPGAs), namely the Xilinx Vertix II family. The design is scalable in order to allow a trade off between rate and distortion. A maximum clock frequency of 112 MHz, with throughput of over 400 pixels/s, was achieved by employing techniques such as parallel operation of independent units and pipelining. A speed up of over 61 times has been achieved versus a MatLab compiled C code implementation on a Pentium IV 1.8 GHZ processor.

Table of Contents

1.0 Description of Operation

1.1 Introduction

As the project progressed, the group encountered various problems that required a re-evaluation of the system design. At each stage of re-evaluation the project scope was reduced in hopes to form an operational system. The major problems encountered with the project were related to the system peripherals. A significant problem related to the use of peripherals is in the difficulty with simulation. In most cases, the peripherals can not be simulated reasonably and must therefore be debugged on the target. The following is a discussion of the various peripheral problems that were encountered, and the subsequent design decisions made. Final comments with hindsight are included in the conclusion.

Stage 1

As expected the initial stage was the most ambitious and encompassed the intended functionality of the system – the networked JPEG2000 file server. At this point in the design, the developers were not familiar with the tools; specifically the Xilinx Embedded Development Kit (EDK), the Xilinx Multimedia board hardware, or the exact details of the IP modules that were intended to be used.

The core of the system was to be a microprocessor. Initially, PIC and ARM replica cores available from OpenSource.org[20] were identified and explored. In the end, due to the tools available and upon recommendations from the professor the Xilinx MicroBlaze was selected. The MicroBlaze microprocessor is 32-bit pipelined RISC processor that is included with the Xilinx Embedded Development Kit (EDK). The MicroBlaze is organized such that the instruction accesses are separate from data accesses. Further to that, both the instruction and data interfaces are further separated into the Local Memory Bus (LMB) and the On-Chip Peripheral Bus (OPB). The LMB is typically used to access the on-chip Block RAM (BRAM), while the OPB is typically used for access to peripherals such as serial ports, and external memory controllers.

To reduce the development time and focus on the hardware implementation of the fast wavelet transformation algorithm the uCLinux [22] operating system was also initially selected. The uCLinux operating system is a scaled down port of Linux for embedded systems that do not have a memory management unit. The uCLinux OS uses the common Linux API and supports TCP/IP networking and file systems.

The primary interface elements included external peripherals such as a UART, Ethernet, ZBT SRAM, and compact Flash. The RS232 port would allow for interfacing to uCLinux for sending Unix-like commands to the operating system. Ethernet would provide the system an external interface to a Local Area Network (LAN). Since the uCLinux operating system image is fairly large (uCLinux kernel + tools < 900 kb) continually re-loading the uCLinux image would be time consuming. Once a stable uCLinux image is built there would be no need to modify it. The compact flash would then used to store this uCLinux image. The ZBT SRAM was partially

to be used for file storage as provided by the operating system, and for the DWT implementation. A block diagram of the proposed system is present in Figure 1 below.

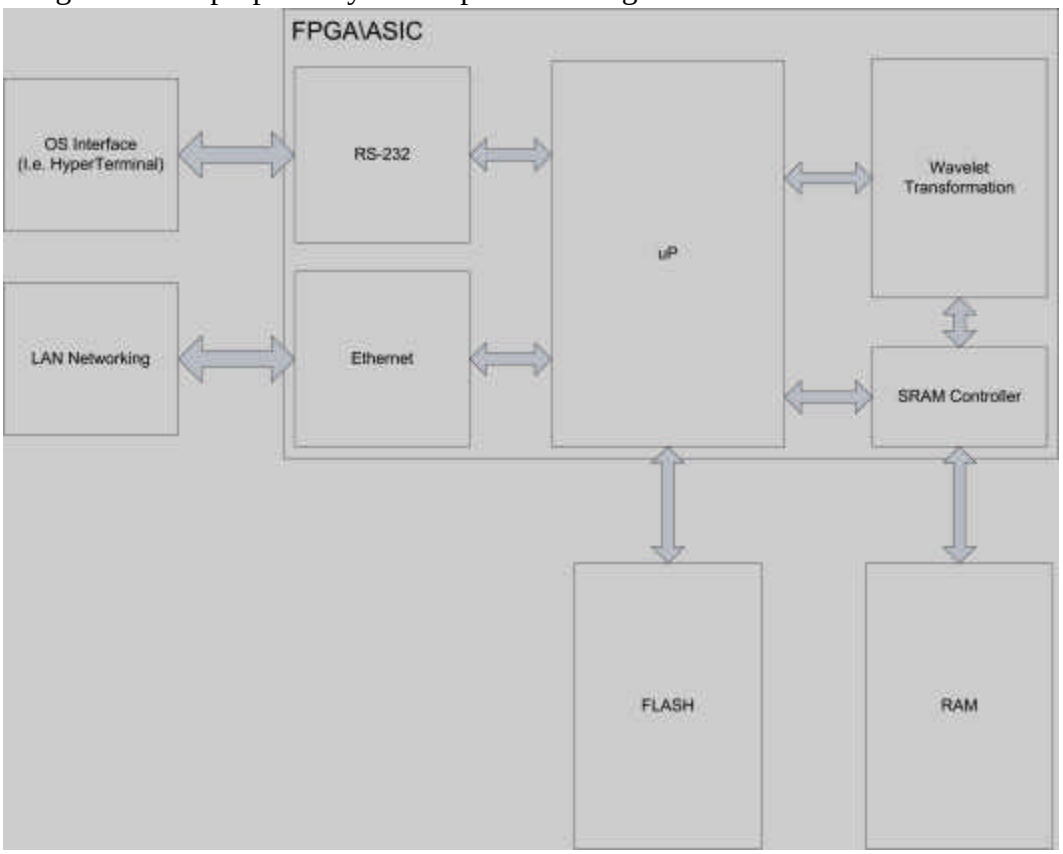


Figure 1. Stage1 System Architecture

The primary goals of this stage included data collection and tool acquaintance. To become more familiar with the EDK, sample MicroBlaze systems that had overlapping functionality with the intended design were built and tested. Data collection consisted of joining the various uCLinux and uCLinux-MicroBlaze distributions and forums, and reading all relevant documentation and FAQs.

Successes

Through the extensive debugging of IP cores that did not work, the group became quite familiar with the Multimedia board and Xilinx Platform Studio (XPS). The group also successfully downloaded and built both the uCLinux-MicroBlaze mbvanilla hardware platform and the uCLinux kernel[1].and had success with the deprecated OPB ZBT controller provided with a sample system provided by Xilinx.

Issues

There were three main issues that caused the group to re-evaluate the design goals and subsequently the system architecture; 1) problems with Ethernet, 2) no current port of uCLinux to the Xilinx Multimedia board, 3) the complexity of the reference JPEG2000 source code.

[1] It took over 30 minutes for the synthesis and routing of the mbvanilla hardware platform.

To test the Ethernet Interface a sample Web Server MicroBlaze system was downloaded from Xilinx. The Web Server application supports Web GET for files and allowed the user to change the state of LEDs on the Multimedia Board through a text box. The first problem with the Web Server application was that it was originally built for EDK version 3.2 and not the EDK version 6.2 that was available. This required that the application be modified so that it was compatible with EDK 6.2. After the modifications were made the Ethernet interface still did not work. Installing EDK 3.2 also proved problematic, as the Web server example still contained errors. This attempt was quickly abandoned.

The Xilinx Multimedia board has an Ethernet chip that provides the physical layer, while the Xilinx Ethernet Media Access Controller (EMAC) provides the Media Access Layer. Along with the evaluation Ethernet controller, Xilinx also provides an internet protocol stack that includes TCP, UDP, and IP. The source code for the Ethernet Driver was debugged to try to resolve the interfacing problems with no avail. Further searching on the internet revealed that other developers have had problems getting this interface to work, and had made custom modifications to the driver in order to make it functional.

At the time of this project, uCLinux-MicroBlaze had not been ported to the Xilinx Multimedia board. The uCLinux-MicroBlaze message groups indicated that a few developers were working on this port, but had not completed this task. A FAQ indicated that it should be possible to port to new hardware within 1-day if the developer knew what they were doing. Given the fact the messages relating to porting to the Multimedia board transpired over several months, this did not appear to be a feasible solution unless a working solution was provided by someone else for immediate consumption.

For benchmarking purposes, reference JPEG2000 C code[X] was to be ported to the uCLinux operating system. The wavelet transformation time between the software and hardware implementations would then be compared for determine the effectiveness of the solution. The JPEG2000 C code would also serve to perform the initial quantization and compression of the image onboard in the MicroBlaze. Closer examination of the Jasper code revealed a complex code structure using pointers that would inhibit the isolation of the wavelet transformation.

Stage 2

Due to the problems encountered within stage 1, the uCLinux operating system and Ethernet were dropped from the design. The Jasper code was also deemed unusable given the time constraints. For benchmark testing, MATLAB simulations would be used to test for effectiveness and transformation correctness. Compact Flash was also scratched from the design because the uCLinux requirement was dropped, and the other project group was experiencing problems with this interface.

For this stage of the design, the MicroBlaze was still selected as the core element that would glue the system together. The revised system architecture is shown in Figure [2] below.

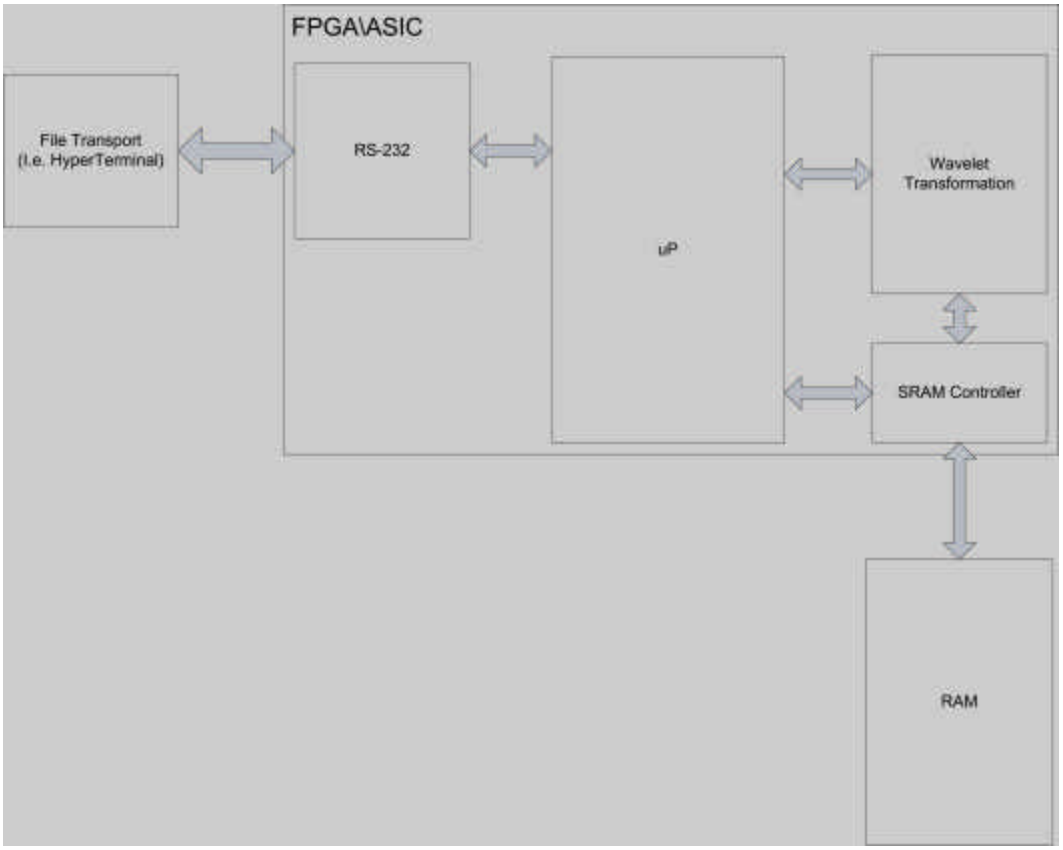


Figure 2. Stage2 System Architecture

The MicroBlaze OPB configuration intended for stage 2 of the design is shown in Figure 3. In the presented configuration, instruction memory is solely contained within the faster on-chip block RAM, but data memory may be located within external RAM or the on-chip RAM. This configuration is recommended for data intense applications that have smaller code sizes as it allows for large data memory, but provides limited instruction memory.

The Xilinx off-the-shelf components consisting of the Microprocessor Debug Module (MDM), External Memory Controller (EMC), GPIO module, and the UART Lite were to be connected to the OPB. The OPB MDM includes a JTAG UART that enables communications with software debugger that comes with the Xilinx EDK. The OPB EMC provides a simplified interface to the MicroBlaze for image storage and retrieval in the external ZBT RAM. The OPB GPIO module is required for simple handshaking communications between the MicroBlaze and the DWT Module.

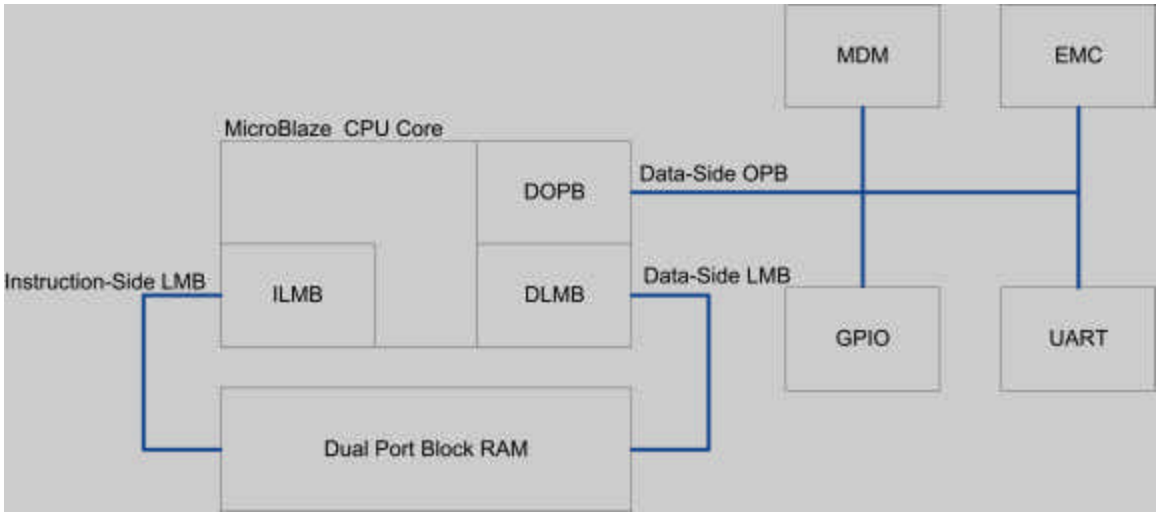


Figure 3. Bus Interface Architecture

The OPB UART is required for communications with the Host PC for the transfer of the raw image file, and the subsequent transfer of the DWT transformed image. The project will use serial (RS232) to transfer the data file to the MicroBlaze. The RS232 driver is implemented using the PC as server and the MicroBlaze as client. A simple SLIP, [24], protocol will be used to ensure communication between the server and client. The client will first send out a START_FILE (0x00) character to indicate the start of data. The end of data will be indicated by an END_FILE (0xFF) character. After each packet of data is sent to the client, the server will also sent out an END (0xC0) character indicating that the packet is sent. If the actual data in the file matches one of the pre defined characters, then a special byte will be sent so that the receiver does not mistake the data as one of the pre defined characters. 0xDD will be used for START_FILE_DATA, which will be sent after the START_FILE character (0x00) if the actual data contain 0x00. 0xDC will be used for END_FILE_DATA, 0xDE will be used for END_DATA. Receiver software running on the MicroBlaze side, would parse the data and store the file in appropriate place for manipulation.

