Texas A&M University
Department of Electrical Engineering

ECEN 654

H.264 Video Decoder Design

Prepared and Revised by
Yoon Seok Yang
Dr. Gwan Choi
Rajballav Dash

October 2010

CONTENTS

1. Laboratory Policies and Report Format

This lab assignment has specific deliverables which are listed later in this manual. All the hardcopy report and electronic submission are due at 5:00pm on the due date. Refer to grading policies to see scoring penalties in case of late submissions. These reports should be complete so that someone else familiar with logic design could use it to verify your work.

This project is to be done individually and each individual will be graded separately. Many of you will be involved in the design of similar blocks so there will be performance credits based on the completeness of your design with respect to others designing the same block. A design submitted by an individual should be unique.

For any doubts and suggestions regarding the project contact me.

The lab report format is as follows:

1. A neat thorough report must be presented to your TA on time. Reports should be submitted on 8.5" x 11" paper, printed on both sides. Your report is a professional presentation of your work in the lab. Neatness, organization and completeness will be rewarded. Points will be deducted for any part that is not clear.

2. Each report should contain the following sections:
a) **Cover Page:** Name, ELEN 654, Title, TA's name and date.

b) **Objectives:** Describe the objective of the work that you are presenting in the report. You should also outline topics in your report in this section. There should be one or two sentences per objective.

c) **Design:** This part contains all the steps required to arrive at your final design. This should include diagrams, tables, equations, explanations, hierarchies, etc. This section should also include a clear written description of your design process. Simply including a Verilog file/synthesized design is not sufficient.

d) **Results/Simulations:** Submit all your results and simulations that you did to verify your design in this section. Hardcopies are required if asked for in the deliverables section.

3. Any design issues/help needed should be notified to the TA as soon as possible which is prior to submission dates. Also, any help needed from the TA should be well inside submission deadlines.

4. You are guaranteed a computer and workspace in 213A Zachry only during your lab period. If you need to work on your design at a time other than your regularly scheduled lab period, the lab in 213A is open with code access throughout the week. However, if another lab section is in progress, ask the TA if he/she has any open lab stations.

5. Attendance at your regularly scheduled lab period is required. An unexpected absence will result in loss of credit for your lab. Your lab instructor may permit rescheduling if you arrange for the change ahead of time.

2. Detailed Grading Policies and Submission Rules

A. Hardcopy Report Submission

1. A neat thorough report must be presented to your lab TA at the due date by 5:00pm in the afternoon.

2. If you need your report during the lab, then you should make another copy of your report before submitting your report to the TA.

3. Penalty for the late report: If you submit your report after 5:00pm on the due date but before 11:59pm on the same date, your report will receive only 90% credit. If you submit your report next day before 5:00pm, you will receive 80% credit. If your report is 2 days late (before 5pm), it receives 50% credit. Further delay results in 'zero' credit.

B. Electronic Submission

1. All files should be zipped together (.zip/.rar only) and submitted in the form of email to the lab T.A. before 5:00pm on the due date.

2. The zipped folder should be sufficient and missing files will incur score penalties. Any files required to be added after the e-mail submission will incur a penalty of 5% per file or late penalty whichever is higher.

3. Each e-mailed folder "must" include the following files:

a) Read Me (5 points): Describes how to **run your code in one line** and brief description of each of your design file.

b) Test Bench File (Filename: TB_<TOP>.v) which you used to test your design (10 points). Each Input block should be well commented and separated by white space.

c) Top Module Verilog File (Filename: <TOP>.v) which includes all your design hierarchies. Also, the folder should contain all other Verilog files required to simulate your design (20 Points).

d) Results like waveforms or simulation results required for your design based on your test-bench or from based on the test cases provided by the TA.(15 Points points);

Your Verilog codes should be well commented and understandable. Also, don't forget to mention your name/e-mail address on top of the file that code. Otherwise, up to 5% points will be deducted due to lack of adherence to the file submission rules.

4. Penalty for the late email submission: If you submit after 5:00pm on the due date but before 11:59pm on the same date, you will receive only 90% credit. If you submit next

day before 5:00pm, you will receive 80% credit. If your submission is 2 days late (before 5:00pm), it receives 50% credit. Further delay results in ‘zero’ credit.

