University of Southern Queensland
Faculty of Engineering and Surveying

ERROR RESILIENT H.264 CODED VIDEO TRANSMISSION OVER WIRELESS CHANNELS

A dissertation submitted by

Timothy Glen Wise

in fulfilment of the requirements

Courses ENG4111 and 4112 Research Project

towards the degree of

Bachelor of Engineering (Software)

Submitted: October 2009

Abstract

The H.264/AVC recommendation was first published in 2003 and builds on the concepts of earlier standards such as MPEG-2 and MPEG-4. The H.264 Recommendation represents an evolution of the existing video coding standards and was developed in response to the growing need for higher compression. Even though H.264 provides for greater compression, H.264 compressed video streams are very prone to channel errors in mobile wireless fading channels such as 3G due to high error rates experienced.

Common video compression techniques include motion compensation, prediction methods, transformation, quantization and entropy coding, which are the common elements of a hybrid video codecs. The ITU-T Recommendation H.264 introduces several new error resilience tools, as well as several new features such as Intra Prediction and Deblocking Filter.

The channel model used for the testing was the Rayleigh Fading channel with the noise component simulated as Additive White Gaussian Noise (AWGN) using QPSK as the modulation technique. The channel was used over several Eb/N0 values to provide similar bit error rates as those found in literature.

Though further research needs to be conducted, results have shown that when using the H.264 error resilience tools in protecting encoded bitstreams to minor channel errors improvement in the decoded video quality can be observed. The tools did not perform as well with mild and severe channel errors significant as the resultant bitstream was too corrupted. From this, further research in channel coding techniques is needed to determine if the bitstream can be protected from these sorts of error rates.

University of Southern Queensland

Faculty of Engineering and Surveying

ENG4111 Research Project Part 1 &
ENG4112 Research Project Part 2

Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Engineering and Surveying, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Engineering and Surveying or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course "Project and Dissertation" is to contribute to the overall education within the student's chosen degree programme. This document, the associated hardware, software, drawings, and other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

Professor Frank Bullen
Dean
Faculty of Engineering and Surveying

Certification

I certify that the ideas, designs and experimental work, results, analysis and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Timothy Glen Wise

Student Number: 0050055729

_____

Signature

_____

Date

Acknowledgments

I would like to thank my supervisor, Dr Wei Xiang for his assistance and guidance throughout this research project.

I also like to acknowledge my son Thomas for providing the necessary distraction and amusement that help me though the time consuming and sometime long hours spent

Finally I would like to thank my friend Bernadette for providing the necessary ear for when I need to talk to someone.

TABLE OF CONTENTS

List of Figures

List of Tables

Glossary of Terms

AVC	Advanced Video Coding
ASO	Arbitrary Slice Ordering
BER	Bit Error Rate
dB	Decibels
CABAC	Context Adaptive Binary Arithmetic Coding
CAVLC	Context Adaptive Variable Length Coding
CGS	Course Gran Scalability
Chroma	Chrominance
CIF	Common Intermediate Format
CODEC	Encoder/Decoder
DVD	Digital Versatile Disk
FGS	Fine Grain Scalability
FIR	Finite Impulse Response
FMO	Flexible Macroblock Ordering
FSK	Frequency Shift Keying
GOB	Group of Blocks
GOP	Group of Pictures
IDR	Instantaneous Decoder Refresh
IEC	International Electrotechnical Commission
ITU	International Telecommunications Union
ISO	International Standards Organisation
JVT	Joint Video Team
JM	Joint Model
JSVM	Joint Scalable Video Model
LOS	Line-of-Sight
Luma	Luminance or Brightness
MPEG	Motion Pictures Expert Group
MSE	Mean Squared Error
MSK	Minimum Shift Keying
NAL	Network Abstraction Layer
NALU	Network Abstraction Layer Unit
PSK	Phase Shift Keying

PPS	Picture Parameter Set
PSNR	Peak Signal to Noise Ratio
QCIF	Quarter Common Intermediate Format
QPSK	Quadrature Phase Shift Keying
RTP	Real-time Transport Protocol
SNR	Signal-to-Noise Ratio, measured in dB
SoS	Sum of Sinusoids
SPS	Sequence Parameter Set
TDL	Tapped Delay Line

Chapter 1

Introduction

1.1 Outline

The International Telecommunications Union - Telecommunications (ITU-T) Recommendation H.264 was first published in 2003 and builds on the concepts of earlier standards such as MPEG-2 and MPEG-4. The H.264 Recommendation represents an evolution of the existing video coding standards and was developed in response to the growing need for higher compression.

Even though H.264 provides for greater compression, H.264 compressed video streams are very prone to channel errors in mobile wireless fading channels such as 3G due to high packet loss rates experienced

From the above statements it can be seen there is need address the error resilience of H.264 compressed video over wireless channels. The purpose and scope of this study is detailed in 1.3 Research Objectives.

1.2 Introduction

Video compression based on the hybrid COders and DECoders (CODEC) has evolved over the last two decades from the initial ITU-T Recommendation H.261 released in 1990 through to the current ITU-T Recommendation H.264/AVC and H.264/SVC published by the international standards bodies ITU-T (International Telecommunication Union) and ISO/IEC (International Organisation for Standardisation / International Electrotechnical Commission) referred to as ISO/IEC 14496–10 (MPEG-4 part 10) Advanced Video Coding (AVC).

This poses a great challenge when H.264 coded video signals that are transmitted over these noisy channels and a multitude of research papers have been written in using the imbedded H.264 error resilience tools when transmitting H.264 bitstreams over wireless fading channels.

Even with these benefits, H.264 compressed video stream is very prone to channel errors in mobile wireless fading channels such as 3G due to high packet loss rates experienced (Lin Liu et al. 2005).

1.3 Research Objectives

The aim of this research project is to investigate the error resilient tools of the ITU-T Recommendation H.264/AVC (2007) and their effect on the error resilience performance for coded video under minor, mild and severe channel errors as experienced in mobile communication environments such as 3G wireless networks. Additional to this the H.264 Recommendation will also be analysed to determine if possible improvements can be made to increase the error robustness.

This research project used Joint Video Team (2009) Joint Model 15.1 (JM 15.1) and Joint Scalable Video Model 9.18 (JSVM 9.18) reference software. The reference software is constantly being updated and therefore the baseline was set at these versions. The reference software was used in conjunction with the Recommendation and the channel model developed under IT++, which originates from the former department of Information Theory at the Chalmers University of Technology, Gothenburg, Sweden. IT++ is a C++ library of mathematical, signal processing and communication classes and functions. Its main use is in simulation of communication systems and for performing research in the area of communications. The library consists of generic vector and matrix classes, and a set of accompanying routines, making IT++ similar to MATLAB.

This software platform will be used to evaluate the performance of H.264/AVC video transmission over wireless channels for sensitivity and quality of service of the coded video stream.

1.4 Outline of the Dissertation

Chapter 2: H.264 Video Compression. This chapter contains a brief description of the H.264 Recommendation, which identifies techniques particular to the H.264 standard, including FMO, ASO, and CAVLC;

Chapter 3: Scalable Video Coding. This chapter gives an overview the Scalable Video Model extension of the H.264 Recommendation

Chapter 4: Mobil Digital Channel Modelling. This chapter gives an overview of the techniques and theory behind the modelling of digital communication channels.

Chapter 5: Simulation. This chapter provides an overview of the simulation used and the process used to perform the simulation.

Chapter 6: Results. This chapter evaluates the performance of the H.264 software over simulated wireless channel.

Chapter 7: Conclusion. This chapter concludes the dissertation and suggests further work in the area of the modelling the error resilience of H.264 Recommendation over wireless fading channels.

Chapter 2

H.264 Video Compression

2.1 Introduction

To represent video scene in digital form, the digital representation of the image seeks to replicate a natural scene with respect to the colour, shape, brightness and texture of the real world. Digital images taken at regular intervals and displayed consecutively produce motion video. This chapter seeks to provide an insight into what is video compression, and the typical video compression techniques used for encoding and decoding consecutive digital images.

Each digital image is divided into smaller component parts known as slices and macroblocks. Smaller sections of the image allow for a more accurate video compression to be achieved but increase in the compression overhead and bitstream sizes are a direct consequence. Composition of these blocks is discussed further in the chapter.

2.2 Video Compression

Richardson (2004) defines video compression as a tool that makes it possible for products from different manufacturers (e.g. encoders, decoders and storage media) to inter-operate. An encoder converts video into a compressed format and a decoder converts a compressed video back into an uncompressed format. ITU-R Recommendation H.264/AVC and H.264/SVC defines bitstream syntax for compressed video and a method for decoding this syntax to produce a displayable video sequence. The recommendation does not actually specify how to encode

(compress) digital video – this is left to the manufacturer of a video encoder – but in practice the decoder is likely to mirror the steps of the encoding process.

The video compression standard most commonly known is the MPEG-2 video coding standard based on ITU-T Recommendation H.262 as is widely used for the transmission of Standard Definition (SD) and High Definition (HD) TV signals over satellite, cable, and terrestrial channels and the storage of high-quality SD video signals onto DVDs and more recently in the Digital Video Broadcasting-Handheld (DVB-H) for mobile devices.

Although MPEG-2 coding standard has been extremely successful it has drawbacks in the compression performance and bitstream rates that can be achieved (Kumar et al. 2006). With the emergence of the H.264 standard, superior compression performance and bitstream rates can be achieved, therefore the H.264 coding standard is becoming more widely used expressly for HD video compression such Blu-Ray.

The H.264/AVC recommendation was first published in 2003. It builds on the concepts of earlier standards such as MPEG-2 and MPEG-4 Visual and offers the potential for better compression efficiency for compressed video and greater flexibility in compressing, and transmitting and storing video. The H.264 Recommendation represents an evolution of the existing video coding standards and was developed in response to the growing need for higher compression of video for various applications such as videoconferencing, digital storage media, television broadcasting, Internet streaming, and communication.

2.3 H.264 Encoding and Decoding

Common video compression techniques include motion compensation, prediction methods, transformation, quantisation and entropy coding, which are the common elements of a hybrid video encoder/decoder as discussed in Richardson (2004). The ITU-T Recommendation H.264 introduces several new error resilience tools, as well as several new features as Intra Prediction and Deblocking Filter and enhancements to the standard hybrid video encoder/decoder as shown in Figure 1.

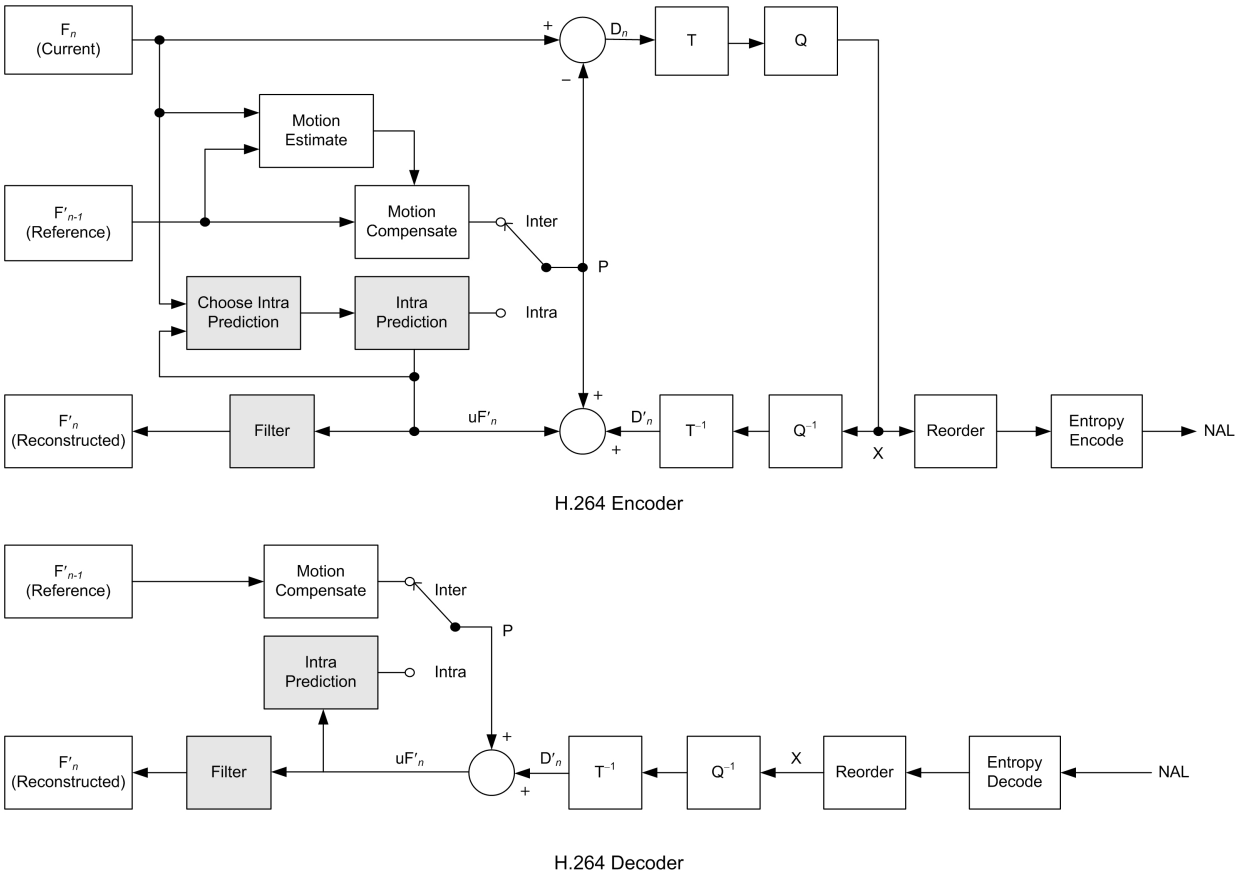

Figure 1 – H.264 Encoder and Decoder (Richardson, 2004)

A colour image is usually sampled in RGB colour space, where the additive primary colours of Red, Green, and Blue are combined in various intensities to for the image. The intensities of the Red, Green, and Blue components are represented individually as an 8 bit number ranging from 0 (minimum intensity) to 255 (maximum intensity). Therefore if all three intensities are set to 0 then white is represented, and consequently if all are set to 255, black is represented. As can be seen the overhead in amount data bits required to represent the colour space in RGB form for each image would be extremely large.

As human visual perception is more sensitive to the luminance of the image than the colour, it is therefore possible to represent the image more efficiently by separating the luminance and colour information. This is known as the YCbCr or YUV colour space, where Y represents the luminance and Cb and Cr represents the deference between the colour intensities.

Figure 2 shows three of the four sampling patterns used by the H.264 recommendation, the forth 4:0:0 or monochrome is only represented by the Y samples. The main sampling used is 4:2:0 as it is widely used in video conferencing, digital television, and digital versatile disk (DVD) storage due to it requiring one quarter the number of colour samples to Y samples compared to 4:4:4 or RGB video.

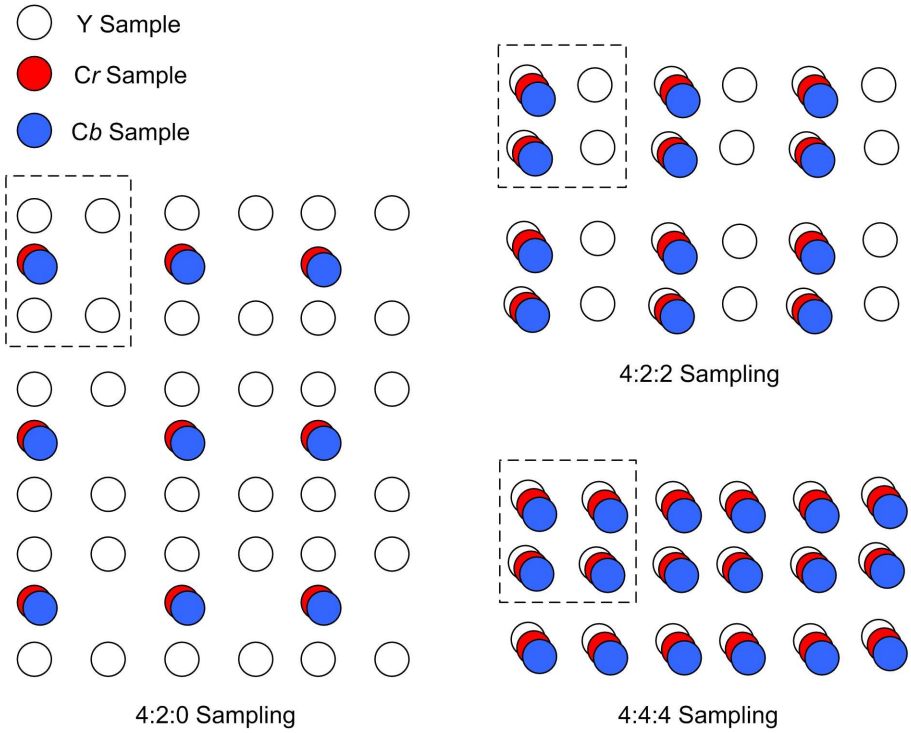

Figure 2 – YUV Sampling Patterns (Richardson, 2004)

The coding of picture in the spatial domain is performed by partitioning a picture into slices. A slice is a sequence of macroblocks, or, when macroblock-adaptive frame/field decoding is in use, a sequence of macroblock pairs. Each macroblock is comprised of one 16x16 luma array and, when the chroma sampling format is not equal to 4:0:0 and parameter `separate_colour_plane_flag` is equal to 0, two corresponding chroma sample arrays. When `separate_colour_plane_flag` is equal to 1, each macroblock is comprised of one 16x16 luma or chroma sample array.

