UNIVERSIDAD DE NAVARRA
ESCUELA SUPERIOR DE INGENIEROS
SAN SEBASTIÁN

Design of Efficient Viterbi Decoders for Communication Transceivers: IEEE 802.11a case study

DISSERTATION

submitted for the Degree of Doctor of Philosophy by

ARITZ ALONSO DOMINGO

under the supervision of

Andoni Irizar Picón
and
Ainhoa Cortés Vidal

Donostia–San Sebastián, July 2016

What we call the beginning is often the end.
And to make an end is to make a beginning.
The end is where we start from.
T.S.Elliot

他會走回早已消逝的歲月
那塊楨著灰塵的彼璃,
如果他能衝破
他一直在懷念著過去的一切.
看得到, 抓不著.
積著茯塵的玻璃
彷佛隔著一塊
那些消逝了的歲月

Acknowledgement

Decía Constantino Cavafis en su poema Ítaca *que cada persona debía aspirar a que su viaje fuese largo y fecundo. En mi caso, concluir esta tesis ha sido un largo camino que he recorrido en compañía de gente que ha dejado una fuerte impronta en mí. Por medio de estas primeras líneas me gustaría expresar mi agradecimiento a todas ellas.*

En primer lugar quiero dar las gracias a mis padres, por haber recorrido conmigo este camino día a día, en los momentos buenos y los no tan buenos. Por haberse sentido ilusionados ante lo que hacía y ayudarme a levantarme cada vez que tropezaba. Por haberme dado la oportunidad de llegar hasta aquí y el esfuerzo que les ha supuesto. En definitiva, por ayudarme a ser quien soy hoy. Gracias.

A mis familiares, a los que están y estuvieron, en especial a mis tíos Eugenio y Rosa y a mi primo Pablo, por haberme motivado a seguir adelante y las muestras de cariño que me han dado durante todos estos años.

A Maite y Juan, por el interés y apoyo desinteresado que me han demostrado.

A Aurelia, porque, no importa la distancia, haber sido una amiga desde que tengo uso de memoria.

Quiero agradecer también a los compañeros con los que he coincidido durante estos años en Tecnun/CEIT en el desarrollo de esta tesis. A los miembros del Área de Comunicaciones y Tratamiento Digital de la Señal, Igone, Fanfi, Luis, Koldo, Ramón, Esti, Naiara, Borja y Pilar, quiero agradecerles lo mucho que he aprendido junto a ellos y la oportunidad que me han brindado para crecer tanto personal como profesionalmente. A Markos, por estar siempre dispuesto a echar una mano o discutir nuestros gustos ci-

nematográficos. A las chicas del laboratorio, Ainara, Ainhoa, Leti y Marta, por haber sido las mejores compañeras que he podido tener.

Agradezco también las horas que he pasado en compañía del grupo del café, a Aitor, Ane Miren, Borja, David, Gurutz, Jon y Lexuri, por haber hecho de estos años una travesía mucho más amena. Me llevo de vosotros los recuerdos más bonitos de estos años. En especial quiero agradecer a Iker, quien me ha demostrado ser un auténtico amigo, y estar siempre dispuesto a ayudarme.

A mis nuevos compañeros de Farsens, en especial a Dani e Ibon, por la ayuda que me han brindado para terminar esta tesis y el interés que han demostrado en ella.

A Ainhoa Cortés quiero agradecerle el esfuerzo y dedicación que me ha dedicado en estos últimos meses de redacción y por haber arrojado luz ante el recurrente temor a enfrentarse al folio en blanco.

Y por último quiero dar mi más sincero agradecimiento a Andoni Irizar, quien, literalmente, me ha acompañado desde el primer día en que pisé la Universidad hasta el momento en que defienda este proyecto de tesis que tienes entre manos. Él es en gran medida el responsable de que en primer lugar llegase a ser ingeniero y, posteriormente, tuviese la oportunidad de realizar el doctorado. Gracias.

Summary

Forward error correcting techniques have become fundamental tools to obtain robust and reliable communication networks. In this reward, convolutional coders belong to a family of codes used in applications such as deep space communications, LTE, GSM, UWB and WLAN. The Viterbi algorithm is a maximum likelihood decoder for convolutional codes. It operates recursively and in each iteration it discards the less probable messages that can have been transmitted. It is estimated that the Viterbi decoder is the most complex entity of the receiver chain of a multicarrier transceiver.

In this research work the architecture of a flexible and parameterizable Viterbi decoder is presented. This flexibility allows us to quickly modify our architecture so that it decodes any given convolutional code. This way we can easily compare our implementation with other alternatives found in the literature. The decoder description does not make use of external or proprietary IPs, so the decoder can be easily ported to any FPGA manufacturer or ASIC technology.

The Viterbi decoder is one of the most important building blocks of the receiver chain of a transceiver, and its performance is a clear indicator of the Bit Error Rate (BER) or Packet Error Rate (PER) we can expect from the system. The parametrization of our decoder implementation allows us to make trade offs between the complexity, area resource utilization, achievable clock speed and decoding capacity of the transceiver. However, making such a parametrical analysis, specially when the entire transceiver architecture is being analyzed under different channel configurations, is a time consuming task. In order to overcome this limitation, in this research work a fast Hardware in the Loop (HiL) evaluation platform has been designed. This platform allows us to quickly compare different decoder configurations and evaluate the performance of the transceiver architecture

in which they are embedded. The HiL platform has proven to significantly reduce the simulation time of other alternatives such as RTL simulators.

The case study of the parametrical analysis has been WLAN 802.11a. In this research work the sources of a WLAN 802.11a compliant transceiver have been obtained. The transceiver architecture is functional up to the MAC layer of the standard, and it includes complex components such as a time and offset synchronizer and equalizer and phase offset tracker. Also, during this research work a simple hardware oriented demapping algorithm has been proposed.

By means of the HiL platform, the Viterbi decoder architecture has been optimized in terms of area resource utilization and its PER performance curves have been obtained for different transmission modes supported by the WLAN standard.

Contents

List of Figures

List of Tables

Glossary

ACS	Add-Compare-Select
ADC	Analog to Digital Converter
AGC	Automatic Gain Control
ASIC	Application Specific Integrated Circuit
AWGN	Additive White Gaussian Noise
BBP	Base Band Processor
BER	Bit Error Rate
BMU	Branch Metric Unit
BPSK	Binary Phase-Shift Keying
CiL	Controller-in-the-Loop
CDMA	Code Division Multiple Access
CLB	Configurable Logic Block
CORDIC	Coordinate Rotation Digital Computer
CPU	Central Processing Unit
CRC	Cyclic Redundant Check
CSI	Carrier Strength Indicator
DAC	Digital to Analog Converter
DUT	Device Under Test

DVB-T	Terrestrial Digital Video Broadcasting
FIR	Finite Impulse Response
FEC	Forward Error Correction
FFT	Fast Fourier Transform
FIFO	First-In First-Out
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GSM	Global System for Mobile communications
GPS	Global Positioning System
GPU	Graphics Processing Unit
HiL	Hardware-in-the-Loop
HDL	Hardware Description Language
IC	Integrated Circuit
IFFT	Inverse Fast Fourier Transform
IIR	Infinite Impulse Response
IoT	Internet of Things
IP	Intellectual Property
ISI	Inter-Symbol Interference
JTAG	Joint Test Action Group
LAN	Local Area Network
LIFO	Last-In First-Out
LSB	Least Significant Bit
LUT	Look-up Table
MAC	Medium Access Control

MMC	Modular Multilevel Converter
MPU	Minimum Path Unit
MSB	Most Significant Bit
OFDM	Orthogonal Frequency Division Multiplexing
PER	Packet Error Rate
PLCP	Physical Layer Convergence Protocol
PLME	Physical Layer Management Entity
PMD	Physical Medium Dependent
PPDU	PLCP Protocol Data Unit
PRNG	Pseudo Random Number Generator
QAM	Quadrature Amplitude Modulation
RAM	Random Access Memory
ROM	Read-Only Memory
RRC	Root Raised Cosine
RSSI	Received Signal Strength Indicator
RTL	Register Transfer Level
SAIF	Switching Activity Interface File
SDR	Software Defined Radio
SNR	Signal-to-Noise Ratio
SoC	System-on-Chip
SPU	Survivor Path Unit
UI	User Interface
UWB	Ultra Wide Band
VHDL	VSIC (Very High Speed Integrated Circuits) Hardware Description Language
WLAN	Wireless Local Area Network

List of Symbols

$\lceil x \rceil$	Largest integer not greater than x
$\lfloor x \rfloor$	Smallest integer not less than x
ϵ	Comparator level in the MPU
ν	Constraint length of the convolutional code
τ	Traceback depth of the convolutional decoder
Φ	Figure of merit used to compare decoder architectures
ζ	Comparator identifier at level ϵ in the MPU
k	Number of bits that are loaded into to the convolutional coder per clock cycle
l_{mpu}	Latency in clock cycles of the Minimum Path Unit (MPU)
m	Number of bits coded in a QAM constellation point
n	Number of coded bits generated by the convolutional coder per clock cycle
p	Parallelization ratio of the ACSU
R	Coding rate
T	Threshold of the adaptive Viterbi algorithm
N_{max}	Number of survivor paths per iteration in the adaptive Viterbi algorithm

$\mathbf{c}$	Source coded symbol
$\mathbf{m}$	Source uncoded symbol
$\mathbf{r}$	Received symbol
$\mathbf{x}$	Demapped received symbol
$G(x)$	Transfer function of the convolutional code

CHAPTER 1

Introduction

Contents

1.1 INTRODUCTION

For the last decades new and more complex digital system have appeared on a yearly basis as predicted by Moore's observation [Moo65]. However, the possibility of designing more complex Integrated Circuits (ICs) has not only be limited to increase their performance on a very specific task. On the contrary: the miniaturization requirements of new devices such as smartphones, tablets or phablets (or those which will become common under the newly coined Internet of Things (IoT)) have only been achieved when technology has been mature enough to integrate into a single chip devices as diverse as Central Processing Units (CPUs), Graphics Processing Units (GPUs), Digital to Analog Converters (DACs), Analog to Digital Converters (ADCs) and wireless transceivers. These ICs are known as System-on-Chips (SoCs).

With the advent of advanced touch-control User Interfaces (UIs) the mobile market has experienced a significant growth that continues nowadays, with several reports [AMJ+14a, AMJ+14b, Chu14, CMS14, Cis14] forecasting shipments between 1.8 and 2.2 milliard units for the year 2015.

As SoCs grew in complexity and performance, end user began to demand application and services with higher bandwidth consumption. High reliable communication systems are necessary for the development applications and services. In this scenario, forward error correcting techniques are fundamental tools that provide robustness to digital communication links.

Convolutional coders are one of the first codes for which an optimum decoding algorithm was discovered. They are preferred over block codes such as Turbo codes for their superior performance with comparable complexity decoding architectures.

The Viterbi algorithm is a maximum likelihood decoder for convolutional codes. It operates recursively and in each iteration it discards the less probable messages that can have been transmitted. It is estimated that the Viterbi decoder is the most complex entity of the receiver chain of a multicarrier transceiver.

Since its discovery, Viterbi decoders have become a fundamental cornerstone in many communication systems. Back in 1986, V.32 modems used an 8 state convolutional code to obtain a coding gain of about 3.5 dB. V.34, an updated revision of the standard, used 16 to 64 state convolutional codes to achieve coding gains of 4.0 to 4.5 dBs [FBEM96]. The Viterbi algorithm is known also for being the first coding scheme used in deep space communications in the Pioneer program [CHIW98]. At the dawn of the new millennia, Viterbi decoders were implemented in around one milliard cell phones worldwide [Jr.05], and a contemporary survey carried out by Qualcomm [Pad05] indicated that approximately 10^{15} bits were decoded by Viterbi decoders on video broadcast applications.

The Viterbi decoder proposals found in the literature aim at optimizing its description in areas such as area and power consumption, decoding capacity or achievable maximum frequency. These goals are generally mutually exclusive and do not analyse their impact on the overall decoder performance.

The purpose of this research work is to obtain, verify and optimize a Viterbi decoder description with respect to the transceiver architecture in which it will be used. Wireless Local Area Network (WLAN) 802.11a will be used as the case study of the research work.

1.2 OUTLINE OF THE RESEARCH WORK

In the following, the organization of this PhD dissertation is outlined:

Chapter 2: State of the art
This chapter summarizes the State of the Art of Viterbi decoders. First, convolutional codes are introduced. Next, the Viterbi algorithm is described. Following, the basic building blocks of hardware Viterbi decoders are enumerated. Finally, Viterbi decoder hardware implementations found in the literature are listed.

Chapter 3: Objectives
This chapter states the main objective of this research work and decomposes it into partial objectives. It also presents the figrues of merit that will be used to compare the different Viterbi decoder architectures found in the literature with that developed in this research work.

Chapter 4: Viterbi decoder architecture
The chapter describes the hardware architecture of the Viterbi decoder implemented in this research work. Both register exchange and traceback implementations are proposed.

Chapter 5: Hardware-in-the-Loop simulations
This chapter describes a Hardware in the Loop simulation platform used to perform a parametrical analysis of the Viterbi decoder and introduces the WLAN 802.11a compliant transceiver, the case study of this research work, hardware modules. The Viterbi decoder is embedded within the WLAN 802.11a compliant transceiver description and then a parametrical analysis of the entire system is performed using the Hardware-in-the-Loop (HiL) simulator to find a balance between system performance and resource utilization. Once the transceiver architecture has been optimized, a more thorough analysis is executed using the HiL platform to obtain the Packet Error Rate (PER) performance curves of the system under multipath fading channels.

Chapter 6: Results

This chapter compares the Viterbi decoder description developed in this research work with other proposals found in the literature.

Chapter 7: Conclusions and areas for further research

This chapter presents the conclusions of this research work and possible areas for further research.

CHAPTER 2

State of the art

Contents

2.1 INTRODUCTION

The continuous advances in integration processes have allowed the deployment of cheaper, more complex digital circuits. At the same time, these new devices have boosted the appearance of applications and services that demand high bandwidth mobile communications. Error correction techniques are fundamental and powerful tools that allow the deployment of highly reliable communication systems.

The Viterbi algorithm, named after Andrew James Viterbi [Vit67], is the key decoding algorithm of convolutionally encoded messages. It is used in a wide range of applications such as speech recognition, LTE, Physical Downlink Control Channel (PDCCH), CDMA and GSM digital cellular, dial-up modems, satellite, deep-space communications [Doo10], Ultra Wide

Band (UWB) communication schemes and 802.11 wireless Local Area Networks (LANs).

Among all the available algorithms aimed at decoding convolutional codes the Viterbi is the most resource intensive. However, it is capable of performing maximum likelihood decoding. A significant research work effort is maintained up to this day aimed at identifying new, optimized hardware architecture implementations of the algorithm.

Analyzing these architectures is a very time consuming task, specially when the decoder is embedded in a communication transceiver. The hardware implementation must not only be verified, but its decoding capacity must be measured in various channel configurations (noise level, multipath delay spread, ...) and, if the communication system allows it, different transmission rates supported by the transceiver. Consequently, the entire system (transceiver architecture and surrounding channel) should be considered when the architecture of the transceiver is being optimized to meet certain area, power consumption or decoding capacity criterion.

2.2 VITERBI DECODER

This section presents the state of the art of Viterbi decoders. First convolutional codes are disclosed. Next, the Viterbi algorithm is introduced. Following, the building blocks of Viterbi decoders are described and finally the different implementations of Viterbi decoders are presented.

2.2.1 CONVOLUTIONAL CODES

A simplified view of the operation of a communication system is depicted in figure 2.1.