3. H.264 Decoder Overview

Introduction:
The H.264 encoding block diagram and algorithmic flowchart is given in the decoder design power point provided along with other files of the project. Please try and understand each block of the encoder before you start to design the decoder. The decoder block diagram is given in Figure 3.1. This block diagram shows the different algorithmic blocks of the H.264 decoder. The complete specification and details of the H.264 decoder can be found from the specification document (Standard_T-REC-H.264-200711.pdf) provided in the project folder. Each block of the decoder is shown in Figure 3.1. As can be seen from below, the decoder consists of the following blocks: NAL decoding, entropy decoding, inverse quantization, inverse transform, intra-frame prediction Block, Inter-Frame prediction block and de-blocking filter. The details of each of these blocks is given separately in block level tutorials which is required to understand to each block separately for implementation purposes. Note that each of these blocks interacts with other blocks. Hence, detailed understanding of date flow among different blocks needs to be understood before implementing a block so that each of the interactions are take care of in the design.

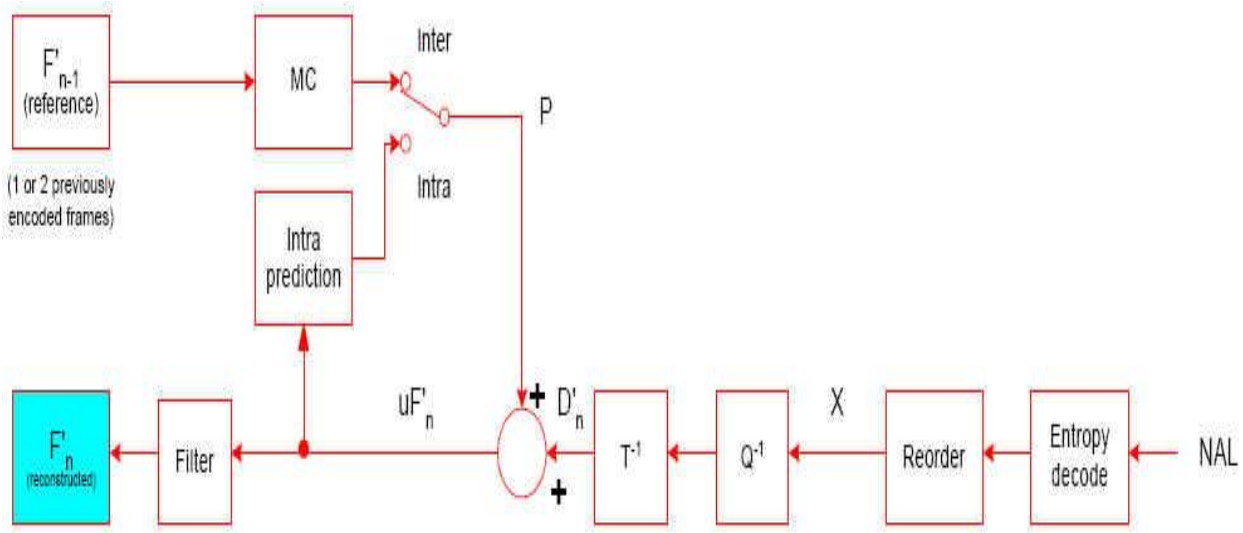

Figure 3.1: H.264 Decoder Block Diagram

Decoder Dataflow Chart:
The flowchart of the complete decoding algorithm is given below in Figure 3.2. The input bit stream goes through a NAL decoding stage which basically inputs the bit stream in a specific format to the decoder. The NAL decoding routine can be seen from the reference C code provided or from the specification manual. The NAL decoded bit stream is fed to

the Entropy decode and Reorder unit which decompresses the bit stream and rearranges the frame into correct order (inverse ordering). This goes through an intra-frame prediction block whose output goes through an inverse quantization and inverse transform block. Similarly, reference frame (previously decoded frame) and current frame is used to for inter-frame prediction which then passes through inverse quantization