This stage also consisted of modules that are not located on the OPB bus. These modules consist of the Digital Clock Manager (DCM), the DWT Module, a ZBT Memory Controller, and a MUX. The DCM module would provide clock frequency scaling and Delay Locked Loop (DLL) functionality for concise and accurate clock control. The system clock generated by the DCM is shared between the OPB and the non-OPB modules. Since the RAM allocated to the original image file is shared, the original design considered the addition of the DWT module on the OPB bus through the use of the OPB Intellectual Property Interface (IPIF) module. The IPIF Module would then arbitrate and control the DWT module access requests to the OPB EMC. Due to the complexity of the IPIF module and the fact that the shard OPB bus represents a computational bottleneck, this design was dropped. Instead, a separate non-OPB ZBT RAM controller was to be instantiated for the DWT Modules sole access. The multiplexing of the data and address lines between the OPB EMC and the ZBT RAM controller is performed by a MUX that is enabled through the OPB GPIO interface. Further communications between the DWT Module and the MicroBlaze were also to be conducted through the OPB GPIO Module. A block diagram indicating the proposed interface between the OPB IP cores and the non-OPB IP cores is presented in the following diagram (Figure 4).

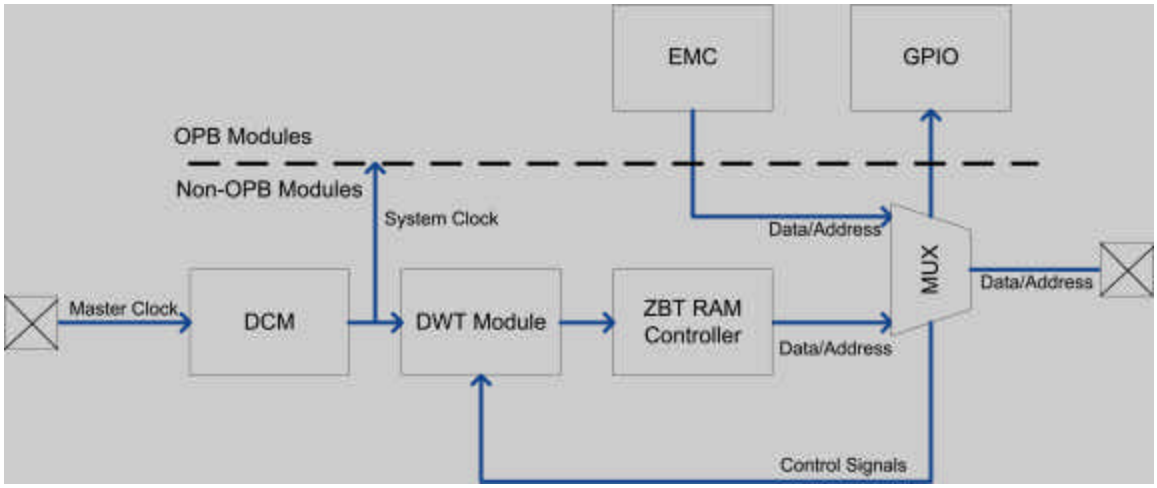


Figure4. Bus Interface Architecture

This stage required the integration of several IP cores; some of which had already been proven, and some that would require further development. The tasks were broken down into three main components; 1) Serial Communications, 2) Memory Access, and 3) DWT integration.

Successes

There are two basic ways to add non-OPB cores to the design of a MicroBlaze system; the peripheral wizard and exporting the XPS project to ISE. For details on exporting XPS projects to the ISE refer to the application note in the Appendix. For added control in architecture development, synthesis, and routing, the team decided that exporting the project to ISE would be the best course of action. An XPS project was successfully ported to ISE as a sub-module, and a DCM was added to the top-level. The system functionality was tested using simple print statements through the XPS software debugger.

To test serial communications, a simple C program was written for the MicroBlaze that would access the UART driver and perform polled byte send and byte receive functions. These functions were tested successfully in a loopback mode with the host PC. Further to this development a C program was written for the host side that would open a file, and send the contents of the file over the Windows COM port using the Windows Serial Port Library. As a sidetrack experiment, and in trying to keep within the intended original design, communications with dial-up-networking using SLIP was attempted. Although the initial AT command handshaking communications were occurring, the host script indicated that the Windows was not sending the dial string to the MicroBlaze. The dial string indicates to the DTE that the host is attempting to make a call. Once the dial the string has been sent the DTE can reply with "CONNECT" and start communications using TCP\IP encapsulated in SLIP.

The Stage 2 design called for two memory controllers, an OPB memory controller and a low level ZBT controller for the DWT module. The EDK provides a sample verilog ZBT controller that was to be used by the DWT. The other project group had already written a very simple toplevel to test this interface, but indicated that it was not working. Our group picked up where they left-off and immediately found that the ZBT controller appeared to be working. Due to the rudimentary nature of the test implemented for the controller, it was impossible to extract timing information or conclude with 100% certainty that the controller was functioning correctly.

Issues

As indicated in stage 1, a top-level with the MicroBlaze using a deprecated ZBT core was built using XPS and was deemed to be functional. When this system was exported to ISE as a sub-module it no longer worked. Initial troubleshooting performed on this problem did not provide any insight as to the root cause.

Even though the ZBT interface did not work, the ISE exported project was still functional in terms of the MicroBlaze. In attempts to debug to the ZBT problem, the sub-level MicroBlaze project was changed several times and subsequently re-synthesized and re-routed. Using a 1.3GHz P4 with 256 MB of RAM, the time from synthesis to BIT file download took more than 25 minutes. Even if all of the OPB components worked the first time, it would undoubtedly take several iterations to integrate the DWT module.

Phase 3

The third and final stage epitomized a near minimal system for DWT demonstration. Due to the turnaround time on a hardware change and the fact that the hardware was still not completely functional, the MicroBlaze system was dropped. The following block diagram (Figure 5) shows the simplified proposed system architecture for stage 3.

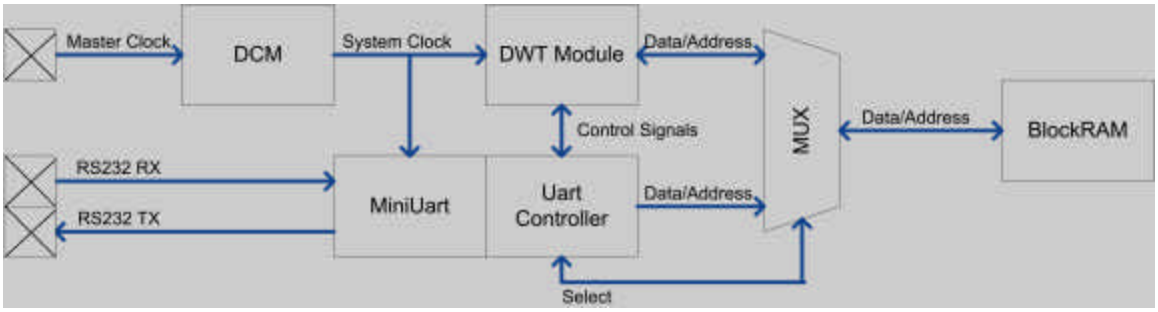


Figure 5. System Design Phase 3

As before, the DCM would be used to provide a stable system clock, and for frequency scaling if required. Instead of external RAM, the up to 128KB (1Mbit) internal RAM would be used for storing the image file and for DWT transformation. The OpenSource.org MiniUart would be used to provide a simplified RS232 interface for sending and receiving the image files from the Host PC. The UART Controller would be responsible for SLIP like framing for data transfers with the Host and would also perform simple control handshaking with the DWT Module.

With the ease of instantiating BlockRAM, the crux to this design is a functioning UART. It was found that the MiniUart IP did not appear to function out-of-the-box. This VHDL code was debugged and modified until a simple loopback system was achieved. On continued development, another significant error in the MinuUart IP was identified. This problem was unable to be resolved because we ran out of time.

In trying to mitigate our risk of not having a system demonstration, development in parallel on another prototype board system was attempted.

- **Xess Board**

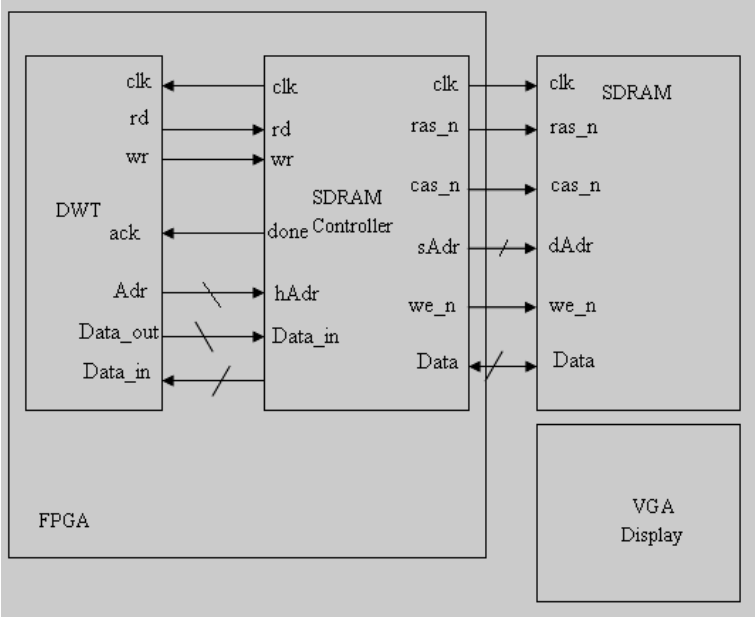


Figure 6. Phase Three system design (using Xess Board)

Our system would look like the diagram in figure 6. The VGA display driver and SDRAM memory controller was obtained from Xess [29]. Images were converted to *.xes format, using img2hex [29],and downloaded to SDRAM. We were able to build a new top level design integrating our DWT module as shown in figure 6; however there were still some handshaking problems we weren't able to solve due to the time constraints.

1.2 Conclusion

In conclusion, the results of the project were disappointing. The group continually encountered design problems that represented significant barriers to overcome. As a work-around for each barrier was designed a new barrier was found. The unfortunate side-effect of the diminishing scope, the results of the project do not represent the effort that went into the journey. In hindsight, we may have looked like gods if we were able to use a ported uCLinux with ease and if all the other IP modules fell into place. But without the trials and tribulations encountered when debugging complex systems, you learn very little about the details of the pieces you are integrating together.

A lesson re-learned was one that usually separates the Engineers from the Hobbyists. When an Engineer encounters a design problem they first define the problem. This step usually involves pouring over documentation, problem simulation, and hardware debugging. In the case of peripherals in a complex computer system, this process almost always leads to the use of complex hardware debugging tools such as logic analyzers, and oscilloscopes. With the expense

of the Multimedia board, the lack of access of hardware debugging tools, and the loose and conflicting documentation provided by Xilinx, the group sometimes had to resort to the time-consuming and often non-profitable habit of trial-and-error. The group was able to perform some debugging and problem solving through the probing of two LEDs wired to various signals, but this process was inhibited by the lack of access to pins; the multimedia board had a protection shield to prevent such debugging that may cause unintentional static discharges if not performed with proper static protection, and was not practical for debugging buses or complex signals.

The group acknowledges that the project was high risk to start with due to the use of hardware and tool chains that were new to the students, the professor, and the teaching assistant. Given the problems encountered with this project, the group makes the following recommendation for subsequent years; with the lack of computing power available for synthesis and routing and the lack of access to hardware debugging tools, it is not recommend to build projects that incorporate large complex IP modules such as the MicroBlaze. Furthermore, it would be best to start with a minimal system and grow outward through the addition of features, instead of the reverse.

1.3 JPEG2000 Codec Architecture

The wavelet transform decomposes a signal in terms of a set of basis functions generated by translation and dilation/compression of a localized mother wavelet. The discrete version of the wavelet transform, known as the discrete wavelet transform (DWT), has found a large variety of signal processing applications. In addition to many appealing features following DWT such as multi-resolution analysis and time-scale representation, a key factor that makes the DWT become practical is the availability of fast algorithm wavelet transforms [1].

Due to its advantageous features, the Discrete Wavelet Transform (DWT) has emerged to become a basic encoding algorithm for recent data compression standards. During the last decade intensive research efforts have been spent both on the theoretical foundations of DWT and on its practical implementation in modern signal and image processing systems. This effort has been successful; as a consequence, the DWT has been selected as the transform coding kernel for the forthcoming JPEG2000 image coding standard [2].

1.3.1 The complete JPEG2000 Standard Algorithm

The new compression standard, JPEG2000, is intended to serve as a general compression algorithm that can be efficiently used over wide range of integrated service networks with different kinds of images. The scalability requirement for the JPEG2000 is intended to allow trading off between compression rates and image quality according to the types of images. Moreover, one of the major features of the JPEG2000 is that it allows both lossy and lossless compression ability within the same framework, as opposed to the JPEG standard [1]. Figure 7 summarizes the complete encoding and decoding steps for the JPEG2000.

The main encoding steps include:

1- Preprocessing:
This step is simply a DC level shifting step. Input sample data should have a nominal dynamic range centered about zero.

2- Forward Transform:

- The input image is divided into blocks, called tiles. The JPEG2000 standard committee has adopted the Discrete Wavelet Transform (DWT) as the transform of choice. The DWT can operate in two different modes:
  1- Irreversible Transform (real-to-real):
  Daubechies 9-tap/7-tap filter
  2- Reversible Transform (Integer-to-Integer):
  Le Gall 5-tap/3-tap filter (Integer Wavelet Transform)
- Two Filtering Modes are supported:
  1- Convolution based.
  2- Lifting Scheme.
- DWT module should be scalable with variable :
  width - Height - Decomposition levels.

Figure 8. Wavelet Forward Transform Example (3 levels decomposition)

3- Quantization step:

$$l_{ij} = \left\lfloor \frac{q_{ij}}{\Delta_b} \right\rfloor \qquad \Delta_b = 2^{R_b - d_b}\left(1 + \frac{m_b}{2}\right)$$

R_b : Dynamic range: depends on the number of bits and the choice of the wavelet

d_b :Exponent

m_b Mantissa

4- Entropy Coding:

The encoding scheme recommended is the Embedded Block Coding with Optimization Truncation of the embedded bit stream (EBCOT). The key features of this encoding are:

- Each subband is divided into rectangular blocks, which are coded independently called code blocks
- Bitstream is organized in a succession of layers
- Each layer corresponds to a certain distortion level
- The quality of the reproduction is proportional to the numbers layers received

1.3.2 DWT Different Realizations Alternativs: A Theoretical Background

This section discusses different realization alternatives for the DWT, with a special focus in the the lifting scheme-based realization that was chosen for our implementation.

Historically, the wavelet transform has gained widespread acceptance in fields of signal processing and image coding [3]. In the wavelet transform, dilations and translations of a mother wavelet are used to perform a spatial/ frequency analysis of the input signal. Different realizations alternatives for the DWT algorithm can be found in the literature, figure 9 shows three different realizations alternatives. These are namely, the Mallat filtebank realization, lattice structure, and the lifting scheme-based realization.