When macroblock-adaptive frame/field decoding is not in use, each macroblock represents a spatial rectangular region of the picture

2.4 Profiles

The H.264/AVC and H.264/SVC recommendation defines a series of profiles that place restrictions on the encoded bitstream. The restrictions placed on the encoded bitstreams are required so that the capabilities needed to decode these bitstreams shall be supported by all decoders conforming to that profile. Encoders are not required to make use of any particular subset of features supported in a profile but are not allowed to use any features not supported by that profile.

The supported profiles for H.264/AVC are referred to as the Baseline, Main, Extended, and Fidelity Range Extensions (FRExt). The FRExt consists of total eight related profiles known as High, High 10, High 4:2:2, High 4:4:4 Predictive, High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra, and CAVLC 4:4:4 Intra profiles making a total of eleven profiles overall. While the supported profiles for H.264/SVC is referred to as the Scalable Baseline, Scalable High, and Scalable High Intra. The Scalable Baseline and Scalable High are based on the H.264/AVC Baseline and High profiles, but with more restrictions on the encoded bitstream.

Even though the constraints imposed by a given profile defines a bitstream syntax it is still possible to require significant variation in the performance of encoders and decoders depending upon the other factors such as the specified size of the decoded pictures. As it is not practical to implement a decoder to handle all possible variations of features used by an encoder for any given profile, additional constraints on the bitstream are imposed by the use of “levels” that are specified within each profile. These levels define constraints such as maximum frame size of a picture, bit rates, macroblocks allowed. Therefore encoders are required to provide conforming bitstreams consistent with their implemented profile and level, while decoders conforming to a specific profile must be able to support all of its features.

The H.264/AVC Main and FREXt profiles and the H.264/SVC High profiles are mainly used for the compression of HD video for storage and/or transmission over fixed wire networks and are not suitable for wireless networks.

Therefore the main focus will be on the H.264/AVC Baseline and Extended profiles and the H.264/SVC Scalable Baseline profile all of which provide the following video coding tools:

- I, P, B, SP, and SI Slices,
- Flexible Macroblock Ordering (FMO),
- Arbitrary Slice Ordering (ASO),
- Redundant Pictures,
- Context Adaptive Variable Length Coding (CAVLC),
- Context Adaptive Binary Arithmetic Coding (CABAC),
- NAL Units and Data Partitioning, and
- Parameter Sets.

2.5 Slices

A picture may be divided into slices, where a slice consists of given number of macroblocks or macroblock pairs when using MBAFF that are ordered consecutively in raster scanned order within a particular slice group. A picture consists of one (FMO Disabled) up to seven slice groups consisting of a given number of macroblocks or macroblock pairs that compose the picture scanned in the order specified by the FMO technique used. A picture divided into three slices is shown in Figure 3.

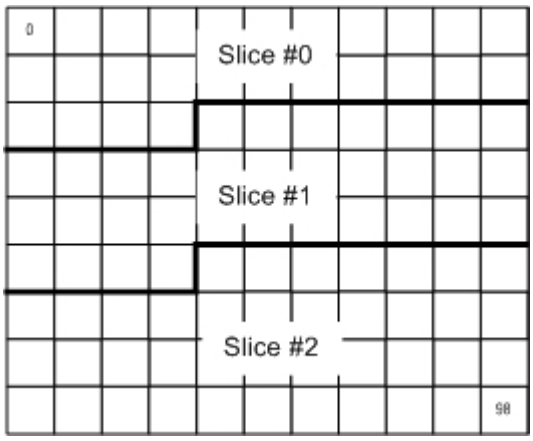

Figure 3 – Slice Partitioning

Therefore although a slice contains macroblocks or macroblock pairs that are scanned consecutively in raster order within a slice group, these macroblocks or macroblock pairs are not necessary in raster order within the picture. The value of the encoder parameter `slice_mode` is used to set the slice coding mode. By default `slice_mode` is equal to zero, which specifies that each slice group consists of one slice. Therefore a picture can be divided into one to seven slices depending if FMO is being used or not. When `slice_mode` is equal to one, each slice consists of a fixed number of macroblocks.

The number of bytes contained in each slice is specified by the encoder parameter `slice_argument`. This parameter can be equal to one up to maximum number of macroblocks contained in the picture. Therefore each slice group can now be divided into a number of slices, though a slice cannot cross a slice group boundary. When MABFF is in use, a macroblock pair is also not to cross a slice boundary.

When `slice_mode` is equal to two, each slice consists of a fixed number of bytes. The number of bytes contained in each slice is specified by the encoder parameter `slice_argument`. Therefore each slice group can now be divided into a number of slices consisting of set number of bytes, though the bytes that comprise one macroblock or macroblock pair when MABFF is in use, are not to cross a slice boundary.

Slices are self-contained in the sense that given the active sequence and picture parameter sets, their syntax elements can be parsed from the bitstream and the values of the samples in the area of the picture that the slice represents can be correctly decoded without use of data from other slices provided that utilised reference pictures are identical at encoder and decoder. Some information from other slices maybe needed to apply the deblocking filter across slice boundaries. Slices are encoded as I, P, B, Switching I (SI), or Switching P (SP) slices.

I Slice

A slice in which all macroblocks contained in the slice are coded using intra prediction. If a block or macroblock is encoded in intra mode, a prediction block is formed based on previously encoded and reconstructed macroblocks. Therefore all prediction is based only on the macroblock within that picture. There is also a special case of I slice called the Instantaneous Decoder Refresh (IDR) picture which clears the contents of the reference picture buffer. The first picture in coded video sequence is always an IDR picture, but the recommendation allows for additional IDR pictures to be sent a given intervals.

P slice

A slice in which macroblocks contained in the slice can be coded using either intra or inter prediction. Inter prediction creates a prediction model from one or more previously encoded video frames. The model is formed by shifting samples in the reference frame called motion compensated prediction. P slices support one motion compensated signal per block.

B slice

A slice in which macroblocks contained in the slice can be coded using either intra or inter prediction. B slices support two motion compensated signals per block.

Additional to the above mentioned slice encoding techniques, H.264/AVC introduced two additional slice types known as switching I slice (SI) and P slice (SP). The discussion on these two slice types is beyond the scope of this paper, though further investigation is warranted on their use for error recovery.

2.6 Flexible Macroblock Ordering

Flexible Macroblock Ordering (FMO) is one of the error resilience schemes used by H.264 as part of the baseline, extended, and scalable baseline profiles. A picture can be partitioned into regions known as slice groups which are subset of the macroblocks within the picture or when Macroblock-Adaptive Frame/Field decoding (MABFF) is in use, a sequence macroblock pairs. This partitioning of the picture's macroblocks into slice groups is specified by the macroblock to slice group map.

The encoder parameter `num_slice_groups_minus1` + 1 specifies the number of slice groups a picture is to be divided into. By default, FMO is turned off by the parameter `num_slice_groups_minus1` being set to zero. When FMO is off, a picture consists of only one slice group with all macroblocks within the picture scanned in raster order being part of this slice group.

To enable FMO, the encoder parameter `num_slice_groups_minus1` is set to a value representing the number of slice groups the picture is to be divided into. A picture can be divided into two up to a maximum of eight slice groups, though there are some limitations to the number of slice groups allowed depending on the FMO technique chosen. The FMO technique is selected by the encoder parameter `slice_group_map_type`, which specifies how the mapping of slice group map units to slice groups is coded. The seven available slice group map types (0 through 6) which are shown in Figure 4.

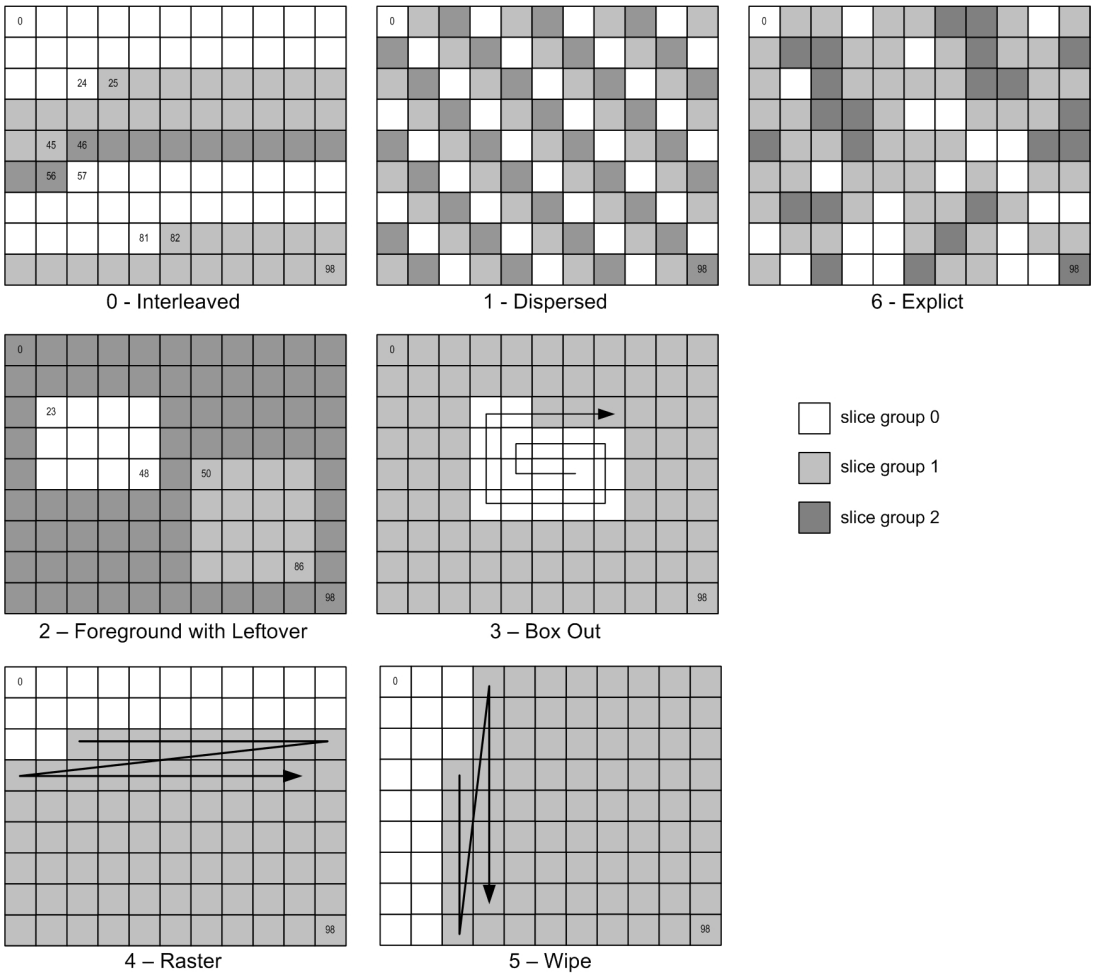

Figure 4 – FMO Techniques

From literature review the use of FMO has been the main focus of providing error resilient H.264 encoded bitstreams with Ogunfunmi and Huang (2005) proposing a novel 3D Macroblock to slice group Allocation Map (MBAmap) that uses the dispersed FMO technique with three slice groups and then spread the macroblocks for one picture across three frames to distribute burst errors. The drawback for this technique is that the each picture requires three frames, thereby increasing the bitstream complexity. Another method proposed by Hoa Chen et al. (2008) is to use adaptive FMO technique selection to choose the best FMO technique based on the picture contents and Rate Distortion Optimisation (RDO) to produce a bitstream the supplies superior error resilience than using one technique alone of wireless networks. The later technique proposed Hoa Chen et al. (2008) is worthy of further investigation.

2.7 Arbitrary Slice Ordering

Since each slice of a coded picture can be approximately decoded independently of other slices of the picture. Depending on the profile in use, arbitrary slice ordering may or may not be allowed. If arbitrary slice ordering is allowed, the slices and data partitions of a coded picture may follow any decoding order relative to each other. Arbitrary slice ordering (ASO) allows slices to be decoded in a different order than their designated display order. Therefore ASO improves upon loss robustness and delay reduction which is particularly important for real-time video streaming across networks that have an out of order delivery of data.

2.8 Redundant Pictures

A redundant picture is an alternative representation of a coded slice, which may use different quantization parameters, different reference pictures, different mode decisions, and different motion vectors than those used in the encoding of the primary slice. If the primary slice is received correctly, the redundant slice is discarded. However, if the primary slice is received in error, the redundant slice can be decoded in order to limit the distortion caused by the error in the primary bitstream. The use of redundant pictures is limited to the baseline profile and the parameter `NumberReferenceFrames` has to be equal to the number of pictures used in the primary Group of Pictures (GOP) specified by the parameter `PrimaryGOPLength`.

2.9 Context Adaptive Variable Length Coding (CAVLC)

CAVLC is a reversible procedure for entropy coding that assigns shorter bit strings to symbols expected to be more frequent and longer bit strings to symbols expected to be less frequent. This is the method used to encode residual, zigzag ordered 4x4 (and 2x2) blocks of transform coefficients. CAVLC is designed to take advantage of several characteristics of quantized 4x4 blocks.

When the parameter `entropy_coding_mode` is set to 0, residual block data is coded using a CAVLC scheme and other variable-length coded units are coded using Exponential Golomb codes which are variable length codes with a regular construction.

2.10 Context Adaptive Binary Arithmetic Coding (CABAC)

CABAC uses Binary Arithmetic Coding which means that only binary decisions (1 or 0) are encoded. A non-binary-valued symbol, such as a transform coefficient or motion vector is converted into a binary code prior to arithmetic coding. This process is similar to the process of converting a data symbol into a variable length code but the binary code is further encoded by the arithmetic coder prior to transmission.

When `entropy_coding_mode` is set to 1, the CABAC arithmetic coding system is used to encode and decode H.264 syntax elements. CABAC achieves good compression performance through selecting probability models for each syntax element according to the element's context then adapting probability estimates based on local statistics, and then using arithmetic coding.

2.11 NAL Units and Data Partitioning

2.11.1 Network Abstraction Layer (NAL)

The bitstream can be in one of two formats, either the NAL unit (NALU) stream format or the byte stream format – as specified in Annex B of the recommendation. The NAL unit stream consists of a sequence of syntax structures called NAL units which are sequenced in decoding order with constraints imposed on the decoding order and contents of the NAL units in the NAL unit stream.

The byte stream format can be constructed from the NALU stream format by ordering the NAL units in decoding order and prefixing each NAL unit with a start code prefix and zero or more zero-valued bytes to form a stream of bytes. The NAL unit stream format can be extracted from the byte stream format by searching for the location of the unique start code prefix pattern within this stream of bytes.

2.11.2 Data Partitioning

Normally each encoded slice is put into exactly one NAL unit, but when using data partitioning, the coded data for a single slice is split up into three partitions, and each partition is put in a separate NAL unit. The first partition A contains the slice header, macroblock types, quantization parameters, prediction modes, and motion vectors. The second partition B contains residual information of intra-coded macroblocks and the final partition C contains residual information of inter-coded macroblocks.

Data partitioning allow the decoder to be able to use information from correctly received partitions when one of the partitions is lost. Stockhammer and Bystrom (2004) conducted research in the use of data partitioning in mobile channels which showed that percentage of lost frames was lowered and probability of decoding poor quality video is reduced.