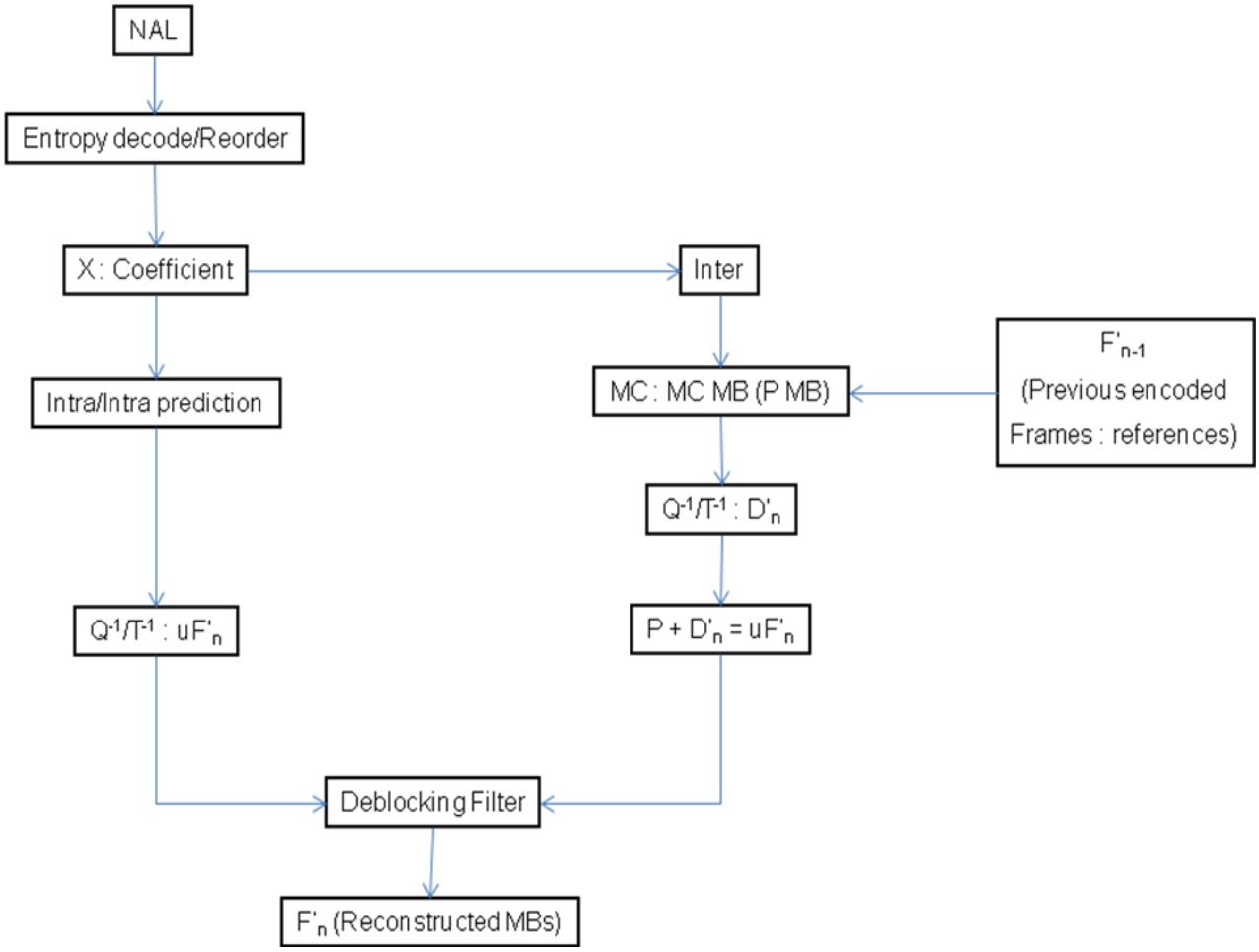

Figure 3.2: H.264 Algorithmic Flow Chart

and inverse transformation blocks. The output then goes through the de-blocking filter and reconstructed. We will discuss the decoder block in detail later in this manual.

Frame Resolution:

The decoder frame resolution and data size is shown in figure 3.3. The example below is a QCIF image of 176x144 resolution. A decoder frame is divided into 16x16 macro-blocks. Hence, there are 11x9=99 macro-blocks. Each macro-block contains 4 luminance blocks and 1 block each of chrominance (Cb and Cr). Hence, macro-block has 6 sub-

blocks. Each sub-block has 8x8 pixels with each pixel as eight bits. Hence, each macro-block has 99x6x8x8 pixels and 99x6x8x8x8 bits as shown below in Figure 3.3.