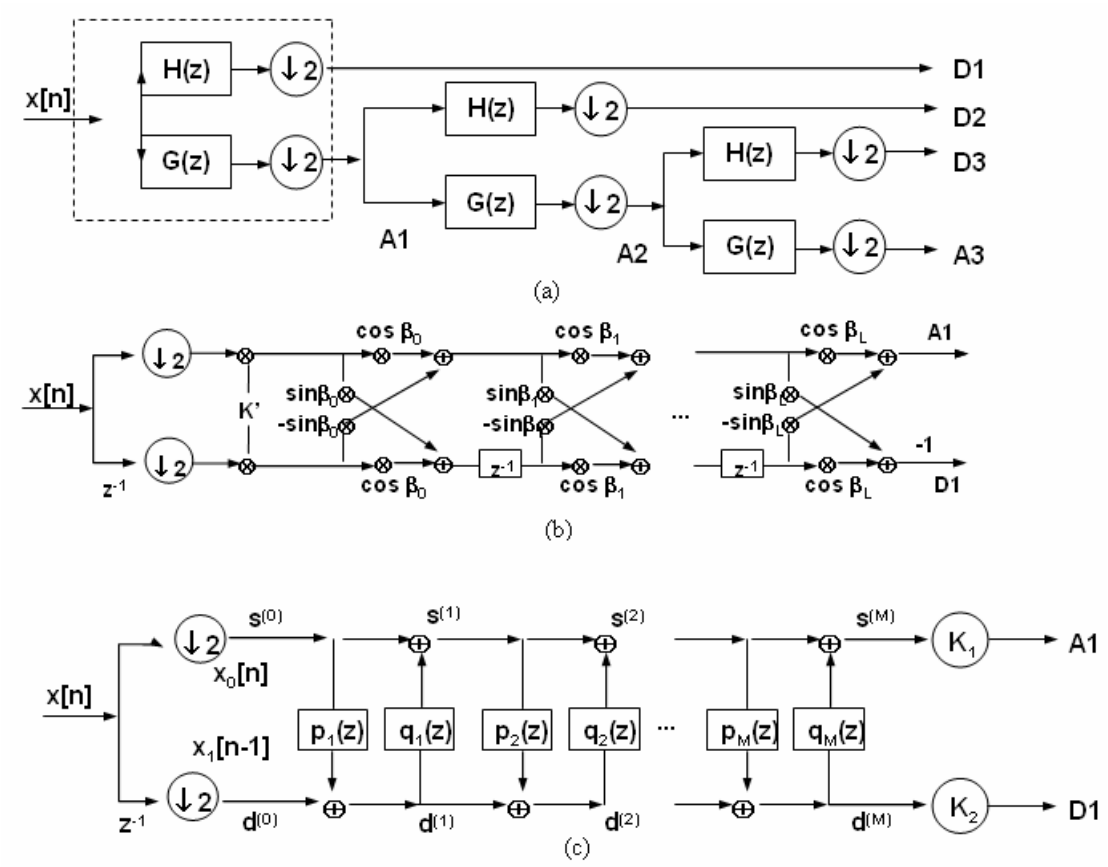


Figure 9. Different Realizations Alternatives for DWT. (a) Mallat FilterBank, (b) Lattice Structure, and (c) Lifting scheme.

- Malat Filter bank realization

The DWT has been traditionally implemented by means of the Mallat filter bank scheme [1]. The algorithm includes two main steps: signal decimating and filtering with a pair of

Quadrature Mirror Filters (QMFs). Figure 10 shows a A one stage wavelet filter bank analysis (left side) and synthesis (right side). The filter bank can be realized using FIR filters. The process consists of performing a series of dot products between the two filter masks and the signal.

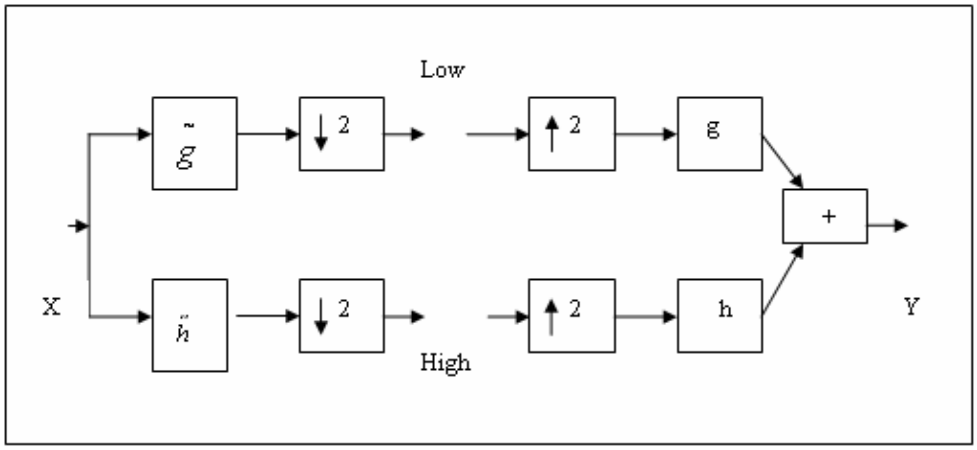


Figure 10. A one stage wavelet filter bank analysis (left side) and synthesis (right side) [1].

- Lifting Scheme-based realization

Sweldens has proposed an alternative framework, called Lifting Scheme (LS) [4], to compute the DWT. The lifting scheme is a computationally efficient way of implementing DWT. It has the advantage of reduced complexity, compared to the conventional convolution based scheme, as illustrated in Table 1. Moreover, all the operations within the lifting steps can be performed in parallel, hence the possibility of a fast implementation.

Table 1. Complexity of lifting scheme vs. convolutional wavelet transform for different wavelet filters

Filter	Muls/Shifts		Additions	
	convolutional	Lifting	Convolutional	Lifting
(5,3)	4	2	6	4
C(13,7)	8	4	14	8
(9/7)	9	5	14	8

.

Wavelet Transform from Filter Bank to Lifting Scheme Implementation :

In our design for the DWT module, we have chosen the lifting scheme approach for the realization of the DWT. Thus, next we will show some details about the mathematical properties of this scheme.

Starting from the Malat filter bank realization , shown in figure 10, the conditions for perfect reconstruction are given by [4] as:

$$h(z)\tilde{h}(z^{-1})+g(z)\tilde{g}(z^{-1})=2$$
$$h(z)\tilde{h}(-z^{-1})+g(z)\tilde{g}(-z^{-1})=0$$

Polyphase representation :

Figure 10 shows that the wavelet decomposition is performed by filtering then downsampling. Clearly, it would be more efficent to do the downsampling before the filtering [1].

Employing the modified FIR filter in the wavelet transform, we can end up with a new matrix representation :

$$\begin{pmatrix} l(z) \\ g(z) \end{pmatrix} = \tilde{P}(z) \begin{pmatrix} x_e(z) \\ z^{-1}x_o(z) \end{pmatrix}$$

where $\tilde{P}(z)$ is the *polyphase matrix*:

$$\tilde{P}(z) = \begin{pmatrix} \tilde{h}_e(z) & \tilde{h}_o(z) \\ \tilde{g}_e(z) & \tilde{g}_o(z) \end{pmatrix}$$

The perfect reconstruction condition is :

$$\tilde{P}(z^{-1})P(z) = I$$

Lifting Representation :

Ingrid Daubchies have proved that, [1], if we start with a complementary filter pair (h,g), the polyphase matrix P(z) can always be factored into a number of lifting steps, which are easier to implement:

$$P(z) = \begin{pmatrix} K_1 & 0 \\ 0 & K_2 \end{pmatrix} \prod_{i=1}^{m} \left\{ \begin{pmatrix} 1 & s_i(z) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ t_i(z) & 1 \end{pmatrix} \right\}$$

Figure 11 shows the realization of the lifting scheme and the complexity of a (M,N) filter.

Lifting Scheme representation for the 5/3 Le Gall wavelet filters:

In our design for the hardware lifting scheme-based DWT module, we have chosen the (5/3) le Gall wavelet filters. The JPEG2000 standard committee has recommended using the Le Gall wavelet filters for the integer mode operation. The following shows how the lifting equations for the le Gall 5/3 filter along with the block diagram can be derived.

Starting with the 5/3 wavelet filters transform equations:

$$\tilde{h}(z) = -\frac{1}{8}z^{-2} + \frac{1}{4}z^{-1} + \frac{3}{4} + \frac{1}{4}z - \frac{1}{8}z^{2}$$
$$\tilde{g}(z) = \frac{1}{4}z^{-2} - \frac{1}{2}z^{-1} + \frac{1}{4}$$

The factorized polyphase matrix for the le Gall filter is :

$$\tilde{P}(z) = \begin{pmatrix} 1 & 0 \\ 0 & -\frac{1}{2} \end{pmatrix} \begin{pmatrix} 1 & \frac{1}{4} + \frac{1}{4}z \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ -\frac{1}{2}z^{-1} - \frac{1}{2} & 1 \end{pmatrix}$$

The lifting equations can then be directly obtained:

$$y_{2i+1} = -0.5(x_{2i} + x_{2i+2} + x_{2i+1}$$
$$y_{2i} = 0.25(y_{2i+1} + y_{2i+3}) + x_{2i,where}\ 0 \le i < N/2$$

Using the above equations, one can sketch how the lifting scheme based realization for the 5/3 leGall filters would look like. Figure 12 represents the basic building block of the 1D-DWT using the Le Gall wavelet filters. Since all coefficients are multiplies of 2, all multiplications and divisions can be replaced by shifting operations. From the figures 12, we can see an interesting property of the lifting representation. Every time we apply a predict or update lifting step we add something to one stream. All the samples in the stream are replaced by new samples and at any time we need only the current streams to update sample values.

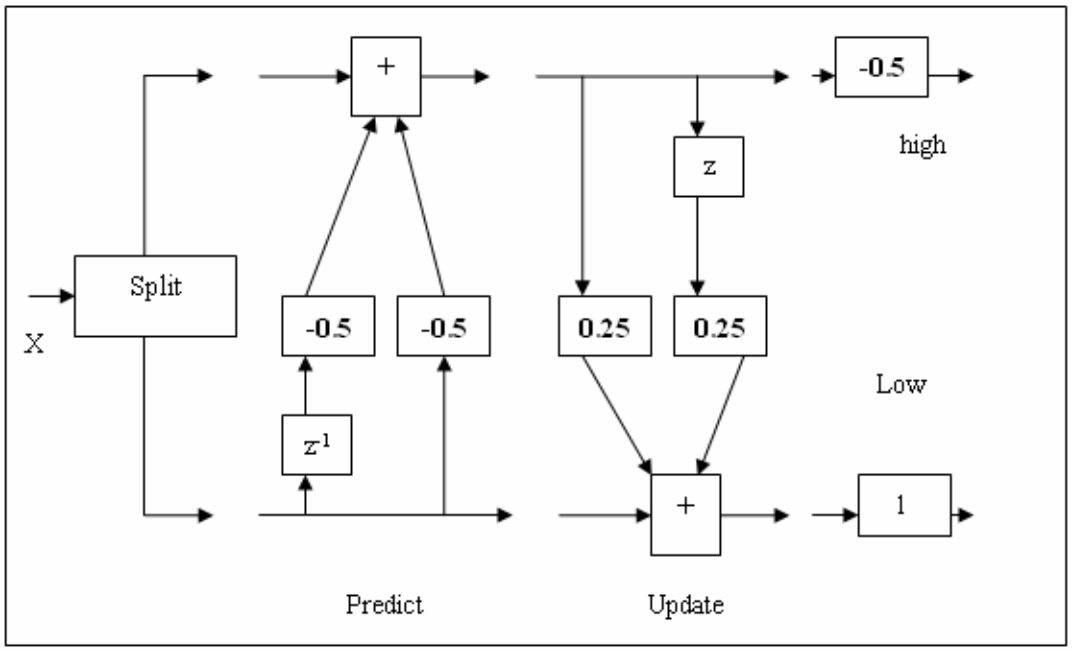


Figure 12 The implementation of the wavelet transform of le Gall filters [4]

In lifting literature, the two basic steps are the *predict step*,and the *update step*. The idea behind this terminology is that lifting of the high-pass subband with the low- pass subband can be seen as prediction of the odd samples from the even samples[1].

- Integer Wavelet Transform (IWT)

The lifting scheme-based realization allows to have what is called integer-to integer transform, or simply Integer Wavelet transform. The transform coefficients of the IWT are exactly represented by finite precision numbers, thus allowing for truly lossless encoding. This would help us reduce number of bits for the sample storage and to use simpler filtering units. Integer wavelet transforms is achieved by rounding off the output of $s_i(z)$ and $t_i(z)$ filters, in figure 11, before addition or subtraction. This would lead to a very beneficial class of transforms, in terms of computational complexity and memory requirements. Yet, they are highly dependant on the choice of the factorization of the polyphase matrix [6]. Equation xx shows the lifting steps for the 5/3 le Gall Integer Wavelet Transform. The rational coefficients allow the transform to be reversible, i.e invertible in finite precision analysis, hence giving a chance for performing lossless compression [26].

$$y_{2i+1} = \lfloor -0.5(x_{2i} + x_{2i+2}) \rfloor + x_{2i+1}$$

$$y_{2i} = \lfloor 0.25(y_{2i+1} + y_{2i+3}) \rfloor + x_{2iwhere} \quad 0 \le i < N/2$$

1.3.3 Proposed Design

In this project, a scalable lifting scheme based Integer wavelet transform unit was implemented. 5/3 le Gall wavelet filters were employed. Design Acceleration has been achieved by techniques like exploiting the parallel nature of sub units, pipelining, and re-usability of data. There were two major design stages towards the final proposed architecture.

- **Parallel Operation**

First, starting from the sequential realization of the algorithm, we tried to further exploit the parallel nature of the algorithm through parallel operation of independent units. Second, the designed was further optimized by introducing pipeline stages. The following highlights the major features of the proposed architecture.

Table 1 shows the number of operations required by the 5/3 filter for each predict or update stage. In our design, we introduce parallel processing of independent sub units. Input samples are accessed through a four-sample wide window, allowing two concurrent predict operations and two concurrent update operations, as shown in figure 12.

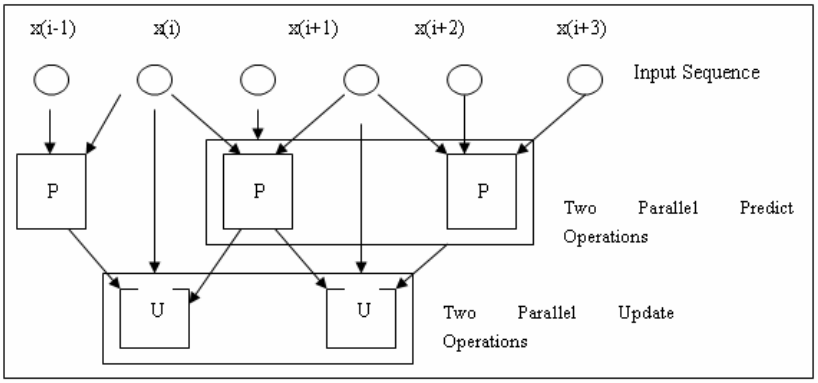


Figure 12. Parallel Processing of input data samples.

Registers were used for temporary storage, and reusability of temporary data. Figures 13 and 14 show the block diagram for the update and predict modules respectively.

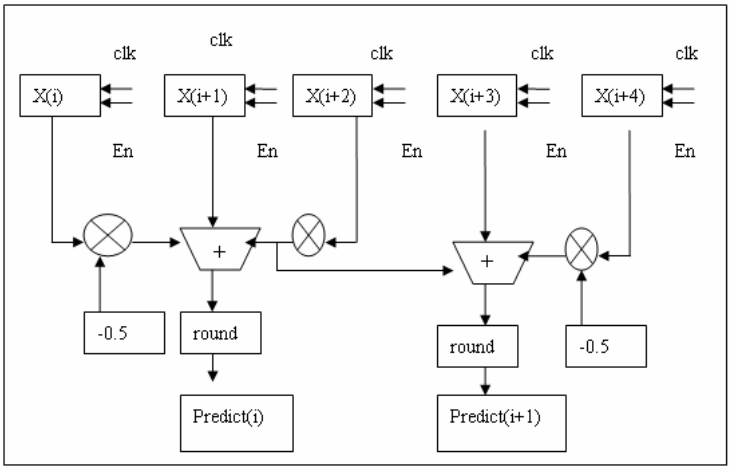


Figure 13. Predict Filter module with two concurrent values being calculated.