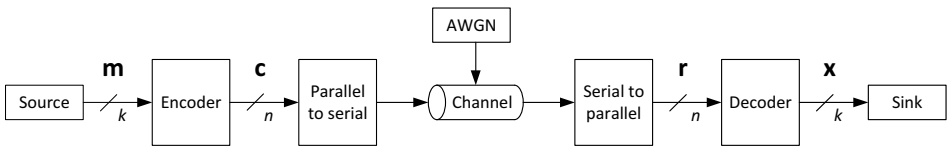


Figure 2.1: *Simplified block diagram of a communication system*

In the transmitter side of the communication system, a data source generates messages $\mathbf{m}$. Each message $\mathbf{m}$ is represented as a vector of k symbols. The symbols of $\mathbf{m}$ can belong to any alphabet, although in digital communications they are generally considered to be bits.

To minimize the probability of errors during data transmission most communication systems implement Forward Error Correction (FEC) mechanisms. The block responsible of protecting the message $\mathbf{m}$ against the mismatches of the channel is the *encoder*. The encoder takes the binary message $\mathbf{m}$ and produces an output n bit wide *codeword* $\mathbf{c}$, where $n > k$. The ratio $R = k/n$ is known as the *coding rate* of the encoder and it can be seen as a tradeoff between the redundancy added to the original message and the capacity of detecting errors during transmission: codes with low R are more redundant and offer more opportunities to detect and correct erroneous bit sequences, but transmit less source *data* bits per *encoded* bit.

The transmitter serializes codeword bits in a parallel-to-serial block so that a single *encoded* bit is transmitted per channel use. In figure 2.1 we have assumed a channel that simply adds Additive White Gaussian Noise (AWGN) to the transmitted signal. At the receiver side, the inverse process is performed. A serial-to-parallel block takes the detected signal and generates a received vector $\mathbf{r}$ of n bits. Note that due to the AWGN in the channel, the codeword $\mathbf{c}$ and received vector $\mathbf{r}$ are not necessarily the same.

A *decoder* takes the received vector $\mathbf{r}$ and, based on the knowledge of the coding rules at the *encoder*, tries to identify which message $\mathbf{m}$ was originally transmitted. The guess generated by the decoder is labelled $\mathbf{x}$ in figure 2.1.

The Bit Error Rate (BER) is a performance metric of such a communication system. It is the ratio between the number of bits $\mathbf{m}$ and $\mathbf{x}$ differs and the length of the message $\mathbf{m}$.

Convolutional codes are one of the earliest codes for which effective decoding algorithms were developed. They are viewed as *stream codes* since generally they operate on continuous streams of bits not partitioned in discrete message blocks or packets. In practice they are preferred to block codes (for example, Turbo codes) because, at a comparable encode/decode complexity, they provide excellent performance [Moo05]. The name *convo-*

lutional code comes from the fact that the operation of these codes can be seen as filtering or convolution.

Let us take for example the architecture of an encoder implementing a $R = 1/2$ convolutional code in figure 2.2. Being a $R = 1/2$ code, every instant of time the convolutional encoder takes a one-bit input *message* and generates a two-bit output *codeword*. As it can be seen, a convolutional encoder consists basically of a shift register and a set of n XOR gates.

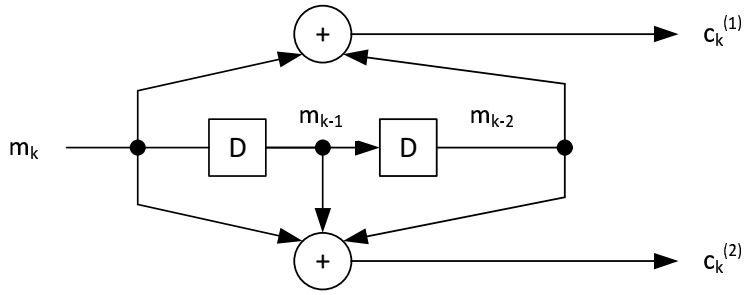


Figure 2.2: *Example convolutional encoder with R=1/2*

The number of registers in the shift register of the convolutional encoder is known as the *constraint length* ν of the code. In essence, the convolutional encoder operates as a Finite State Machine (FSM). Each instant of time a new input message bit is fetched into the encoder and the content of its register is shifted one position to the right. The content of the shift register represents the *state* of the FSM. For a convolutional code with a constraint length ν, the total number of possible states in the encoder is 2^ν.

The output codeword of the encoder is calculated every time instant as the XOR operation between the input of the encoder at that time instant and the logical value of certain positions in the shift register. The *transfer function* $G(x)$ of the convolutional code is the matrix representation of the elements that are taken into account when calculating the output of the encoder. The elements of the transfer function matrix $G(x)$ are generally expressed in polynomials of x. The degrees of the terms of the polynomial indicate the positions in the shift register that are considered to calculate the codeword. The highest degree of the polynomial is then ν, and the input to the encoder represents the *independent term* of the polynomial.

For the encoder in figure 2.2 we have that

$$\begin{aligned} G(x) &= [m_k + m_{k-2} \qquad m_k + m_{k-1} + m_{k-2}] \\ &= [1 + x^2 \qquad\qquad\qquad 1 + x + x^2] \end{aligned} \tag{2.1}$$

In general, $G(x)$ is a $k \times n$ matrix. Since in our example $k = 1$ and $n = 2$, we have that the *transfer function* matrix in (2.1) is a two-element row vector.

Not all the convolutional codes can be represented by polynomials. An encoder that has only polynomial entries in its transfer function matrix is said to be a *feedforward* encoder or a Finite Impulse Response (FIR) encoder. An encoder that has rational functions in its transfer function matrix is said to be a *feedback* or Infinite Impulse Response (IIR) encoder.

For feedforward encoders, it is common to indicate the connection polynomials as vectors of numbers representing the *impulse response* of the encoder. The binary representation of the transfer function of equation (2.1) is represented by the vectors

$$\begin{aligned} G(x) &= [m_k + m_{k-2} \qquad m_k + m_{k-1} + m_{k-2}] \\ &= \big[[101] \qquad\qquad\qquad [111]\big] \end{aligned} \tag{2.2}$$

where the Most Significant Bit (MSB) of the binary vector represents the input to the encoder and the Least Significant Bit (LSB) of the binary vector indicates the ν-th element of the shift register of the encoder.

These binary vectors are often expressed compactly in octal form, where the triplets of bits are represented using the integers from 0 to 7. In this form, the encoder in (2.1) is represented as

$$G(x) = \big[[101] \quad [111]\big] = \big[5 \quad 7\big] \tag{2.3}$$

A convolutional code is unequivocally represented by the triplet (n, k, ν) and its transfer function $G(x)$.

For encoding and decoding purposes, there are two helpful representations of the the transfer function of a convolutional code: the *state* diagram and the *trellis* diagram.

The state diagram of the convolutional code is a graph that depicts the state transitions and the generated codewords as a function of the input

bits to the encoder. Figure 2.3 shows the state diagram of the example convolutional encoder presented in figure 2.2.

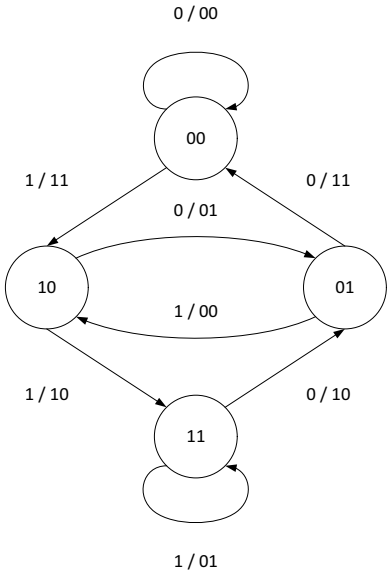


Figure 2.3: *State diagram*

Each state transition is represented with an arrow connector. Above each arrow two quantities separated by slashes are annotated. The leftmost quantity is the input to the encoder at a given time instant that triggers the state transition. The rightmost quantity is the generated codeword due to the state transition.

For example, from figure 2.3, we can deduce that when the convolutional encoder is at state "01" it will transition to state "00" when a logical zero is applied to its input. In this transition the encoder will generate the codeword "11". Otherwise, if a logical one is applied to the encoder, then it will transition to state "10" and generate the output codeword "00".

The trellis diagram of the convolutional code is a rearrangement of the former state diagram where the notion of time instant is added to the state transitions. Let us consider the trellis diagram of our example convolutional code depicted in figure 2.4.

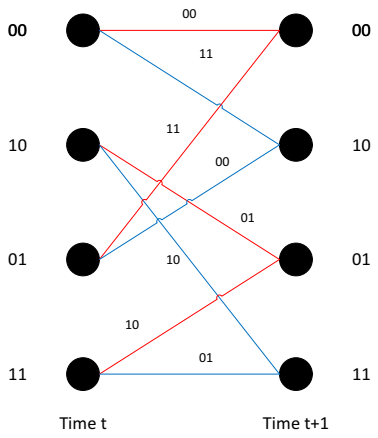


Figure 2.4: *One-state Trellis*

The diagram shows two sets of all the possible states of the encoder. The leftmost column is the initial state of the convolutional encoder at time instant t. The rightmost column is the final state of the convolutional encoder after the transition has occurred at time instant $t+1$. In the figure initial and final states are connected by coloured lines. The color of these lines represent the logical value of the input to the encoder at the initial time instant: a red line indicates a logical zero input bit and a blue line represents a logical one input bit. Above each line the codeword generated due to the state transition is written.

One-state transition trellis diagrams such as that in figure 2.4 are stacked one after the other to represent the evolution of the encoder over a specific time interval. For example, if our example encoder, beginning at the initial state "00", were to encode the message $\mathbf{m} = [1, 1, 0, 0, 1, 0, 1, 0]$, the trellis diagram shown in figure 2.5 would be generated.

In the figure above the binary representation of the states of the encoder have been substituted by their corresponding unsigned decimal representation. The colored line in the figure represents a *path* through the trellis that unequivocally identifies the encoding of the message $\mathbf{m}$.

As it can be deduced from these diagram representations of the convolutional code, there is a very tight connection between the input message $\mathbf{m}$ to the encoder, the generated output codeword $\mathbf{c}$ and the state tran-

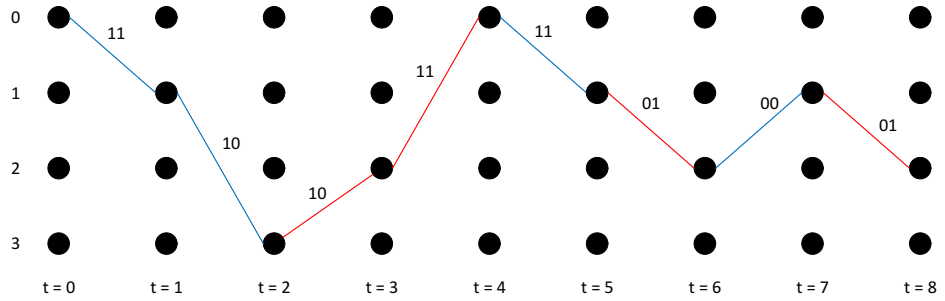


Figure 2.5: *Example path in the Trellis diagram*

sition sequence depicted by the *path* through the trellis diagram. For a given initial state condition in the convolutional encoder, knowing any of the former sequences (input message, output codeword or path through the trellis) implies the knowledge of the remaining sequences.

Thus, a convolutional decoder is the block that, based on the received codeword signal $\mathbf{r}$, tries to identify the path in the trellis that originated it. After the most probable path in the trellis has been identified, the decoder reconstructs the original message by identifying the state transitions in the trellis path with the inputs to the encoder that trigger them.

2.2.1.1 Parallelization of convolutional codes

Some decoding algorithms are based on parallel implementation of the convolutional encoder. Whereas a *serial* convolutional encoder takes a binary word of k bits and generates a codeword of n bits, a *parallel* implementation of the convolutional encoder takes a block of p consecutive input binary words of k bits and generates a block of p codewords of n bits.

For the same input message $\mathbf{m}$, both serial and parallel encoder implementations produce identical outputs. Consequently, the two encoder implementations have identical constraint length ν and number of states 2^{ν}. The implementation of the parallel convolutional encoder is modified slightly to accommodate to the increased number of inputs and outputs. For the example convolutional encoder introduced in this chapter, its parallel implementation with $p = 2$ is shown in figure 2.6.

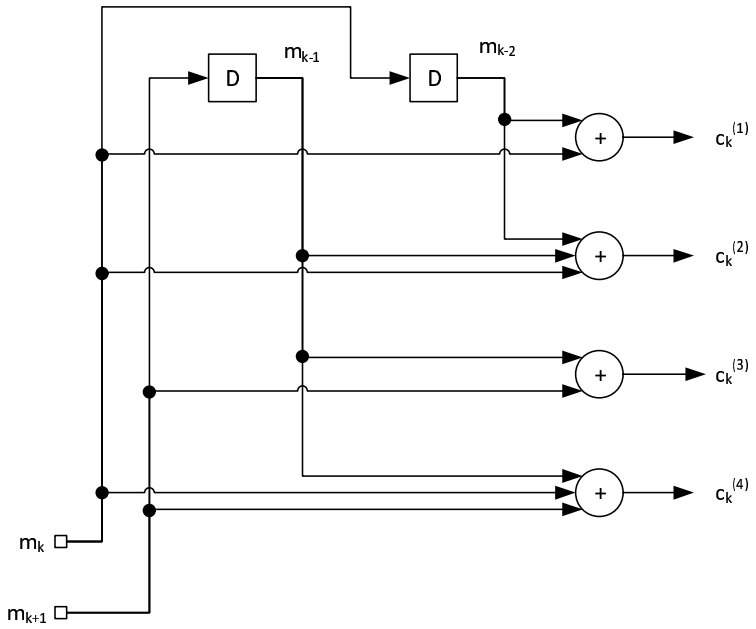


Figure 2.6: *Example convolutional encoder with parallelization index $p = 2$*

From the figure, input port m_k corresponds to the input of the serial encoder at time instant t and input port m_{k+1} corresponds to the input of the serial encoder at time instant $t + 1$. Similarly, output ports $c_k^{(1)}$ and $c_k^{(2)}$ of the parallel encoder correspond to the output of the serial encoder at time instant t and output ports $c_k^{(3)}$ and $c_k^{(4)}$ of the parallel encoder correspond to the output of the serial encoder at time instant $t + 1$.

The trellis diagram of the parallel implementation of the convolutional encoder is obtained by rearranging the states of the serial implementation. The trellis diagram of the parallel implementation of the example convolutional encoder with $p = 2$ is shown in figure 2.7.

For the parallel encoder, the lines connecting the initial and final states of the trellis diagram are split into p colored segments. The color of the first segment indicates the logical value of the binary message word at time instant $t = 0$. A red line indicates that at that time instant a logical zero was input to the parallel convolutional encoder. A blue line, on the other

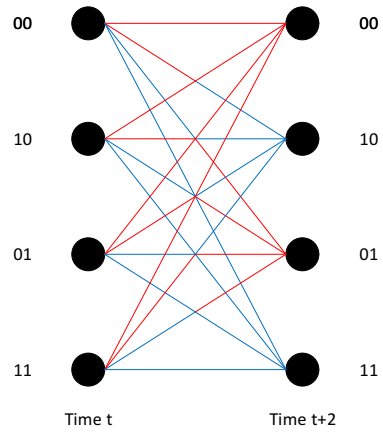


Figure 2.7: *Parallel Trellis diagram*

hand, indicates that a logical one was input to the parallel convolutional encoder. The color of the last segment indicates the logical value of the message input to the parallel convolutional encoder at time instant $t = p-1$.

The generated codewords due to the state transitions in the parallel implementation of the convoluitonal encoder with $p = 2$ are shown in table 2.1.

There is a limit in the level of parallelization p that can be obtained in the convolutional code. The parallelization p must never surpass the constraint length ν of the code. Otherwise, different input messages blocks would produce identical state transition, and therefore, the decoding process would become infeasible.