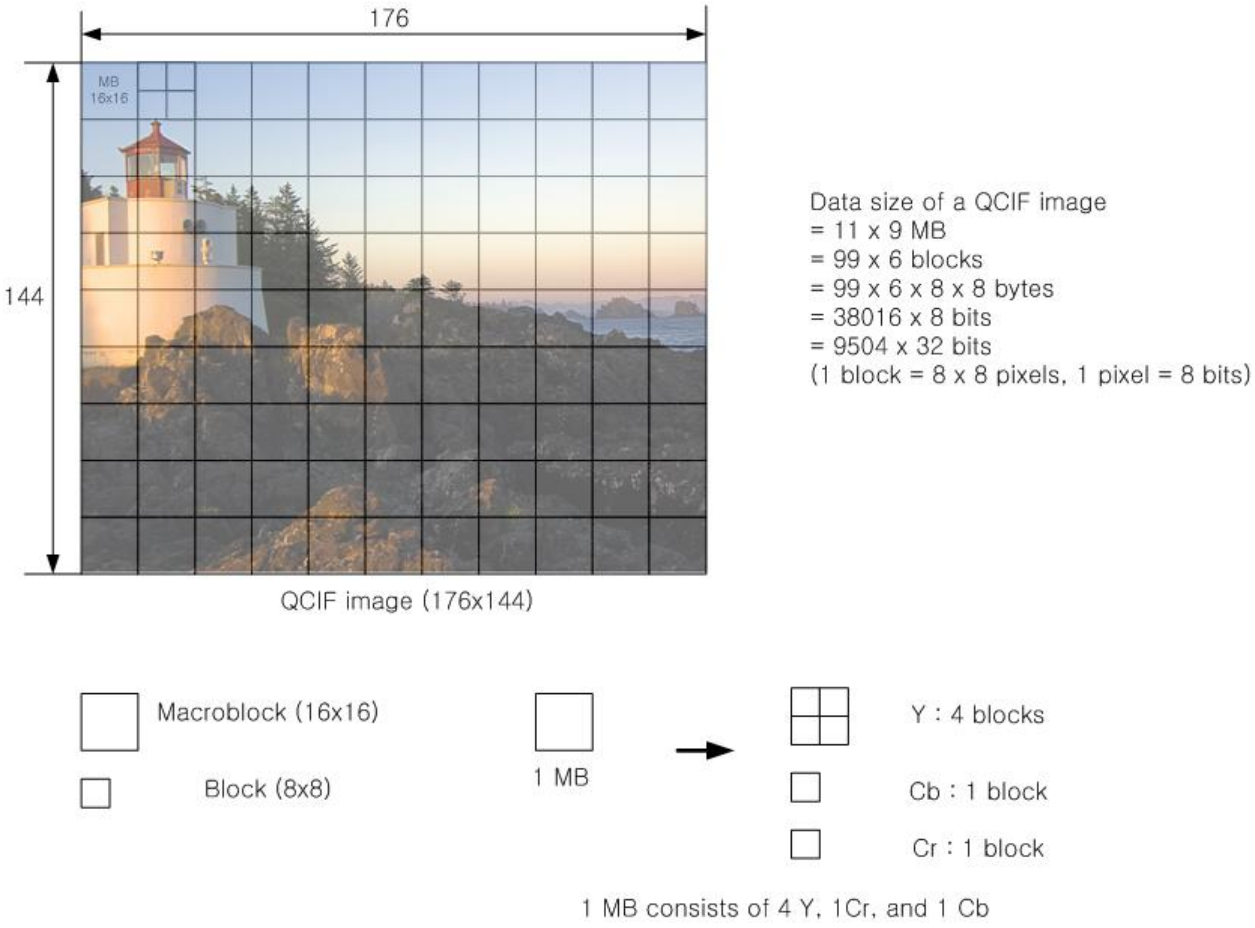

Figure 3.3: H.264 Decoder Frame Resolution

H.264 Decoded Data Flow:

The H.264 decoder showed above in Figure 3.4 shows a detailed data flow between different blocks. The bit stream then goes through NAL decoding/Syntax Parsing which is then fed to the entropy encoding block. Context Adaptive Variable Length Coding is done which goes through the inverse scan and inverse quantization blocks before being inverse transformed and then summed with the output from the inter/intra-frame prediction. This is then passed through a de-blocking filter block which with the help of the decoded frame buffer control writes to the decoded frame buffers. For more details on the overview of the decoder reference files has been provided in the Tutorial>Introduction and Overview Section.