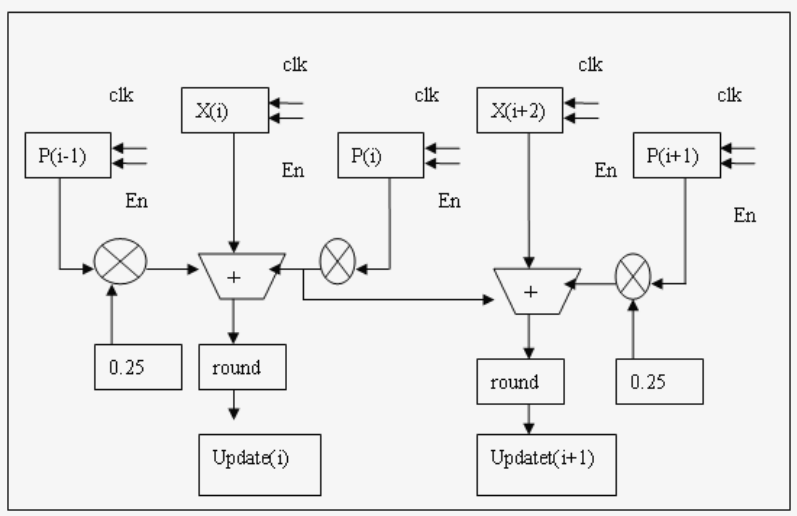


Figure 14. Update Filter Module with two concurrent values being calculated.

- **Pipelining the Optimized DWT**

By pipelining the DWT, the highpass and lowpass coefficients can be computed during the time the memory is being accessed, rather than having to wait until the reads are complete, computing the coefficients, and then writing them. Due to the nature of the lifting scheme DWT, the low pass coefficients depend not only on the high pass coefficient in the stage immediately before them, but also the pixel values of higher stages. As a result, it is necessary to ensure that the pipeline data is valid at all times.

By examining the coefficient equations, we can determine which values are necessary to compute the coefficients:

$L_N = f(H_{N-1}, P_{2N-1}, H_N)$

$H_N = g(P_{2N-1}, P_{2N}, P_{2N+1})$

The smallest window we can look at is two pixels. In this design, two pixels are read in and two coefficients are computed. When outputting L_N and H_N, L_{N+1} and H_{N+1} are being computed. This requires P_{2N-1}, P_{2N} and P_{2N+1} to be available. P_{2N+1} and P_{2N+2} are read in, and then fed directly into another set of registers resulting in P_{2N} and P_{2N-1}, since two pixels are read in every clock cycle. Note that the design ensures that L_N and H_N become valid during the same clock cycle.

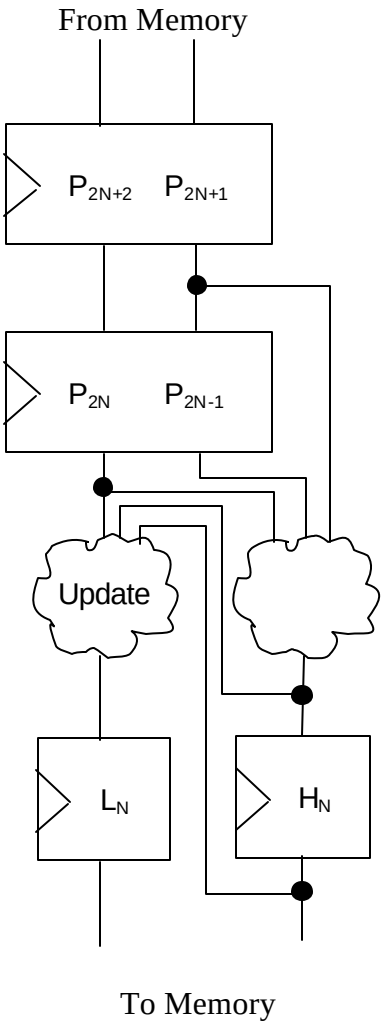


Figure 15. Pipelined DWT with a window of 2 pixels

For a window of four pixels, the concept is the same, only more work is done in parallel.

$L_N = f(H_{N-1}, P_{2N-1}, H_N)$

$$L_{N+1} = f(H_{N-1}, P_{2N-1}, H_N)$$

$$H_N = g(P_{2N-1}, P_{2N}, P_{2N+1})$$

$$H_{N+1} = g(P_{2N+1}, P_{2N+2}, P_{2N+3})$$

When L_N, L_{N+1}, H_{N+1} and H_{N+2} are available on the outputs, L_{N+2}, L_{N+1}, H_{N+3}, and H_{N+4} are being computed. This requires P_{2N+3} and P_{2N+5} through P_{2N+9} to be available. Pixels P_{2N+6} through P_{2N+9} are read in each clock cycle, and two pixels are delayed to give the other two required pixels.

By delaying H_{N+2} one stage before outputting it, we can have H_N, L_N, H_{N+1} and L_{N+1} become valid all on the same clock cycle, which makes the control system easier to design.

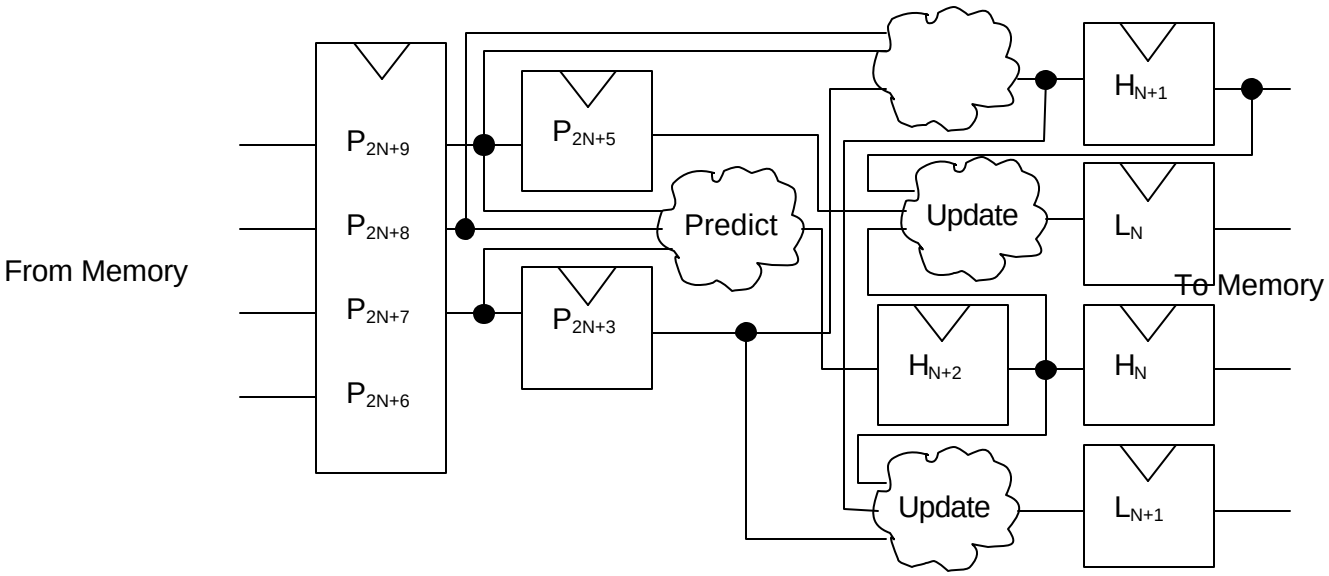


Figure 16. Pipelined DWT with a window of 4 pixels (final design)

Notice that the critical path from P_{2N+6} or P_{2N+7} to L_{N+1} is through two stages of combinational logic. If the clock frequency needs to be increased, the pipeline can be redesigned with another stage to break up the path at the expense of an increased latency. This is left as an exercise to the reader.

Filling the Pipeline:

The wavelet transform specifies that L_1 should be calculated using H_1, P_1, and H_1. This is equivalent to inputting the beginning of the pixel stream as P_3, P_2, P_1, P_2, P_3, rather than beginning with P_1. This will end up generating one extra high pass coefficient, but the control system simply ignores it.

Emptying the Pipeline:

Similarly to filling the pipeline, when reading in the last pixels, the 2nd last and 3rd last pixels are repeated in reverse order after the last pixel. IE, the end of the pixel stream will look like P_{N-3}, P_{N-2}, P_{N-1}, P_N, P_{N-1}, P_{N-2}. In some cases this will have the effect of calculating extra coefficients. The control system simply keeps track of how many values it has written, and stops after it has written all the expected coefficients. Any extra values are ignored. For an even number of input pixels, there will be an equal number of high pass and low pass

coefficients. For an odd number, the algorithm results in one more low pass coefficient than the number of high pass coefficients.

Further Improvements (better memory organization)

If dual-port RAM is available or there is more than one bank of RAM, reads and writes can be carried out simultaneously, approximately halving the number of clock cycles needed for execution.

Memory access is the major bottleneck to the design. The pipeline requires 4 clock cycles for to read in new values, but only one clock cycle to compute the new value. By intelligent storage of the image in memory, even faster speeds may be obtained. Simply storing more consecutive pixels per word will not work, as the wavelet transform must be able to read the memory in either rows or columns. However, if the pixels are stored in square blocks, not only is it possible to always read multiple pixels per read, but it allows the system to work on multiple rows or columns in parallel. For example, the first 32-bit word in memory could represent the first two 8-bit pixels of the first row, and the first two pixels of the second row.

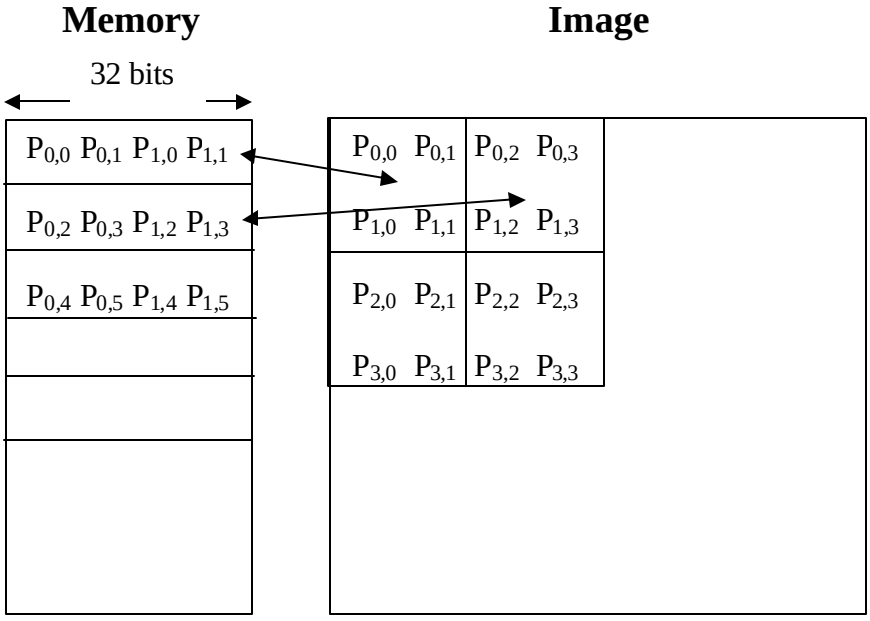


Figure 17. Improved memory organization

When reading rows, the first two words of memory are read in, and the first two bytes of each word are used to calculate the first two high-pass coefficients and the first two low-pass

coefficients of the first row. Meanwhile, the first four coefficients of the second row are being processed in parallel.

Another complication comes from the fact that when writing the coefficients out, they are not written to consecutive locations. However, since we are calculating two rows in parallel, we will end up with one full block of high-pass coefficients, and one full block of low-pass coefficients. This method imposes the limitation that the image height and width must be a power of two.

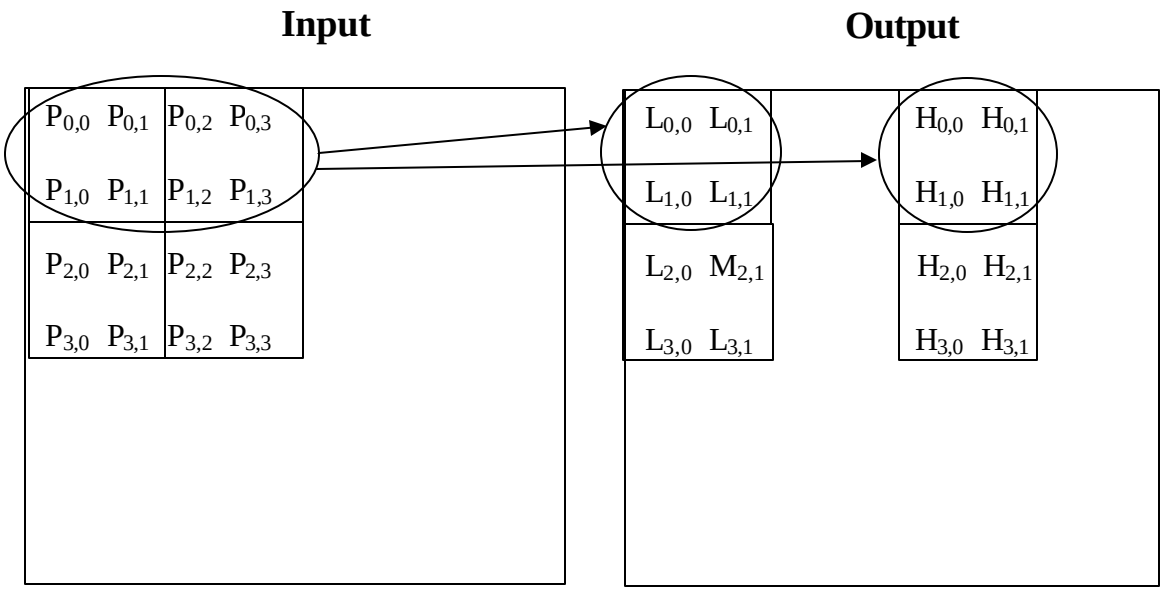


Figure 18. Writing coefficients with improved memory organization

Although there was not enough time to implement the design, it is conceptually faster than the existing design, and should allow for a speedup of 4x by requiring half the number of cycles to read in four pixels, and by allowing two columns or rows to be processed in parallel. The pipeline for the DWT calculation would stay the same, but it would be instantiated twice. Only the control system and memory controller would need to be changed in order to process blocks. Furthermore, in the same manner 128-bit words could be used to access 16 pixels at a time, and provide an even greater speedup.

the 2D-DWT was obtained by performing the designed 1D-DWT on both the rows and columns. Figure 19 shows how would the 2D-DWT module operate on an input image, and what would be the final output.