2.2.2 VITERBI ALGORITHM

The Viterbi algorithm was proposed by Andrew Viterbi [Vit67], but it was not until [For73] that its optimality as a maximum likelihood sequence decoder was appreciated. Communication system applications include examples such as maximum likelihood sequence estimation in the presence of Inter-Symbol Interference (ISI) [For72] and optimal reception of spread-spectrum multiple access communications [Ver84]. It has also been used for hidden Markov modeling [DPH93] and survey applications [MS00].

Initial/End state	00	10	01	11
00	0000	0011	1101	1110
10	0111	0100	1010	1001
01	1100	1111	0001	0010
11	1011	1000	0110	0101

Table 2.1: *Generated codewords for the parallel convolutional encoder($p = 2$)*

The key idea behind the Viterbi algorithm is that a coded sequence $\mathbf{c} = [c_0, c_1, \ldots]$ or its associated received input sequence $\mathbf{r} = [r_0, r_1, \ldots]$ in figure 2.1 corresponds to a single path through the encoder trellis. Due to mismatches in signal reception, the received sequence $\mathbf{r}$ may not correspond with the same path through the trellis. Moreover, the received sequence may imply an *illegal* path through the trellis. A path that includes states transitions other than those defined by the state diagram of the code.

The Viterbi algorithm tries to find the *legal* path through the trellis that is closest to the received sequence $\mathbf{r}$. Instead of taking a brute force approach, where all the possible paths through the trellis would need to be compared with the received sequence $\mathbf{r}$, the Viterbi algorithm operates in an efficient, recursive form. In each decoding time instant the algorithm selects a set of only 2^ν paths among all the possible paths that can be generated up to that point. The 2^ν paths that the Viterbi algorithm selects in each iteration are known as the *survivor paths* at that time instant. There is one survivor path per state in the convolutional code. Every decoding time, all possible paths that converge to a given state and are not the *survivor path* are discarded by the Viterbi algorithm for further analysis.

Competing paths ending at a same state in the encoder are compared in terms of their *metrics*. Encoder sequences that differ significantly to the received sequence have a higher metric cost than encoder sequences that resemble the received sequence. Consequently, the Viterbi algorithm selects the paths with minimum metric. Typically, the Hamming distance is the preferred method when calculating the metrics. The Hamming distance

	000	001	010	011	100	101	110	111
000	0	1	1	2	1	2	2	3
001	1	0	2	1	2	1	3	2
010	1	2	0	1	2	3	1	2
011	2	1	1	0	3	2	2	1
100	1	2	2	3	0	1	1	2
101	2	1	3	2	1	0	2	1
110	2	3	1	2	1	2	0	1
111	3	2	2	1	2	1	1	0

Table 2.2: *Hamming distances between all possible 3-bit wide codewords*

between two binary vectors (or points) is the number of positions in which their logical values differ. Table 2.2 shows the Hamming distance between all possible 3-bit codewords. The metric of the survivor path ending at a given state of the encoder is known as the *path metric* of that state.

When the decoding time advances, the Viterbi algorithm extends the survivor paths of the previous time instant with the transitions described by the trellis diagram of the coder. The metrics of the new candidate paths are calculated by adding the path metric of their precursor state with the *branch metric* of the transition: the Hamming distance between the codeword of the state transition (defined in the trellis diagram) and the received sequence $\mathbf{r}$ at that time instant. For each of the 2^{ν} states, the algorithm selects, among all the possible transitions defined by the trellis, the path with minimum accumulated metric.

Let us better illustrate the operation of the Viterbi algorithm by means of an example extracted from [Moo05]. Consider again the example convolutional encoder described in figure 2.2. The state and trellis diagrams of this encoder were given in figures 2.3 and 2.4.

When the data sequence

$$\begin{aligned} m &= [1, 1, 0, 0, 1, 0, 1, 0, \ldots] \\ &= [m_0, m_1, m_2, m_3, m_4, m_5, m_6, m_7, \ldots] \end{aligned}$$

is applied to the encoder it generates the following codewords:

$$\begin{aligned} \mathbf{c} &= [11, 10, 10, 11, 11, 01, 00, 01, \ldots] \\ &= [\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3, \mathbf{c}_4, \mathbf{c}_5, \mathbf{c}_6, \mathbf{c}_7, \ldots] \end{aligned}$$

The coded output sequence is transmitted through a channel. Due to the noisy characteristics of this channel the binary representation of the received sequence $\mathbf{r}$ and the codewords $\mathbf{c}$ differ. Let us assume that during the signal transmission, the received sequence becomes

$$\begin{aligned} \mathbf{r} &= [11, 10, \underline{0}0, 1\underline{0}, 11, 01, 00, 01, \ldots] \\ &= [\mathbf{r}_0, \mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3, \mathbf{r}_4, \mathbf{r}_5, \mathbf{r}_6, \mathbf{r}_7, \ldots] \end{aligned}$$

where the underlined bits in $\mathbf{r}$ indicate the positions where the logical values of the codewords $\mathbf{c}$ have toggled their values.

The Viterbi algorithm operates as follows.

At time instant $t = 0$ the sequence $\mathbf{r}_0 = 11$ is received. It is known that state 0 is the initial state of the convolutional coder. Based on the trellis diagram (figure 2.4) of the convolutional code, we know that only states 0 and 1 can be reached at time instant $t = 1$ from the initial state of the encoder. The generated codewords due to these transitions are "00" and "11" respectively. The Viterbi algorithm calculates the *path metrics* of these hypothetical transitions as the Hamming distance between the received sequence ($\mathbf{r}_0 = 11$) and the generated output codewords. In our case, the *path metrics* are 2 and 0 respectively. As a result, at time instant $t = 1$ there are only 2 possible paths, with metrics 0 and 2, as shown in the figure 2.8.

The metric that converges to a specific state in a given instant of time are known as the *branch metric* of the state at that time instant.

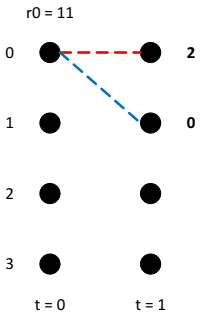


Figure 2.8: *Path extension at time instant* $t = 1$

At time instant $t = 1$ the sequence $\mathbf{r}_1 = 10$ is received. Again, all existing paths are extended at time $t = 2$ as defined by the trellis diagram by adding the path metric to each branch metric. Note that at time $t = 2$, a total of 4 paths are available, each ending in one of the 4 possible states of the trellis

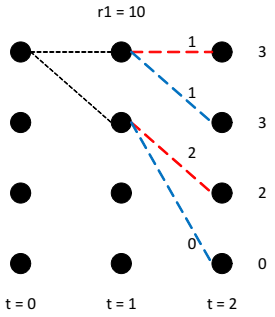


Figure 2.9: *Path extension at time instant* $t = 2$

At time instant $t = 2$ the sequence $\mathbf{r}_2 = 00$ is received. Once again, each path is extended by adding the path metric to each branch metric. In the figure 2.10, the branch metric of each path is represented by the value above each state at time instant $t = 2$, and the branch metrics of all path possible paths to time instant $t = 3$ are the value above the doted colored lines

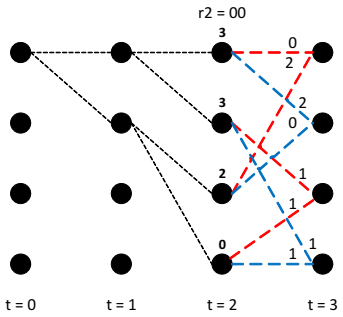


Figure 2.10: *Path extension at time instant* $t = 3$

Note that at time $t = 3$ there are several possible paths that end up in the same node of the trellis diagram. The Viterbi algorithm selects the paths with the best (lowest) accumulated metric at each node, as these are the paths that are closer to the received sequence $\mathbf{r}$. These paths are known as the survivor paths at time instant $t = 3$. The remaining paths will be discarded for further analysis. After the survivor paths have been selected, the trellis diagram becomes as follows:

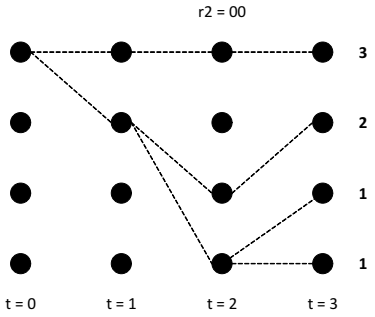


Figure 2.11: *Survivor path selection at time instant* $t = 3$

At time instant $t = 3$ the sequence $\mathbf{r}_3 = 10$ is received. The algorithm continues extending the paths by adding the path metric to each branch metric

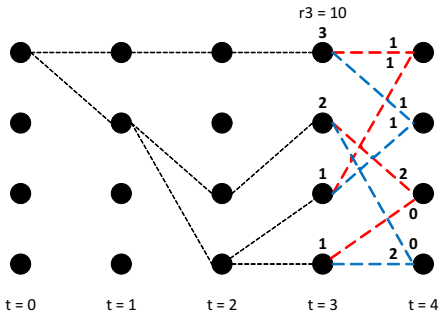


Figure 2.12: *Path extension at time instant* $t = 4$

As before, the paths with best (lowest) metric at each state are selected.

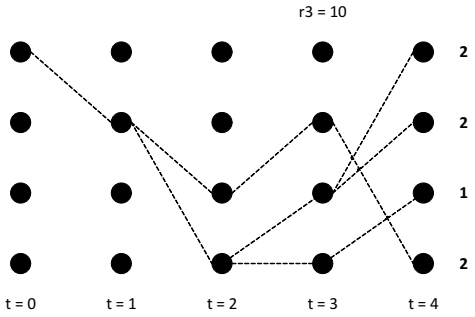


Figure 2.13: *Survivor path selection at time instant* $t = 4$

If we observe the previous figure, we will notice a fundamental characteristic of the survivor paths: from now on all the survivor paths share an identical set of state transitions in the trellis. Indeed, all the survivor paths at time instant $t = 4$ share a common state transition from state 0 to state 1 at the beginning of the encoding.

The decoding process continues. At time instant $t = 4$ the sequence $\mathbf{r}_4 = 11$ is received. The paths are extended by adding the path metric to each branch metric

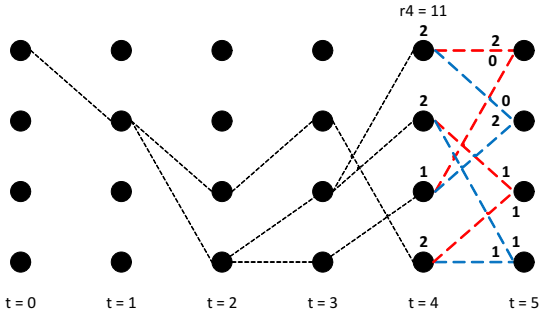


Figure 2.14: *Path extension at time instant* $t = 5$

This time, there are several paths ending in states 2 and 3 with the same minimum (best) metric of 3. Since only one path must be selected, the survivor paths can be chosen randomly or using a pre-defined rule. In this case the selection algorithm has no impact on the decoding capability and performance of the Viterbi algorithm. The next figure depicts the selected survivor paths

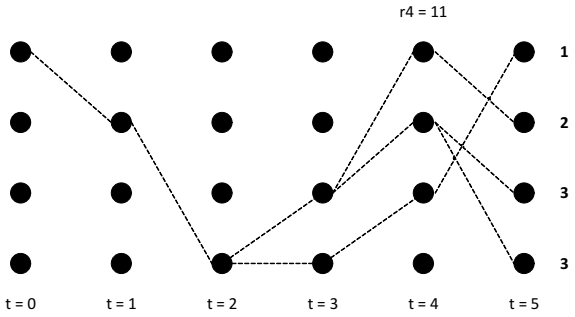


Figure 2.15: *Survivor path selection at time instant* $t = 5$

At time instant $t = 5$ the sequence $\mathbf{r}_5 = 01$ is received.

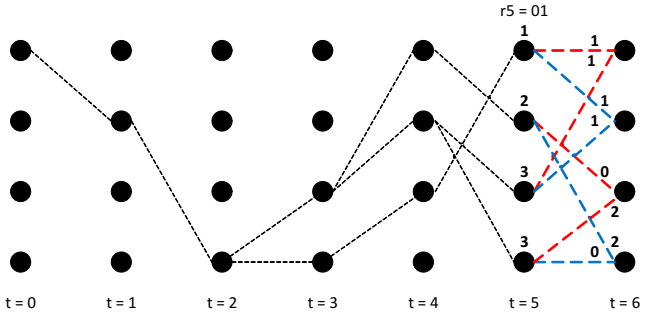


Figure 2.16: *Path extension at time instant* $t = 6$

After survivor path selection, the state transition diagram becomes:

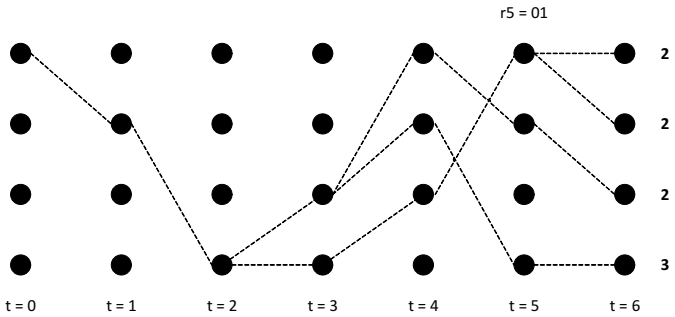


Figure 2.17: *Survivor path selection at time instant* $t = 6$

At time instant $t = 6$ the sequence $\mathbf{r}_6 = 00$ is received.

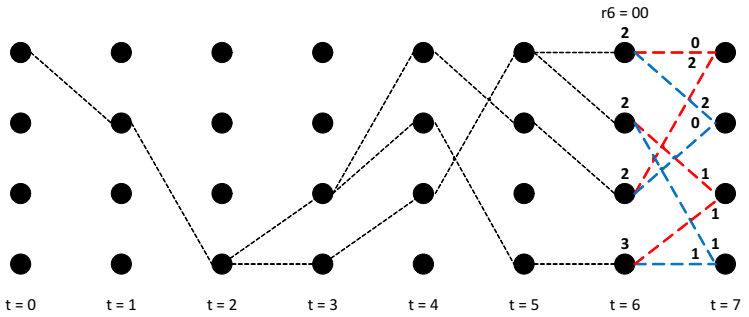


Figure 2.18: *Path extension at time instant* $t = 7$

After survivor path selection, the state transition diagram becomes:

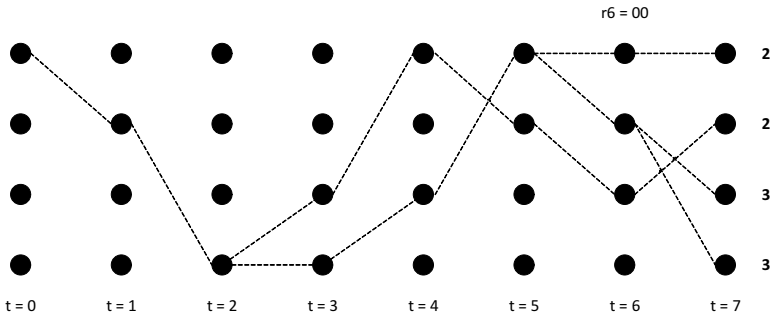


Figure 2.19: *Survivor path selection at time instant* $t = 7$

At time instant $t = 7$ the sequence $\mathbf{r}_7 = 01$ is received.