2.12 Parameter Sets

In the H.264 recommendation the Video Coding Layer (VCL) was separated from the NAL. NAL units are classified into VCL and non-VCL NAL units. The VCL NAL units contain the data that represents the values of the samples in the video pictures, and the non-VCL NAL units contain any associated additional information such as parameter sets, which contains important header data which is expected to rarely change and offers the decoding of a large number of VCL NAL units. There are two types of parameter sets:

- sequence parameter sets, which apply to a series of consecutive coded video pictures called a coded video sequence

- picture parameter sets, which apply to the decoding of one or more individual pictures within a coded video

In previous coding standards, if a few key bits of information such as sequence header or picture header information were lost due to errors, this caused the entire bitstream to be corrupted. This loss of data had a severe negative impact on the decoding process. If the parameter gets corrupted the same effect will be observed, but because it is separated from the main picture information it can be protected against errors in a specialised manner.

Chapter 3

Scalable Video Coding

3.1 Introduction

The ITU-T Recommendation H.264/AVC standard is now well established, but some initiations exist in regards to video streaming. With the emergence of new technologies in mobile communications, an ever increasing amount of video streaming content is being used. To support the different screen resolutions and network bandwidths, the video stream has to be encoded multiple times to provide different bitstream rates and resolutions for each application.

The current revision 3 of the ITU-T Recommendation H.264 (2007) also contained extensions to ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC to specify Annex G – Scalable Video Coding (SVC). The SVC extension introduced three additional profiles (Scalable Baseline, Scalable High, and Scalable High Intra). The SVC extension provides scalability at a bitstream level to support functionalities such as bit rate, format, and power adaptation, graceful degradation in lossy transmission environments as well as lossless rewriting of quality-scalable SVC bit streams to single-layer H.264/AVC bit streams.

With a moderate increase in decoder complexity relative to single-layer H.264/AVC, these functionalities provide enhancements to transmission and storage applications. SVC has achieved significant improvements in coding efficiency with an increased degree of supported scalability relative to the scalable profiles of prior video coding standards

H.264 Scalable Video Coding (H.264/SVC) introduces scalability features that enable encoders to produce a single bitstream with that provides layers for multiple

temporal, spatial, and SNR scalability, while maintain the high compression efficiency.

3.2 SVC Overview

Most components of H.264/AVC are re-used in H.264/SVC which includes motion compensated and intra prediction, transform and entropy coding, deblocking filter, and NAL (Network Abstraction Layer) unit. The base layer of an SVC bitstream is generally coded in compliance with H.264/AVC though new tools are added for supporting temporal, spatial, and quality scalability also known as Signal to Noise Ratio (SNR) scalability.

The JSCM SVC encoder supports two different coding modes, single-layer coding mode, and scalable coding mode. Although single-layer bit-stream can also be generated in the scalable coding mode, the single-layer coding mode provides more flexibility but lacks the support the generating scalable bit-streams. When the encoder is in single-layer mode, an AVC compatible bit-stream is generated that can be decoded using the H.264/AVC decoder.

3.2.1. Single Layer Coding

To provide for single-layer coding mode, the configuration file contains the parameter `AVCMode`, when set to 1 will only allow single layer coding, which is also referred to as Multiview coding mode, since this mode was implemented to support multiview coding. When the encoder is run in single-layer mode, an AVC compatible bit-stream is generated that is compatible with the H.264/AVC decoder. The configuration file parameters for the single-layer coding are subset of the ones provide for H.264/AVC, but do not provide for the same flexibility. When the single-layer coding mode used, the scalability tools can not be used, but the coding structure is not restricted to dyadic prediction structures as used in the H.264/AVC.

To control the picture slice and coding modes, the sequence format string is used, which is similar to the Hieratical Coding provide by H.264/AVC. The sequence coding structure together with Memory Management Control Operation (MMCO)

and Reference Picture List Reordering (RPLR) commands are specified in the `SequenceFormatString` parameter in encoder configuration file.

3.2.2. Scalable Coding

H.264/SVC provides scalable video bitstreams that contains a non-scalable base layer and one or more enhancement layers. An enhancement layer may enhance the temporal resolution, the spatial resolution, or the quality of the video content represented by the lower layers or part of those layers. Spatial and temporal scalability describe cases in which layers within the bitstream represent the source content with a reduced picture size (spatial) or frame rate (temporal) resolution respectively. In quality scalability, a substream within the bitstream provides the same spatial and temporal resolution as the global bitstream, but with a lower fidelity or SNR.

The scalable layers can be aggregated to a single Real-time Transport Protocol (RTP) stream, or transported independently. The concept of video coding layer (VCL) and Network Abstraction Layer (NAL) is inherited from H.264/AVC. The VCL contains the signal processing functionality, such as transform, quantization, motion-compensated prediction, loop filter, and inter-layer prediction. A coded picture of a base or enhancement layer consists of one or more slices, where the NAL encapsulates each slice generated by the VCL into one or more Network Abstraction Layer Units (NAL units).

3.3 Spatial Scalability

For spatial scalability, each layer within the bitstream represents one of the spatial resolution formats supported by the scalable encoder. Each layer is identified by a layer identifier, or dependency identifier. If the identifier is equal to 0, the bitstream only contains the base H.264/AVC compatible layer and is limited in what is allowed. If the scalable baseline profile is used, then the `ProfileIdc` parameter must equal to 66, 77, or 88 for the base layer to maintain compatibility with the H.264/AVC

bitstreams. Even though, there are limitations on encoder parameters can be specified, e.g. FMO parameter `slice_group_map_type` can only be set to 2.
For temporal scalability, Schwatz et al. (2006) describe temporal scalability as being achieved by an oversampled pyramid approach. The pictures of different spatial layers are independently coded with layer specific motion parameters. However, in order to improve the coding efficiency of the enhancement layers, additional inter-layer prediction mechanisms have been introduced. These prediction mechanisms have been made switchable so that an encoder can freely choose which base layer information should be exploited for an efficient enhancement layer coding. Since the incorporated inter-layer prediction concepts include techniques for motion parameter and residual prediction, the temporal prediction structures of the spatial layers should be temporally aligned for an efficient use of the inter-layer prediction. To archive this all NAL units contain information from a given a time instant form an access unit and thus have to be follow each other inside an SVC bit-stream.

3.4 Temporal Scalability

A bitstream is said to provide temporal scalability when the set of its access units can be partitioned into a temporal base layer, and one or more temporal enhancement layers. A video sequence can be temporal scaled by reducing the number frames encoded thereby reducing the frame rate. To achieve this, temporal scalable bitstreams are be generated by using hierarchical prediction structures, without any changes with respect to H.264/AVC. Any picture can be marked as reference picture and used for motion-compensated prediction of following pictures independent of the corresponding slice coding types. These features allow the coding of picture sequences with arbitrary temporal dependencies.

For temporal scalability, Schwatz et al. (2006) describe that the coding and display order of pictures is completely decoupled, as in that any picture can be marked as a reference picture and used for motion-compensated prediction of following pictures, independent of the corresponding slice coding types. These features allow the coding of picture sequences with arbitrary temporal dependencies.

Temporal scalable bitstreams can be generated by using hierarchical prediction structures as in H.264/AVC. These key pictures are coded in regular intervals by using only previous key pictures as references. The pictures between two key pictures are hierarchically predicted, so that that the sequence of key pictures represents the coarsest supported temporal resolution. This can be refined by adding pictures of following temporal prediction levels. With temporal scalability the hierarchical prediction structures also provide an improved coding efficiency compared to classical IBBP coding structure of H.254/AVC.

3.5 Quality Scalability

.

For quality scalability the picture can be encoded using either Course Gran Scalability (CGS) or Fine Grain Scalability (FGS). CGS achieves scalability based on the spatial scalability concepts, with the only difference in that up-sampling and scaling operations typical of inter-layer prediction are avoided. However, CGS is not able to provide satisfactory performance, especially when the bit rate ratio between successive quality layers is small.

FGS can be is achieved by the use of progressive refinement slices. Each of these slices represents a refinement of the residual signal, corresponding to a bisection of the quantization step size

Chapter 4

Mobile Digital Channel Modelling

4.2 Introduction

A typical digital communication system as shown in Figure 5 is described by Moon (2005) consisting of a fairly general framework for a single digital communication link. In this link, digital data from a source such as the H.264 encoder is encoded and modulated for communication over a channel. At the other end of the channel, the data is demodulated, decoded and sent to a sink. The elements in this link all have mathematical descriptions and theorems from information theory which govern their performance. This chapter will provide overview of channel modelling.

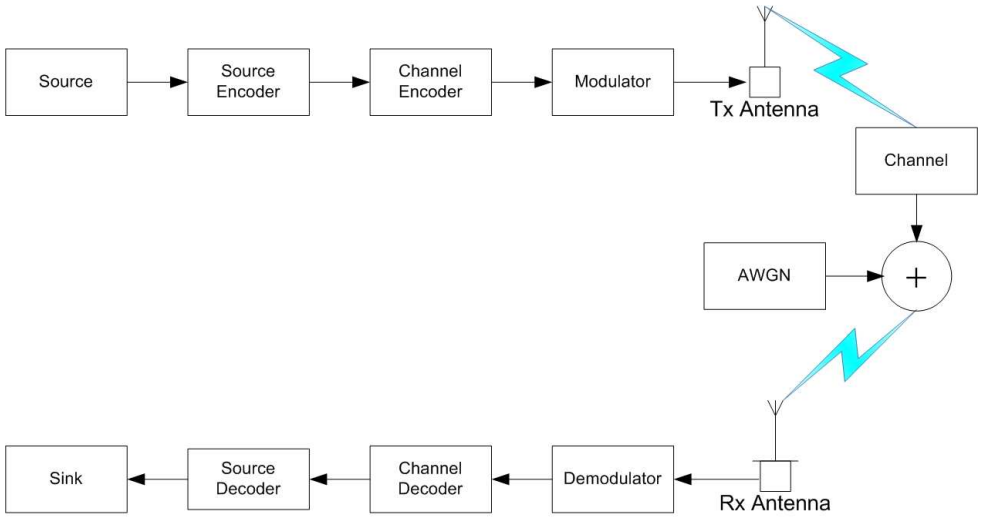

Figure 5 – Typical Digital Communication Chanel (Moon, 2005)

4.3 Mobile Communication Channel

A mobile communication channel is effected by two types of fading, large scale and small scale. This large scale fading manifests itself in the channel through attenuation of the signal due to Path Loss and Shadowing. Path Loss is the mean loss of power as a function of distance of the receiver from the transmitter, while Cavers (2003) describes shadowing as the variation of the power about the path loss. Shadowing is attributed to large obstacles such as hills and tall buildings between the transmitter and receiver.

Sklar (1997) proposed that large scale fading due to path loss and shadowing can be represented by:

$$L_p(d) = L_s(d_0) + 10n\log_{10}\left(d/d_0\right) + X_\sigma$$

where d is the distance from the transmitter, d_0 is the reference distance located in the far field of the antenna, n is the path loss exponent, and X_σ denotes a zero-mean Gaussian random variable with standard deviation σ. Typically the value of d_0 is taken to be 1 km for large cells, 100 m for microcells, and 1 m for indoor channels.

On the other hand, Sklar (1997) describes small scale fading as manifestations due to time variance behaviour of the channel and/or time spreading of the signal. For mobile communications, the channel is time variant because of motion between the transmitter and receiver results in propagation path changes, with the rate of change of this propagation accounting for the rate of change of the fading. This results in either as fast or slow fading of the signal within the time domain.

While time spreading occurs due to individual signals being received spread in time as the result of being scattered of surrounding objects around the receiver. This results in either flat or frequency selective fading of the signal within the frequency domain.

If multiple reflective paths or scatters are large in number and there is no non fading Line-of-Sight (LOS) component, the envelope of the received signal is statistically described by a Rayleigh probability distribution function and is termed Rayleigh fading as used in the Jakes fading channel model. If the received signal contains a significant non fading LOS component the small scale fading envelope of the received signal can be described by a Ricean probability distribution function and is termed Ricean fading as used in the Rice fading channel model.

So the effect on the received signal is due to the:

- Mean path loss as a function of distance from the transmitter.

- Variations about the mean path loss (typically 6–10 dB), or large scale fading such as shadowing.

- Rayleigh or small scale fading margin (typically 20–30 dB).

As can be seen, Rayleigh small scale fading is the most significant contribution to loss of signal power. Rayleigh fading models of communication channels assume that the magnitude of a signal that has passed through the channel will fade according to a Rayleigh distribution produced by the radial component of the sum of two uncorrelated random Gaussian variables. Therefore Rayleigh flat fading models have been shown to be good for simulation on the effect of propagation of a signal through a communications channel, such as that used by wireless devices.

4.4 Rayleigh Fading Channels

Clarke (1968) cited in Iskander (2008) describes a radio communication model for flat fading in urban/suburban environments, which assumes a fixed transmitter with a vertically polarized antenna and a mobile terminal. In Clarke's model the electric field incident (*E)* on the mobile antenna consists of *N* angular spread horizontal plane waves which are called scatterers:

$$E_z(t) = E\sum_{n=1}^{N} C_n \cos(\theta_n) - E\sum_{n=1}^{N} C_n \sin(\theta_n),$$

where $\theta_n = 2\pi f_n t + \varphi_n$ are the phases of the received scatterers and $f_n = (v/\lambda)\cos\alpha_n$ the Doppler shift of the n^{th} scatterer.

Therefore the maximum Doppler shift experienced can be expressed by $f_n = v/\lambda$. This model has been shown to be Rayleigh-distributed, with probability density function:

$$p_R(r) = \frac{2r}{2\sigma^2} e^{-r^2/2\sigma^2}, \qquad r \geq 0$$

Gans (1972) developed a power spectral theory for the mobile radio channel based on Clarke's model to show that the Doppler shifts of the carrier frequency in the frequency domain on each scatterer over time is spectrum spread. This effect is known as Doppler spreading of the signal and is indirectly proportional to the channel coherence time.

4.4.1 Jake's Model

To produce fading simulators designed to model both the Rayleigh fading scatterer distribution and Doppler spreading, most are designed around the Sum-of-Sinusoids (SoS) or Filtered Gaussian Noise (FGN) methods.

Jakes (1974) popularised a model for Rayleigh fading based on the SoS method. Let the scatterers be uniformly distributed around a circle at angles α_n with N scatterers arriving at the moving receiver. Therefore the Doppler shift experienced by scatterer *n* is:

$$\omega_n = \omega_m \cos(\alpha_n)$$

where the maximum Doppler shift $\omega_m = 2\pi\lambda v$ can be found from the carrier frequency wavelength (λ), and the velocity of motion of the receiver (v). By using arrival angles $\alpha_n = 2\pi n/N$ there is a quadrennial symmetry in the magnitude of the Doppler shift which can be modelled with $N_0 + 1$ complex oscillators, where:

$$N_0 = \frac{\left(\frac{N}{2} - 1\right)}{2}$$

Thus Jake's model is represented by:

$$T(t) = K\left\{\frac{1}{\sqrt{2}}\left[\cos\alpha + I\sin\alpha\right]\cos\left(\omega_M t + \theta_0\right) + \sum_{n=1}^{N_0}\left[\cos\beta_n + I\sin\beta_n\right]\cos\left(\omega_n t + \theta_n\right)\right\}$$

4.4.2 Dent's Model

Dent (1993) showed that certain limitations exist with the deterministic nature of the Jake's model and the correlation of the waveforms. To remove these correlations Dent proposed to remove the correlation between the waveforms by using Walsh-Hadamard code words to provide quadrennial symmetry for all Doppler shifts. Using arrival angles $\alpha_n = 2\pi\left(n - 0.5\right)/N$ and Jake's procedure, this leads to the following model:

$$T(t) = \sqrt{\frac{2}{N_0}}\sum_{n=1}^{N_0}\left[\cos\beta_n + I\sin\beta_n\right]\cos\left(\omega_n t + \theta_n\right)$$

Iskander (2008) describes s(t) as a low pass input to a TDL channel, then the low pass output y(t) is obtained as the convolution between s(t) and g(t,τ) so that:

$$y(t) = \sum_{i=1}^{N} g_i(t)\, s(t - \tau_i) \qquad \text{where} \qquad g_i(t) = a_i \cdot e^{j \cdot 2\pi f_D \cos(\theta_i) t}$$

4.5 Channel Characteristics

Radio wave propagation in the mobile environment can be described by multiple paths which arise due to reflection and scattering in the mobile environment. The physical model of a mobile channel is based on multiple reflections each with its own amplitude, phase delay, and Doppler shift which can be summarised by:

$$y(t) = \sum_{i} a_i \cdot e^{j 2\pi f_D \cos(\theta_i) t} \cdot s(t - \tau_i)$$

4.5.1 Doppler Spread

If a mobile receiver moves through a random field, then changes in signal level and phase, with the rate of the changes proportional to the velocity of the mobile receiver. This is referred to as flat fading, where the signal bandwidth is narrow so that small delays in τ_i do not affect the signal, therefore $s(t - \tau_i) \approx s(t)$. This represents the most common mobile channel, the flat fading channel represented by:

$$y(t) = s(t) \cdot \sum_{i} a_i \cdot e^{j 2\pi f_D \cos(\theta_i) t}$$

4.5.2 Delay Spread

If the Doppler spread is very small or the mobile receiver is stationary, it can be considered that phases of the scatterers are constant. If the signal is an impulse in time, the reflections spread the signal on reception causing delay spread. This represents a mobile channel that has an impulse response that can be represented by a Finite Impulse Response (FIR) filter:

$$y(t) = \sum_i a_i \cdot e^{-j\phi_i} \cdot s\left(t - \tau_i\right)$$

where ϕ is a random phase associated with the signal arrival.