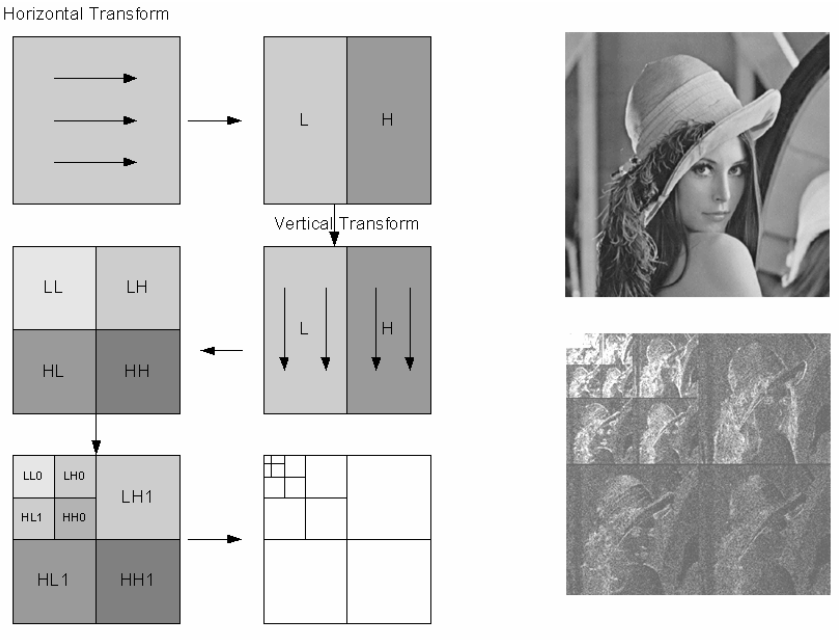


Figure 19. 2-D Recursive Pyramid DWT

1.3.4 Precision Issues

As for the conventional DWT realizations, partial transform results need to be represented with a high precision. This raises storage and complexity problems. On the other side, Integer wavelet transforms (IWT) results in integer intermediate results. Thus, it is possible to use integer arithmetic without encountering rounding problems [27,28]. Consequently, based upon precision studies from the literature for the 5/3 filter IWT [28]; a fixed precision of 8 bits per pixel were selected. The error introduced by this precision has been proved, through comparing software and hardware implementations, to be within an accepted range to most applications.

2.0 Requirements

2.1 Part Requirements

To complete the project the following hardware components are required:

- Reprogrammable FPGA
- External RAM
- Compact Flash (16MB will be sufficient)
- RS232 Interface
- Ethernet

These requirements are met by the Virtex II\MicroBlaze Multimedia board [7].

2.2 Resource Requirements

2.2.1 Peripheral Resource Requirements

VHDL Component	Logic Cells	Comments
Microblaze	900[8]	Virtex-II (125MHz, 82D-MIPS)
LMB Memory Controller	3 Slices, 6 LUTs, 2 FFs[9]	
LMB Memory Controller	3 Slices, 6 LUTs, 2 FFs[9]	
OPB EMC	310 Slices, 254, LUTs, 445 FF's[10]	From data sheet, assuming maximum resources used.
UART Lite	108 LUTs, 57 FF's[11]	From data sheet, assuming maximum resources used.
OPB Bus with Arbiter	436 Slices, 668 LUTs, 145 FFs[13]	From data sheet, assuming maximum resources used.
MDM	188 Slices, 292 LUTs, 204 FFs[14]	From data sheet, assuming maximum resources used.
Block RAM (BRAM) Block	Block RAMs 64[15]	From data sheet, assuming maximum resources used.
Local Memory Bus (Instruction Side)	*	Local Memory Bus (Instruction Side)
Local Memory Bus (Data Side)	*	Local Memory Bus (Data Side)
DCM	108 LUTs, 57 FFs	From data sheet, assuming maximum resources used.
MiniUart	155 LUTs, 96 FFs	synthesized

*The data sheet does not provide the number of gates used in the LMB design. You cannot compile the LMB buses without a master device attached to the bus. The synthesis output does not provide a breakdown of what modules used how many LUTs, FFs, etc. The number of gates used by the master device is variably dependant on the clock frequency.

2.2.2 DWT Resource Requirements

This section presents the initial attempts to provide a rough estimation for the resources required to implement a 2D-DWT using the Le Gall 5/3 filters, following the specifications outlined by the JPEG 2000 standard committee in [2].

- JPEG 2000 standard committee specifications for the wavelet module:

Starting with the specifications outlined in [1], the DWT module is supposed to have following parameters:

1. Picture Width (x)
2. Picture Height (y)
3. Levels of Decomposition (maximum :m)
4. Mode of operation (Integer :for 5/3 le Gall filters , Real : for 9/7 daub filters).

Since we will limit our implementation to only one wavelet family, thus we will have only one mode of operation using the 5/3 le Gall filters. Thus our module should be parameterized with the picture width, picture height, and levels of decomposition (we will set maximum allowable number of levels to 5).

- Computational Complexity for a single lifting step

If the signal is numbered from 0 and if even terms are considered to be the lowpass values and the odd terms the highpass values, we can interpret the le gall polyphase matrix in the time domain as:

$$y_{2i+1} = -0.5(x_{2i} + x_{2i+2}) + x_{2i+1}$$
$$y_{2i} = 0.25(y_{2i+1} + y_{2i+3}) + x_{2i,where}\ 0 \le i < N/2$$

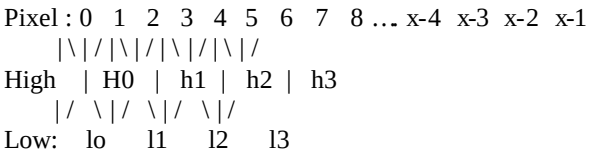
where s are the signal values, and s are the transformed signal values. Note that the odd samples are calculated from even samples, and even samples are calculated from the updated odd samples. Using the above equations, we can calculate the number of multiplications (shifts) and additions required.

# of Multiplications / Shifts	# of Additions
2	4

- Design Parameters Assumptions
  1. A precision of 8 bits will be used; since we are dealing with integer arithmetic in the case of the 5/3 le Gall filters.
  2. External memory is used where original image is stored, and the resulting image will also be stored in the external memory at different locations.

i. Resources Estimation for the lifting-based 2-D DWT module

The lifting scheme-based DWT works as follows:



The 2D-DWT is performed by applying a 1D-DWT step on each row. Then 1D-DWT is performed on the resulting coefficents coloumnwize. During a 1D-DWT step, for an N input elements we have N/2 high-pass coefficients and N/2 low-pass coefficients.

We define a cost function in terms of the number of multiplications (shifts) and additions required to perform a 2D-DWT on an input image with width (x) and height (y) :

1D-DWT on each row :

We have y rows, each row contains x pixels. Each 1D-DWT will require x/2 single lifting steps to obtain x/2 low-pass and high-pass coefficients. Thus the cost function is:

Cost function_rows = (x/2) * (2 MM + 4 AA)

Where MM: cost of one multiplier , and AA: cost of one adder

When 1D-DWT is applied to each coloumn, we obtain a similar cost function :

Cost function_cols = (y/2) * (2 MM + 4 AA)

Thus the final cost function for a 1-level 2D-DWT will be :

Cost function_2ddwt_1level = (x/2 + y/2) * (2 MM + 4 AA)

For an m-level 2D-DWT, we get :

Cost function_2ddwt_m level = (x+y)/ (2^[m-1]) * (2 MM + 4 AA)

With the assumptions that maximum allowed decomposition level is m, we can obtain final cost function :

Cost function_final = $\Sigma^{7}_{m=1}$ { (x+y)/ (2^[m-1]) * (2 MM + 4 AA) }

The multiplications (MM) are to be implemented using shifts. The multiplication by (1/4) is a shift right 2 bits, while multiplication with (1/2) is a shift right one bit. Thus the Multiplications cost (2MM) is simply : two shifts by 1 bit and 2 bits. As for the adders (AA), we need 4 8-bit adders. We can start with using the adder provided by the vhdl standard libraries, moreover, we can go for desigining a pipelined adder to speed up the operation. This will be decided based on the obtained speed.

For the Adder (AA):

A typical standard 8-bit adder has been synthesized on the vertex 2. .

# of Flip flops	8
# of BELs	24

For the Multipliers (MM) :

Since multiplication is done by powers of two, the multiplication can be achieved using shift right operation.

# of Flip flops	6
# of BELs	1

Now, let's define an area cost function for each of the adder and shifter..

For Adder : AA_{area} = 8 FFs

For Mult. : MM_{area}= 6 ffs

Now we can substitute in the general cost function equation derived above. Assuming the picture width (x) is 64 and height (y) is 64. We finally obtain an estimated cost for the area .

Estimated Area Consumption = $\sum^{5}_{m=1}\{ (64+64)/ (2^{[m-1]}) * (2 *6 + 4 *8) \}$

= 1092 ffs.

B- Address controller Module

This is supposed to be a Mux that would produce the exact read and write memory addresses during the operation of the DWT module. According to the intermediate state of the DWT module, the correct read/ write address is calculated using :

- Source offsets at which the row/ col. Inputs are stored.
- Destination offsets at which the low and high pass coefficients will be stored.
- The step size between when reading/ writing conseqective pixels.

A typical address assignment at a specific state would be something like this:

Src_address <= start_offset_src + step_src

A rough estimation of the resource consumption of such controller is summarized in the table below. Since we will have two controls for rows and cols respectively, the figures estimated were multiplied by two.

# of FFs	32 * 2 =64
# of BELs	24 * 2 =48

C- External Memory

The DWT module will work on an image stored in an external memory. Resulting coefficients are written back to memory. For an image with width (x) and height (y), we will need 2*x*y bytes of storage.

Thus for a 64 * 64 image, we need : 2*64*64= 32768 Bytes

** All these calculations assuming the image is a gray scale image. Incase of an RGB image, all the results will be multiplied by 3.

3.0 FPGA User I/O Pins

Pin-Out (Only Accesses 1 Bank of the ZBT RAM)

```
# UART
Net RX LOC=C8;
Net TX LOC=C9;
Net RS232_req_to_send LOC=B8;

# GPIO
NET LED<0> LOC=B27;
NET LED<1> LOC=B22;

# External Memory Constraints
# Bank 0
NET "SRAM_ADDR<0>" LOC = "T23";
NET "SRAM_ADDR<1>" LOC = "U23";
NET "SRAM_ADDR<2>" LOC = "AB29";
NET "SRAM_ADDR<3>" LOC = "AA29";
NET "SRAM_ADDR<4>" LOC = "AA27";
NET "SRAM_ADDR<5>" LOC = "AB27";
NET "SRAM_ADDR<6>" LOC = "H25";
NET "SRAM_ADDR<7>" LOC = "G25";
NET "SRAM_ADDR<8>" LOC = "G28";
NET "SRAM_ADDR<9>" LOC = "H29";
NET "SRAM_ADDR<10>" LOC = "U27";
NET "SRAM_ADDR<11>" LOC = "T27";
NET "SRAM_ADDR<12>" LOC = "V29";
NET "SRAM_ADDR<13>" LOC = "U29";
NET "SRAM_ADDR<14>" LOC = "T24";
NET "SRAM_ADDR<15>" LOC = "T25";
NET "SRAM_ADDR<16>" LOC = "U28";
NET "SRAM_ADDR<17>" LOC = "F28";
NET "SRAM_ADDR<18>" LOC = "L23";
# BANK0_A0 (Data Bus for Byte-A)
```

```
NET "SRAM_DATA<0>" LOC = "T30";
NET "SRAM_DATA<1>" LOC = "P28";
NET "SRAM_DATA<2>" LOC = "R25";
NET "SRAM_DATA<3>" LOC = "R29";
NET "SRAM_DATA<4>" LOC = "R27";
NET "SRAM_DATA<5>" LOC = "R23";
NET "SRAM_DATA<6>" LOC = "N30";
NET "SRAM_DATA<7>" LOC = "K26";
# BANK0_B0 (Data Bus for Byte-B)
NET "SRAM_DATA<8>" LOC = "M25";
NET "SRAM_DATA<9>" LOC = "J29";
NET "SRAM_DATA<10>" LOC = "K27";
NET "SRAM_DATA<11>" LOC = "L24";
NET "SRAM_DATA<12>" LOC = "H27";
NET "SRAM_DATA<13>" LOC = "H26";
NET "SRAM_DATA<14>" LOC = "K25";
NET "SRAM_DATA<15>" LOC = "H28";
# BANK0_C0 (Data Bus for Byte-C)
NET "SRAM_DATA<16>" LOC = "J25";
NET "SRAM_DATA<17>" LOC = "J26";
NET "SRAM_DATA<18>" LOC = "J28";
NET "SRAM_DATA<19>" LOC = "K24";
NET "SRAM_DATA<20>" LOC = "J27";
NET "SRAM_DATA<21>" LOC = "K29";
NET "SRAM_DATA<22>" LOC = "L25";
NET "SRAM_DATA<23>" LOC = "L26";
# BANK0_D0 (Data Bus for Byte-D)
NET "SRAM_DATA<24>" LOC = "P30";
NET "SRAM_DATA<25>" LOC = "P23";
NET "SRAM_DATA<26>" LOC = "P27";
NET "SRAM_DATA<27>" LOC = "T29";
NET "SRAM_DATA<28>" LOC = "R24";
NET "SRAM_DATA<29>" LOC = "R28";
NET "SRAM_DATA<30>" LOC = "U30";
NET "SRAM_DATA<31>" LOC = "T28";
NET "SRAM_CLKEN" LOC = "G30";
NET "SRAM_WEN" LOC = "F26";
NET "SRAM_BEN<0>" LOC = "J24";
NET "SRAM_BEN<1>" LOC = "H24";
NET "SRAM_BEN<2>" LOC = "F29";
NET "SRAM_BEN<3>" LOC = "G29";
NET "SRAM_CEN" LOC = "G26";
NET "SRAM_OEN" LOC = "F30";
NET "SRAM_ADV" LOC = "K23";
```

Stage 3.

```
Net MASTER_CLOCK_P PERIOD =37037 ps;
Net MASTER_CLOCK_P LOC=AH15;
Net ALTERNATE_CLOCK_P LOC=AD16;

# MEMORY CLOCK FEEDBACK LOOP
Net MEM_CLK_FBIN_P LOC=AE15;
Net MEM_CLK_FBOUT_P LOC=AH14;

# DCM RESET EXTENSION
Net EXTEND_DCM_RESET_P LOC=AH7;

# LED LIGHT - ACTIVE LOW
Net USER_LED0_Z LOC=B27;
Net USER_LED1_Z LOC=B22;

# RS232
Net RS232_RX LOC=C8;
Net RS232_TX LOC=C9;
Net RS232_req_to_send LOC=B8;
```

4.0 Features to be Added/Deleted

The first stage requires the integration of the microprocessor, the communication interfaces, and the external memory components. In this stage, the uCLinux OS is also loaded onto the microprocessor. To show the functional form at the system level a C program ported to uClinux will perform all the JPEG2000 transformation and compression. The JPEG2000 compression time will be recorded for future benchmarking. The completion of this stage is critical to the success of the project.

The second stage includes the integration of the fast implementation of the wavelet transformation. The results of the fast hardware implementation are compared to the benchmark software implementation in stage one.

Stage three includes the addition of the USB interface. The device will interface to a web camera in USB host mode and request, convert to the JPEG2000 format, and store still images in non-volatile memory.

A final addition to the project may include appending an IDE interface for Hard Disk Drive (HDD) storage of the images.