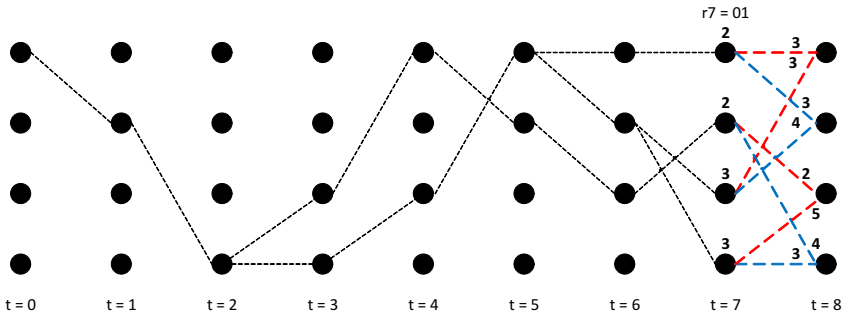


Figure 2.20: *Path extension at time instant* $t = 8$

After survivor path selection, the state transition diagram becomes:

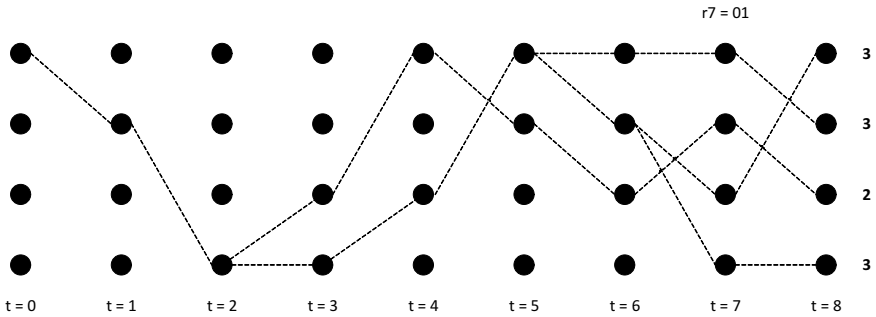


Figure 2.21: *Survivor path selection at time instant* $t = 8$

The decoding is completed after the reception of sequence $\mathbf{r}_7$ by selecting the state at the last stage with best (lowest) metric.

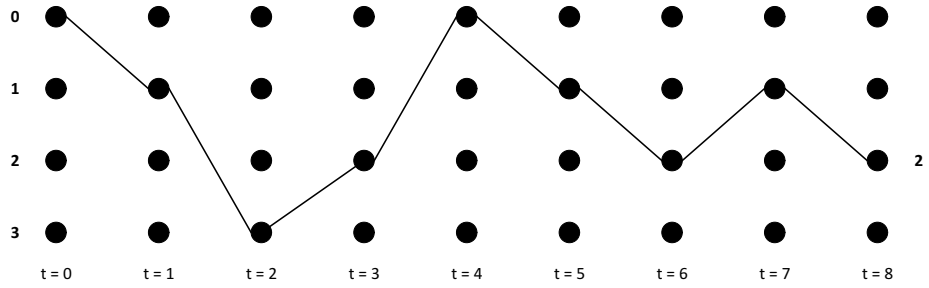


Figure 2.22: *Survivor path with minimum accumulated metric of the decode process*

Beginning with this last state, the trellis diagram is traversed backwards along the survivor path until the first stage in the encoding process is reached, and then it is traversed forwards once more along the survivor path while the input message sequence that originates the obtained state transitions is calculated.

The colored line in the figure below represents the reconstructed message **m** after the Viterbi algorithm has been applied to the received message **r**. Red lines correspond to a logical zero in the message sequence, while blue lines represent logical ones in the message sequence.

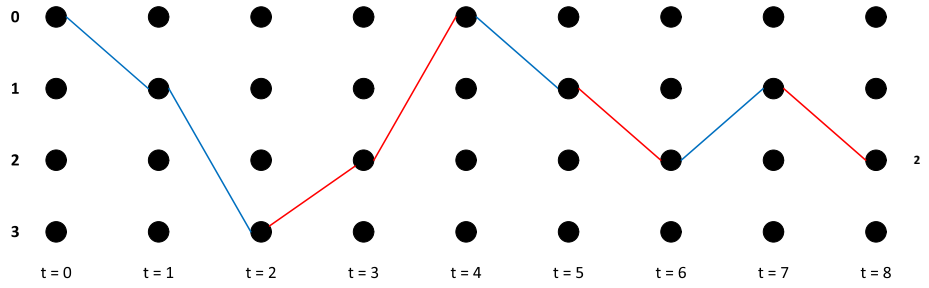


Figure 2.23: *Decoded message* **r**

Comparing the previous result with the contents in figure 2.5 we observe that the Viterbi algorithm can successfully decode an erroneously received noisy sequence **r**.

2.2.2.1 Adaptive Viterbi Algorithm

The adaptive Viterbi algorithm [MW96] is a variation of the classic algorithm. It is aimed at reducing the power consumption and memory requirements of decoders implementing the Viterbi algorithm [VPG13, BSK+13, BSL11b].

The adaptive Viterbi algorithm adds two additional criterions when selecting the survivor paths. First, a threshold T is defined. All candidate paths whose accumulated branch metric exceeds (is *worse* than) this threshold are automatically discarded for further analysis. Even the survivor path of a given state can be discarded if its accumulated metric exceeds this value. Secondly, the number of survivor paths stored at each decoding iteration is reduced from 2^{ν} to a value N_{max}. From the set of survivor paths not exceeding the predefined threshold T, only the first N_{max} paths with best (lowest) accumulated metric are considered for further analysis.

Several Adaptive Viterbi decoders can be found in the literature [STGB02a, TSR+05a, LDZL11a, CC01, STGB02b].

2.2.3 VITERBI DECODER BUILDING BLOCKS

From the example of the previous section, we deduce that the key operations performed by the Viterbi algorithm are the following:

- To calculate the branch metrics between the input sequence $\mathbf{r}$ at a given time instant t and all the possible codewords generated by the encoder.
- To extend the survivor paths at time instant $t-1$ with the state transitions defined by the trellis diagram and calculate the metrics of all these candidate paths as the addition of the branch metric of the precursor state and the branch metric due to the state transition at time instant t.
- To select the path with minimum accumulated metric among all the candidate paths ending at a given state in the encoder at time instant t and to repeat this setp will all the 2^{ν} states in the code.
- To Store into memory the new set of survivor paths.

Figure 2.24 shows the typical block architecture of a Viterbi decoder. The architecture is divided into the following functional blocks:

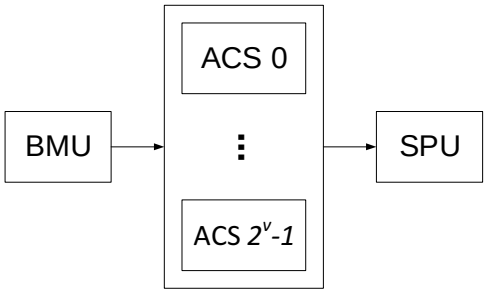


Figure 2.24: *Simplified block diagram of a Viterbi decoder*

- The Branch Metric Unit (BMU) is responsible for calculating the Hamming distances between the received sequence **r** and all the possible codewords of the code.
- A set of 2^{ν} Add-Compare-Selects (ACSs) (one per state in the code) extend the metrics that reach each state in the code by adding the accumulated path metric to the branch metric associated with the state transitions defined in the state and trellis diagrams in figures 2.3 and 2.4, compare the resulting metrics and select the survivor path as the path reaching the state with best (lowest) accumulated metric.
- The Survivor Path Unit (SPU) updates the historic of survivor paths with the outcome of the ACSs and regenerates the estimated message sequence **x**.

The following sections describe each functional block in detail.

2.2.3.1 Branch Metric Unit

The BMU calculates the branch metrics for all state transitions defined in the trellis diagram of the code. It does so by calculating the Hamming distances between the received sequence **r** and all possible codewords in

the convolutional code. For binary convolutional codes with coding rates $R = k/n = 1/2$ the total number of codewords generated by the encoder is 2^n.

The metrics calculated by the BMU can be improved if the functional blocks in the receiver chain provide the decoder with further information about the received signal. For this discussion we will need to expand the initial transceiver architecture first introduced in figure 2.1.

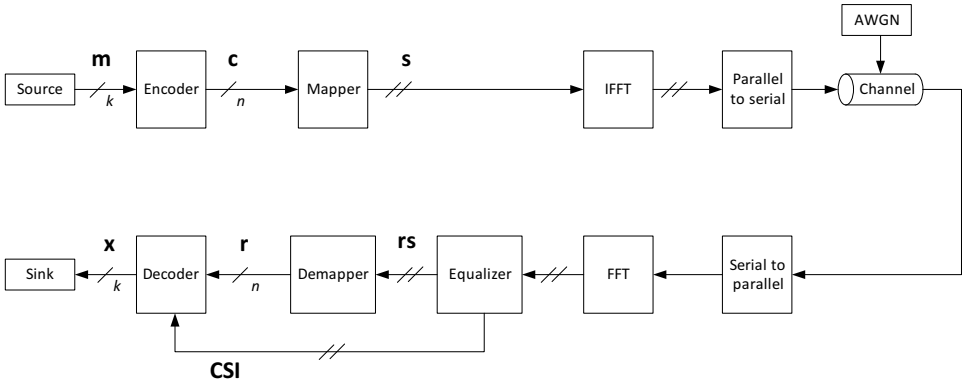


Figure 2.25: *Block diagram of an OFDM transceiver*

Figure 2.25 depicts a basic block diagram of an Orthogonal Frequency Division Multiplexing (OFDM) based communication system. The key differences between this architecture and the early transceiver shown in figure 2.1 are the following:

- The codeword generated by the convolutional encoder is loaded into a *mapper*. This unit takes a block of m codeword bits and generates a symbol $\mathbf{s}$ of an M-ary Quadrature Amplitude Modulation (QAM) constellation, where $M = 2^m$.
- The set of constellation points its loaded into an Inverse Fast Fourier Transform (IFFT) unit. The IFFT is the core element of OFDM based communication systems, as it performs both modulation and frequency multiplexion.
- At the receiver side, a Fast Fourier Transform (FFT) core demodulates the received signal and generates a set of complex symbols.

These symbols are affected by the impulse response of the channel. The channel response is different for all the subcarrier frequencies of the OFDM modulation. An *equalizer* is needed to estimate and compensate this effect.

- Finally, a *demapper* takes the equalized complex symbols and returns the binary representation of the constellation points that are closer in Euclidean distance to them.

Figure 2.26 shows the basic architecture of a BMU. As it can be seen, an array of 2^n elements calculate in parallel all the branch metrics in the trellis diagram for the input sequence **r**. An optional input port labelled *CSI* provides the BMU the channel estimation on multicarrier transceivers.

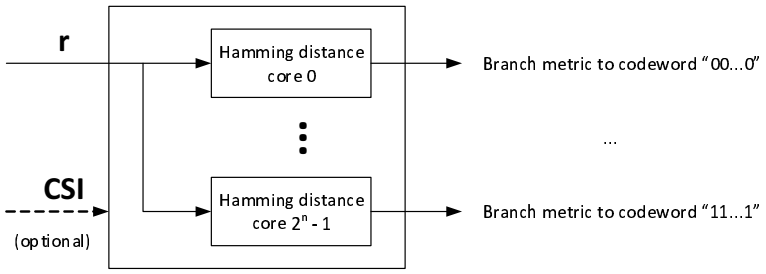


Figure 2.26: *Block diagram of the Branch Metric Unit (BMU)*

Depending on the encoding of the input sequence **r**, Viterbi decoders are classified as *hard* or *soft* decoders. On the other hand, Viterbi decoders that use Carrier Strength Indicator (CSI) are known as CSI-aware decoders. The following sections describe different BMU implementations depending on their inputs **r** and optional CSI.

2.2.3.1.1 Hard decoder

Hard decoders treat the demapped constellation points **r** as a set of logical ones and zeros. Let us consider the modulation and demodulation scheme of a Binary Phase-Shift Keying (BPSK) constellation depicted in figure 2.27. In the transmitter side, a mapper generates a constellation point s_0 whenever it is given a logical zero input bit. Similarly, it generates a constellation point s_1 when given a logical one input bit. Due to the

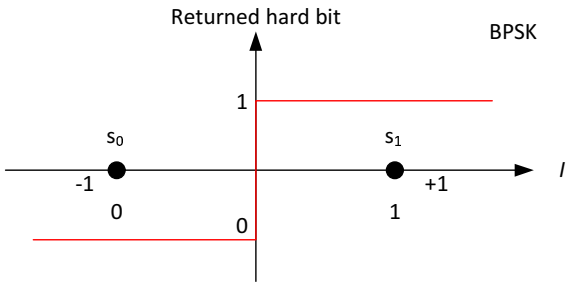


Figure 2.27: *Demap function of a hard decoded bits*

noisy characteristics of the channel, the symbols seen by the receiver differ from s_0 and s_1. The demapper, however, takes the following assumption: if the in-phase component of the received constellation point is negative, then it is considered that a logical zero was transmitted. Similarly, if the in-phase component of the received constellation point is positive, then it is considered that a logical one was transmitted. The *hard* decoding nomenclature comes from the fact that the demapping function is non-continuous and changes abruptly in the decision boundary between s_0 and s_1.

Hard-decoders have a simpler hardware architecture (the Hamming distance between 2 input bits is trivial) but are more error prone in noisy scenarios where the received symbols fall in the decision boundary between s_0 and s_1.

2.2.3.1.2 Soft decoder

Soft decoders use a set of demapping functions similar to that depicted in figure 2.28. Here, the demapper, instead of returning a logical one or zero, calculates a *weight* or *soft* bit, indicating how close the received symbol matches the constellation point s_0 or s_1: if the received symbol is close to s_0, then, with high probability, the transmitted bit was a logical zero. Similarly, if the received symbol is close to s_1, then, with high probability, the transmitted bit was a logical one. The certainty of the soft bits decreases as the received symbol falls in the proximity of the decision boundary between s_0 and s_1

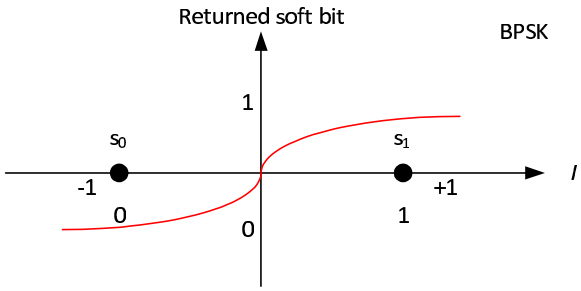


Figure 2.28: *Demap function of a soft decoded bits*

Soft decoders take into account these weight factors to better generate the branch metrics. Soft-decoders require more hardware resources than hard-decoders. However, soft-decision decoding is the generally preferred methods because of its superior performance. It has been determined that soft-decision decoding provides 2 to 3 dB gain over hard-decision decoding [Moo05]. The decoding capacity is the difference (in dBs) of the transmitted signal power required by two different decoders to obtain an identical BER level.

So far floating point precision has been considered when representing the soft-bit values. However, real decoders implement fixed-point logic. By converting the softbit metrics to small integer quantities, it is possible to efficiently accumulate the metrics. It has been found in [HJ71] that quantizing each softbit with 3 bits (eight quantization levels) results in a loss in coding gain of around only 0.25 dB. It is possible to trade metric computation complexity for performance, using more bits of quantization to reduce loss or using less bits for faster, smaller implementations.

2.2.3.1.3 Carrier Strength Indicator aware decoders

Multiple carrier communication systems can provide further information to improve the calculation of the branch metrics. In a multiple carrier system such as that in figure 2.25, each element of the received sequence **r** is decoded from different, independent carriers. The transmission channel affects each subcarrier independently. For instance, some carriers are attenuated more than others. At the receiver side the *equalizer* is necessary to estimate and compensate the impulse response of the channel. When

the channel response is provided to the Viterbi decoder, then the decoder is referred as a CSI aware design.

CSI aware decoders improve their metric calculations by weighting the branch metrics with the channel impulse response. Bits decoded from very attenuated subcarriers have a smaller weight on the branch metrics than those that are decoded from less attenuated subcarriers since they are more prone to mismatches due to the noise.