4.5.3 Frequency Selective Fading

Previously it was shown that a mobile channel may experience Doppler spread or Delay spread. Suppose the mobile receiver is moving, or even the other objects are moving with respect to the receiver, then the scatterers are therefore constantly varying in both frequency and power due to the time variance of the channel. This channel is a time variant linear filter, where the response observed at given time t to and impulse τ is:

$$y\left(t\right) = \sum_{i=1}^{N} a_i \cdot e^{j2\pi f_D \cos(\theta_i)t} s\left(t - \tau_i\right)$$

This represents the well known tapped-delay line model as shown in Figure 6, where each fading process $g_k(t)$ is complex Gaussian, with a Doppler power spectrum modelled by Jake's or Rice Rayleigh fading model and t_s is the time delay for that multipath signal reaching the mobile receiver.

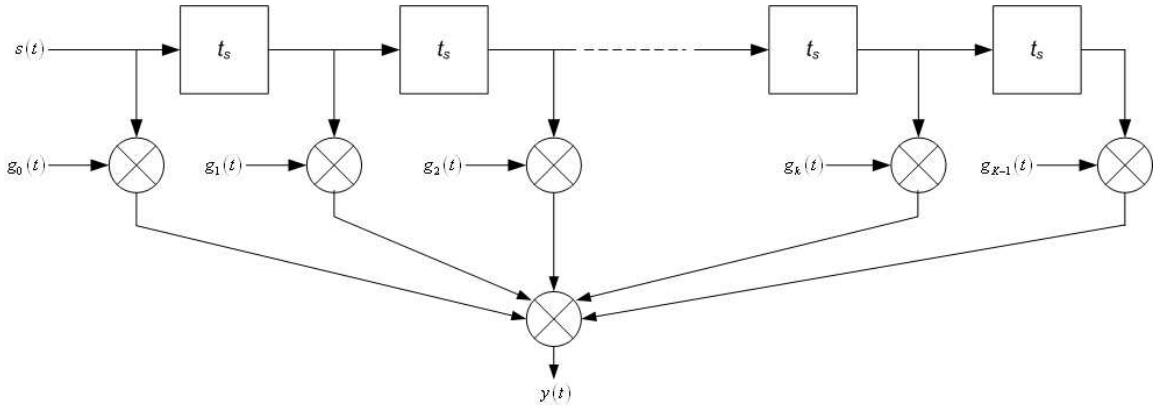

Figure 6 – Tapped Delay Line Model (Iskander, 2008)

Chapter 5

Simulation

5.1 Introduction

So far the background of H.264 video coding and channel modelling has been covered in previous chapters, so the task now is to realise this to design a software platform to perform testing. Once the model is implemented, the testing will mainly iterative with the process illustrated in Figure 7.

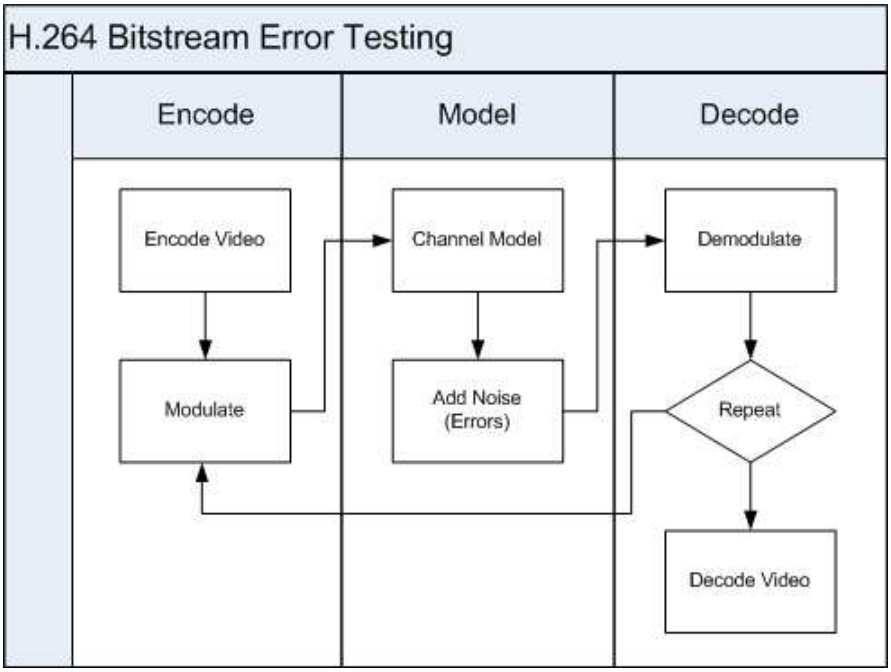

Figure 7 – Test Flow Diagram

This chapter will discuss the modulation technique used as well as the channel model implemented. As the main goal is creating errors on a channel, an overview of Additive White Gaussian Noise (AGWN) will be covered. Following implementation, different encode video sequences will be passed through the model to test the performance of H.264 encoded bitstream with the variety of the error resilience tools also tested.

5.2 Channel Capacity

The channel capacity is defined by the maximum rate at which data can be transmitted over a given communication channel bounded by its constraints. Stallings (2005) highlighted that there are four concepts that relate to the channel capacity:

- Data rate in bits per second (bps), at which data can be communicated.
- Bandwidth of the transmitted signal as constrained by the transmitter and the nature of the transmission medium, expressed in cycles per second, or Hertz.
- The average level of noise over the communications path, and
- The rate at which errors occur.

If the channel can be considered noise free, the limitation of the data rate is the bandwidth of the signal. Nyquist states that if the rate of signal transmission is two times the bandwidth (*2B),* then a signal with frequencies no greater than *B* is sufficient to carry the signal rate. This assumption is based on a signal is represented by two discrete levels 0 or 1 or one bit, though in reality though different modulation techniques a signal can be represented by more than one bit. Therefore with multiple bit signalling, the Nyquist Bandwidth formulation for channel capacity becomes:

$$C = 2B \log_2 M$$

where *C* is the capacity of the channel in bits per second, *B* is the bandwidth of the channel in Hertz (Hz), and *M* is the number of discrete signal elements.

Now considering the presence of noise can corrupt one or more bits, then the given data rate will have effect on the error rate. As the data rate increases, bits become shorter in time and more bits will be effected by the given noise pattern, increasing the error rate.

From this it can be assumed that for a given level of noise and by increasing the signal strength, it would improve the ability to receive the correct data. The reasoning for this is the signal to noise ratio (S/N) expressed in decibels as:

$$S/N\, dB = 10 \log_{10} Signal/Noise$$

Therefore the signal-to-noise ratio sets the upper limit on the achievable data rate, and therefore corresponding channel capacity as expressed by Shannon as:

$$C = B \log_2 \left(1 + S/N\right)$$

The channel capacity referred to by Shannon is for an error free channel. Though Shannon proved that if the actual information rate on a channel is less than the error-free capacity, then it is theoretically possible to use a suitable signal code to achieve error free transmission through the channel.

5.3 Noise

A common impairment to the quality of a received signal is noise. Any received signal will consist of the transmitted signal modified by various distortions introduced as unwanted signals between the transmission of the signal and its reception. Stallings (2005) says that noise can be divided into four categories:

- Thermal noise
- Intermodulation noise
- Crosstalk
- Impulse noise

5.3.1 What is E_b/N_0

There is a parameter related to signal-to-noise that is more convenient in determining data rates and error rates and is the standard quality measure for digital communication system performance (Stallings, 2005). This parameter is the ratio of signal error per bit to noise power density per Hz, or E_b/N_0. The ratio E_b/N_0 is important because the Bit Error Rate (BER) for digital data is a decreasing function of this ratio. Therefore the performance in terms of BER versus *Eb/No,* also depends on the way in which the data is encoded onto the signal. For a given signal the noise in the channel is sufficient to alter the value of a bit, then for constant signal and noise strength, an increase in data rate increases the error rate. The advantage of *Eb/No* compared to *S/R* when determining the BER is that *Eb/No* does not depend on the bandwidth of the channel.

5.4 Channel Model

Now that the channel impulse response has been defined as a Rayleigh flat fading channel, a representation of the channel to be used for testing is shown in Figure 8.

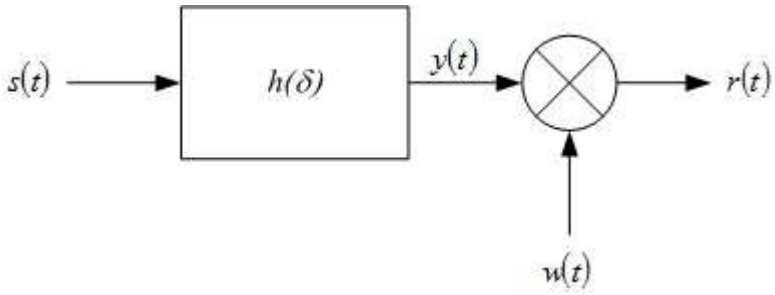

Figure 8 – Typical Chanel Model

5.4.1 Additive White Gaussian Noise (AWGN)

To simulate a mobile channel, the effects of multipath fading and noise on mobile channel need to be determined. The simplest channel model is the Additive White Gaussian Noise (AWGN) channel. In this channel, the desired signal is degraded by thermal noise associated with the physical channel itself, as well as electronics at the transmitter and receiver.

Also channels experience noise and the component *w(t)* is the Additive White Gaussian Noise (AWGN) that is to be added to the transmitted signal to simulate the addition of noise within the channel.

5.4.2 Rayleigh Fading Channel

As mentioned in chapter 4, a mobile channel can be represented by a variety of models depending on the characteristics of the channel. Even though the TDL channel model is the most widely accepted model for a 3G mobile channel, the complexity of the simulation is beyond the scope of this paper. Thus it will be considered that the bandwidth of the transmitted signal is narrow enough to consider that $s(t-\tau_i) \approx s(t)$ and can therefore be represented by a flat fading channel without a LOS component.

Thus the Rayleigh flat fading model is considered a good simulation of the wireless communication channel for the purpose of testing. The channel will be considered to be in a non LOS urban environment as the purpose is to research the error resilience of the H.264 encoded bitstream. As there is no LOS component, the Doppler spread of the scatterers can be represented by the Jakes model.

5.4.3 Modulation

There are several modulation techniques chose from including Frequency Shift Keying (FSK), Minimum Shift Keying (MSK) and Phase Shift Keying (PSK). The modulation technique chosen was four level PSK known as Quadrature Phase Shift Keying (QPSK), which uses phases separated by multiples of $\pi/2(90°)$. This produces two bits for every signal element;

Chapter 6

Results

5.5 Introduction

There are no definitive guides regarding the effective use of the error resilience tools used by the JM and JSVM reference software, so the purpose of the chapter is to see if these tools have an effect on the performance of the decoder on bitstreams that have been effect by channel errors.

The use of these tools hand an effect on the encoded file size, in that in some cases the bitstream would encounter more errors than a bitstream encoded with the default parameters. Some comparison of the using the JM and JSVM reference software was performed to make a comparison of their ability to decode bitstreams when containing errors.

The testing gave some interesting results even though the JM and JSVM reference decoder software has habit of crashing when encountering a bitstream that had enough errors to corrupt important sections of the bitstream.

5.6 JM Performance

The JM 15.1 reference software was tested using the extended profile using the RTP bitstream with one partition. The sequence was coded with only the first picture encoded as an I slice and the remainder in PBPB slice order with P slices being used as reference pictures. The akiyo_qcif sequence was used which has a resolution of 177 × 144 pixels that is broken up into 99 macroblocks. To keep coding times to minimum, the sequence encoded consisted of a 100 frames.

The encoding strategies used were compared against the default configuration which is one slice per picture and no error resilience. For the remainder of the tests, each frame was encoded as two slices per picture consisting of a maximum of 50 macroblocks, and then using the dispersed FMO technique over three slice groups and finally Data Partitioning using three partitions. The results of the first series of tests are shown in Figure 9.

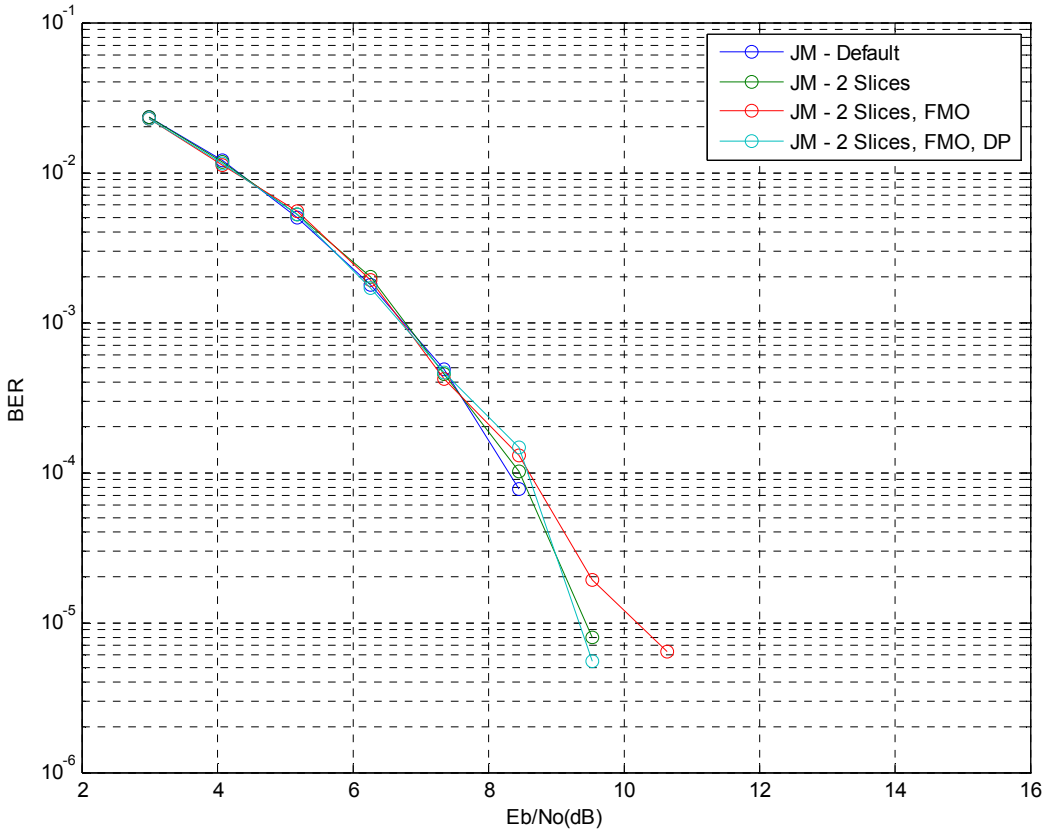

Figure 9 – Test Results JM Bitstream at E_b/N_0 of 3 to 15 dB, Tests 1 through 4

As can be seen the default configuration performed the best, achieving a BER of 7.5 $\times 10^{-5}$ at E_b/N_0 of 8.45 dB although the bitstream could not be decoded even with the small amount of errors present, it was not until the BER reached zero at 9.55 dB that sequence could be decoded. The use of two slices per picture increased the encoded bit rate from 26.05 kbit/s to 27.88 kbit/s, thus increasing the bit error probability. This resulted in a BER of 7.8 $\times 10^{-6}$ at 9.55 dB, but the sequence was able to be decoded.

Using FMO increased the BER to 1.9×10^{-5} at E_b/N_0 of 9.55 dB due to the encoded bit rate jumping to 31.56 kbit/s but the sequence was able to be decoded. The last test used DP which resulted in the increasing of the bit rate to 32.05 kbit/s, resulting in a BER of 5.5×10^{-5} at 9.55 dB. Even with the higher bit rate, the sequence again was able to be decoded.