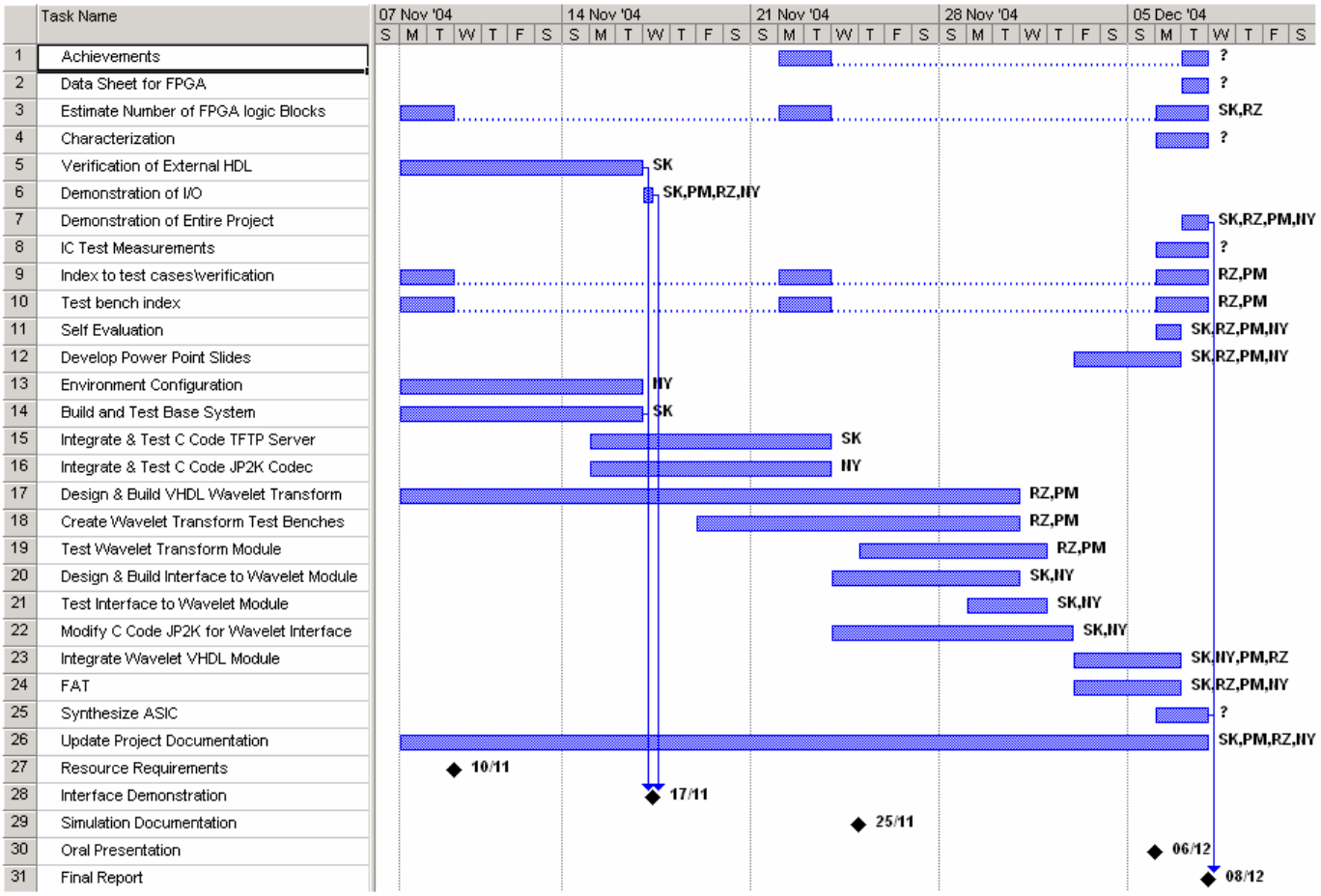
5.0 Parts List

Aside from the Xilinx Virtex II\MicroBlaze Multimedia board (DO-V2000-MLTA) the following items are required:

1. Ethernet Cable
2. RS232 Cable
3. A PC for interfacing with the project via Ethernet/RS232

6.0 Scheduling

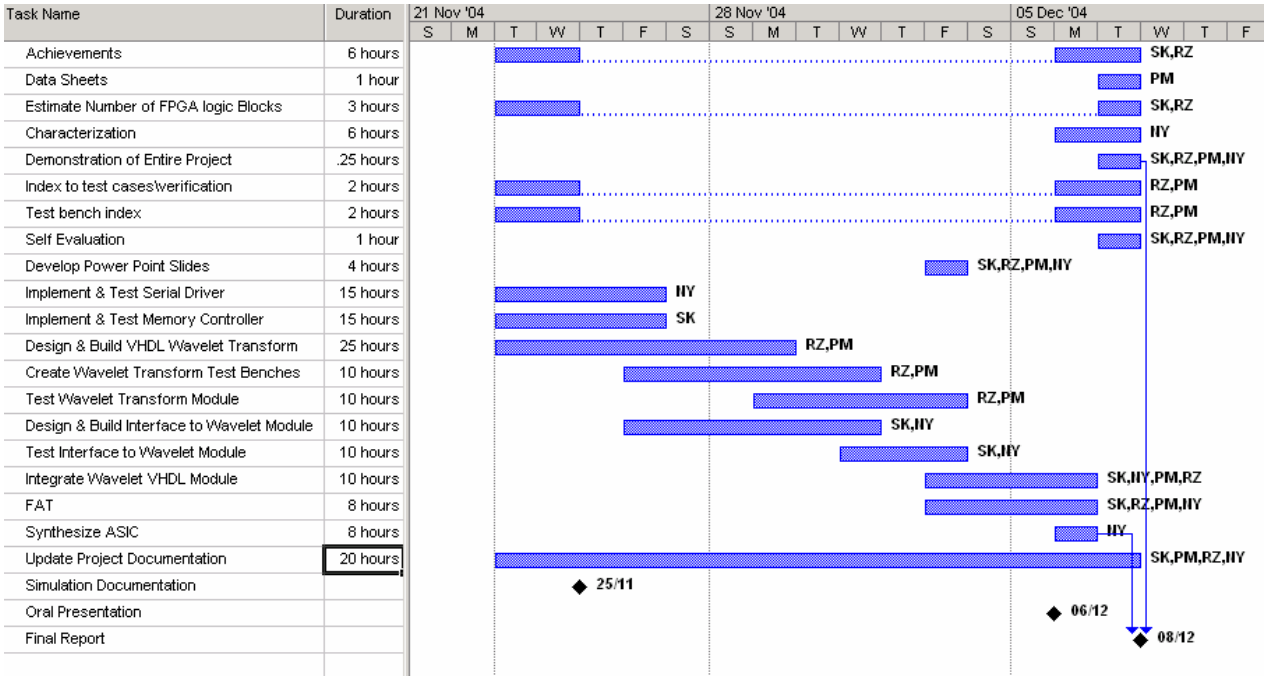
6.1 Old Schedule



The old schedule reflected the optimistic design whereby file server software was to run on the MicroBlaze processor, and file transfers via Ethernet were attainable. The original design also called for C code to perform the initial quantization and compression of the JPEG2000 image in the MicroBlaze. Due to unforeseen problems implementing Ethernet and porting the Jasper source code, the project scope has been changed so that more realistic expectations can be met.

The new schedule reflects the changes in the project. Instead of Ethernet, a simple custom file transfer protocol will be implemented over RS232. A wavelet transform will be performed on the image, and the transformed image will be store in RAM. The resulting transformed image will then be transferred to a PC through the serial interface. The transformed image and will then be compared to a DWT transformed image from MATLAB to test for correctness.

6.2 New Schedule



7.0 Meeting Minutes

Date:	October 22, 2004
Time:	1:00pm to 2:30pm
Location:	ECERF W5-005
Attendees:	PM, SK, NY

Our specified product will be an image server that takes in uncompressed images from an external camera (USB webcam?), and compresses them to JPEG 2000. It stores the files locally (IDE harddrive or compact flash card), and serves them on a network (via HTTP or FTP). Possible applications could be in surveillance/security, or this could be used as the starting point for a streaming video compressor.

The project will be built up in stages.
Stage 1:

- JP2 Codec implemented in software on a microprocessor core.
- Server code for uploading uncompressed images and downloading compressed images via ethernet interface.
- All storage in external ram.
- File system and transfer protocols finalized.
- Tests comparing uncompressed and compressed images (error measurements).
- Speed/timing tests.
- Determine which parts of the codec take the longest.

Stage 2:
- Hardware acceleration of codec added.
- Comparison of images compressed in software and in hardware (to verify accuracy).
- Speed/timing tests.

Stage 3:
- Camera interface added.

Stage 4:
- Harddrive/compact flash storage added.

For the project to be considered successful, we must at least reach stage 2, since stage 1 consists nearly entirely of combining other peoples' IP. Stage 3 and 4 may be implemented in reverse order depending on the difficulty of the tasks (ie, storage might be accomplished easily).

Issues and tasks:
- Choose an Eval Board
- Choose IP cores
- Interface Code
- Network Server - Transfer Protocols, DHCP?
- Off-chip RAM/On-chip ROM
- Codec implementation and acceleration
- Operating System?
- File System
- Camera Interface/USB host mode

Assigned Tasks:
PM:
Research IP cores: microprocessor, RAM, ROM, Ethernet and USB interfaces. Meeting Minutes

SK:
Research source code for a RTOS, FTP/TFTP server, command parser, filesystem. Create the outline for the project.

NY:
Research the Evaluation boards; write up the description of operation including the added/deleted features.

RZ:
Complete the Design Hierarchy. There is JPEG2000 source code and documentation at http://www.ece.uvic.ca/~mdadams/jasper/ . Please determine whether this code supports both encoding and decoding, and whether it supports lossless and lossy compression. Start thinking about how we will use the code.

We will meet on Monday from 11-2 to report our research results and determine how the report will be completed.

Date:	October 25, 2004
Time:	11:00am to 1:00pm
Location:	Cameron Library 3.52
Attendees:	PM, SK, NY, RZ

Task Updates:
SK: I found a couple of embedded RTOS's and some different protocol stacks. I was unable to find any file system firmware or command parsers. Depending on the processor used we may be able to use Linux or a derivative thereof.

NY: There are many evaluation boards available to order through various vendors. The boards cost approximately $100-$200 US. It was hard to find information regarding the evaluation boards that Dr. Elliott may have available.

PM: Recommended that we use the OpenRisk microprocessor from OpenSource. This processor is supported by a simulator and full toolset. Information about Ethernet and RAM controllers can be retrieved from the ECE511\ECE522 application notes. These modules are also available at OpenSource.com.

RZ: Explained the process of JPEG2000 compression.

Decisions Made:
1. Use the OpenRisk microprocessor (contingency is a MIPS open source microprocessor).
2. Use uCLinux for the OS, protocol stack, file system, etc.
3. Add an RS-232 port, as a contingency for the Ethernet Interface
4. Addition of Flash for program storage
5. Only implement the wavelet transformation in hardware; utilize the open source JPEG2000 code and port to the OpenRisk microprocessor.

New Tasks:
SK: Draft up a schedule. Update the specification regarding the software. Update the Minutes.
PM: Find the various VHDL\Verilog modules and start to create the Top Level VHDL Diagram
RZ: Create the interface for the Wavelet Transformation Module.
NY: Complete the generic specification with additions/subtractions. Look into setting up CVS.
All: Get recommendations regarding the evaluation board from Dr. Elliott. Complete the pin-out once the evaluation board is known.

Date:	November 06, 2004
Time:	11:00 am to 1:30pm
Location:	CMC lab
Attendees:	PM, SK, RZ

The following has been discussed or done :
1- We have investigated the board and the EDK software.
2- Discussed the requiremenets for the resources requirement report.
3- Assign tasks.

SK : Build and Test Base system. Prepare new schedule and the resource estimation for some of the modules.
NY : Environment configuration. JP2K interface with the wavelet module.
PM : Go through the specification report and figure out the modifications required. Design of Wavelet module Testbenches.
RZ : Provide more details about the DWT algorithm. Design and build the Discrete wavelet transform module. Provide the software simulation for the DWT.

Date:	November 23, 2004, 4:00 pm to 5:30pm November 24, 2004, 2:00 pm to 3:30pm
Location:	CMC lab
Attendees:	PM, SK, RZ, NY

Discussed issues with memory controllers.
Discussed possible optimizations for 1D DWT architecture.
Reiterated the need for effective communication.

Assigned Tasks:
PM: Look at optimizations for 1D DWT module. Update documentation to include new contributions from members, and proofread documentation.
RZ: Continue work on 2D DWT module. Provide documentation for progress so far on wavelet module.
SK: Continue work on the memory controller. Update abstract, description of operation, architectural description, schedule, and provide information on status of memory controller.
NY: Design a serial protocol for passing information to/from the FPGA, and write C code for the PC and board RS232 transfer interfaces. Provide documentation for serial protocol.

8.0 References

1. Stéphane Mallat , A Wavelet Tour of Signal Processing, 2nd Edition
2. SKODRAS,A.N., CHRISTOPOULOS,C.A., and EBRAHIMI,T.: "JPEG2000: The Upcoming Still Image Compression Standard", (invited paper), Pattern Recognition Letters, Vol. 22, pp. 1337-1345, Oct. 2001.
3. Gnavi, Stefano, Barbara Penna, Marco Grangetto, Enrico Magli, and Gabriella Olmo: "Wavelet Kernels on a DSP: A Comparison between Lifting and Filter Banks for Image Coding," EURASIP JASP 2002:9 (2002) 981-989.
4. Daubechies and W. Sweldens. Factoring wavelet transforms into lifting steps. J. Fourier Anal. Appl., 4(3):245-267, 1998.
5. M. Grangetto, E. Magli, G. Olmo., - "Finite precision wavelets for image coding: lossy and lossless compression performance evaluation.", ICIP 2000 - IEEE International Conference on Image Processing, Vancouver, Canada, (FILE: 1544.pdf).
6. "Micro Blaze Soft Processor Core," Retrieved from www.xilinx.com/microblaze, on October 27, 2004.
7. "Xilinx Multimedia Board," Retrieved from www.xilinx.com/products/boards/multimedia/, on October 27, 2004.
8. MicroBlaze RISC 32-Bit Soft Processor, August 21, 2002, page 1
9. LMB Block RAM (BRAM) Interface Controller, DS452 (v1.6), January 26, 2004, page1
10. OPB External Memory Controller (EMC), DS421(v1.7.1), December 15, 2003 , page 1
11. OPD UART Lite, DS422(v2.5.1), July 29, 2004, page 1
12. OPB Ethernet Media Access Controller (EMAC), DS435 (V1.23.2), January 9, 2004, page 1
13. On-Chip Peripheral Bus v2.0 with OPB Arbiter (v1.10), DS401 (v2.6.3), January 21, 2004, page 1
14. Microprocessor Debug Module (MDM), DS450 (v1.1), January 15, 2004, page 1
15. Block RAM(BRAM) Block, DS444 (v1.0),July 22,2003,pag1
    "memory loads and dumps from/to text file", extracted from www.informatik.uni-hamburg.de/ vhdl/ models
16. Local Memory Bus (LMB) v1.0, DS445 (v1.4) January 24, 2004, page 1
17. MicroBlaze Hardware Reference Guide, March 2002, page 13
18. MicroBlaze Software Reference Guide, April, 2002, pp 69-80
19. Digital Clock Manager (DCM) Module, DS(v1.1), December 19, 2003, page 1
20. "MiniUart Revision 1.2", Retrieved from http://www.opencores.org/cvsweb.shtml/miniuart on December 5, 2004
21. "UCLinux Support Site", Retrieved from http://uclinux.openchip.org/ on October 20, 2004.
22. "MicroBlaze ucClinux Project Home Page", Retrieved from http://www.itee.uq.edu.au/~jwilliams/mblaze-uclinux/ on October 15, 2004.
23. "The JasPer Project Homepage", Retrieved from http://www.ece.uvic.ca/~mdadams/jasper/, on October 27, 2004.
24. "RFC1055", Retrieved from http://www.faqs.org/rfcs/rfc1055.html on November 15, 2004.
25. "Multimedia Board Design Examples", Retrieved from http://www.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?sSecondaryNavPick=Design+Tools&key=micro_blaze on November 23, 2004.
26. M.Martina, G.Masera, G.Piccinini, M.Zamboni, "A VLSI Architecture for IWT (Integer Wavelet Transform)," Proceeding of 43rd MidWest Symposium on Cicuits and Systems, Lansing MI, USA, August 2000.