According to the results in [ACS+08], an extra decoding capacity gain of around 2 dB can be expected from Viterbi decoders that implement CSI over pure softbit based Viterbi decoders. The improvement in the decoding capacity comes at the expense of further computational requirements in the ACS units. To the author's knowledge, CSI aware Viterbi decoder hardware implementations have not been proposed in the literature.

2.2.3.2 Add-Compare-Select Unit

The ACS unit is the core element of the Viterbi decoder. It is responsible for extending the path though the trellis by *adding* the branch metrics of the survivor paths of the previous decoding time instant with the branch metrics due to the state transitions, comparing the resulting metrics of all the candidate paths converging into a given state in the trellis, and selecting the path with better (lowest) accumulated metric.

ACS units are computationally demanding modules. It has been estimated that the Viterbi decoder is the most computationally intensive module in an OFDM based receiver [MGJ04], even more than the FFT, synchronization or equalization modules.

The basic architecture of a ACS unit is shown in figure 2.29. From the convolutional code definitions of section 2.2.1 we know that a convolutional code with a constraint length of ν has a total of 2^ν states. The most direct approach to the Viterbi algorithm instantiates 2^ν ACS units, each responsible of the add, compare and select operations that occur in a state of the code. The *radix* of the ACSs indicates the number of candidate paths the ACS must analyse to select the survivor path. For the serial convolutional codes, the Viterbi decoder implements radix-2^k ACSs. For parallel implementations of the convolutional encoder as shown in section 2.2.1.1, the Viterbi decoder implements radix-2^{pk} ACSs.

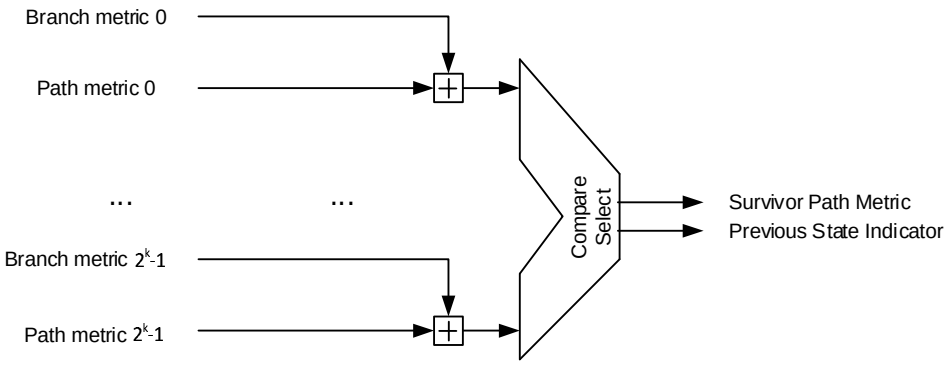


Figure 2.29: *ACS unit block diagram*

As it can be seen in figure 2.29, each ACS unit essentially consists of a series of adders, comparators and multiplexers. Since in a convolutional code with $R = k/n = 1/2$ each state in the trellis can be reached from 2^k precursor states, the hardware requirements of the ACS unit are:

- 2^k accumulators that extend the path metrics of the survivor states with the branch metrics.
- A compare tree that compares the metrics of all candidate paths.
- A 2^k:1 multiplexor that identifies the survivor path.

The ACS operation must be completed before new branch metrics are calculated by the BMU. High throughput Viterbi decoders are generally operated with a clock frequency equal to the data rate of the communication link. In this scenario, the ACS operation must be completed in a single clock cycle, with no possible pipelining opportunities. Consequently, the ACS is the bottleneck of very high speed Viterbi decoders. On these systems, the logic of the ACS must be optimized to minimize its propagation delay as low as possible. A possible way of improving this limitation is by modifying the order in which the ACS executes its operations: if the compare operation is executed first followed by the select and add operations, then some authors [HYS14] suggest that the ACS slightly improves its timing results.

So far the Viterbi decoder has been assumed to decode a continuous data stream. This implies that the accumulated metrics of the survivor paths

increase as time advances. On real implementations of Viterbi decoders this is a severe drawback. The finite precision of the accumulators inside the ACS will overflow sooner or later if the decoding elapses for a prolonged period of time. When an accumulator associated to a state on the trellis overflows its accumulated metric is abruptly reduced. This, in turn, can introduce decoding errors in the following iterations of the algorithm. The state that has overflowed will produce paths with lower metrics than those states that have not overflowed. Since the Viterbi algorithm selects the paths with minimum accumulated metric, the decoder can mistakenly select as survivor paths those paths that are generated from a state that has previously overflowed.

The previous state indicator in figure 2.29 is an identifier of the precursor state in the trellis that originated the survivor path. This information is needed by the last functional block in the Viterbi decoder: the SPU.

2.2.3.3 Survivor Path Unit

The SPU unit is responsible for updating the historic of survivor paths in the trellis and generating the estimated transmitted sequence $\mathbf{x}$ depicted in figure 2.1.

In convolutional codes, data is typically encoded in a stream. Once the encoding begins, it may continue indefinitely. If such a data stream were to be decoded using the Viterbi algorithm, the paths through the trellis would have to have as many state transitions as the length of the message $\mathbf{m}$. For long data stream sequences this could require an extraordinary amount of data to be stored since the decoder would have to store 2^{ν} paths whose lengths increase every decode cycle. Additionally, this results in an enormous decoding latency. Strictly speaking it would not be possible to output any decoded values until the maximum likelihood path was selected at the end of the decoding.

However, it is not necessary to wait until the end of the transmission. In general when the Viterbi algorithm has iterated a sufficiently large amount of time, the set of survivor paths share a common initial state transitions in their trellis diagram. This can be seen in the decoding example in section 2.2.2 on page 22. In this example, after time instant $t = 5$, the state transitions of all survivor paths are identical between time instants 1 and

2. Therefore, at time instant $t = 4$ it is safe to make a decision about the message transmitted during time instant $t = 1$ and $t = 2$.

The number of decode iterations that are allowed in the Viterbi algorithm before an output decision is made is called the traceback (or decoding) depth τ of the decoder. The value τ can then be seen as the latency introduced by the Viterbi decoder operation. Consequently, it is only necessary for the Viterbi algorithm to maintain a small window of the trellis diagram of the survivor paths. This window goes from the current time instant t back to time instant $t - \tau + 1$

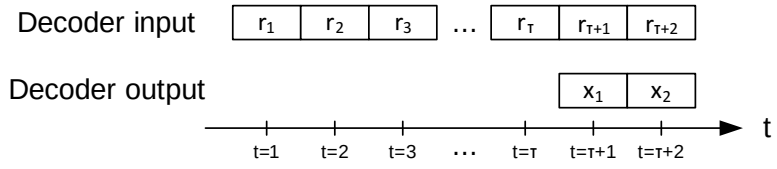


Figure 2.30: *Viterbi decoder latency*

At a system level the Viterbi decoder can be seen as a delay unit. This delay is represented in figure 2.30. From time instant $t = 1$ the receiver chain provides the decoder with new received vectors $\mathbf{r}$. However, the allowed traceback depth introduces a latency of τ cycles. It is not until time instant $t = \tau + 1$ where the decoder takes a decision on the code bits $\mathbf{r}_{t-\tau}$, where $\mathbf{r}_{t-\tau}$ is the received sequence $\mathbf{r}$ at time instant $t - \tau$.

Even though it is possible to make incorrect decoding decisions on a finite decoding depth, this error, known as truncation error, can be minimized if the decoding depth of the decoder is sufficiently large. It has been found in [For74, HC77] that if a decoding depth of about five to ten times the constraint length of the code is employed, then the performance loss due to the truncation error is very small compared with the full length solution.

There are a few options when a decision about the output $\mathbf{x}_{t-\tau}$ has to be made at time t [Wic95]:

- Output $\mathbf{x}_{t-\tau}$ on a randomly selected survivor path.
- Output $\mathbf{x}_{t-\tau}$ on the survivor path with best metric.

- Output $\mathbf{x}_{t-\tau}$ that occurs most often among all the survivor paths.
- Output $\mathbf{x}_{t-\tau}$ on any path.

However, for sufficiently large values of τ the performance difference among these alternatives is very small.

According to the way survivor path historic data is managed, the SPU can be based on a *register exchange* implementation or a *traceback* implementation.

2.2.3.3.1 Register exchange implementation

The register exchange implementation of the SPU operates by storing the message associated to each survivor path in a structure similar to a shift register. Each shift register is τ elements wide and there is a total of 2^{ν} shift registers, one per state in the encoder.

For example, the contents of the shift registers of the register exchange implementation at time instant $t = 5$ of the convolutional code example in section 2.2.2 are depicted in figure 2.31.

	t=5	t=4	t=3	t=2	t=1
State 0	0	0	1	1	1
State 1	1	0	0	1	1
State 2	0	1	0	1	1
State 3	1	1	0	1	1

Figure 2.31: *Register exchange implementation example. Register values at $t = 5$*

The oldest decoded message bit of the survivor path is stored in the least significant position in the shift register.

When the ACS unit selects the next survivor path in the next decoding time instant, the contents of all shift registers of the SPU are shifted one position to the right. This process is illustrated in figure 2.32.

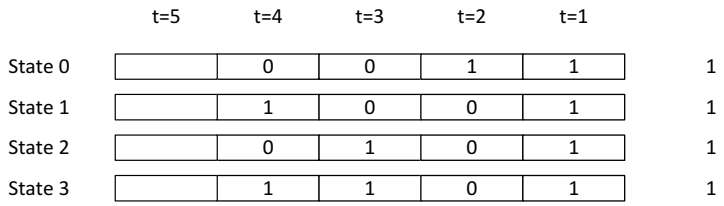


Figure 2.32: *Register exchange implementation example. Data shift at time intant* $t = 6$

In figure 2.32, since a decoding latency sufficiently large has been permitted, the output of all shift register converges to an identical decoded message $\mathbf{x}$ at time instant $t = 1$. This value is the Viterbi decoder output $\mathbf{x}$ for the received vector $\mathbf{r}$ at time instant $t = 1$.

After that, the contents of all shift registers are updated by copying the values of the state associated to the precursor state as pointed by the ACS units. In our decode example, the survivor path selection at time instant $t = 6$ highlighted in figure 2.17 is as follows: the survivor paths ending at states 0 and 1 at time instant $t = 6$ are the extension of the survivor path ending at state 0 at time instant $t = 5$. Consequently, the value of the shift register associated to state 0 is copied into the shift registers associated to states 0 and 1. Similarly, the survivor path ending at state 2 at time instant $t = 6$ is the extension of the survivor path ending at state 1 at time instant $t = 5$. Therefore, the values of the shift register associated to state 2 are updated by copying the value of the shift register associated to state 1. Finally, since the survivor path associated to state 3 at time instant $t = 6$ is the extension of the survivor path ending at state 3 at time instant $t = 5$, the contents of the shift register associated to state 3 is copied in itself. This update process is illustrated in figure 2.33.

Finally, the transmitted message $\mathbf{m}$ that generates the state transitions indicated by the ACS units are stored in the most significant elements of the shift registers. This process is illustrated on figure 2.34.

The register exchange implementation of the SPU has the lowest overall decoding latency of τ cycles [MZMD13].

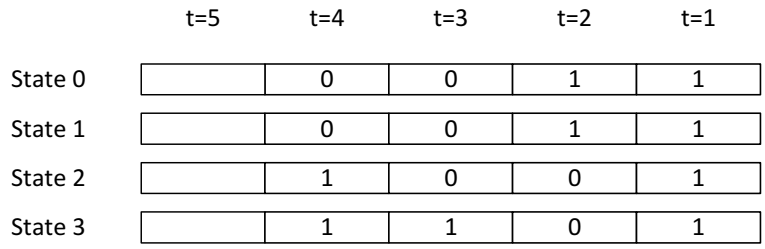


Figure 2.33: *Register exchange implementation example. Shift register content update at time instant $t = 6$*

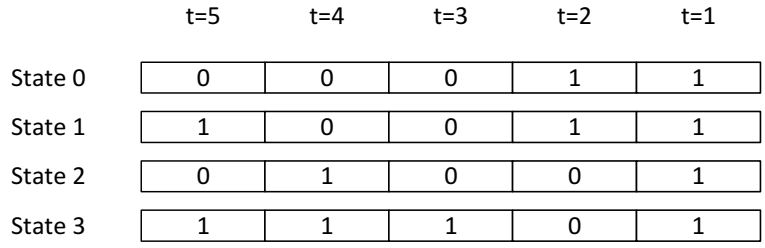


Figure 2.34: *Register exchange implementation example. Store decoded bits at time instant $t = 6$*

2.2.3.3.2 Traceback implementation

Traceback implementations of the SPU are based on Last-In First-Out (LIFO) memories that store the historic of state transitions estimated by the ACS units.

Every clock cycle, the LIFO memory is loaded with a vector of 2^{ν} elements. Each element of the vector is associated to a single state of the convolutional code. During each iteration of the algorithm, the element of the vector stores a pointer to the precursor state of the survivor path ending at the state associated to the vector element. Since in a code with 2^{ν} states and a coding rate of $R = k/n$ 2^k different states can be reached from the same precursor state, the memory of the *traceback* implementation of the SPU is organized as a $\left[\tau \times (2^{\nu})k\right]$ LIFO memory, where τ is the traceback depth of the decoder.

For example, figure 2.35 contains the contents of the memory of a traceback implementation at time instant $t = 5$ of the convolutional code example in section 2.2.2

	State 0	State 1	State 2	State 3
t = 5	2	0	1	1
t = 4	2	2	3	1
t = 3	0	2	3	3
t = 2	0	0	1	1
t = 1	0	0	-	-

Figure 2.35: *Traceback implementation example. Memory contents*

Note that at time instant $t = 1$, there were no precursor states for states 2 and 3 of the trellis in figure 2.8. Consequently, the elements associated to states 2 and 3 at time instant $t = 1$ in the LIFO memory of the traceback unit in figure 2.35 show null data.

After τ iterations, the decoding process is freezed. The survivor path of the decoder is reconstructed by identifying the ACS with minimum accumulated metric and by tracing *backwards* the contents of the LIFO memory (hence the name of the SPU implementation) from the last decoding iteration to the first decoding iteration.

If in our example, a decision were to be made at time instant $t = 5$, figure 2.36 depicts the traceback process. First, the state with minimum accumulated metric is identified. In our example, we know from figure 2.15 that the survivor path with minimum accumulated metric at time instant $t = 5$ ends at state 0. Then, the contents of the LIFO memory is traced back from this state. At time instant $t = 5$, the precursor pointer stored at the vector element associated to state 0 in the LIFO memory is 2. Similarly, at time instant $t = 4$ the precursor pointer stored at the the vector element associated to state 2 in the LIFO memory is 3. The backtrace process continues until all the addresses of the LIFO memory are read. In figure 2.36, the memory elements accessed are highlighted in red. Finally, all state

transitions are rearranged in reverse order. For our case, the traced back state transition sequence associated to the survivor path ending at state 0 at time instant $t = 5$ is $0 \rightarrow 1 \rightarrow 3 \rightarrow 3 \rightarrow 2 \rightarrow 0$.

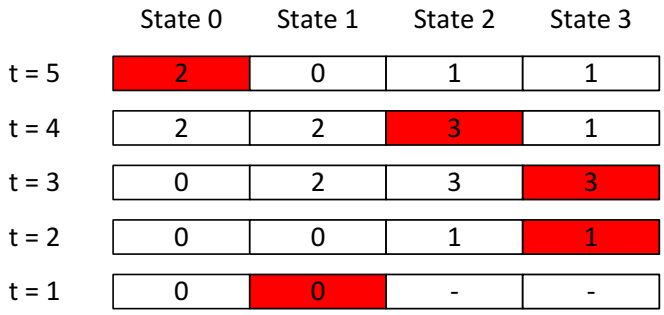


Figure 2.36: *Traceback implementation example. Traceback step*

Once the survivor path has been reconstructed the decoded message can be generated by associating the state transitions in the reconstructed path with the input messages to the encoder as pointed by the trellis diagram.