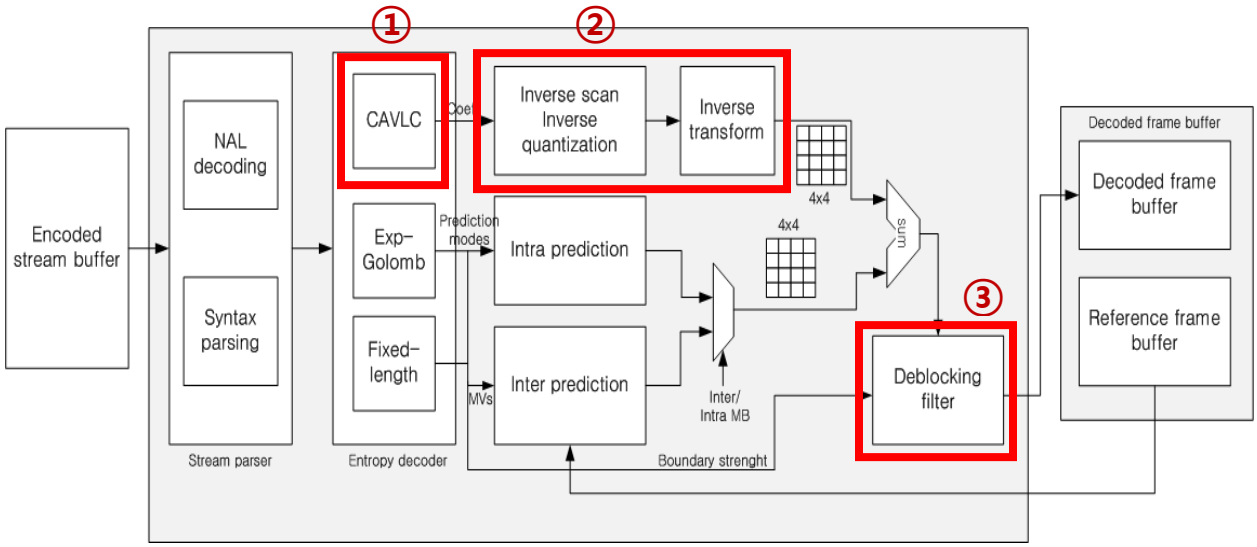

Figure 3.7: H.264 Decoder Data Flow Diagram

4. H.264 Blocks: Specs and Design Guidelines

This document doesn't discuss details of each block of H.264 encoding and decoding process. Students can choose one of three blocks, 1)CAVLC, 2)Inverse scan, inverse quantization and inverse transform, and 3)De-blocking filter for their design implementation as shown in Fig.3.7. When you select your design block, you should study and understand the functional operation of the design block. You can analyze the behavior of the block using reference C code step by step. You might print out input and output reference data from the C model to create test benches. You are responsible to deliver your final project report including verilog source files, test benches, and detail discussion of your design block.

4.1 Context Adaptive Variable Length Coding

Context Adaptive Variable Length Coding (CAVLC) is an entropy encoding technique used in H.264 decoder to compress bit streams in the residual block data while other elements are coded using Exp-Golomb codes.

4.1.1 Suggested Reading

The CAVLC process can be understood in detail from the following documents:

i. **h264_Context_Adaptive_Variable_Length_Coding.pdf (Doc1):** Read for CAVLC Encoding/Decoding details. (You can skip the Exp-Golomb Coding part as it doesn't contribute to the understanding of CAVLC but is necessary to understand the H.264 entropy encoding techniques.)

ii. **Standard_T-REC-H.264-200711.pdf (Doc2):** Section 9.2 (Page 210) describes the CAVLC specifications. Refer to tables/algorithms provided here for any details that are omitted in the first document.

4.1.2 Block Description

In this block, you are required to design the encoding block and the decoding block. The encoding block compresses the bit streams at the encoder and the decoding block decompresses the bit streams. At the encoding side, the CAVLC module take a 16x16 macro-block as input along with some control signals (if required) and gives a compressed bit stream as output. The decoder on the other hand takes the compressed bit stream as input and decompresses it to get back the original macro-block. Please refer to examples described in Doc1 to understand details of the encoding/decoding process. Some of the tables referred to in this document can be found in Section 9.2 of Doc2. These modules should be synchronized to a clock and should have a reset signal to initialize memory elements.