27. G.Kuzmanov, B. Zafarifar, P. Shrestha, S.Vassiliadis, "Microarchitectural Extension for Lifting-based DWT," Proc. "PROGRESS 2002: Networked Embedded Systems All Over the Place", Utrecht, The Netherlands, October 2002, pp. 108-116.

28. Michael Adams, Faourzi Kossentini, " Reversible Integer-to-Integer Wavelet Transforms for Image Compression: Performance Evaluation and Analysis," IEEE Transactions on Image Processing, VOL. 9, NO. 6, JUNE 2000.

29. Xess Board SDRAM and VGA driver, obtained from www.xess.com/downloads.

9.0 Feedback and Comments to Instructor

~~October 28, 2004~~

- ~~The team has concerns regarding access to the Multimedia boards; the first stage may require a lot of one-on-one time with the hardware. Also, some of the team members do not have access to ECERF outside of normal operating hours.~~

November 24, 2004

- Access to the multimedia board is still a design bottleneck. (Sean) has soldered a connector for the JTAG interface, but we still don't have power for the JTAG interface (PS2 connector). Ideally, another computer with Windows could be setup in the lab. Alternatively, someone could by a PS2 extension cable or a USB to PS2 conversion kit ☺. Are any of the graduate students in our class able to drag their PC's into that lab?

10.0 Datasheets

See Attached.

11.0 Results of Experiments and Characterization

11.1 Experiments results (speed up vs. area trade-offs)

The initial design for the 1D Wavelet transformation operated sequentially: data was read in from memory, processed, and then written back to memory. In the first stage, a modification has been made to make use of the parallel nature of the algorithm. The version that is now being developed will pipeline these operations to increase the throughput. However, since we may not have access to dual-port RAM, we still may need to operate reads and writes sequentially. In addition, more speed may be obtained by operating on more pixels at one time in parallel.

The memory access times determine the absolute speed limit, as data cannot be processed faster than it is read in and subsequently written out. Depending on the type of memory controller we are able

to use, this may be the bottleneck of our design. As a result, as many values as possible are stored in registers to keep our memory accesses to a minimum.

Key features of the final design :

- Parallelism is included to be able to operate on more pixels at one time.
- Extra registers are used to allow data re-usability and minimize memory accesses.
- With the same 2D-DWT controller, the 1D-DWT was further optimized by adding pipeline stages.
- In order to minimize execution time, coefficients were calculated simultaneously with memory accesses.
- Four pixels were read in, and then four pixels were written out with the calculations done during the next set of reads.
- By using dual port RAM, reads and writes could be done simultaneously. In this manner, execution time of the 1D transform is only a few clock cycles more than the time it takes to read through all pixel values.

11.2 Numerical Simulations

This section outlines the preliminary steps taken towards the complete verification of the hardware system. Matlab was used to perform simulations for the 2D-DWT lifting scheme algorithm using 5/3 Le Gall filters. A number of of test images have been used to validate the algorithm before reducing it to hardware. An Inverse DWT step was also used to verify the perfect reconstruction of transformed image. Matlab routines for software simulation and for generating image text files for the Vhdl test bench are provided in appendix (D). Figures 20 and 21 show the forward DWT simulation results on a test image.

(a) original (b) 1-level wavelet decomposition

Figure 20. Forward 2D-DWT example

11.3 Design verification and Simulations

Final Design Flow and Verification:

This section presents our final design flow and verification strategy. Results, comparisons, and analysis are provided with each stage. Figure 24 shows the overall design flow, starting from passing a test image text file to the VHDL simulator, then simulating the DWT module. The output image is then exported back the Matlab for further analysis.

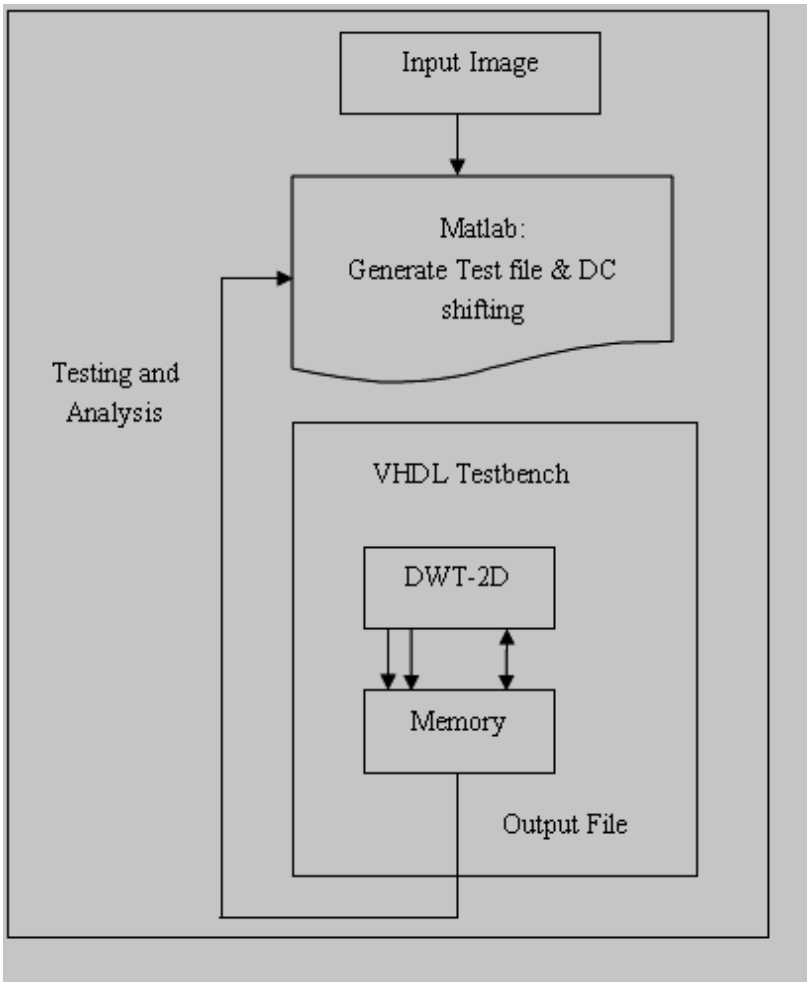


Figure 24. The overall Design flow and verification strategy

Next , we details the simulation results and performance analysis at different stages of the project.

11.3.1 Testvector-level Simulations for 1D-DWT

Our first experiments were performed on the basic 1D-DWT module. The module was instantiated with input test vectors and the resulting output vectors were comapared to the results obtained from the Matlab model of the code, for the same test vectors. Table 2 summarizes some of the results.

Table 2.Preliminary evaluation for the hardware vs. software output for 1D-DWT

Sample software vector:	Sample hardware vector
[-5 -24 26 23 25 23]	[-5 -24 26 23 25 23]
[-3 -24 9 33 29 24]	[-3 -24 9 33 29 24]
[-49 -46 -61 -76 -86 -73]	[-49 -46 -93 -76 -86 -105]
[-73 25 -95 -121 -122 -108]	[-105 25 -127 -121 -122 -108]

11.3.2 file-Level Simulations for 1D-DWT

Our experiments next extends to apply the basic 1D-DWT module to a compelete image file. Waveforms were obtained at different design stages of the 1D-DWT. Figure 26 shows the waveforms for the final 1D-DWT module after adding the pipeline stages.

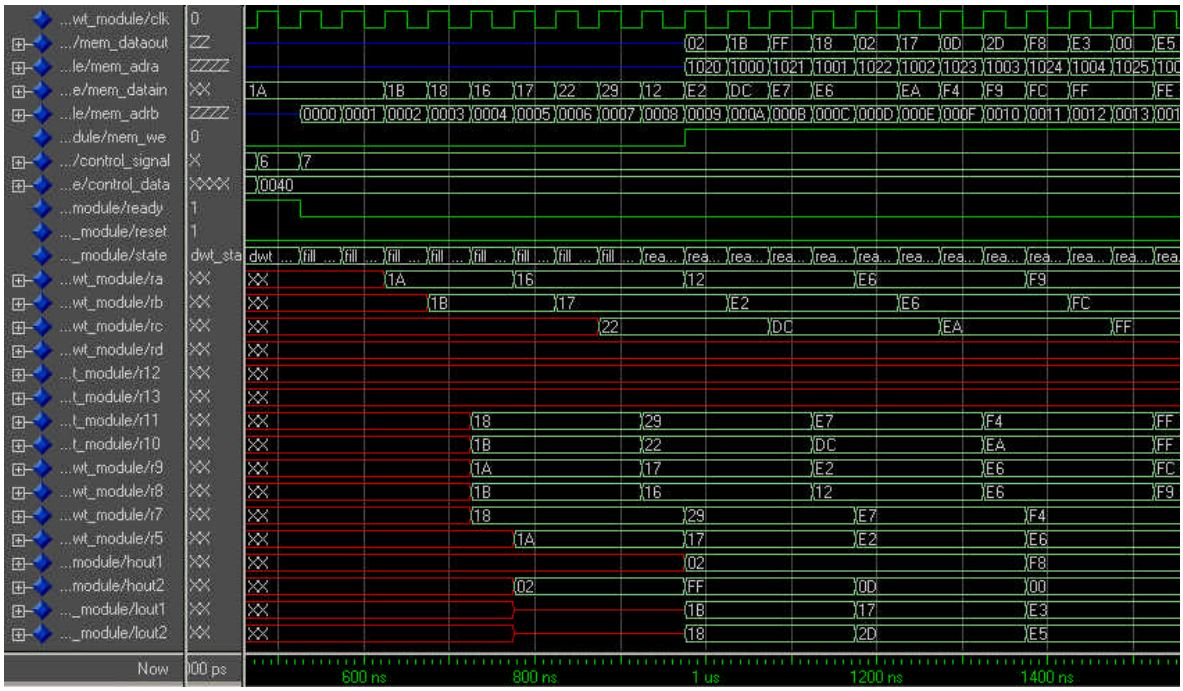


Figure 26. Waveform for the 1D-DWT (with pipelining)

Experiments were next extended to perform the file level simulation for the compelete 2D-DWT module. Two-dimensional discerete wavelet transform is simply performed by performing the 1D-DWT module twice; on the rows and then on the columons. First, waveforms was examined to verify the correct operations of the control signals. Second, the resulting transformed text file was read by the Matlab for our final stage analysis.

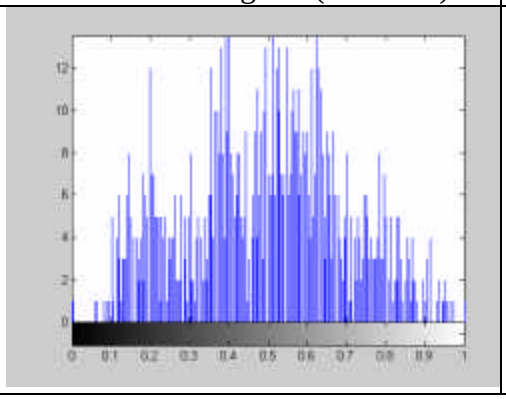
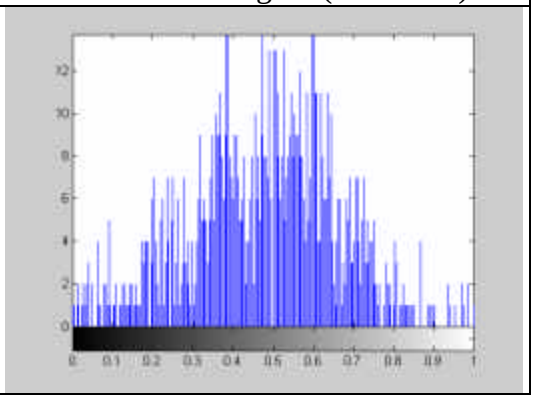
11.3.4 Compelete Performance Analysis

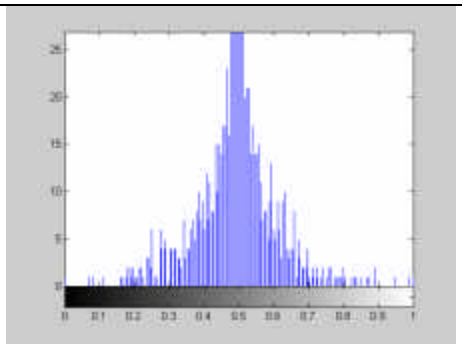
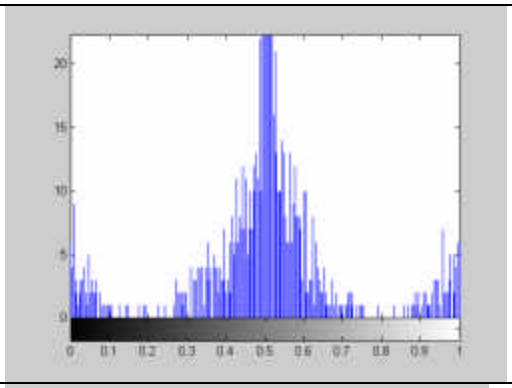
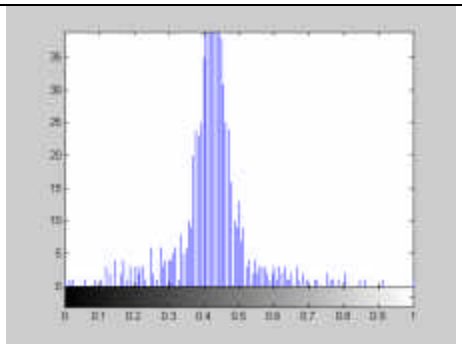
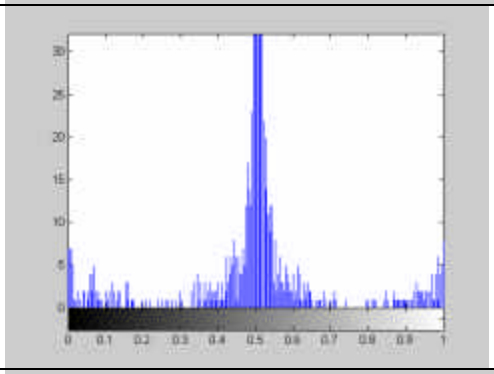
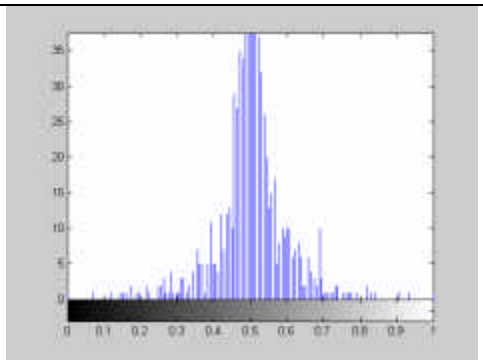
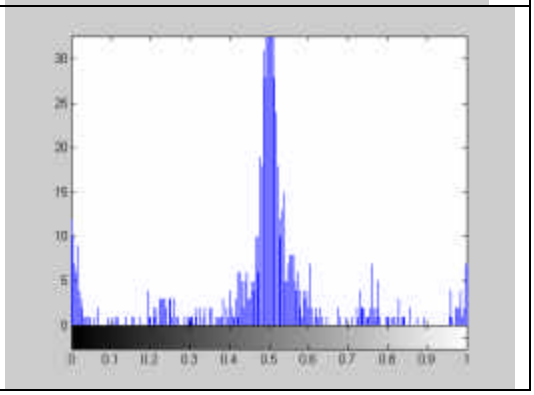
First Analysis was conducted to evaluate how much the rounding error affects the overall performance of the hardware implementation with a data depth of 8 bits. This was achieved by comparing the output image from the VHDL simulator against that obtained by the software simulation, where more precision for the intermediate stages were allowed. Table 3 shows preliminary sub-bands comparisons using test vectors. Table 4 shows the histogram of different sub-bands for the hardware and software outputs. There is a high degree of similarity, with some errors introduced at the image boundaries.