The next series of tests performed with the JM Reference software was use two slices per picture each using dispersed FMO in three slice groups. For the first test the JM software encoder was modified to allow CABAC in the extended profile to compare it with the default CAVLC. For the next test macroblock line intra update was used to perform extra intra macroblock updates for one Group of Blocks (GOB) for every frame. This was again used the test, but resend picture parameter set flag was set to resend the PPS before every primary coded picture. The last test was enable rate control using the default rate control parameters. The results for these series of tests are shown in Figure 10.

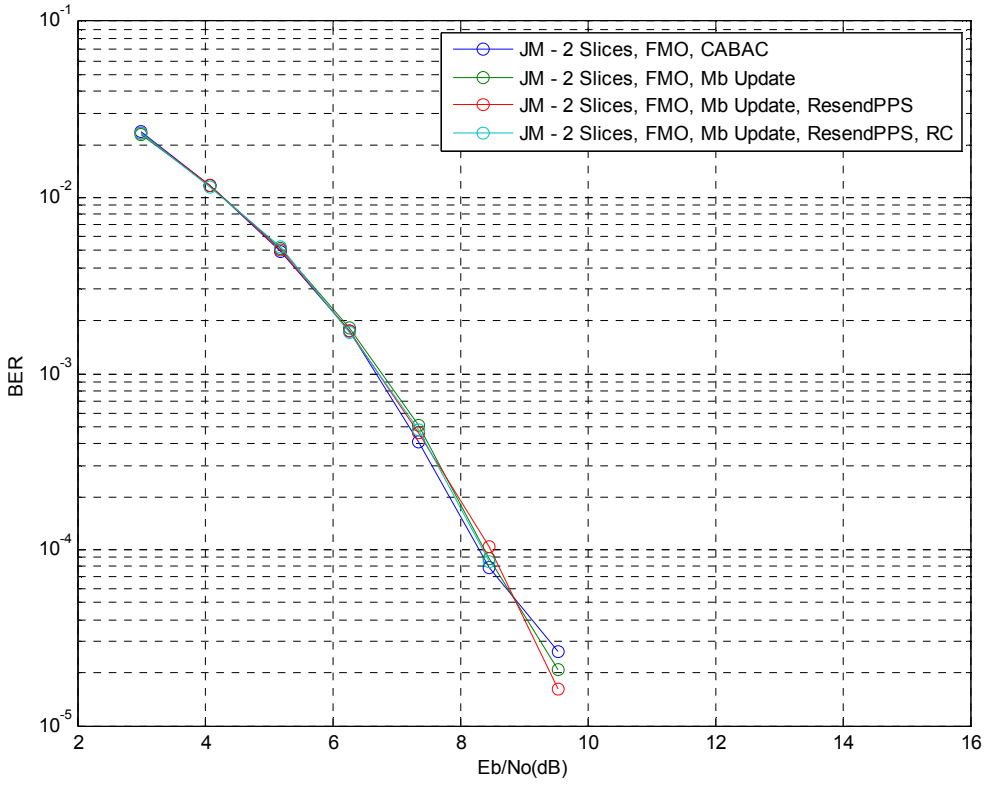

Figure 10 – Test Results JM Bitstream at E_b/N_0 of 3 to 15 dB, Tests 5 through 8

When CABAC was used the encoded stream had a bit rate of 30.45 kbit/s while achieving a BER of 2.5×10^{-5} at E_b/N_0 of 9.55 dB. The decoding of this sequence was possible but there were notable error concealments within the picture compared to the reference sequence as shown in Figure 11.

Figure 11 – Akiyo_qcif at E_b/N_0 9.55 dB, using CABAC

When extra intra macroblock updates was enabled, the encoded sequence bit rate significantly increased to 71.79 kbit/s, thereby resulting in a BER of 2×10^{-5} at 9.55 dB. Even with the high BER, the sequence was decoded, but again there were notable error corrections within the picture compared to the reference sequence as shown in Figure 12.

Figure 12 – Akiyo_qcif at E_b/N_0 9.55 dB, using extra MB Intra Updates

When using the resend PPS option, the bit rate achieved for the encoded bitstream was 72.74 kbit/s resulting in a BER of 1.6×10^{-5} at 9.55 dB, though the sequence was able to be decoded. Finally for JM, the rate control option was used, with the encoded bitstream having a bit rate of 49.36 kbit/s, which was close to the target bit rate of 45.02 kbit/s. This resulted in a BER of 8.6×10^{-5} at 8.45 dB and zero at 9.55 dB. For this test the sequence was decoded and while error concealment was visible as shown in Figure 13, the decode sequence mirrored the reference sequence with the errors observed partly due to the rate control employed by the encoder. This by far produced the best results for the JM codec when used in an error prone environment.

Figure 13 – Akiyo_qcif at E_b/N_0 9.55 dB, using Rate Control

5.7 JSVM Performance

To provide for comparative results the JSVM 9.18 reference software was tested. The JSVM reference software was tested using the extended profile for the base layer and scalable baseline profile for the scalable layers. The sequence was coded as specified in Table 1 for the single layer mode.

Reference frames:	4									
Format string:	A0P4B1B3b2R-0-0-0R-1+0-2									
Coding Types:	IDR	B	B	B	P	IDR	B	B	B	P …
Stored as reference:	1	1	0	1	1	1	1	0	1	1 …
Coding Order:	0	2	4	3	1	5	7	9	8	6 …

Table 1 – JSVM Single Layer Mode Coding Sequence

For scalable layer mode the, each temporal layer was encoded with the default slice order. The akiyo_qcif sequence was again used, with a difference in that two sequences were used. The akiyo_qcif sequence at a frame of 30 Hz was used for the single layer mode and for all layers L > 0 for the scalable layer mode, while an akiyo_qcif sequence at a frame of 15 Hz was used for the scalable layer mode for the base layer. Again to minimize coding times, the sequence encoded consisted of a 100 frames.

For the JSVM the first two tests where performed in single layer mode to produce AVC compatible bitstreams. The difference between the two was the symbol mode used being CAVLC for the first test and CABAC for the second. The next four tests were performed in scalable layer mode, the first using the default SVC coding parameters. For the remainder of the tests, each frame in scalable layers was encoded as two slices per picture consisting of a maximum of 50 macroblocks, and then using the foreground with leftover FMO technique over two slice groups. The FMO test sequences were encoded using CAVLC and CABAC symbol modes. The base layer does not support B slices, FMO, and/or CABAC when encoding, therefore these tests are only evaluating the scalable layers for error resilience. The results of the series of tests are shown in Figure 14.

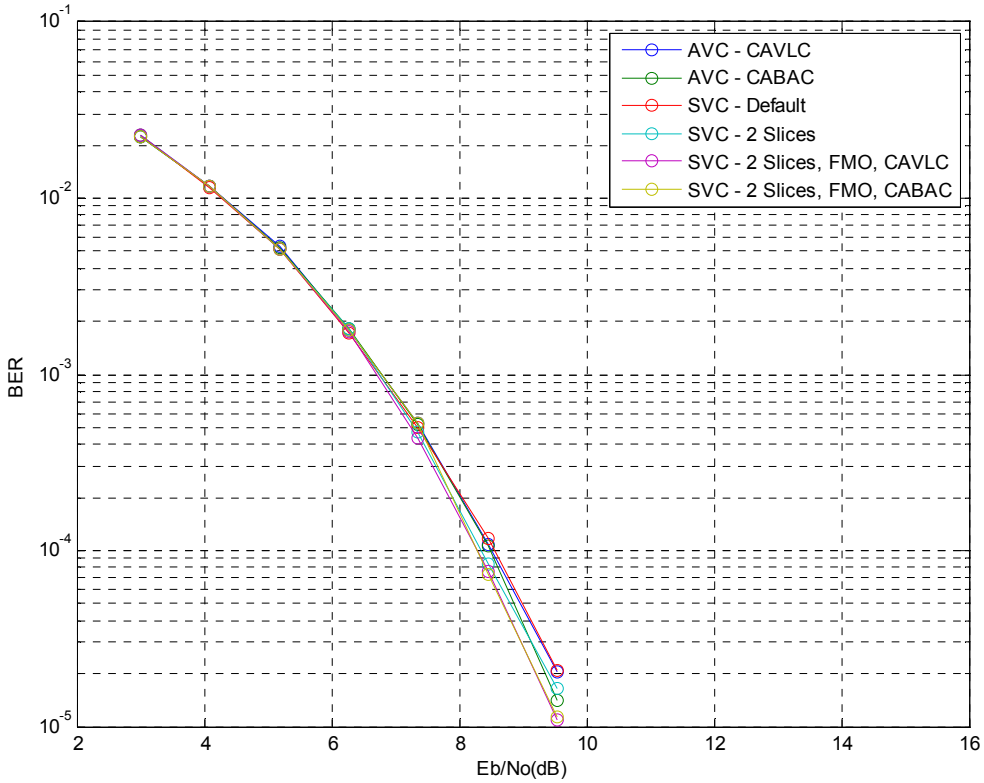

Figure 14 – Test Results JSVM Bitstream at E_b/N_0 of 3 to 15 dB, Tests 1 through 6

To perform a comparable comparison the bitstream extracted for the scalable layer mode test was DTQ(1,4,0) or Layer 8, the highest layer in the bit stream. For the first two tests the average bit rate was 117.5808 kbit/s and 106.9968 kbit/s achieving a BER of 2×10^{-5} and 1.5×10^{-5} respectably at 9.55 dB. Even though this was the case, only a few frames could be extracted before the bitstream collapsed. But from this the CABAC encoded sequence provided an improved bit rate.

The scalable layer mode encoded bitstream using default parameters, with the bit rates for each layer shown in Table 2. The BER achieved for this test was 2.1×10^{-5} at 9.55 dB, though no sequence could be extracted until the BER reached zero.

Layer	Bitrate (kbit/s)	Minimum Bitrate (kbit/s)
176x144 @ 1.8750	21.2186	21.2186
176x144 @ 3.7500	23.8108	23.8108
176x144 @ 7.5000	25.8552	25.8552
176x144 @ 15.0000	27.3528	27.3528
176x144 @ 1.8750	82.0586	82.0586
176x144 @ 3.7500	93.4362	93.4362
176x144 @ 7.5000	102.3096	102.3096
176x144 @ 15.0000	109.0368	109.0368
176x144 @ 30.0000	114.4320	114.4320

Table 2 – JSVM Layer Bitrates for Default Coding

The scalable layer mode encoded bitstream using two slices per picture, with the bit rates for each layer shown in Table 3. The BER achieved for this test was 1.6×10^{-5} at 9.55 dB, though no sequence could be extracted until the BER reached zero.

Layer	Bitrate (kbit/s)	Minimum Bitrate (kbit/s)
176x144 @ 1.8750	21.8314	21.8314
176x144 @ 3.7500	24.6600	24.6600
176x144 @ 7.5000	27.1176	27.1176
176x144 @ 15.0000	29.3880	29.3880
176x144 @ 1.8750	83.4921	83.4921
176x144 @ 3.7500	95.9862	95.9862
176x144 @ 7.5000	106.6536	106.6536
176x144 @ 15.0000	116.7480	116.7480
176x144 @ 30.0000	126.0144	126.0144

Table 3 – JSVM Layer Bitrates for Multiple Slices

The scalable layer mode encoded bitstream using two slices per picture using FMO and CAVLC encoding, with the bit rates for each layer shown in Table 4. The BER achieved for this test was 1.08×10^{-5} at 9.55 dB, though no sequence could be extracted until the BER reached zero.

Layer	Bitrate (kbit/s)	Minimum Bitrate (kbit/s)
176x144 @ 1.8750	21.8314	21.8314
176x144 @ 3.7500	24.6600	24.6600
176x144 @ 7.5000	27.1152	27.1152
176x144 @ 15.0000	29.3712	29.3712
176x144 @ 1.8750	84.1821	84.1821
176x144 @ 3.7500	97.3062	97.3062
176x144 @ 7.5000	109.2288	109.2288
176x144 @ 15.0000	121.8648	121.8648
176x144 @ 30.0000	135.1392	135.1392

Table 4 – JSVM Layer Bitrates for Multiple Slices, FMO, & CAVLC

The scalable layer mode encoded bitstream using two slices per picture using FMO and CABAC encoding, with the bit rates for each layer shown in Table 5. The BER achieved for this test was 1.1×10^{-5} at 9.55 dB, though this time the sequence could be extracted which no notable difference between the reference sequence and the extract sequence.

Layer	Bitrate (kbit/s)	Minimum Bitrate (kbit/s)
176x144 @ 1.8750	21.8314	21.8314
176x144 @ 3.7500	24.6600	24.6600
176x144 @ 7.5000	27.1152	27.1152
176x144 @ 15.0000	29.3784	29.3712
176x144 @ 1.8750	77.7171	84.1821
176x144 @ 3.7500	90.2723	97.3062
176x144 @ 7.5000	102.0144	109.2288
176x144 @ 15.0000	114.9168	121.8648
176x144 @ 30.0000	128.6184	135.1392

Table 5 – JSVM Layer Bitrates for Multiple Slices, FMO, & CABAC

The next series of tests performed was to extract the layers from the bitstream and test them individually over the error prone channel. From this, the scalable layers that provided the best results in decoding the sequence was layer DTQ(1,4,0) as shown in Figure 15 for encoded bitstreams using multiple slices per picture.

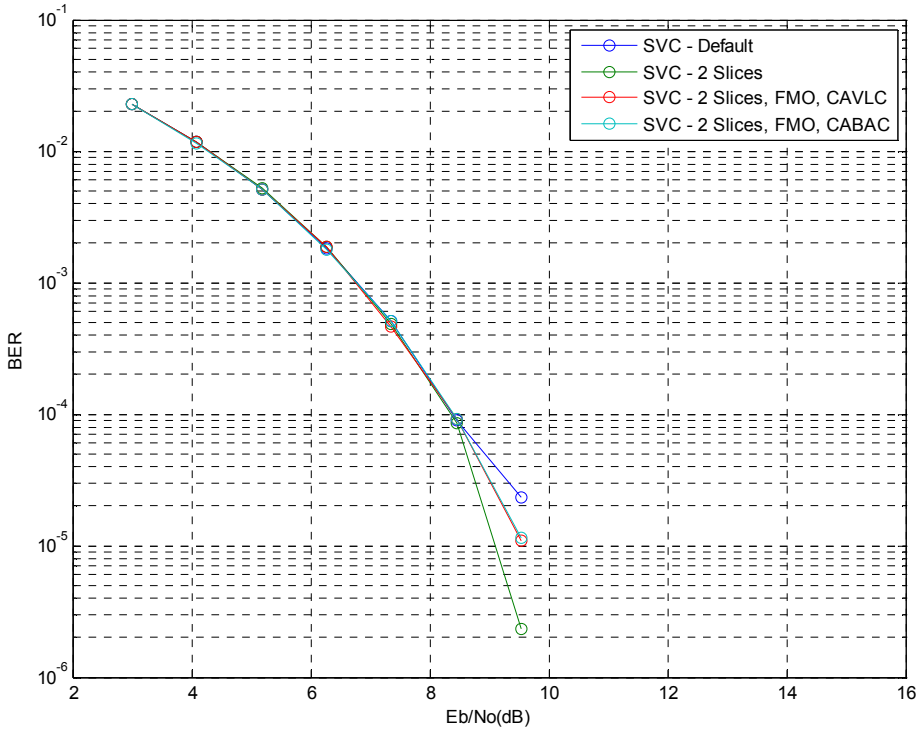

Figure 15 – Test Results JSVM Bitstream at E_b/N_0 of 3 to 15 dB layer DTQ(1,4.0)

For the base layer, the best performer was layer DTQ(0,2,0) as shown in Figure 16, with all four test bitstreams performing equally well, though the perceived quality of the picture is reduced due to the spatial and temporal reduction. The remainder of the layer figures are contained in Appendix C.