4.1.3 Important Things to Note

There are some important things to note in these documents:

i. In Doc1, Page 4 the description to find out the encoding of levels for non-zero coefficient is not sufficient and assumes the knowledge of

Level_VLCx tables which are not provided. Hence, please refer to Page 213, Section 9.2.2 of Doc2 to understand fully how to encode levels of the remaining non-zero coefficients.

ii. Doc1/Doc2 refers to frequency components in macro-blocks. Frequency components decrease from left to right and top to bottom in a macro-block.

iii. Refer to examples in Doc1 for the decoding process which follows from the encoding algorithm.

4.2 Deblocking Filter

A filter is applied to every decoded macro-block in order to reduce blocking distortion. The Deblocking filter is applied after the inverse transform in the encoder (before reconstructing and storing the macro-block for future predictions) and in the decoder (before reconstructing and displaying the macro-block). The filter has two benefits: (1) block edges are smoothed, improving the appearance of decoded images (particularly at higher compression ratios) and (2) the filtered macro-block is used for motion-compensated prediction of further frames in the encoder, resulting in a smaller residual after prediction. Picture edges are not filtered.

4.2.1 Suggested Reading

The filtering process can be understood in detail from the following documents:

i. **h264_Deblocking_Filter.pdf (Doc3):** Read for Deblocking filter details and examples of filtering operation.

ii. **Standard_T-REC-H.264-200711.pdf (Doc2):** Section 8.7 (Page 193) describes the Deblocking process specifications. Refer to tables/algorithms provided here for any details that are omitted in the first document.

4.2.2 Block Description

For this block, you are required to design the Deblocking filter block. The filter block remains the same at the encoder and decoder side. The filter module takes a frame as input which has 11x9 macro-blocks (refer to "Frame Resolution" in Section 3 of this document for details) along with following control signals:

i. Filter_Intra: If this signal is '1' then p or q is intra coded else neither p nor q is intra coded.

ii. Filter_Coded: If this signal is '1' then p or q contains coded coefficients else neither p nor q contains coded coefficients.

iii. Filter_Reference_Frame: If this signal is '1' then p and q have same reference frame else p and q may be either from different reference frames or a different number of reference frames.

iv. Filter_Motion_Vector: If this signal is '1' then p and q have identical motion vectors else different motion vectors.

v. Quantization_Parameter: This input is required to calculate the thresholds alpha and beta discussed in Section 8.7.2.2. of Doc2.

Quantization_Parameter in this case is a fixed value which remains same for the entire frame.

The filter module should be synchronized to a clock and should have a reset signal to initialize memory elements. The input list provided above is not exhaustive as depending on the specification it may be required to additional control inputs. The output of the Deblocking filter should also be a filtered frame.

4.2.3 Important Things to Note

There are some important things to note in these documents:

i. In Doc1, Page 1 the filter order described is important. Also, the filter window moves one step at a time across the frame.
ii. The thresholds missing in Page 2 of Doc3 is described in detail in Section 8.7.2.2 of Doc2. Questions marks in Doc3 refer to alpha, beta in Doc2.
iii. Mathematical operations like Clip has been provided in "Conventions", page 13 of Doc2.

4.3 Transform and Quantization

Each residual macro-block is transformed, quantized and coded. Previous standards such as MPEG-1, MPEG-2, MPEG-4 and H.263 made use of the 8x8 Discrete Cosine Transform (DCT) as the basic transforms. The "baseline" profile of H.264 uses three transforms depending on the type of residual data that is to be coded: a transform for the 4x4 array of luma DC coefficients in intra macro-blocks (predicted in 16x16 mode), a transform for the 2x2 array of chroma DC coefficients (in any macro-block) and a transform for all other 4x4 blocks in the residual data.

4.3.1 Suggested Reading

The transform and quantization process can be understood in detail from the following documents:

i. **h264_Transform_Quantization.pdf (Doc4):** Read for details regarding the transform and quantization process. Equations have been provided in this document to aid your understanding and implementation process. This document also provides examples of transform and quantization process.
ii. **Standard_T-REC-H.264-200711.pdf (Doc2):** Section 8.5 (Page 169) describes the Transform and Quantization process specifications. The sequence of parsing of data from a macro-block needs to be understood clearly.