Table 3.Preliminary evaluation for the hardware vs. software output for the 2D-DWT

Sub-band	Sample software vector:	Sample hardware vector
LL	[27,24,24,45,-29,-27,-27,-12,-3,0]	[27,24,24,45,-29,-27,-27,-12,-3,0]
LH	[-12, 91,-14, 14,-16,113,-19, -119]	[-12,91,-14,14,0,1,-19, -7]
HH	[-8, -9, -6,3,-2, 0,2,-3,4,1]	[120,-9,-6,3,-2,0,-126,109,4,-111]

Table 4. Histogram of different sub-bands for software vs. hardware output

	Sub-band Histogram (Software)	Sub-band Histogram(Hardware)
LL		

LH		
HL		
HH		

Moreover, the Peak Signal-to-Noise Ration (PSNR) for the corresponding sub-bands. Results. A considerably high PSNR has been obtained, as shown in table 5.

Table5. PSNR for various sub-bands

Sub-band	LL	LH	HL	HH
PSNR	47.7993	47.9702	47.9960	47.9756

11.4 Synthesis and Implementation results

Initially, we intended to target our module for the Vertex2 chip on the Xilinx Multimedia board. However due to unforeseen design problems, we attempted to perform the implementation on Xess board with the Spartan2 Chip. Thus, we have targeted our design towards two FPGA

devices; namely Vertex2 and Spartan2. The synthesis results of our DWT module are summarized in tables 6 and 7. Experiments have also shown a speed up of over 61 times has been achieved versus a MatLab compiled C code implementation on a Pentium IV 1.8 GHZ processor.

Table 6. Synthesis results for the parallel version (no pipelining)

Target Device	: Virtex-II XC2V2000-FF896	Spartan: XC2S100
Number of Slices	763 out of 10752 7%	769 out of 1200 64%
Total # of LUTs	1385 out of 21504 6%	1381 out of 2400 57%
Number of FFs	427 out of 21504 1%	438 out of 2400 18%
Maximum Clock Frequency	108.962MHz	61.229MHz
Power consumtion (mw)	562	232

Table 7. Synthesis results for the final pipelined version

Target Device	Vertex 2	Spartan: XC2S100
Number of Slices	255 out of 10752 2%	268 out of 1200 22%
Total # of LUTs	473 out of 21504 2%	482 out of 2400 20%
Number of FFs	149 out of 21504 1%	153 out of 2400 6%
Maximum Clock Frequency	112.931MHz	59.284MHz
Power consumption (mw)	562	111

11.5 Achievements

November 24, 2004:

System level

A significant amount of time was spent trying to get the Ethernet Interface working. It was very much anticipated that this interface would work out-of-the-box. Xilinx also provides a web\ethernet example for the V2000 Multimedia Board that is built for EDK version 3.2. This version does not build for EDK 6.2 due to the use of deprecated and un-supported cores and changes in syntax. After careful examination of the web example it was successfully ported (it synthesized) to EDK version 6.2. Unfortunately, the Ethernet part still did not function. Alongside the troubleshooting of the web example a comprehensive search on the internet regarding the EMAC IP was conducted. Several people have tried to get this core to work with no avail. There was one success story from the University of Mexico that used the EMAC core

with the Vertex IIp and the PowerPC core. They indicate that the XilNet source code must be modified to get the interface to work.

From the web server example exercise, a few small achievements can be surmised:

1. The Xilinx Platform Studio (XPS) was well learned.
2. It was learned how to import custom IP modules into an XPS project.
3. Source code was located and run on the external RAM using the OPB ZBT RAM controller.
4. As an initial step towards mitigating the risk of not having an Ethernet interface, Dial-up Networking over SLIP was further explored. A polled serial driver was created and superficially tested.

Discrete Wavelet Transformation:

The basic 1D-DWT block was designed and simulated. Efforts were made to speedup the design. The achievements, up-to-date , can be summarized:

1. Design and simulation of a 1D-DWT module.
2. Modifications were made to add parallelism to the initial design.
3. Efforts are made to obtain a pipelined version of the design.
4. For the sake of further analysis matlab functions were designed:
   a- A matlab routine that reads the text file, resulting from the modelsim and converts it into image.
   b- A matlab code for the DWT.

There are still a number of open problems , the group is working on:

1. Developing a pipelined version of the design.
2. Extending the design to the 2D-DWT.
3. There is a scaling problem between the image obtained by the hardware and that is generated by the matlab code.
4. The current 1D-DWT should be extended for the 2D-Dwt.

December , 10, 2004

The main achievements in this project can be summarized as follows:

At the Design level:

1. A Lifting based DWT (Integer-to-Integer) unit was implemented in FPGA, namely Xilinx VertexII FPGA and Xilinx SpartanII FPGA.

2. Maximum Clock Frequency of 112 MHZ was achieved, with throughput of over 400 pixel/s, using techniques as parallel operation, data re-usability, and pipelining. The Design is scalable to Comply with the specification of the JPEG2000 standard, thus allows trade off between rate and distortion, and has lossless compression ability.
3. A speed up of over 61 times has been achieved versus a MatLab compiled C code implementation on a Pentium IV 1.8 GHZ processor.

At the Integeration and Demonstration level:

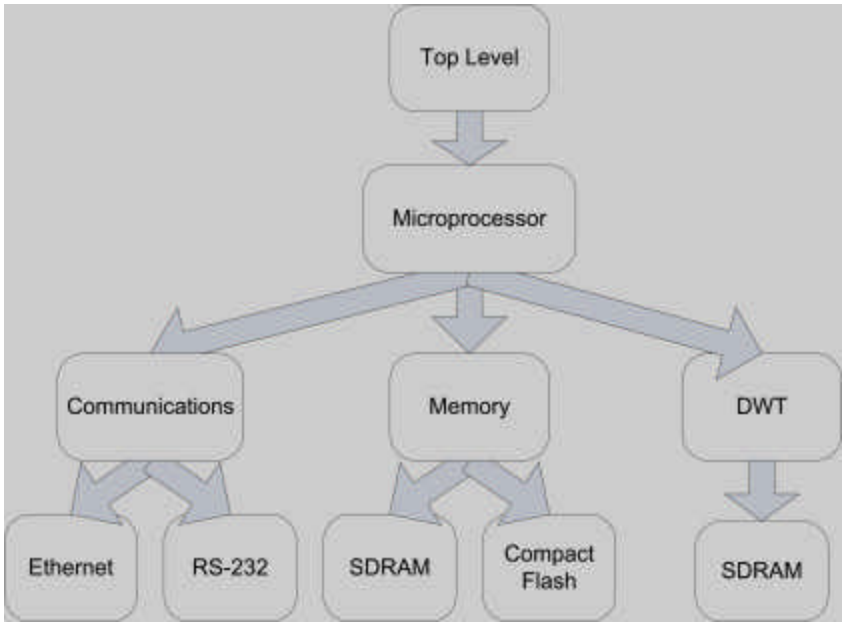
Two systems have been integerated, these are:

1. For the Multimedia Board: The image to be transformed was pre-initialized in the on-chip Block RAM. The resulting transformed image is sent to a host PC via the RS232 serial port.

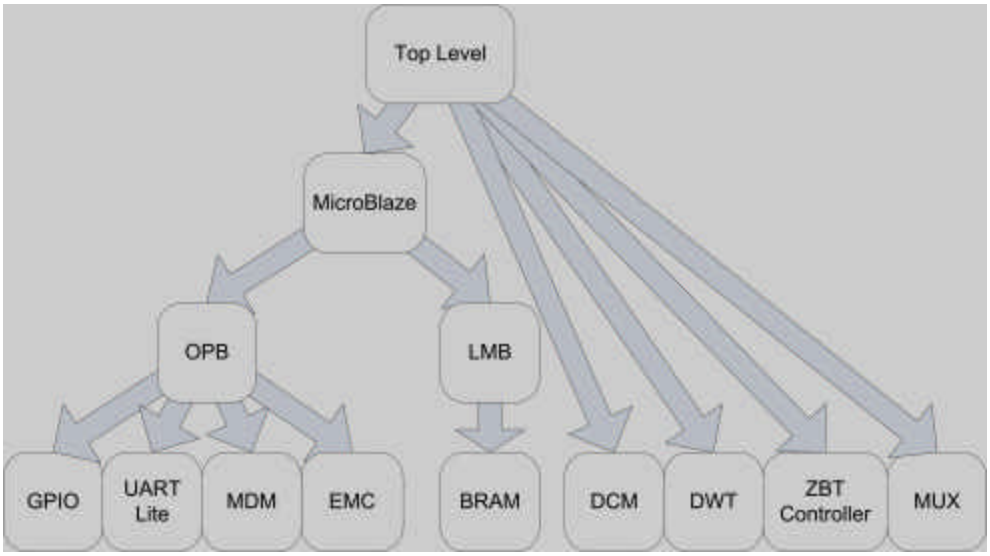
2. For the Xess Board : The image to be transformed was sent via the parallel port to the on-board SDRAM. The discrete wavelet transform was next performed and the resulting transformed image is sent to VGA display.

12.0 Design Hierarchy

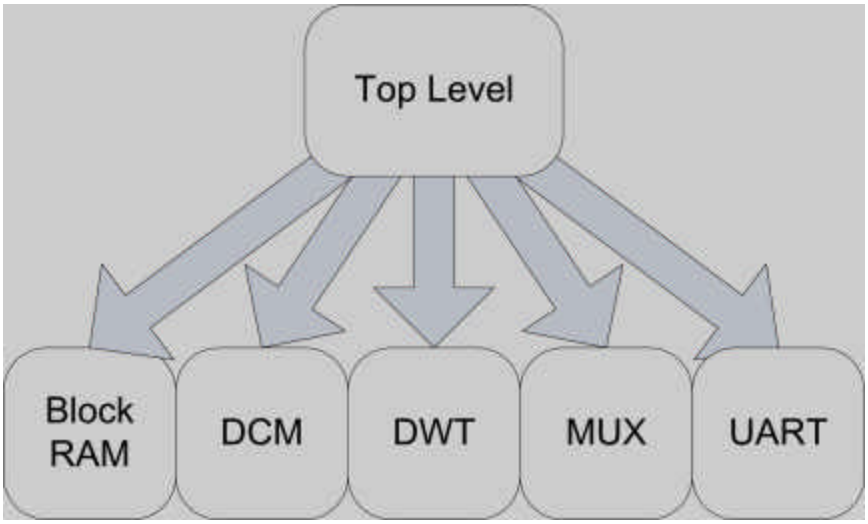
Stage 1.



Stage 2.



Stage 3.



13.0 Index to VHDL Packages and Code

1. system_stub.vhd : top-level system design (compiled)
2. blockram_cg.vhd : VHDL wrapper for LogiCore block RAM (compiled and tested)
3. clockgen.v : Clock generator for the multimedia board (compiled and tested)
4. miniUART from opencores.org (compiled and working with some bugs)

1. miniUART.vhd : Top level UART
2. uart_lib.vhd : Declarations for the UART
3. TxUnit.vhd : Transmit Unit
4. RxUnit.vhd : Receive Unit
5. clkUNIT.vhd : baud rate generator

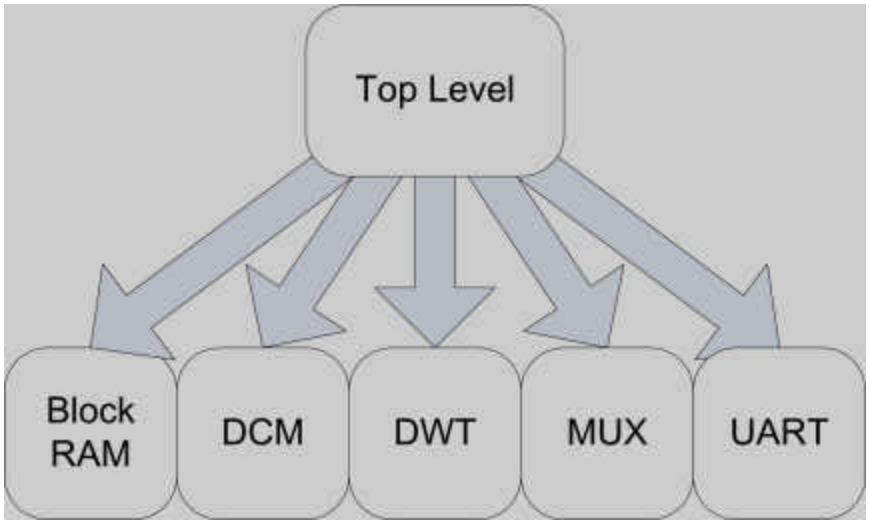
5. DWT2D Module
   1. dwt2d.vhd : Top level for the 2d-dwt (simulated)
   2. dwt1d.vhd : Performs the 1D-DWT (simulated)
   3. dwt2d_tb.vhd : testbench file for testing the 2D-dwt (simulated)
   4. dwt_package.vhd : package containing the constants, functions and procedures. (simulated)
6. Xess board-based System:

   1. toplevel_xess.vhd : Top level for the 2d-dwt (compiled and tested with errors)
   2. dwt2d.vhd : Same as in (5) (compiled and tested)
   3. SDRAM.vhd : sdram controller (compiled and tested)
   4. vga.vhd : vga driver (compiled and tested).

The vhdl design code is provided in appendix (A), while the matlab codes, testbench and C codes is in appendix (B). Appendix (C) contains the synthesis reports.

14.0 Top Level VHDL Design

The current top-level design consists of a miniUART component, a DWT2D component, and a block RAM component. The block RAM is initialized through a file. The DWT performs the calculations on the data and then the resulting memory is transmitted out through the serial port, refer to stage 3 of the design heirarchy.



As for the top-level design, for the xess board-based system, the design consists of an SDRAM controller component, a DWT2D component. The SDRAM is loaded with the image

from the parallel port. The DWT performs the calculations on the data then the resulting memory content is displayed on the VGA display using VGA driver.

15.0 Index to Simulations

Figure 24 shows the overall flow of our design strategy. Different simulations and tests were performed to verify the correct operation of the module under test and evaluate its performance. A number of testbenches and matlab routines have been written. These are found in appendix B, and are indexed as follows:

1. 1D_DWT_testvector_level testbench : Test vectors were entered from using force command.
2. 1D_DWT_testbench : The 1D-DWT module (operating on image file)
   2.1 dwt_mem.vhd : Apply the 1D-DWT over the whole text file.
   2.2 toplevel_1d_tb.vhd : toplevel file for the testbench of the 1D-DWT
3. 2D_DWT_testbench : The 2D-DWT module
   3.1 toplevel.vhd : toplevel file for the testbench for the 2D-DWT
4. Matlab routines for analysis and comparison:
   4.1 dwt_1d.m : performs 1D-dwt
   4.2 dwt_2d.m : performs the 2d-dwt
   4.3 filter_even : even filter step.
   4.4 filter_odd : odd filter step
   4.5 PSNR.m : calculates the psnr and histogram for the image obtained by the software and hardware implementation.
5. Server.c : C file created for the serial port communications with the Multimedia board.

16.0 Index to Synthesis

The timing and area synthesis reports are found in appendix (C) and are indexed as follows:

1. dwt_2d.syr : Main DWT_2d module synthesis results
2. top_level_MM : Compelete system for the Multimedia board.
3. top_level_xess : Top level for the xess board.

17.0 Schematics

The project does not require any circuitry beyond that provided by the multimedia board.

Appendix B VHDL Packages and Code

Appendix C Testbenches ,C, and Matlab Functions