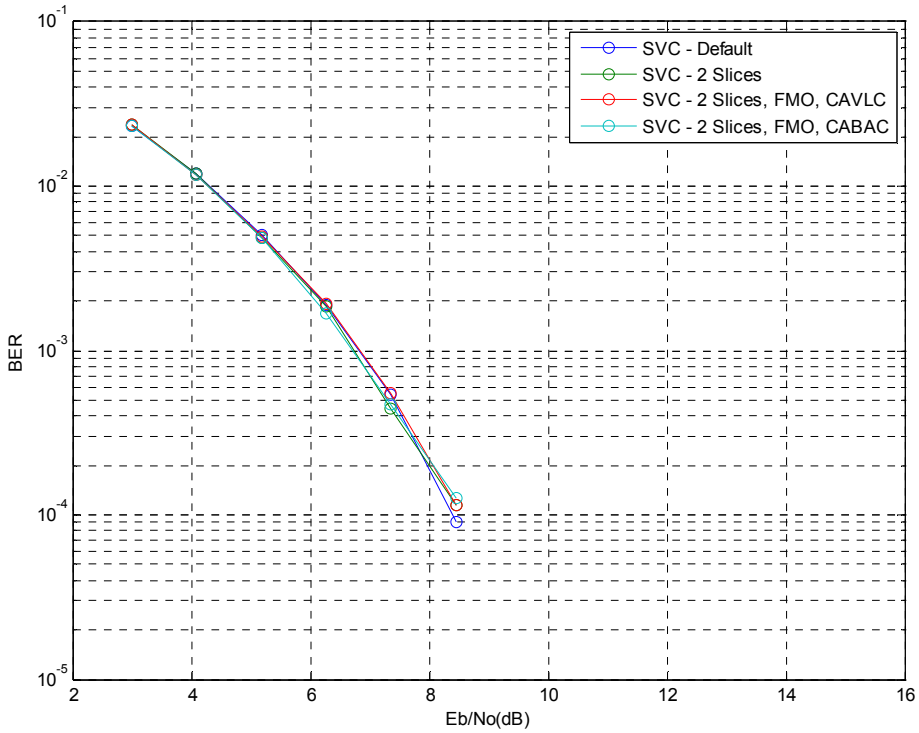

Figure 16 – Test Results JSVM Bitstream at E_b/N_0 of 3 to 15 dB layer DTQ(0,2.0)

Chapter 7

Conclusions

This research project investigated the error resilience of the International Telecommunication Union's (ITU) H.264/AVC Recommendation and Annex G Scalable Video Coding (SVC) over error prone communications channels. The aim was to test the encoded bitstreams by subjecting them to minor, mild, and severe channel errors.

The minor channel errors where produced by passing the bitstream though AWGN channel. The resultant Bit Error Rates (BER) of 10^{-4} experienced were consistent with those shown in Stallings (2005) for an AWGN channel for E_b/N_0 value of 8 to 9 dB. Once the errors were introduced, the JM 15.1 and JSVM 9.18 error resilient tools were employed to determine the error resilience of the encoded to these errors. The conclusion was that these tools performed well for mid channel errors though more investigation is required for some of the less known tools for Macroblock protection and Rate Control.

To determine if channel coding could reduce these errors, a QPSK soft demodulator was used with Turbo codes. From this the BER was reduced to zero for the AWGN channel, resulting in complete bitstreams being decoded even after being passed through a channel with an E_b/N_0 value of 3 dB.

The mild and severe channels errors where produced by using Rayleigh flat fading with the Jakes model. From this the resultant BER experienced was similar to those again shown in Stallings (2005). From results a bit error probability of approximately 50% was experienced, which could not be handled by the error resilient tools specified in the H.264 Recommendation.

Again some channel coding was attempted but the results proved inconclusive as the bitstream still had significantly high BER. This warrants further research in implementing Forward Error Correcting codes within the encoder and decoder to protect the vital parts of the bitstream such as the Sequence Parameter Sets, Picture Parameter Sets, and Slice headers.

A another significant advantage would be to further investigate channel correction coding such as LPDC and Turbo coding to a greater extent to see if a H.264 encoded bitstream could be protected and then recovered from a channel that produces severe channel errors.

Therefore there is a considerable amount of future work that can be conducted. With the use channel encoders and decoders, the processing speed of the decoding process becomes an issue. The encoder and decoder speed could be improved by implementing a multi thread encoder and decoder through the use of dynamic link libraries instead of the static libraries currently used by the JSVM reference software.

Bibliography

Biglieri, E 2005, *Coding for Wireless Channels*, Springer Science+Business Media, Inc.

Cavers, J.K 2002, *Mobile Channel Characteristics*, Kluwer Academic Publishers

Dent, P Bottomley, G.E Croft, T 1993, Jakes fading model revisited, *Electronics Letters*, vol. 29, issue. 24, pp. 1162 – 1163, viewed 26 Apr 09, IEEE Xplore, item 10.1049/el:19930777

Gans, M.J 1972, A power-spectral theory of propagation in the mobile-radio environment, *Vehicular Technology, IEEE Transactions on*, vol. , issue 1, pp. 27 – 38, viewed 20 Apr 09

Hao Chen, Zhen Han, Ruimin Hu & Ruolin Ruan 2008, 'Adaptive FMO selection strategy for error resilient H.264 coding', *International Conference on Audio, Language and Image Processing, ICALIP 2008,* pp. 868-872, viewed 20 Jan 09, IEEE Xplore, item 10.1109/ICALIP.2008.4589969

International Telecommunication Union 2007*, Advanced video coding for generic audiovisual services*, ITU-T Recommendation H.264, International Telecommunication Union, Geneva, Switzerland

Iskander, C (Hi-Tek Multisystems) 2008, A MATLAB® based Object-Oriented Approach to Multipath Fading Channel Simulation, MATLAB® Central, viewed 09 Apr 09 <http://www.mathworks.com/matlabcentral/fileexchange/18869>

Jakes, W.C (ed.) 1974, Microwave Mobile Communications, John Wiley & Sons

Kumar, S, Liyang Xu, Mandal & MK, Panchanathan, S 2006, 'Error Resiliency Schemes in H.264/AVC Standard', *Journal of Visual Communication and Image Representation***,** vol. 17, issue 2, pp. 425-450

Lin Liu, Sanyuan Zhang, Xiuzi Ye & Yin Zhang 2005, Error Resilience Schemes of H.264/AVC for 3G Conversational Video Services', *Proceedings of the Fifth International Conference on Computer and Information Technology 2005 (CIT'05'),* pp. 657-661, viewed 31 Oct 08, IEEE Xplore, item 10.1109/CIT.2005.113

Moon, TK 2005, *Error Correction Coding Mathematical Methods and Algorithms*, John Wiley & Sons, New Jersey.

Ogunfunmi, T & Huang, WC 2005, 'A flexible macroblock ordering with 3D MBAMAP for H.264/AVC', *IEEE International Symposium on Circuits and Systems, 2005, ISCAS 2005,* vol. 4, pp. 3475- 3478, viewed 23 Dec 08, IEEE Xplore, item 10.1109/ISCAS.2005.1465377

Richardson, IEG 2004, *H.264 and MPEG-4 Video Compression: Video Coding for Next-generation Multimedia*, John Wiley & Sons.

Schwarz, H, Marpe, D & Wiegand, T 2006, Overview of the Scalable H.264/MPEG4-AVC Extension, *IEEE International Conference on Image Processing, 2006*, pp. 161-164, viewed 18 Mar 09, IEEE Xplore, item 10.1109/ICIP.2006.312374

Spinsante, S, Gambi, E & Falcone, D 2007, Scalable extension of the H.264 video codec: Overview and performance evaluation, *15th International Conference on Software, Telecommunications and Computer Networks, 2007. SoftCOM 2007*, pp. 1-5, viewed 18 Mar 09, IEEE Xplore, item 10.1109/SOFTCOM.2007.4446126

Stallings, W 2005, *Wireless Communications and Networks*, Pearson/Prentice Hall

Stockhammer, T & Bystrom, M 2004, 'H.264/AVC data partitioning for mobile video communication'*, 2004 International Conference on Image Processing, ICIP '04'*, vol. 1, pp. 545-548, viewed 14 Jan 09, IEEE Xplore, item 10.1109/ICIP.2004.1418812

Sklar, B 1997, Rayleigh fading channels in mobile digital communication systems, Part I Characterization, *Communications Magazine, IEEE*, Vol. 35, No. 7, pp. 90-100, viewed 15 Apr 09

Sullivan, G & Wiegand, T 2004, 'Video Compression - From Concepts to the H.264/AVC Standard', *Proceedings of the IEEE,* vol. 93, no. 1, pp. 18-31, viewed 18 Feb 2009, IEEE Xplore, item ISSN: 0018-9219

Tourapis, AM, Sullivan, G, Sühring, K, Oelbaum, T & Leontaris, A 2009, *H.264/MPEG-4 AVC Reference Software Manual (JVT-AD010),* Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG (ISO/IEC JTC1/SC29/WG11 and ITU-T SG16 Q.6)

Unknown 2008, *JSVM Software Manual (JSVM 9.16),* Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG

Wiegand, T, Sullivan, GJ, Bjøntegaard, G & Luthra, A 2003, 'Overview of the H.264/AVC Video Coding Standard', *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560-576, viewed 15 Jan 09, IEEE Xplore, item ISSN: 1051-8215

Appendix A – Project Specification

University of Southern Queensland

FACULTY OF ENGINEERING AND SURVEYING

ENG4111/4112 Research Project
PROJECT SPECIFICATION

FOR: **Timothy Wise**

TOPIC: Video over Wireless - Error Resilient H.264 Coded Video Transmission over Wireless Channels

SUPERVISOR: Wei Xiang

SPONSORHSIP: USQ

PROJECT AIM: The project aim is to investigate the error resilience performance of the H.264 standard for coded video under minor, mild and severe channel errors. The scope of the project also involves setting up a software platform to evaluate the performance of H.264 video transmission over wireless channels for sensitivity and quality of service of the coded video stream.

PROGRAMME: **Issue A, 19 Mar 09**

1. Research the hybrid video compression architecture relating to the ITU-T Recommendation H.264.
2. Research the error resilience tools and features employed by the ITU-T Recommendation H.264.
3. Develop a program, written in C/C++/C#, utilising available H.264 Reference Software to evaluate H.264 transmission over simulated wireless channels.
4. Evaluate the software by subjecting the encoded bitstream to minor, mild and severe channel errors.
5. Evaluate the software for mobile applications using Scalable Video Coding (SVC).

As Time Permits

6. Optimise the software for use with mobile technologies, where available memory and processor speeds are factors in performance.

AGREED _____ (student) _____ (supervisor)

Date: 19/03/2009 Date: (/ 4 / 2009

Co-examiner: _____ 02/04/09

Appendix B – Source Code

H.264.cpp

```
/**********************************************************************
 * H.264.cpp : Defines the entry point for the console application.
 *
 *
 * Author: Timothy Wise  (0050055729)
 * Date:   10 Jul 2009
 * --------------------------------------------------------------------
 *
 * This application simulates a communication channel
 * A file is read in and converted to binary bits and
 * then sent over the channel.
 *
 * --------------------------------------------------------------------
 */

#include "stdafx.h"
#include "ChannelModel.h"

// it++ communication libaray
#include <itpp/itcomm.h>

using namespace itpp;


// These lines are needed for use of stdio
using std::cin;
using std::cout;
using std::endl;
using std::ofstream;


/*********************************
 * The main function entry point  *
 *********************************/

int _tmain(int argc, _TCHAR* argv[])
{
      uint8_t data;          // A byte
      wchar_t *temp;
      char name[128];        // Name of input file
      int index = 0;
      bvec bvTemp, bvData;   // Binary vectors

      // Read in H.264 bitstream file from command line
      if (argc >= 2)
      {
            temp = argv[1];
      }
      else
      {
          cout << "Useage: h.264 <name> " << endl;
            cout << "  <name> The input h.264 bitstream file";
            cout << endl;
```

```
            exit(0);
        }

        // Convert from a wchar_t pointer to a char array
        size_t origsize = wcslen(temp) + 1;
        size_t convertedChars = 0;
        wcstombs_s(&convertedChars, name, origsize, temp, _TRUNCATE);

        // Open binary input stream
        bifstream inFile(name);

        // Size of the input file in bytes
        int size = inFile.length();

       /*******************************************
        * Read in the file byte by byte and convert *
        * to a binary string                        *
        *******************************************/

        while (index < size)
        {
            inFile >> data;      // Read in a byte

            bvTemp = dec2bin(8, data);  // Convert to bits

            // Store in binary vector
            bvData = concat(bvData, bvTemp);

            index++;
        }

        // Close the input file
        inFile.close();

        // Create the channel model
        ChannelModel *Channel = new ChannelModel();

        //Channel->InitSpreadingCodes(4,4);

        // Simulate for 12 Eb/N0 values.
        vec EbN0dB = linspace(3, 15, 12);
        Channel->SetChannelNoise(EbN0dB);

        // Transmit the bitstream over the channel
        Channel->Transmit(bvData, AWGNChannel, name);

        // Print out the BER results
        Channel->getResults();

        delete Channel;
        return 0;
}
```

ChannelModel.h

```
/*************************************************************************
 * ChannelModel.h
 *
 *
 * Author: Timothy Wise  (0050055729)
 * Date:   10 Aug 2009
 * -----------------------------------------------------------------------
 *
 * The header file for the channel model class.  his class
 * implements different channel models
 *
 * -----------------------------------------------------------------------
 */

#ifndef ChanMod_H
#define ChanMod_H

#pragma once

#include <itpp/itcomm.h>
#include "JakesChannel.h"
#include "ChannelCoder.h"

using namespace itpp;

// These lines are needed for use of:
using std::cout;
using std::endl;
using std::ofstream;

// The channel models to support
enum Channels
{
      FIRChannel,     // Finite Impulse Response
      TDLChannel,     // Tapped Delay Line
      AWGNChannel,    // Adative White Guassian Noise
      RICEChannel,    // Rice Fading Channel
      JAKESChannel    // Jakes Fading Channel
};


/*
 * The ChannelModel class
 */
class ChannelModel
{
private:
      bool spread;    // Specify if spreading is to be used

      vec N0;         // The AWGN noise variance
      vec EbN0dB;     // The Eb/No values to use

      vec ber;        // The Bit-Error-Rate
      vec err;        // The number error bits received
      vec good;       // The number of good bits received
      vec total;      // The total bits recieved
```

```
        BERC berc;      // The bit error counter

        // The multicode spreading of the signal
        Multicode_Spread_2d mcSpread;

        // The channels
        TDL_Channel tdlChannel;    // TDL Channel
        AWGN_Channel awgnChannel;  // AWGN Channel
        JakesChannel jakesChannel; // Jakes Fading Channel
        ChannelCoder channelCoder; // The channel coder

public:
    ChannelModel(void);    // Default Constructor
    ~ChannelModel(void);   // Defaulst Destructor

    void getResults();
    void SetChannelNoise(vec noise);

    void InitJakesChannel(double normDoppler);
    void InitSpreadingCodes(int factor, int numCodes);
    void InitTDLChannel(const CHANNEL_PROFILE profile, double chip,
double normDoppler);

    int Transmit(bvec uncodedBits, const Channels channel, char*
name);
};

#endif // #ifndef ChanMod_H
```