4.3.2 Block Description

In this block, you are required to design the encoding transform and quantization block along with the inverse transform and quantization block which will be used for decoding. At the encoding side, this module takes a 16x16 macro-block as input along with some control signals (if required) and gives a transformed and quantized macro-block as output. The decoder on the other hand takes the transformed and quantized macro-block as input and does inverse transform and quantization on it to get back the original macro-block.

Please refer to examples described in Doc1 to understand details of the encoding/decoding process. For any clarifications, refer to the specification Doc2. These modules should be synchronized to a clock and should have a reset signal to initialize memory elements.

4.3.3 Important Things to Note

There are some important things to note in these documents:

i. Read the sequence in which parts of a macro-block needs to be processed by this module. All select (decision making) signals should be inputs to this block.
ii. All operations and variables are in 16 bit integer arithmetic as described in Doc1 and Doc2.
iii. In your implementation, all hardware optimization/mathematical approximation steps need to be implemented exactly adhering to the calculations provided in the specification.
iv. Mathematical operations like Clip among many others has been provided in "Conventions", page 13 of Doc2.

5. H.264: Resources – Tutorials, Reference & Specification Docs

*** red color doc.: important materials**

Folder	Filename	Description
Tutorial		
- Decoder_Blocks		Tutorial Files for Decoder Blocks are in this folder
	h264_Context_Adaptive_Binary_Arithmetic_Coding.pdf	Tutorial for Binary Arithmetic Coding
	h264_Context_Adaptive_Variable_Length_Coding.pdf	Tutorial for Variable Length coding
	h264_Deblocking_Filter.pdf	Tutorial for De-blocking Filter
	h264_Inter-Prediction.pdf	H.264 Inter-Frame Prediction Tutorial
	h264_Intra_Prediction.pdf	H.264 Intra Frame Prediction Tutorial
	h264_Transform_Quantization.pdf	Tutorial for Transformation and Quantization Blocks
- Introduction and Overview		
	csvt_overview.pdf	Overview of the H.264 / AVC Video Coding Standard
	h.264-tutorial.pdf	Overview of the H.264/AVC (SLIDES)
	h264_overview.pdf	White paper: Overview of the H.264/AVC
	h264-AVC-Standard.pdf	The H.264/MPEG4 Advanced Video Coding Standard and its Applications
	VideoCodecConcepts.pdf	Basics of Video Coding
	Standard_T-REC-H.264-200711.pdf	The Original Standards Document – Very Important for understanding details and specification
source image		
	mother-daughter_qcif.300.yuv	Sample raw video file in binary format
	mother-daughter_qcif.300.264	Sample H.264 Encoded Video File in binary format
	mother-daughter_qcif.300.264	Sample H.264 Encoded Video File in Hexadecimal (ASCII file format)

	.txt	(Input to H.264 RTL Test Bench Module)
tool		
	Software_Manual.pdf	Manual for H.264 Sample C Code
	Jm14.2.zip	H.264 Decoder: Sample C Code for reference
	hex2bin.c	C code which converts a Hexadecimal ASCII file to binary file
	bin2hex.c	C code which converts the Binary File to Hexadecimal File
- H.264_Sample_ Player (For Window)		
	DGAVCIndex.exe	Sample player for H.264 Encoded Data
- Raw file player (For Window)	RawSwquence.exe	Sample player for Raw data (YUV file)
- Raw file player (For linux)	videometer	Sample player for Raw data (YUV file)
Other Misc Files		
	H.264_Final_Project.pdf	H.264 Final Project Manual (.pdf)
Library Files		
	osu018_stdcells.lib	Library file in liberty format. This file can be converted to database file which Synopsys can use.
	osu018_stdcells.db	Database library file with cell information which Synopsys uses for synthesis. This file should be the target and link library in Synopsys Design Analyzer.
	osu018_stdcells.v	Verilog file containing specified delays for each cell using which you can run post-synthesis simulations.
	osu018_stdcells.tlf	Timing Library in .tlf format.
	osu018_stdcells_se.v	This gives the structural port list of cells in your library which Cadence Silicon Encounter uses.
	osu018_stdcells.lef	This file contains geometry which Cadence Silicon Encounter uses to place cells in your design.
	encounter.conf	This is a sample configuration file which is used to configure your Silicon Encouter Physical Design