In our example, the state transition $0 \rightarrow 1 \rightarrow 3 \rightarrow 3 \rightarrow 2 \rightarrow 0$ corresponds to the a decoded message $\mathbf{x} = 11001$.

The traceback implementation of the SPU has twice the latency of the register exchange implementation: τ cycles are necessary to generate the survivor path and another τ cycles are necessary to output de decoded message.

In cases where decoding can not be freezed during the traceback state, the First-In First-Out (FIFO) memory of the SPU is implemented as an array of memories in ping-pong configuration (one memory stores the newest survivor path transitions while the other is being traced backwards) or by using multiple-port Random Access Memories (RAMs) (one port in the RAM memory is used as a write pointer of state transitions of the new survivor paths while the other pointer is used as a red pointer for the traceback state).

2.2.4 VITERBI DECODER IMPLEMENTATIONS

The Viterbi decoder is the most complex element of the receiver in a digital transceiver [MGJ04]. Consequently, a lot of optimization effort is carried out during the design stage to obtain an architecture that meets system requirements such as decoding capacity, data throughput and area resource utilization.

The decoding capacity is the BER figure that a system can achieve for a given channel noise levels. On the Viterbi decoders, the constraint length ν of the convolutional code is the first indicator of the decoding capacity that can be achieved in the system: decoders with higher values of ν are more robust against noise in the channel. However, higher values of ν lead to more complex Viterbi decoder implementations, which require more hardware resources and, generally, result in slower implementations.

The size of the Viterbi decoder is directly related with the manufacturing costs of the design. These costs are the ultimate indicator that will decide wether a hardware architecture is suitable for mass consumption or not. The convolutional code implemented by the Viterbi decoder is an initial gauge of the performance and the cost, in terms of area, of the design. For example, the constraint length of the code hints about the decoding capacity of the decoder, while it also expresses the total number of ACS units required in the design. The traceback depth of the algorithm illustrates about the truncation error of the implementation and also gives an idea about the memory requirements of the SPU and the delay of the core. The works proposed in the literature try to find a trade-off between the maximum decoding capacity predicted by the convolutional code and the costs in terms of area of the implementation of the Viterbi decoder.

In this section the state of the art of Viterbi decoders found in the literature is presented. Depending on the optimization goal each Viterbi decoder pursuits, decoders are classified into area efficient decoders, high data throughput decoders and reduced latency decoders. Finally, reconfigurable decoders are introduced.

2.2.4.1 Area efficient Viterbi decoders

The most demanding components of a Viterbi decoder in terms of area are the ACS units followed by the SPU. In its simplest radix-2 version, an

ACS unit requires two adders, one comparator and a 2:1 multiplexor. This hardware resources requirements increase exponentially with the constraint length of the code. Therefore, minimizing the number of ACS units in the decoder will reduce the manufacturing cost of the device. The first solution to narrow the influence on the area resource utilization of the decoder is to constraint the designs to convolutional codes with manageable constraint lengths. Keeping the constraint length of the convolutional code as low as possible has, however, a negative impact on the maximum decoding capacity achievable by the decoder, as codes with low states are more error prone than codes with higher constraint lengths.

Another solution that tries to reduce the area cost of the ACSs relies on instantiating less ACS than the total number of states in the trellis and sharing the available units between the different states of the convolutional code. This approach is very popular on Viterbi decoders implementing the adaptive algorithm.

Table 2.3 summarizes the hardware resource consumption of different ACS units found in the literature. The authors in [BSK+13] and [BSL11a] present two sets of results: the initial hardware consumption of a typical ACS unit and the resource utilization of their modified adaptive ACS unit. All ACSs in the table are designed for a convolutional code with $R = 1/3$ and a 3-bit wide softbit word and have been implemented in a Xilinx xc3s500 Field Programmable Gate Array (FPGA).

As can be seen in table 2.3, adaptive ACSs have higher costs in terms of area compared to traditional ACSs, requiring two to five times the logic of traditional ACS. This is because adaptive ACSs need to compare the resulting survivor path length with the threshold level of the algorithm, an operation that is not needed on the traditional ACS implementation. However, adaptive ACSs have four benefits over traditional ACSs. First, the power consumption of Viterbi decoders based on adaptive ACSs is at least half the total power of Viterbi decoders based on traditional ACSs. Second, the memory requirements of the SPU of adaptive Viterbi decoders is reduced over traditional ACS based Viterbi decoders because in the former less survivor paths are generated at each decoding cycle. Third, the more area resource intensive adaptive ACSs lead to faster hardware architectures. In [BSK+13] for example, the adaptive ACS unit is at least three times faster than its traditional counterpart. Note that the authors in [BSL11a] present no data regarding the register utilization of the tradi-

	[BSK+13] ACS	[BSK+13] Adaptive ACS	[BSL11a] ACS	[BSL11a] Adaptive ACS
4 input LUTs	36	81	8	47
Registers	20	24	-	8
Critical path delay [ns]	18.45	3-6	-	-
Power consumption[mW]	45	23.58	143	38

Table 2.3: *Comparison of normal and adaptive ACSs with* $code_n = 2$, $code_k = 1$ *and* $softbit_{bw} = 3$ *in Xilinx xc3s50 FPGA*

	FPGA	ν	Clock [MHz]	Decode Speed [Mbps]
[STGB02b]	xc4036	7	40.5	0.33
[ZB03]	xcv300	8	101	12.5
[TSR+05b]	xcv1000	12	13	0.415
[BPBS06]	xc2v2000	8	32.26	2

Table 2.4: *Summary of adaptive Viterbi decoders found in the literature implementing hard-decoding with $k = 1$ and $n = 2$*

tional ACS unit because they consider that its functionality falls beyond the scope of the ACS unit itself. This does not mean, however, that these registers are to be omitted in the overall Viterbi decoder architecture. Finally, the extra area requirements of each adaptive ACS can be overcome by the fact that adaptive decoders can operate with less ACS units (they only require up to N_{max} units) which in turn reduces the memory requirements of the SPU and therefore, its size.

Area savings in the Viterbi decoder can be obtained by sharing the available ACS unit among the states in the trellis. On the one hand, this solution reduces the number of ACS units that are required on the decoder. On the other hand, it reduces the decoding throughput of the decoder, as more clock cycles are required to calculate all the survivor paths in the code. Table 2.4 summarizes the implementation results of several Viterbi decoders taking this approach. All the decoders presented in the table implement a hard-decoding BMU and are based on a $R = k/n = 1/2$ convolutional code.

As can be seen in table 2.4, sharing ACS units between different states in the trellis has a penalty on the data throughput that can be achieved by the design. Indeed, since several clock cycles are required so that the available ACSs obtain all survivor paths, the ratio between the system clock speed and decoding throughput is proportional to the reduction in ACS units.

For example, in [STGB02b] a Viterbi decoder with 128 states and a single ACS unit is generated. Therefore, the decoder requires 128 clock cycles to generate all the survivor paths at each decoding instant. Since the decoder is clocked with a 40.5MHz clock, this means that the decoding speed of the solution is reduced to $40.5/128 \approx 0.33$ MHz. A more extreme case found in the table is [TSR+05b]. Here, a Viterbi decoder for a convolutional code with $2^{12} = 4096$ states is required. Instead of instantiating 4096 ACSs, the authors choose to instantiate a more manageable quantity of only 128 ACSs. Therefore, each ACS unit is shared between $4096/128 = 32$ states and thus, for a clock speed of 13 MHz, the decoding throughput of the system is reduced to $13/32 \approx 0.415$ Mbps.

Not all adaptive Viterbi decoders implementations found in the literature reduce the number of available ACS units to minimize the area resource utilization of the decoder. Since the adaptive Viterbi algorithm reduces the number of survivor paths generated, the memory requirements to keep track of these paths is reduced in adaptive decoders, which in turn reduces the area requirements of their SPU. Since no ACS sharing is done, these decoders obtain a decoding throughput that matches their system clock. Examples of these solutions are the proposed works in [CV03] and [LDZL11a]. The former obtains a Viterbi decoder that reaches a decoding throughput of 60.5 Mbps on a Virtex-II FPGA, while the latter obtains a decoder achieving a decoding throughput of 202.587 Mbps on a xc5vlx300 Virtex-5 FPGA.

2.2.4.2 Data throughput enhanced Viterbi decoders

Data throughput is the amount of data that can be decoded per unit of time. Even if giving a hard value from which a system is considered to be high throughput is questionable and open to debate, during this research work we will consider that systems achieving decoding throughputs above several megabits per second to be high throughput.

High decoding throughput works found in the literature obtain this goal by using two different techniques: simplifying the architecture of the decoder, which results in faster implementations, and obtaining parallel implementations of the decoder architecture.

The work in [GXC+13] belongs to the first group: those that maximize data throughput by obtaining faster implementations. Here, the Viterbi de-

coder reaches decoding throughputs of 1.25 Gbps. However, the decoding throughput is obtained at the expense of a simplified decoder architecture. For instance, the decoder implements a hard decoding BMU and the constraint length of the convolutional code is $\nu = 2$. Hard decoding reduces the size of the accumulators of the ACS units. This helps increasing the achievable clock speed of the design, which in turn enhances the decoding throughput of the decoder. However, as was mentioned in section 2.2.3.1.2, hard decoders have a 3 dB decoding loss compared to decoders implementing soft decoding. Besides, the reduced constraint length of the decoder limits the decoding capacity of the convolutional code. Therefore, in this case, a significant tradeoff between the throughput and decoding capacity of the decoder is made.

The proposed work in [SV13] is similar to that in [GXC+13]. Here, the decoder achieves a maximum decoding throughput of 394Mbps with a system clock rated at the same speed. However, to do so the convolutional code being implemented in the Viterbi decoder is simplified: the proposed Viterbi decoder implements a hard-decoding BMU and is based on a four state ($\nu = 2$) convolutional code with $R = k/n = 1/2$. Therefore, the design in [SV13] minimizes the hardware resource utilization and achieves a very high speed implementation at the cost of lower decoding capacity. When implemented in a Xilinx Virtex-7 xc7vx330t-ffg1157-3 FPGA, the decoder requires only 198 registers and 390 Look-up Tables (LUTs) and achieves a maximum clock speed of 394 MHz. As a reference, a typical 64 state Viterbi decoder such as that in [Xil11c] requires 2573 LUTs on a Virtex-6 FPGA.

Authors in [SSV14], on the other hand, propose a convolutional code based communication system that reaches data speeds of 4 Gbps. This high data throughput is obtained by parallelizing the transceiver architecture: the total bandwidth of the system is divided into eight parallel, independent lower speed streams. This property relaxes the constraints on the Viterbi decoder, as this unit must now only have to deal with data streams of 500 Mbps. Compared to the former [SV13], the reduced data throughput of the Viterbi decoder in [SSV14] allows implementing much more robust decoding architectures. For instance, here the decoder implements an 8-level soft-decoding BMU and a convolutional code with $\nu = 6$. Compared to [SV13] where a hard Viterbi decoder with $\nu = 2$ was implemented, the decoder in [SSV14] has a much better decoding capacity at the expense

of a higher hardware resource utilization to implement the eight Viterbi decoder instances.

In [VNS12b] a similar approach to that in [SSV14] is presented: the total bandwidth requirement of the application is divided into independent lower speeds streams. In [VNS12b], however, the total number of parallel streams is not fixed and is a function of the maximum throughput the Viterbi decoder achieves with different softbit quantization levels: when less bits are used to represent the softbit values, the achieved Viterbi decoder reaches higher clock speeds (and slightly worse decoding capacity) and therefore, less units need to be instantiated in the system.

An opposite technique to obtain data throughputs higher than the system clock is the parallelization of the convolutional code. This technique was introduced in section 2.2.1.1 and basically rearranges the states in the trellis diagram so that each ACS can generate the survivor paths of due to consecutive input symbols to the encoder. In this way, if a radix-2^k ACS can generate survivor paths corresponding to k input symbols to the encoder, a radix-2^{pk} can generate survivor paths corresponding to pk input symbols to the encoder. As a consequence, the decoding throughput of a radix-2^{pk} based ACS is p times its base clock speed. However, parallelization of the ACS has a significant impact on the area resource utilization of the parallel ACS. Table 2.5 lists the hardware resource utilization of ACSs of different radix. As can be seen, the number of adders in a parallel ACS increases lineally with p, the number of comparators in the ACS increase exponentially and a $p:1$ multiplexor is required. Therefore, we conclude that parallelization of the convolutional has a high impact on the area utilization of the decoder and it is only suited in scenarios where high data throughput with respect to the system clock speed is pursuit such as in [TNHN99] and [NG13].

Following the discussion of the parallelization of the ACS unit, the authors in [BK13] present 3 different architectures that implement radix-2^{pk} ACSs with $p = 2$ and $k = 1$. Type 1 architecture is a straight forward translation into hardware of the parallel trellis diagram of figure 2.7. Type 2 architecture calculates the maximum value among four input operands by using LUTs which take six MSBs of each relative comparison results among two operands. Finally, type 3 architecture reorder the logic elements of the type 1 architecture as described by [Par99] to obtain a faster

p	Radix-2^{pk}	Adders	Comparators	Multiplexor size
1	radix-2	2	1	2:1
2	radix-4	4	3	4:1
3	radix-8	8	7	8:1

Table 2.5: *Resource utilization of ACSs of different radices for convolutional codes with $k = 1$*

ACS	Gate count	
	400 MHz	250 MHz
Type 1	3011	2040
Type 2	3460	2848
Type 3	4173	2703

Table 2.6: *Gate count comparison of different radix-4 ACSs at different clock speeds in [BK13]*

implementation. Table 2.6 summarizes the area resource utilization of some implementations of the various ACS architectures at different clock speeds.

The results in table 2.6 indicates that the most area efficient parallel ACS unit is type-1. The area resource utilization of the type-2 and type-3 ACS implementations differ with the clock frequency: at lower frequencies, the type-3 has a slight gain in area resource utilization over type-2. However, as the clock frequency increases, so does the hardware requirements of type-3 ACSs and it becomes the most demanding implementation in terms of area resource utilization.

Table 2.7 compares the area resource utilization of all three parallel ACSs when they have been implemented at their highest achievable clock speed. As seen in the table, the fastest and most resource intensive parallel ACS implementation is type-3. On the other side, the ACS implementation

	Maximum clock speed [MHz]	Gate count
Type 1	270	2583
Type 2	263	3242
Type 3	400	4173

Table 2.7: *Gate count comparison of different radix-4 ACSs at their achievable maximum clock speed in [BK13]*

with less area resource requirements correspond to type-1, but it achieves a clock speed a 37.5% slower than that of the type-3 implementation. Type-2 ACS implementation offers an intermediate area resource utilization compared to type-1 and type-3 ACSs, but overall it is the slowest architecture of all, obtaining a reduction in the operating clock of around 2.5% compared to type-1 implementation.

The results in the first column in the table were obtained when the ACSs were implemented to obtain the maximum achievable clock speed. As can be seen, type-3 ACS offers the minimum path delay of all three architectures: 2.5ns. Type-1 and type-2 architectures, on the other hand, have a very similar behaviour in terms of their achievable clock speed of around 3.7ns. With respect to the area resource utilization, at identical clock speeds type-1 ACS is the most area efficient architecture for all clock speeds and depicts an advantage of around 25% over type-2 ACS architecture and 30% over type-3 ACS architecture.