ChannelModel.cpp

```
/*****************************************************************
 * ChannelModel.cpp
 *
 *
 * Author: Timothy Wise  (0050055729)
 * Date:   05 Aug 2009
 *---------------------------------------------------------------
 *
 * This class is used to model the channel
 * Some channels implemented are the AWGN, Jakes, and TDL
 * Expansion is provided to model Rice and FIR channels
 *
 *---------------------------------------------------------------
 */

#include "StdAfx.h"
#include "ChannelModel.h"


// Default constructor
ChannelModel::ChannelModel(void)
{
    spread = false;   // Don't perform spreading unless init

    vec noise("0");

    SetChannelNoise(noise);  // Set defualt AWGN noise to 0.0
}


// Default destructor
ChannelModel::~ChannelModel(void)
{

}


/*******************************************************
 * ChannelModel::SetChannelNoise(vec noise)
 *
 * This method sets the channel Eb/N0 noise values
 *
 * Input:
 *     vec noise - The vector containing the noise values
 *
 */

void ChannelModel::SetChannelNoise(vec noise)
{
    double Ec = 1.0; //The transmitted energy per QPSK symbol is 1.
    double Es = Ec / 2.0;   //The transmitted energy per bit is 0.5.
    double k = 1.0/std::log(4.0);

    EbN0dB = noise;   //Simulate for 10 Eb/N0 values from 0 to 9 dB.

    vec EbN0 = pow(10, EbN0dB/10); //Calculate Eb/N0 in a linear
scale instead of dB.
```

```
    N0 = Es * pow(EbN0, -1.0);      //N0 is the variance of the
(complex valued) noise.

    // Set up the bit error counters for the number of
    // noise figures used

    ber.set_size(EbN0dB.size(), false);      // Bit error counter
    ber.clear();

    err.set_size(EbN0dB.size(), false);    // Number of bits in
error counter
    err.clear();

    good.set_size(EbN0dB.size(), false);   // Number of good bits
counter
    good.clear();

    total.set_size(EbN0dB.size(), false);  // Total number of bit
counter
    total.clear();
}


/*************************************************************
 * ChannelModel::InitSpreadingCodes(int factor, int numCodes)
 *
 * This method initialises the CDMA spreading codes
 *
 * Inputs:
 *     int factor - The code spreading factor
 *     int numCodes - The number of codes to use
 *
 */

void ChannelModel::InitSpreadingCodes(int factor, int numCodes)
{
     spread = true;       // Perform spreading

    //Initialize the spreading:
    int SF = factor;                    // The spreading factor is 4
    int Ncode = numCodes;               // Number of codes in the
multi-code spread
    smat spreadCodesI, spreadCodesQ;    // The I and Q spreading
codes

    // Set the spreading codes:
    spreadCodesI.set_size(Ncode, SF, false);
    spreadCodesQ.set_size(Ncode, SF, false);

    // Calculate the spreading codes
    smat allCodes = to_smat(hadamard(SF));

    for (int sc = 0; sc < Ncode; sc++)
    {
        spreadCodesI.set_row(sc, allCodes.get_row(sc));
        spreadCodesQ.set_row(sc, allCodes.get_row(sc));
    }
    mcSpread.set_codes( to_mat(spreadCodesI), to_mat(spreadCodesQ));
}


/*******************************************************
 * ChannelModel::InitJakesChannel(double normDoppler)
```

```
 *
 * This method sets the Jakes channel notmalised Doppler
 *
 * Input:
 *     double normDoppler - The normalised Doppler fd*T
 *
 */

void ChannelModel::InitJakesChannel(double normDoppler)
{
    jakesChannel.setNormDoppler(normDoppler);
}


/*******************************************************
 * ChannelModel::InitTDLChannel(const CHANNEL_PROFILE profile,
double chip, double normDoppler)
 *
 * This method initalises the TDL Channel model
 *
 * Input:
 *     const CHANNEL_PROFILE profile - The COST259 channel profile
 *     double chip - The transmitter chip rate
 *     double normDoppler - The normalised Doppler fd*T
 *
 */

void ChannelModel::InitTDLChannel(const CHANNEL_PROFILE profile,
double chip, double normDoppler)
{
    // set sampling time at a half of chip rate (0.5 / chip)
    double Ts = 0.5/chip;

    // select the channel profile model
    Channel_Specification channelSpec(profile);

    // initialize with the defined channel profile
    tdlChannel.set_channel_profile(channelSpec, Ts);

    // set the normalized Doppler; fading type will be set to
Correlated
    // and Rice_MEDS method will be used (default settings)
    tdlChannel.set_norm_doppler(normDoppler);
}


/*******************************************************
 * ChannelModel::::getResults()
 *
 * This method returns the simulation results
 *
 *
 */

void ChannelModel::getResults()
{
    char out[128];

    //Print results:
    cout << endl;
    cout << " EbN0dB  |  Total  | Correct | Errors |  BER " << endl;
      cout << "-----------------------------------------"     <<
endl;
```

```
      for (int i=0; i < EbN0dB.length(); i++)
      {
            sprintf_s(out, " %5.2f   |  %5.0f |  %5.0f | %5.0f  | ",
                      EbN0dB(i),
                      total(i),
                      good(i),
                      err(i));

            cout << out << ber(i) << endl;
    }
}


/*******************************************************
 * ChannelModel::Transmit(bvec bitsToSend, const Channels channel,
char* name)
 *
 * This method starts the simulation of the chossen channel
 *
 * Input:
 *     bvec bitsToSend - The bits to transmit
 *     const Channels channel - The channel model to use
 *     char* name - The name of the received file
 *
 */

int ChannelModel::Transmit(bvec bitsToSend, const Channels channel,
char* name)
{
    // Scalars
    int index;
    uint8_t data;

    // Vectors:
    bvec bvTemp(8);              // Temporary bin vec of 8 bits
    bvec receivedBits;          // Received binary bits from decoder
    cvec receivedAWGN;          // Complex AWGN channel received
signal
    cvec receivedSignal;        // Complex channel received signal
    cvec receivedSymbols;       // Complex received symbols from de-
spreader
    cvec transmittedSignal;     // Complex transmitted chips from
spreader
    cvec transmittedSymbols;    // Complex transmitted symbols from
encoder

    char filename[28];          // Name for the output file
    bofstream outFile;          // Output file stream


      // Channel coefficients are returned in the 'coeff' array of
complex values
    Array<cvec> coeff;

    channelCoder.SetMethod(NONE);

      // Loop through to simulate all noise variance values
specified
    for (int i = 0; i < N0.length(); i++)
    {
        cout << endl << "Simulating point nr " << i + 1 << " with a
variance of ";
```

```
        cout << std::sqrt(N0(i)) << endl;

        berc.clear();               // Clear the bit error counter
        awgnChannel.set_noise(N0(i));   // Set the AWGN noise
variance

        channelCoder.Encode(bitsToSend, transmittedSymbols);

        // This is where we do the multi-code spreading
        if (spread == true)
        {
            transmittedSignal = mcSpread.spread(transmittedSymbols);
        } else {
            transmittedSignal = transmittedSymbols;
        }

        // Pass the signal through the selected channel
        switch (channel)
        {
            case FIRChannel:
                receivedSignal = transmittedSignal;
                break;

            case TDLChannel:
                receivedSignal = tdlChannel(transmittedSignal,
coeff);
                break;

            case AWGNChannel:
                receivedSignal = transmittedSignal;
                break;

            case RICEChannel:
                receivedSignal = transmittedSignal;
                break;

            case JAKESChannel:
                receivedSignal =
jakesChannel.Generate(transmittedSignal);
                break;

            default:
                receivedSignal = transmittedSignal;
                break;
        }

        // Simulate noise on the signal
        receivedAWGN = awgnChannel(receivedSignal);

        // This is where we do the multi-code spreading
        if (spread == true)
        {
            //The multi-code despreading:
            //The second argument tells the despreader that the
offset is zero chips.
            //This offset is usefull on channels with delay.
            receivedSymbols = mcSpread.despread(receivedAWGN, 0);
        } else {
            // or just receive the symbols
            receivedSymbols = receivedAWGN;
        }

        channelCoder.Decode(receivedSymbols, receivedBits, N0(i));
```

```
        // Count the number of bit errors
        berc.count(bitsToSend, receivedBits);

        ber(i) = berc.get_errorrate();     // Get the error rate
        err(i) = berc.get_errors();        // Get the number bits in
error
        good(i) = berc.get_corrects();     // Get the number of
correct bits
        total(i) = berc.get_total_bits();  // Get the total number
of bits sent

        cout << "Bit Error Rate (BER) = " << berc.get_errorrate() <<
endl;

        sprintf_s(filename, "test%i_%s",i, name);
        outFile.open(filename);

        index = 0;

        // Convert bits back into bytes
        // and writes to the output file
        while (index < receivedBits.length())
        {
            for (int j=0; j < 8; j++)
            {
                bvTemp[j] = receivedBits.get(index);
                data = bin2dec(bvTemp);
                index++;
            }
            outFile << data;
        }
        outFile.close();
    }
    return 0;
}
```

ChannelCoder.h

```
/*************************************************************************
 * ChannelCoder.h
 *
 *
 * Author: Timothy Wise  (0050055729)
 * Date:   20 Sep 2009
 *
 * ----------------------------------------------------------------------
 *
 * This the header file for the channel coder class.  This class
 * implements different channel coders
 *
 * ----------------------------------------------------------------------
 */


#ifndef ChanCoder_H
#define ChanCoder_H


#pragma once


#include <itpp/itcomm.h>

using namespace itpp;


// Enummeration of the coders
enum Coders
{
      NONE,
      RS,
      TURBO,
      LPDC
};


/*
 * The ChannelCoder class
 */

class ChannelCoder
{
private:
    Coders codeMethod;  // The coding method

    // Classes:
    QPSK qpsk;                  // The QPSK modulator/demdulator
    Turbo_Codec *Turbo;         // The Turbo coder
    Reed_Solomon *ReedSolomon;  // The reed-solomon coder

public:
    ChannelCoder(void);  // Default Constructor
    ~ChannelCoder(void); // Default Destructor
```

```
    // Method to set coding method
    void SetMethod(const Coders method);

    // Method to encode the bits
    void Encode(const bvec &input, cvec &output);

    // Method to decode the recieved signal
    void Decode(const cvec &input, bvec &output, double N0);
};

#endif // #ifndef ChanCoder_H
```

ChannelCoder.cpp

```
/*****************************************************************
 * ChannelCoder.cpp
 *
 *
 * Author: Timothy Wise  (0050055729)
 * Date:   20 Sep 2009
 *
 * ---------------------------------------------------------------
 *
 * This class provides for the channel coder
 * Some hannle codes implement include Reed-Solomon and Turbo codes
 * Further expansion for LPDC is provided
 *
 * ---------------------------------------------------------------
 */

#include "StdAfx.h"
#include "ChannelCoder.h"


// Default Constructor
ChannelCoder::ChannelCoder(void)
{
	codeMethod = NONE;

	Turbo = NULL;
	ReedSolomon = NULL;
}


// Default Destructor
ChannelCoder::~ChannelCoder(void)
{
	delete Turbo;
	delete ReedSolomon;
}


/*******************************************************
 * ChannelCoder::SetMethod(const Coders method)
 *
 * This method sets the encoder and decoder to use
 *
 * Input:
 *     const Coders method - The coding method to use
 *
 */

void ChannelCoder::SetMethod(const Coders method)
{
	int m = 4;		//Reed-Solomon parameter m
	int t = 2;		//Reed-Solomon parameter t

	ivec generator(2);		// Turbo coder generator
	int constraintLength;		// Turbo coder constraint length
	ivec interleaverSequence; // Turbo interleaver sequence
```

```
        // The coding method to use
        codeMethod = method;

        // Initalise the appropiate coder
        switch (codeMethod)
        {
            case NONE:
            default:
                break;

            case RS:
                if (ReedSolomon == NULL)
                {
                    ReedSolomon = new Reed_Solomon(m, t);
                }
                break;

            case TURBO:
                generator(0) = 013;
                generator(1) = 015;

                constraintLength = 4;
                interleaverSequence =
wcdma_turbo_interleaver_sequence( 320 );

                if (Turbo == NULL)
                {
                    Turbo = new Turbo_Codec();
                }
                Turbo->set_parameters(generator,
                                      generator,
                                      constraintLength,
                                      interleaverSequence);
                break;

            case LPDC:
                break;
        }
}


/*************************************************************
 * ChannelCoder::Encode(const bvec &input, cvec &output)
 *
 * This method performs the channel encoding
 *
 * Inputs:
 *     const bvec &input - A reference to binary input vector
 *     const bvec &output - A reference to binary output vector
 *
 */

void ChannelCoder::Encode(const bvec &input, cvec &output)
{
      bvec codedBits;     // A vector of binary coded bits

      switch (codeMethod)
      {
          case NONE:
          default:
              // No encoding performed
              codedBits = input;
              break;
```

```
            case RS:
                // Encode the binary signal
                ReedSolomon->encode(input, codedBits);
                break;

            case TURBO:
                // Encode the binary signal
                Turbo->encode(input, codedBits);
                break;
        }
        // Modulate the bits to send
        qpsk.modulate_bits(codedBits, output);
}


/*************************************************************************
 * ChannelCoder::Decode(const bvec &input, cvec &output, double N0)
 *
 * This method performs the channel decoding
 *
 * Inputs:
 *     const bvec &input - A reference to binary input vector
 *     const bvec &output - A reference to binary output vector
 *     double N0 - The noise variance to use in the soft demodulator
 *
 */

void ChannelCoder::Decode(const cvec &input, bvec &output, double
N0)
{
        double Ec = 1.0;        // The transmitted energy per QPSK
symbol is 1.
        bvec receivedBits;      // Received binary bits from demodulator
        vec receivedSignal;     // Complex channel received signal

        switch (codeMethod)
        {
            case NONE:
            default:
                // Demodulate the symbols to bits
                output = qpsk.demodulate_bits(input);
                break;

            case RS:
                // Demodulate the symbols to bits
                receivedBits = qpsk.demodulate_bits(input);

                // Decode the the received signal
                ReedSolomon->decode(receivedBits, output);
                break;

            case TURBO:
                // Demodulate the symbols to soft bits
                receivedSignal = qpsk.demodulate_soft_bits(input, N0);

                // Decode the the received signal
                Turbo->set_awgn_channel_parameters(Ec, N0);
                Turbo->decode(receivedSignal, output);
                break;
        }
}
```

JakesChannel.h

```
/*************************************************************************
 * JakesChannel.h
 *
 *
 * Author: Timothy Wise  (0050055729)
 * Date: 21 Sep 2009
 * -----------------------------------------------------------------------
 *
 * The Class header file for the JakesChannel class
 *
 * -----------------------------------------------------------------------
 */

#ifndef Jakes_H
#define Jakes_H


#pragma once


#include <itpp/itcomm.h>

using namespace itpp;


/*
 * The JakesChannel class
 */

class JakesChannel
{

protected:
    bool initFlag;       // Is the generator initialised

    int numFreqs;        // Number of Doppler frequecies
    double bFreq;        // Doppler frequencies
    double bDoppler;     // Normalised maximum Doppler frequency

    vec Amp;             // Doppler amplitudes
      vec Theta1;          // Doppler real phases
    vec Theta2;          // Doppler imaginary phases

public:
    // Default constructor
    JakesChannel();

    // Destructor
    ~JakesChannel(void) {}

    // Initalise the generator
    void Initalise(void);

    // Set number of Doppler frequencies
    void setNoFrequencies(int numFreq);

    // Set the normalised Doppler
```

```
    void setNormDoppler(double normDoppler);

    // Get the number of Doppler frequencies
    int getNoFrequencies() const { return numFreqs; }

    // Return the normalised Doppler
    double getNormDoppler() const { return bDoppler; }

    // Generate a no_samples values from the fading process
    cvec Generate(cvec &input);
};

#endif // #ifndef Jakes_H
```

JakesChannel.cpp

```
/*****************************************************************
 * JakesChannel.cpp
 *
 *
 * Author: Timothy Wise  (0050055729)
 * Date:   21 Sep 2009
 * ---------------------------------------------------------------
 *
 * This class implements a Jakes model fading genertor
 *
 * ---------------------------------------------------------------
 */

#include "StdAfx.h"
#include "JakesChannel.h"

// Default Construtor
JakesChannel::JakesChannel()
{
      numFreqs = 16;       // Number of Doppler Frequencies
      bDoppler = 0.1;      // The Doppler Spread
}


/*******************************************************
 * ChannelCoder::Initalise(void)
 *
 * This method initalises the Jakes model
 *
 *
 */

void JakesChannel::Initalise(void)
{
    // Calculate the Doppler frequencies
    bFreq = 2*pi*bDoppler;

    // Calculate the speading angles
    Theta1 = randu(numFreqs)*2*pi;
    Theta2 = randu(numFreqs)*2*pi;

    // Calculate the doppler amplitudes
    Amp = (2*pi*numFreqs-pi+(randu(numFreqs)*2*pi))/(4*numFreqs);

    initFlag = true; // generator ready to use
}


/*******************************************************
 * ChannelCoder::setNoFrequencies(int numFreq)
 *
 * This method sets the number Doppler frequencies
 *
 * Input:
 *    int numFreq - The number of frequencies
 */
```

```
void JakesChannel::setNoFrequencies(int numFreq)
{
      it_assert(numFreq >= 7,
            "Jakes: Too low number of Doppler frequencies");

      numFreqs = numFreq;
      initFlag = false;
}


/*********************************************************
 * ChannelCoder::setNormDoppler(double normDoppler)
 *
 * This method sets the number normalised Doppler
 *
 * Input:
 *    double normDoppler - The normalised Doppler
 */

void JakesChannel::setNormDoppler(double normDoppler)
{
      it_assert((normDoppler > 0) && (normDoppler <= 1.0),
            "Jakes: Normalised Doppler out of range");

      bDoppler = normDoppler;
      initFlag = false;
}


/*********************************************************
 * ChannelCoder::Generate(cvec &input)
 *
 * This method sets generates the Jakes coeeficants and
 * applies them to the input signal
 *
 * Input:
 *    cvec &input - The complex vector input signal
 */

cvec JakesChannel::Generate(cvec &input)
{
    vec x;
    vec y;
    cvec output;

    if (initFlag == false)
        Initalise();

    int noSamples = input.size();

    // Set output vectors sizes
    // and fill with zeros
    x.set_size(noSamples, false);
    y.set_size(noSamples, false);
    output.set_size(noSamples, false);

    double c = 2.0/numFreqs;

    // Caculate the sum of sinusoids
    // using Jakes method
    for (int i = 0; i < noSamples; i++)
    {
        x(i) = sum(std::sqrt(c)*cos(bFreq*i*cos(Amp)+Theta1));
```

```
            y(i) = sum(std::sqrt(c)*cos(bFreq*i*cos(Amp)+Theta2));
        }

        x = elem_mult(x,x);   // simular to the MATLAB .*
        y = elem_mult(y,y);


        // Calculate correlated fading samples
        for (int i = 0; i < noSamples; i++)
        {
            output(i)= std::sqrt(x(i)+y(i))/std::sqrt(2.0);
        }

        // Apply to the input signal
        return(elem_mult(input, output));
    }
```