		Environment.
sample code		
	decoded_frame_buffer0.v	First Part of the Decoder Output Frame Buffer Memory
	decoded_frame_buffer1.v	Second Part of the Decoder Output Frame Buffer Memory
	decoded_frame_buffer_ctrl.v	Controller required to write to the frame buffers
	encoded_stream_buffer.v	Video file in H.264 format is written into this buffer. It read by the NAL decoder to supply input to the rest of decoding blocks
	h264_dec.v	TOP Level of the H.264 Decoder Logic Block
	tb_h264_dec.v	Test Bench File for the H.264 Decoder in which memory blocks and top level logic is instantiated
	timescale.v	File containing Time Scale directive for simulation

6. Project Deliverables

Code No	Due Date And Submission	Actions/Submission Details
DIV1 (Points =10)	Oct 22, 2010. (Hardcopy)	1. Study the specification and algorithm of the decoder modules you are going to design and understand its relative importance with respect to the entire H.264 decoder.
		2. Choose your design block and submit a report describing the blocks that you are going to design.
DIV2 (Points =15)	Oct 29, 2010 (Hardcopy)	1. Develop an Algorithmic Pseudo-Code for your blocks. Discuss the inputs to the blocks and outputs going out from your block. You should be able to describe in detail the data flow (interaction) of that block with other blocks. Also, describe clearly the functionality of the block in the form of algorithmic pseudo code (Take help from C code provided to you and specification document provided to you). Draw data flow diagram of logic within your blocks. Provide steps you will follow to successfully implement and test your design. Steps should be in the form of flowchart. (No softcopy submissions required)
PORT3 (Points =30)	Nov 5, 2010 (Hardcopy, electronic submission for verilog files)	1. Decide port list of your blocks based on your pseudo code and data flow diagram submitted earlier.
		2. Your naming convention of boundary signals should adhere as far as possible to the naming convention in the specification document. Please refer to specification document before you submit your port list.
		3. Provide block, timing, and state diagrams of your units along with sub-modules and other logic sub-blocks that you are going to use for your design. Provide exact port list of your blocks and draw its relative position with the encoder/decoder showing detailed signal flow between various decoder blocks. Also, print Verilog files of modules/sub-modules that you are going to build along with your hardcopy report. These verilog files may not have logic defined inside them but should contain inputs and output lists based on your block diagram. Attach a sample test bench file to the report as well. Test cases for this verilog test bench can be generated from reference C code and will be used later to test your

		design. (It should contain instances of test cases at least two test cases which you will use to test your design.) Your test bench file should be complete in all respect as this stage. Explain your choice of test cases used in the test bench. (No softcopy submissions required.)
SIM4 (Points =30)	Nov 23, 2010 (Hardcopy, electronic submission for verilog files)	1. Design your block by writing RTL based on the pseudo-code that you submitted earlier. Your RTL should be well commented and synthesizable.
		2. Use your test bench to run simulations on your design. You may test the various functionalities of your design by writing appropriate test cases for each one of them.
		3. Debug your design if there are any errors found during simulation. The final code submitted should be error free as far as possible. Your code will be verified against an extensive set of test cases.
		4. Submit your test benches along with your design files in a zipped folder. Mention clearly in one line how to run your simulations. (Hardcopy should include the description of your code.)
FULL5 (Point= 15)	Dec 3, 2010 (Hardcopy, electronic submission for files)	1. Synthesize your RTL based design using Synopsys Design Compiler using the library files that are provided to you. Physical design can be done using those library files using Cadence Silicon Encounter.
		2. Run post-synthesis simulations on your synthesized net list.
		3. Submit a zipped folder with your synthesized net list file in verilog format and your earlier test bench file. Your delay-annotated simulation output from synthesized net list should exactly match the output of your behavioral RTL which was submitted earlier.
		4. Submit a hardcopy report explaining everything that you did in this project systematically. Also, explain results of various test cases that you ran. Your report should be as comprehensive as possible with respect to objectives achieved, results, conclusions and reasons behind objectives that weren't achieved.

Note 1: All submissions should be done individually. Each report should describe what you have done and achieved in your design module.
Note 2: All report submission should mention the submission code number given in the table above.