Authors in [VNS12a] explore further more the parallelization of the convolutional code and present the implementation results of Viterbi decoders for various type-1 ACS radix. These implementation results are summarized in table 2.8. As can be seen, for the radix-2 case, the throughput and clock frequency of the system are identical. When the ACSs are updated to radix-4 the maximum clock speed achievable by the decoder is reduced a 44%. However, due to the parallelization of the algorithm, the overall throughput of the system is increased by a 12% with respect to the radix-2 based implementation. This increase in the throughput of the decoder comes at a higher cost in terms of hardware resource utilization: the radix-4 implementation requires 2.4 times as many LUTs and 58% more registers

	LUT	FF	Freq [Mhz]	Thoughput [Mbps]
Type 1 Radix-2	2253	1550	356	356
Type 1 Radix-4	5498	2450	200	400

Table 2.8: *Logic area and throughput comparison of Viterbi decoders implementing type-1 ACSs of different radixes in [VNS12a]*

as the radix-2 based implementation. Therefore, we conclude once more that parallelization of the convolutional code increments the data throughput of the system at the expense of a higher cost in area of the design. Consequently, this solution is suited for applications that give preference to data throughput over area.

2.2.4.3 Latency optimized Viterbi decoders

The decoding latency of the SPU is described as the time (in number of clock cycles) that the SPU requires from the moment new data is generated by the ACSs until a decision is validated at the output of the core. Table 2.9 summarizes several Viterbi decoders: [SV13] and [MZMD13] implement register exchange based SPUs, while references [NBA13] and [HLS14] implement traceback based SPUs. As can be seen, traceback based implementation are the only ones that require RAM resources, while the register utilization is greater on register exchange based implementations. Let us take for example [SV13] and [HLS14]. The former implements a register exchange based SPU and the latter implements a traceback based SPU. Both of them use hard decoding on the BMU and implement convolutional codes of identical complexity (in both cases $k = 1$ and $n = 2$). However, even if the traceback depth on [SV13] (32) is slightly lower than that on [HLS14] its register utilization is around 3.5 times that of the traceback based implementation. With respect to the decoding latency of all implementations, all register exchange based decoder have a latency of around τ cycles, while traceback based decoders have a latency in the order of 2τ cycles. The advantage in the decoding latency of reference [NBA13] over [HLS14] is because the former uses RAMs with multiple read/write ports that help obtaining the survivor path transitions much quicker than the

	FPGA	Softbit	SPU type	ν	τ	Registers	LUTs	RAMs	Latency	Clock [MHz]
[SV13]	xc7vx33t	1	RE	2	32	198	390	0	τ	394
[MZMD13]	xa3s500	-	RE	6	18	2356	6261	0	$\tau + k$	118
[NBA13]	xa3s500	-	TB	6	18	-	-	2	1.5τ	97
[HLS14]	xc6vcx75t	1	TB	6	35	55	113	2	2τ	165

Table 2.9: *Summary of the latency of FPGA Viterbi decoder implementations found in the literature with* $R = k/n = 1/2$

single-port RAM based [HLS14]. As can be seen, register exchange based SPU are preferred over traceback SPU on latency sensitive applications.

Traceback depth based SPUs present better power consumption figures than register exchange based SPU [CRBD13]. This is because at each decoding iteration, the contents of all registers in the register exchange SPU can potentially toogle their logical value, while on traceback based SPU only a bunch of pointers and the contents of an address in RAM is updated. To minimize the power consumption, several works [EDE02, EDE04] propose a modified register exchange array that minimizes toogling. This modification, once more, comes at an increased cost in terms of area resources.

Finally, with respect to the BMU, decoders that implement hard decoding require less area than those that implement soft-decoding. This is because, for soft-decoding, the more quantization levels on the received symbols $\mathbf{r}$ require wider adders in the BMU and accumulators in the ACSs. As has been previously mentioned, soft decoders have a 2 to 3 dB gain over hard decoders [HJ71]. This increase in the decoding capacity of the decoder comes at the cost of an increased demand on hardware resources. Consequently, area sensitive implementations (and, to some extent, time critical implementations) prefer hard decoders over soft decoders at the expense of a worse decoding capacity. The same principle could be applied for BMUs implementing CSI-aware decoding: the extra information due to the CSI translates in larger adders and wider accumulators in the BMU and ACSs. Adding CSI information to the BMU can improve the decoding capacity of the decoder by around 2dBs at the expense of more area resource utilization.

2.2.4.4 Reconfigurable Viterbi decoders

Most of the Viterbi decoder proposals found in the literature are based on FPGA technology [Min11, EdDLSH+13, ASA+12, MM15, XJMCZYD04]. The reconfigurability of the FPGAs makes it an adequate platform to evaluate one of the latest trends in Viterbi decoder implementations: reconfigurable Viterbi decoders.

Reconfigurable Viterbi decoders [LDZL11b, PA14a, PN14, CSN14] are those decoders whose hardware elements (distance calculators in the BMU, ACS units, ...) can be dynamically enabled, disabled and rewired on real

time so that the convolutional code implemented by the Viterbi decoders can be adjusted to the noise characteristics of the transmission channel. This behaviour makes reconfigurable Viterbi decoders an interesting solution for applications such as Software Defined Radio (SDR). The decoding capacity achieved by this decoder can be optimized dynamically as well. In terms of area consumption, these decoders show no improvements because hardware resources are always allocated for the most demanding configuration of the decoder and can not be released when the decoder operates on a lighter configuration.

Reconfigurable Intellectual Properties (IPs) are another set of Viterbi decoders. They offer a wide range of possible configurations and the same set of source files can be reused to generate Viterbi decoders with different characteristics. Their behaviour is defined at compile time: once configured the reconfigurable IPs generate a hardware implementation of a Viterbi decoder with a fixed architecture. Their greatest advantage is, therefore, reusability, since the same set of source files can be used to obtain decoders with different properties. One of the most popular commercial Viterbi decoder IP is the LogiCORE Viterbi decoder [Xil11c] by Xilinx.

The LogiCORE Viterbi decoder is an IP tied up to Xilinx FPGAs. Their IP supports feedforward convolutional codes with constraint lengths ranging from three to nine. It also supports coding rates R ranging from 1/2 to 1/7. The BMU of the IP supports hard and soft decoding. The soft-decoding BMU supports puncturing and input symbols of up to eight bits.

The number of ACSs in the IP is governed by what Xilinx defines as the *mode* of the decoder. Parallel mode decoders implement a total of 2^{ν} ACSs. On serial mode decoders, the number of available ACSs is reduced to one. Therefore, serial mode saves area at the expense of the decoding throughput of the core.

The IP can generate SPUs based on the traceback algorithm only, and do not support register exchange based implementations. Using a traceback based SPU, the IP has a decoding latency of 2τ cycles. However, when the traceback depth of the IP is set to a multiple of six a reduced latency mode is available. In this special mode the latency of the core is reduced by half at the expense of a slight speed penalty.

The number of necessary dual port RAM elements in the SPU increases with the constraint length ν of the convolutional code. For constraint lengths lower than seven a single block RAM is necessary. When the constraint length of the code equals nine, then the necessary amount of block RAMs increases to eight.

The IP also supports what Xilinx denotes as multi-channel decoding. Multi-channel decoding allows sharing the same Viterbi decoder hardware with up to three independent data streams that have been encoded using the same convolutional code. The only restriction is that data from the three streams must be provided sequentially to the decoder: in the first clock cycle, a symbol from the first stream is loaded into the decoder. At the second and third clock cycles, data from the second and third data stream is loaded. At the fourth clock cycle, the next symbol from the first stream is loaded into the decoder.

Multi-channel decoding allows reducing the area utilization of a design by using the same hardware resources with independent data streams. The only limitation is that the Viterbi decoder must be sourced with a clock signal whose frequency is, depending on the number of supported channels, two or three times that of the data rate of each data stream.

Dual rate decoding is another technique supported by the IP aimed at reducing the area requirements of the system. When the IP is configured in dual data rate mode, the decoder can decode two independent data streams coded with different transfer functions $G(x)$ as long as the constraint length ν and traceback depth τ for both encoders are identical.

Tables 2.10, 2.11 and 2.12 summarize the implementation results of the Xilinx reconfigurable Viterbi decoder IP when a decoder with $R = k/n = 1/2$, $\nu = 6$, $\tau = 96$ and 3 soft bits is generated respectively in a Virtex-6, Spartan-6 and Virtex-5 FPGAs with modes (parallel, serial and 3-channel decoder).

The serial mode has the lowest resource utilization and can achieve higher clock speeds than the parallel implementation. However, the data throughput of the parallel mode is higher than that of the serial mode. The 3-channel mode, on the other hand, generally uses as many LUTs as the parallel implementation and twice the registers. It is the mode that obtains the highest clock speeds, but since the IP decodes three independent

	Decoder mode		
	Parallel	Serial	3-channel
LUTs	2573	1326	2410
FFs	1863	1497	3434
RAMs	2	2	2
Clock speed [MHz]	347	422	478
Throughput [Mbps]	347	35	159/channel

Table 2.10: *Implementation results of the Xilinx Viterbi decoder [Xil11c] with $R = k/n = 1/2$, $\nu = 6$, $\tau = 96$ and 3 soft bits on a Virtex-6 6VLX75T-3 FPGA*

	Decoder mode		
	Parallel	Serial	3-channel
LUTs	2442	1196	2914
FFs	1980	1590	3507
RAMs	2	2	4
Clock speed [MHz]	126	158	179
Throughput [Mbps]	126	13	59/channel

Table 2.11: *Implementation results of the Xilinx Viterbi decoder [Xil11c] with $R = k/n = 1/2$, $\nu = 6$, $\tau = 96$ and 3 soft bits on a Spartan-6 XC6SÑX45T-2 FPGA*

	Decoder mode		
	Parallel	Serial	3-channel
LUTs	2457	1326	2410
FFs	1538	1497	3434
RAMs	2	2	2
Clock speed [MHz]	272	364	387
Throughput [Mbps]	272	30	129/channel

Table 2.12: *Implementation results of the Xilinx Viterbi decoder [Xil11c] with $R = k/n = 1/2$, $\nu = 6$, $\tau = 96$ and 3 soft bits on a Virtex-5 5VLX30-3 FPGA*

data streams, its data throughput is divided by three. Consequently, data decoding throughput is maximized with the parallel mode of the IP.

As can be seen from the tables, the achievable clock speed of the design benefits from more modern FPGA technology. The Virtex-6 based implementation is the fastest design of the three, having a lead of around 11.5% over the second fastest implementation, the Virtex-5 based design. This advantage can be explained by the fact that Virtex-6 FPGAs are based on 6 input LUTs while the older Virtex-5 is based on 4 input LUTs. When compared to the Spartan-6 implementation, the Virtex-6 achieves a design that is nearly 2.75 times faster. Even if both Virtex-6 and Spartan-6 are based on the same 6 input LUT design, the slices found in Virtex-6 FPGAs have better resource capabilities than those found in the Spartan-6 FPGA. This difference in the routing capabilities of the FPGAs explain he advantage of the Virtex-6 implementation.

In this section a quick review of Viterbi decoders implementations found in the literature has been presented. Viterbi decoder implementations discussed here try to find a trade-off between the complexity of the convolutional code they implement (which indicates the decoding capacity achievable by the decoder), the hardware resource utilization of the decoder and its achievable clock and data decoding rate. The discussion of platforms

aiming at reducing the verification time necessary to evaluate the performance of this architectures will be the subject of the next section.

2.3 HARDWARE-IN-THE-LOOP SIMULATIONS

2.3.1 INTRODUCTION

In the early years of digital design, the time needed to place a new product into market from scratch was dictated mainly by synthesis and implementation tools. As design tools became mature the verification stage, where the design is ensured to behave as expected and its performance is analysed, became the most time consuming step in digital design [KMPR06]. System verification is a very meticulous process. Cases have been reported where a single misplaced transistor has reduced considerably the performance of entire digital designs [Int11]. In the case of digital transceivers, testers can be overwhelmed by the total number of transmission modes the digital transceiver has to comply with and by the number of real world channel conditions the device must be tested under for each transmission mode.

With the improvement of the synthesis tools, the verification stage has become the design bottleneck and can involve around the 60% or the 70% of the development time [SL97]. As a system's design matures the need to prototyping it physically increases. Any reduction in the verification stage can lead to a considerable shorter time to market.

Many techniques have arisen to speed up simulation and reduce verification time. Among them, methodologies involving hardware in the control loop of the simulation have proven to be the only ones to break the inverse relationship between accuracy in simulation and performance in verification stages [ALR11]. These methodologies are known as Hardware-in-the-Loop (HiL) simulations.

2.3.2 HIL SIMULATION CHARACTERISTICS

HiL simulations lie in the principle of executing the most complex and time consuming tasks of the simulation on dedicated hardware while the lightest ones are executed in software.

Figure 2.37 shows the basic block diagram of a HiL based simulator . As

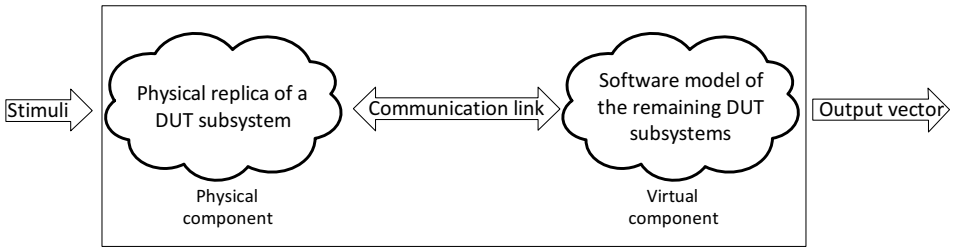


Figure 2.37: *Simplified block diagram of a HiL simulation*

can be seen, the Device Under Test (DUT) being simulated is split in two different subsystems: the *virtual* and *physical* components of the simulator. The virtual component groups all the elements of the simulation that are run on software. The physical component groups all the elements that run on dedicated hardware. Just as in regular pure software simulations, stimuli to the DUT is fetched to the simulator, and simulation results are read from it.

The complexity of the subsystems found in the DUT will be the metric by which it will be decided wether they will be implemented in the physical or virtual components of the HiL platform. This way, subsystems whose software emulation is a very time consuming task will preferably be implemented on hardware. Similarly, when obtaining a software model of the subsystem is infeasible, either due to its complexity or because its behaviour with other subsystems in the DUT is not fully understood, it will be preferably be implemented on hardware.

The communication link depicted in figure 2.37 shares data among the physical and virtual components of the simulator and synchronizes their operation. The bandwidth of this communication link is therefore the real bottleneck of the system and it directly affects the acceleration in simulation time of the platform [OP05].

HiL simulations are becoming an interesting prototyping tool due to their many inherent advantages.

With respect to the cost of the simulation, HiL simulations are a more cost effective solution than physical prototyping because, on the one hand,

they require less hardware and, on the other hand, are quicker to build. Besides, HiL simulators enable embedded system testing at much earlier stages of the development process when errors are easier and less expensive to correct [SI05].

With respect to the verisimilitude of the simulation, HiL simulators often achieve fidelity levels unattainable through purely virtual simulations. Obtaining functionally accurate models of the architectures or algorithms being analysed is essential in any type of simulation. However, during the design stage of a project, obtaining such models is not an easy task. Some times, achieving high levels of accuracy requires complex and time consuming models in the simulation. Other times the behaviour of the component being modelled is not fully understood. In this cases, by prototyping in hardware, HiL simulators get a competitive advantage over other solutions.

With respect to the simulation speed achievable by the simulation platforms, HiL simulations of complex physical devices run faster than purely virtual simulations of the same devices. HiL simulations could theoretically be executed at real-time speeds. However, communication between the physical (hardware) and virtual (software) layers have shown to be the limiting factor of these kind of simulators [OP05].

The simulation speed acceleration of the HiL platforms is the basis of two of their defining characteristics: repeatability and comprehensiveness. With respect to the repeatability, systems that normally operate in highly variable environments (for example, communication systems) can generally be tested in controlled conditions through HiL simulation. With respect to the comprehensiveness, HiL platforms generally make it possible to simulate a given system over a much broader range of operating conditions than what is feasible via purely virtual (software) prototyping.