Appendix C – BER Results JSVM Layers

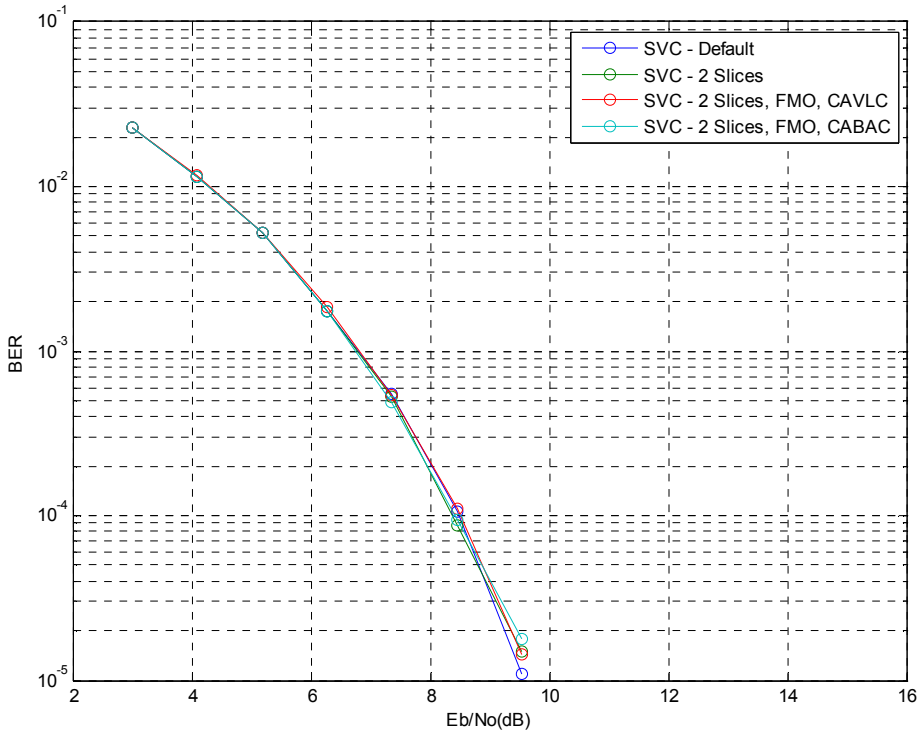

Figure 17 – Test Results JSVM Bitstream at E_b/N_0 of 3 to 15 dB layer DTQ(1,3,0)

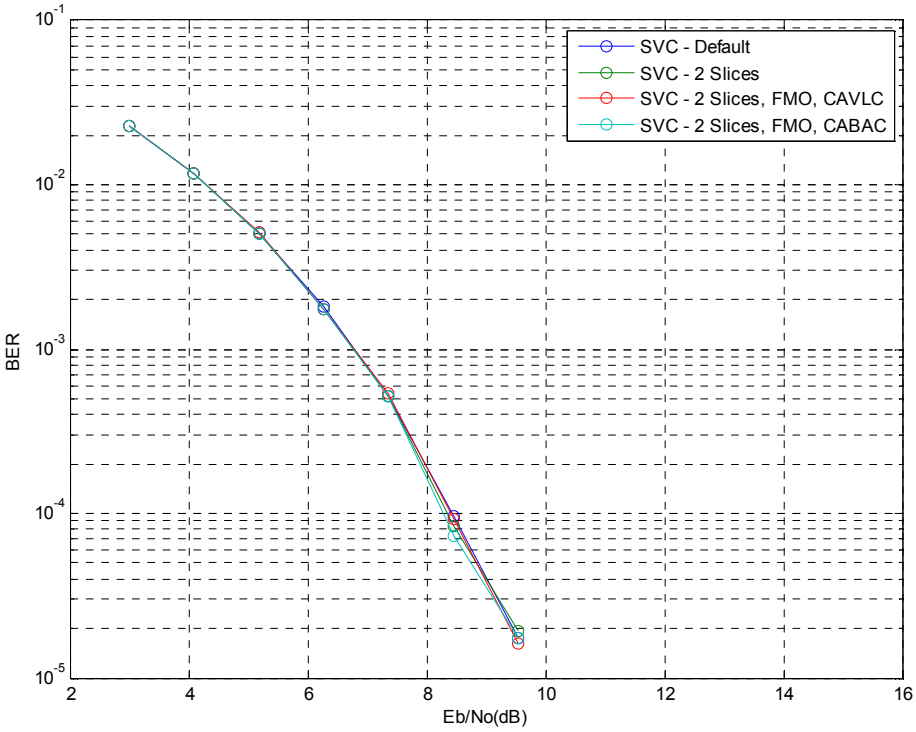

Figure 18 – Test Results JSVM Bitstream at E_b/N_0 of 3 to 15 dB layer DTQ(1,2,0)

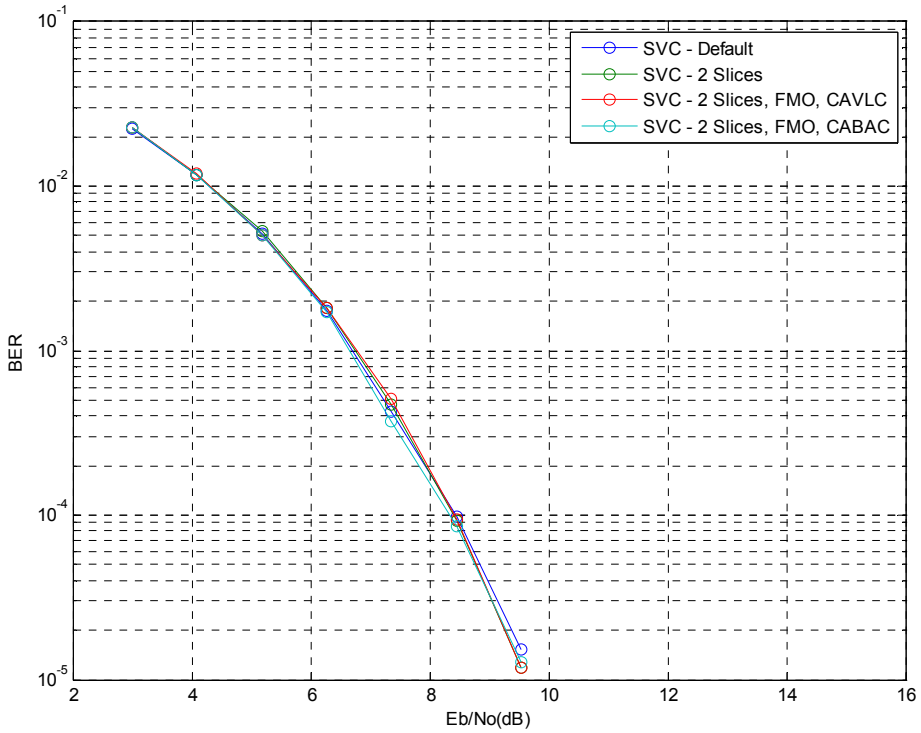

Figure 19 – Test Results JSVM Bitstream at E_b/N_0 of 3 to 15 dB layer DTQ(1,1,0)

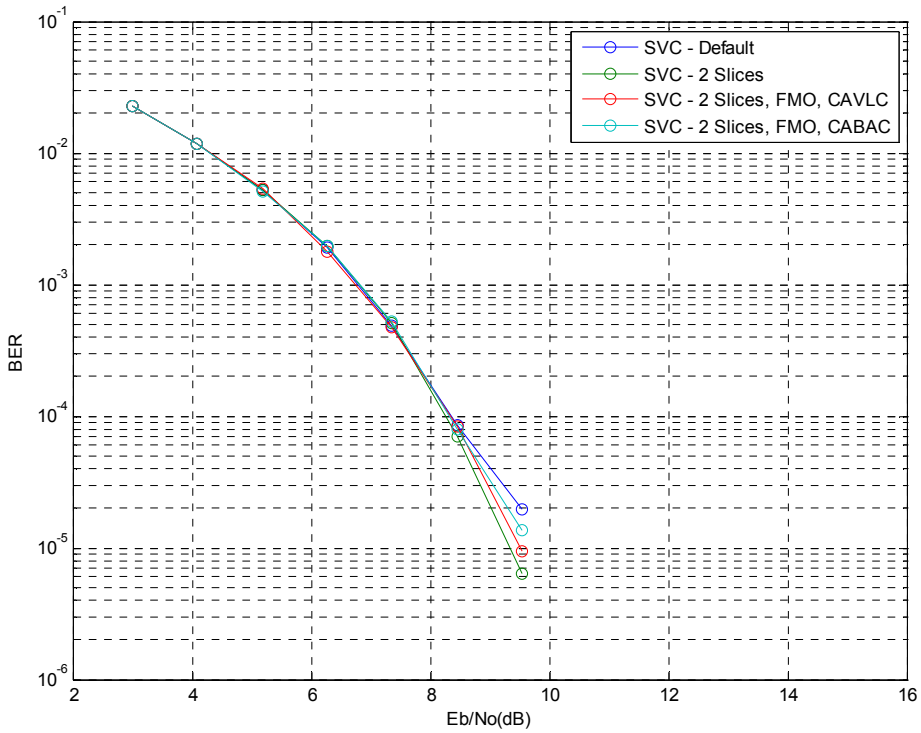

Figure 20 – Test Results JSVM Bitstream at E_b/N_0 of 3 to 15 dB layer DTQ(1,0,0)

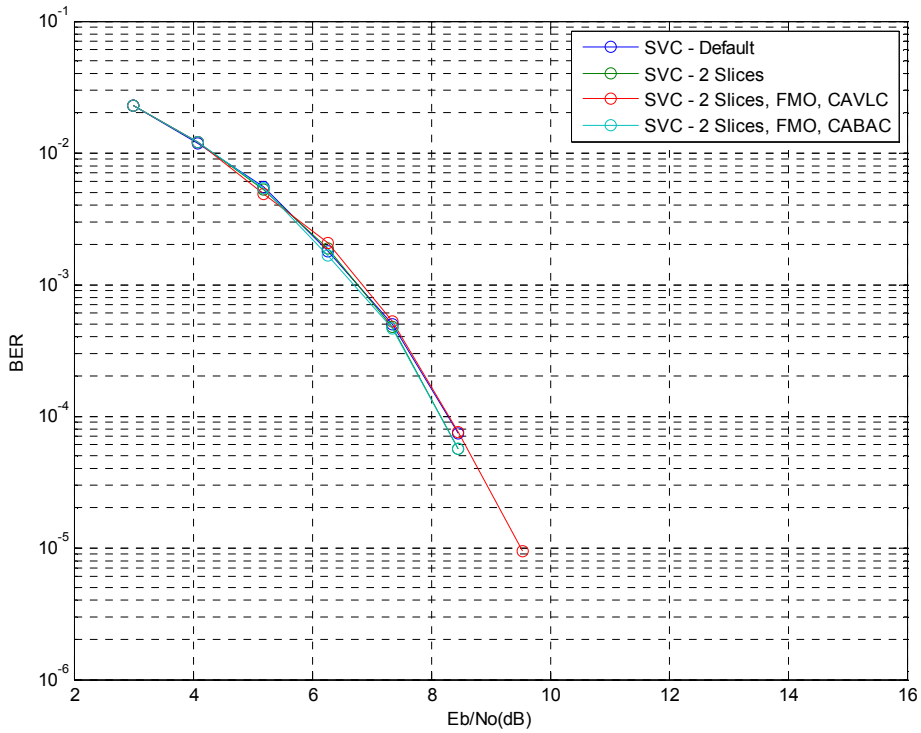

Figure 21 – Test Results JSVM Bitstream at E_b/N_0 of 3 to 15 dB layer DTQ(0,3,0)

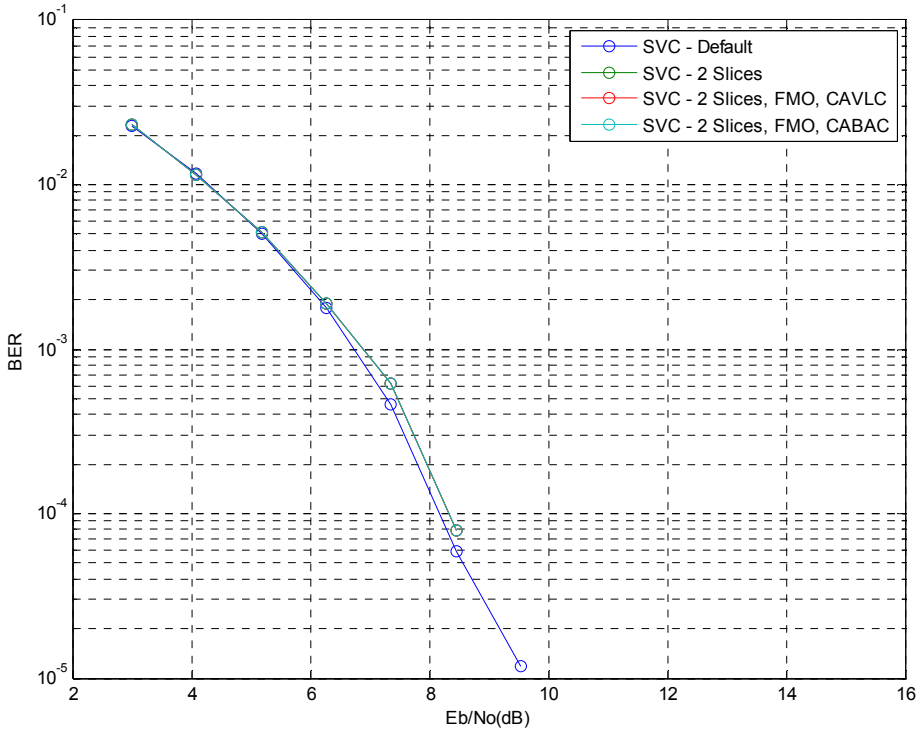

Figure 22 – Test Results JSVM Bitstream at E_b/N_0 of 3 to 15 dB layer DTQ(0,1,0)

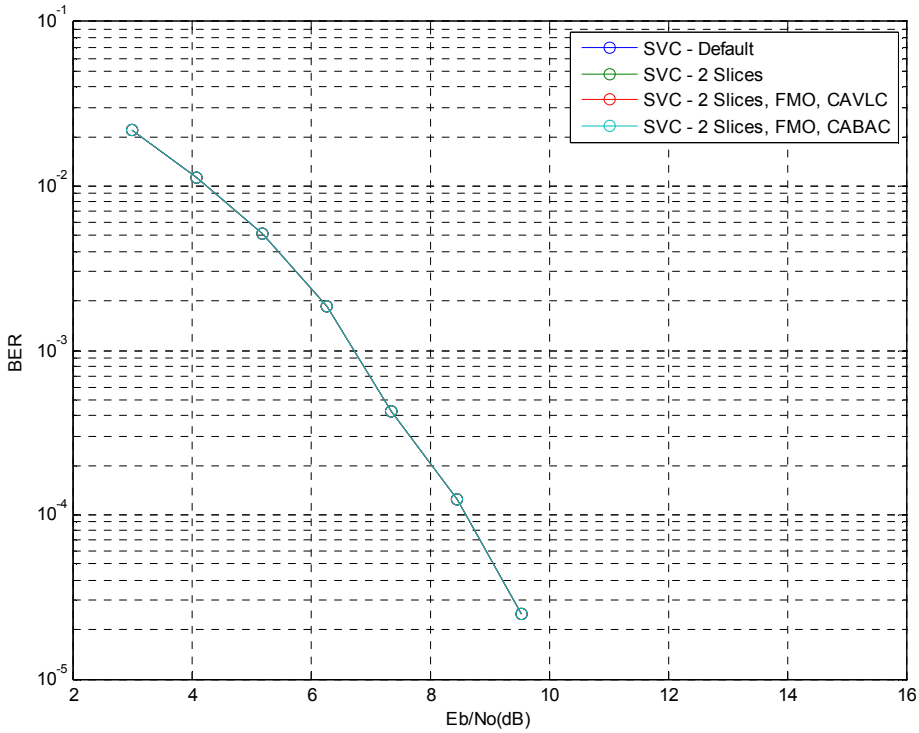

Figure 23 – Test Results JSVM Bitstream at E_b/N_0 of 3 to 15 dB layer DTQ(0,0,0)