Finally, HiL simulation allows different teams to develop different parts of a system in hardware without losing sight of integration issues.

2.3.3 HIL SIMULATION USE CASES

HiL are very common to enhance the capabilities of the control loop of a simulator platform. In industries such as vehicular and power electronics the control loop of the simulator is in charge of several hundreds of subsystems [YPAB14]. The high number of peripherals being controlled,

along with the bandwidth necessary to transmit control and data signals, make purely software based control loops unfeasible and thus, HiL simulators become an interesting alternative. When dedicated hardware is instantiated in the control loop of the simulator, HiL platforms are known as Controller-in-the-Loop (CiL).

In the aforementioned work in [LGSB14] a HiL simulator for a Modular Multilevel Converter (MMC) application is presented. In the words of the authors, real-time control of MMC applications is difficult due to two factors: the large number of subsystems involved and amount of measurement and control signals that must be shared with the controlled. These requirements can exceed the processing capabilities of software based solutions [SOBM+15]. To overcome these difficulties, the authors propose an FPGA based HiL simulator testbench. The testbench consists of a Virtex-7 FPGA operating at a clock speed of 200 MHz. The FPGA is connected to the subsystems of the MMC with a fiber optic connection that reaches communication speeds of 4.25 GBps.

In [JUB+11] another simulator for power electronics is presented. Here, the controller is described in Register Transfer Level (RTL), which makes it portable among different FPGA manufacturers and Application Specific Integrated Circuit (ASIC) technologies, but the HiL platform itself is based on System Generator, a proprietary toolbox of Xilinx. By adopting vendor specific IPs, the time necessary to build the HiL simulator is reduced (the library provides, among others, verified mechanisms to interconnect the virtual and physical components of the HiL platform) at the expense of a solutions that is tied to this specific FPGA manufacturer. The authors in this work have opted for a Joint Test Action Group (JTAG) link between the physical and virtual components of the HiL platform.

In [ABAAA15], HiL simulator for photovoltaic systems is discussed. Compared to the early work in [JUB+11], here the authors have opted for a HiL system that is entirely based on DSP builder, a set of proprietary IP blocks provided by the FPGA manufacturer Altera. Therefore, both the controller and the HiL platform rely on technology provided by the FPGA manufacturer, which, on the one hand, reduces implementation time (no RTL code has to be hand written) but, on the other hand, reduces the portability of the solution even more, as all the system is dependent on Altera technology.

In [YPAB14], another application for power grid environments is described. Here, the number of nodes to be controlled is reduced with respect to the previous references, and thus, this work contains a microprocessor based CiL platform that measures the power consumption on the network in real time and makes the best decision about the best routing of the power resources on the fly.

In [KBJR14] a HiL based flight attitude control unit is described. The HiL verification platform benefits the design in several ways. First, it reduces manufacturing costs because the flight controls are designed and simulated virtually. Next, defects can be identified in advance and subsequently rectified and then sent for the manufacturing process. Finally, critical controls can be tested with real time scenarios in advance.

The HiL simulation of the flight controller is done using MATLAB and Simulink-xPC. The simulink-xPC tool box provides connection with external hardware such as aileron, elevators and rudder actuators. The controlled kernel is compiled in Matlab/Simulink and the resulting kernel executable is loaded in a target PC that controls the actuators in the design by means of data acquisition cards.

HiL CiLs are not limited to power electronics and vehicular applications. For instance, in [SWJS15] the analysis of a HiL platform for tracking the headers of a hard disk drive is introduced, and [BKM+15] is an example of HiL simulators for health application.

The work [SWJS15] is similar to [ABAAA15] in the sense that both the DUT and the HiL platform rely on building block provided by the manufacturer (the authors in [SWJS15] rely on Xilinx's System Generator), which reduces implementation time but restraints the portability of the platform.

In [BKM+15] a HiL simulator for cardiac pacemakers is discussed. Here, the virtual component of the platform consists on a PC running a heart model in Simulink, while the physical component of the platform consists of a dedicated microcontroller evaluation board running a prototype version of the firmware of the pacemaker controller that is being investigated.

The communication link between the physical and virtual component of the simulator is based on a standard JTAG interface. This interface

allows a moderate data communication speeds between the components of the HiL platform.

The control loop of the simulator allows adapting the response of the pacemaker in realtime. Consequently, the authors can have a better understanding on the impact several subtle modifications in the pacemaker control model have on the heart model. The pacemaker controller parameter configuration is performed by a slow speed serial UART interface.

The HiL platform has been built using IP building blocks generated by the authors. Therefore, the proposed platform in [BKM+15] leads to a very flexible HiL platform, as the design can easily be ported to other hardware environments.

The increase in simulation speed provided by HiL platforms is also exploited by some authors such as [AGBL+13] and [LLHW08] to significantly reduce simulation time.

In [AGBL+13] an HiL based accelerator is presented for linear system blocks. The physical component of the simulator consists of several cores capable of computing floating point operations with single, double and custom precision following the IEEE 754 standard. The HiL platform has been implemented on a Xilinx Virtex-5 FPGA using the proprietary IPs found in the System Generator tool of the same vendor. Communication between the physical and virtual components of the simulator is performed using a JTAG link.

The authors in [LLHW08] propose a different application for FPGA based HiL accelerators: verification and performance analysis of wireless receivers. Similar to [LGSB14], the application in [LLHW08] has a very demanding communication link bandwidth between its physical and virtual components. Consequently, the authors have opted to implement fast gigabit ethernet interfaces as the communication link between the components. Besides, the description of the DUT of the physical component is based on hand written RTL code only, which augments the portability of the DUT to any other hardware platforms.

On the other hand, the receiver implemented on the physical component of the HiL platform does not fit into a single FPGA. Instead, the DUT is split into an array of FPGAs. This way, the platform gains expandability. However, no tools have been found that divide the design into the FPGAs

available in the array. Consequently, this division must be done by hand, which is a time consuming task and platform specific.

2.3.3.1 Summary of the Hardware-in-the-Loop platform use cases

Table 2.13 summarizes the features of different HiL platform use cases found in the literature.

The bandwidth provided by standard JTAG and UART interfaces is sufficient for moderately fast HiL platforms. However, in more bandwidth demanding scenarios like the receiver architecture in [LLHW08] and [LGSB14] fast gigabit ethernet or fiber optic interfaces are required.

Concerning the DUT implemented in the physical component of the simulator, some authors like [JUB+11] and [LLHW08] rely purely on hand written RTL descriptions. This decision is more time consuming on the early development stages of the design. However, it offers more portability and the simulation results are closer to the final implementation of the DUT than those DUT that have been build around proprietary IP libraries or automatic code generators such as [SWJS15] and [ABAAA15].

Concerning the HiL platform, two clear groups can be distinguished: those that rely on a custom generated platforms that allow portability to different hardware environments, like [BKM+15] and [LLHW08], and those that rely on the tools provided by the FPGA manufacturers that substantially reduce the implementation time of the HiL platform, like [SWJS15] and [ABAAA15].

As it can be seen in the table, must use cases are focused at obtaining a high performance CiL platform, with few exceptions such as [AGBL+13] and [LLHW08] aimed purely at verification and acceleration purposes.

Some of the works found in the table such as [JUB+11] and [ABAAA15] perform an initial parametrical analysis of the DUT before its implementation in the HiL platform. This is done to obtain an optimized DUT architecture with respect to a certain goal (area requirements, achievable clock speed, system performance, ...) before the simulation begins. This parametrical analysis is done using software models of the DUT, which in the case of complex DUTs is a time consuming task and can delay the beginning of the HiL simulation.

	[JUB+11]	[BKM+15]	[SWJS15]	[ABAAA15]	[LLHW08]	[AGBL+13]	[LGSB14]
Data link	JTAG	UART	JTAG	JTAG	Ethernet	JTAG	Fiber Optic
DUT code	Portable RTL	Portable C++	Proprietary SysGen	Proprietary DSP Builder	Portable RTL	Proprietary SysGen	Portable RTL
HiL platform	Proprietary SysGen	Portable Software	Proprietary System Generator	Proprietary DSP Builder	Portable RTL	Proprietary SysGen	Portable RTL
Parametrical analysis	Software simulation	Non applicable	Non applicable	Software simulation	Non applicable	Non applicable	Non applicable
Use case	Control	Control	Control	Control	Verification	Acceleration	Control

Table 2.13: *Summary of HiL platforms*

2.4 CONCLUDING REMARKS

In this chapter a review of the state of the art concerning Viterbi decoders and HiL based simulators has been presented.

With respect to the Viterbi decoder, a trend has been identified in the literature that tries to obtain very high speed communication systems based on convolutional codes. However, in the search of fast communication systems, the reviewed bibliography makes significants tradeoffs.

On the one hand, hardware architectures that achieve very high decoding throughputs do so by opting towards decoding convolutional codes with low constraint lengths and hard-decoder architectures. Consequently, the decoding capacity of these high speed decoders is compromised. Surprisingly, the decoders that achieve the highest communication speeds are those that offer the worst overall BER figures.

On the other hand, communication systems that simultaneously achieve high communication speeds and provide high levels of FEC do so by splitting the total bandwidth of the communication system into independent, slower parallel links. By reducing the communication speed at each parallel line, more robust convolutional codes are considered, and the decoders can use more powerful (and resource intensive) techniques such as soft-demapping for branch metric calculations. However, parallelization of the communication system has a significant cost on the transceiver architecture. First, the parallelization of the transceiver architecture must be planned early on the development stage. Secondly, parallelization comes at a very high cost in terms of area consumption since the transmission and reception chains must be duplicated several times.

Most of the Viterbi decoders found in the literature are rigid. This means that they have defined to decode data from a particular convolutional code in mid and the optimization techniques applied to them are geared towards meeting the their system specifications. If the decoder were to be reused in a different application many of its modules would need to be required, which increases implementation time. Reconfigurable decoders give a more flexible solution, but IPs such as [Xil11c] do not provide low latency options such as register exchange based SPUs.

With respect to the HiL simulation platforms, different software and FPGA based HiL simulators have been presented. The advantages and

disadvantages of tools relying on portable or vendor specific technologies have been discussed. Also, an opportunity for early parametrical system analysis has been identified that can potentially benefit from the inherent fast execution time of HiL simulators.

CHAPTER 3

Objectives

Contents

3.1 INTRODUCTION

Forward Error Correction (FEC) is a technique that improves the reliability of a communication system by adding structured redundancy to the message being sent. This redundancy helps the receiver to make a better decision about the message that was originally transmitted. The Viterbi algorithm is a maximum likelihood sequence decoder of convolutional codes that is used in communication standards such as the 802.11 family, UWB, CDMA, GSM and LTE.

The Viterbi decoder is the most complex component of the receiver chain in a transceiver [MGJ04]. Therefore, the development time of the receiver chain is mostly occupied by this module. Most of the Viterbi

decoders found in the literature in section 2.2.4 have a rigid design and have been optimized towards a very specific goal (area utilization, decoding throughput or decoding capacity), so if the decoder were to be used in another application much of the work would need to be re-done.

In this research work the description of a flexible and portable Viterbi decoder architecture will be proposed. The flexibility of the decoder will allow to quickly port it to any given application, which will reduce the implementation time. In addition, a Hardware-in-the-Loop (HiL) platform will be proposed to perform a fast parametrical analysis on the decoder architecture. Unlike other proposals found in the literature, this parametrical analysis will be performed over the entire transceiver architecture in which the decoder is embedded, so that all the effects of the transceiver implementation are taking into account during the optimization stage.

3.2 FIGURES OF MERIT

When a convolutional code is chosen for the transmitter of a communication system, its definition gives a raw indicator of the complexity of the Viterbi decoder that will be instantiated at the receiver. When the Viterbi decoder is being implemented, the designer will obtain the hardware architecture that better suits its optimization goals (area resource utilization, achievable maximum clock speed, data decoding throughput, power consumption, ...).

In order to compare different Viterbi decoder architectures, we need to compare at an architectural level and in terms of the performance achieved by the transceiver in which they will be implemented. In this section, two different figures of merit will be presented. These metrics will be latter used to compare different decoder implementations found in the literature with the one developed in this research work.

3.2.1 PER

Two figures of merit are common when comparing the performance of a communication system: Bit Error Rate (BER) and Packet Error Rate (PER). On communication system working on continuous data streams, BER is the main figure of merit. It calculates, on average, the number of

erroneous received bits and is given by

$$BER = \frac{Bits\ in\ error}{Total\ received\ bits} \tag{3.1}$$

Communication systems that operate with data packets like Wireless Local Area Network (WLAN) 802.11a have mechanism such as Cyclic Redundant Checks (CRCs) that identify packets that have been erroneously received. If a single bit on such communication systems is received in error, the entire packet is discarded and, generally, a data retransmission is requested which, in turn, reduces the achievable maximum data throughput of the system. Therefore, for the WLAN 802.11a standard used during this research work, PER will be chosen as the main figure of merit of the communication link.

The 802.11 standards define a maximum tolerable PER for the receiver and for 802.11a [IEE99] [Roh04] [OP99] this rate is set to 10%. Consequently, during this research work, the only valid implementations will be those that achieve a PER of 0.1 or more.

3.2.2 METRIC Φ

Comparing different Viterbi decoders is not a trivial task. Depending on the convolutional code being implemented, the obtained decoder architecture can be more or less complex. In addition to that, the optimization goal pursuit (area, speed, throughput, ...) leads to different decoder architectures, and even the technology in which the decoder is being implemented (Application Specific Integrated Circuit (ASIC), part name of the Field Programmable Gate Array (FPGA), ...) produces different results.

In order to give a common reference for the following discussion, in this research work the metric Φ is proposed. This metric is oriented to FPGA implementations and provides an estimation of the data throughput that can be achieved for a given system hardware complexity. It is given by

$$\Phi = \frac{Data\ Throughput\ [Mbps]}{LUTs + Registers} \tag{3.2}$$

The data throughput is the amount of data that a single decoder instance can decode per unit of time. As it can be seen, the complexity of the

system is measured as the number of Look-up Tables (LUTs) and registers necessary in the FPGA. For traceback based decoder implementations the number of occupied block Random Access Memories (RAMs) has been left out of the equation. This decision has been taken because logic elements (LUTs and registers) are more limited than RAM resources even in modern FPGAs. Besides, design parts that are implemented solely on block RAMs can operate at much higher frequencies than those parts that are implemented using logic resources.

Let us take for example the highest part of the Xilinx Series-7 FPGAs, the Virtex xc7v200t [Xil15]. This FPGA contains 305400 slices. Since each slice on the Series-7 products consists of eight registers and four six-input LUTs, that makes for a total of 2443200 registers and 1221600 LUTs. This FPGA also contains 1292 36kb block RAMs, which makes a total of 47628288 RAM bits. Therefore, there is nearly a 19.5 to one ratio between RAM bits and registers, and a 39.5 ratio between total RAM bits and available LUTs. Besides, block RAMs in these FPGAs can operate with frequencies as high as 600 MHz, while achieving designs that reach these frequencies is very difficult [SV13]. For this reason, equation (3.2) does not take into account the RAM resources taken by the Viterbi implementation.

3.3 OBJECTIVES

In this section we will define several partial objectives to obtain and analyze the description of a flexible Viterbi decoder architecture.

- A highly parameterizable Viterbi decoder architecture will be designed in VSIC (Very High Speed Integrated Circuits) Hardware Description Language (VHDL). This decoder architecture will be suited for FPGA and ASIC implementations, although in this research work we will only focus on the FPGA implementation. The decoder will provide the following characteristics:
  - Support for hard and soft decoding.
  - Implementation of Carrier Strength Indicator (CSI) aware Branch Metric Units (BMUs).
  - Support for register exchange and traceback implementations of the Survivor Path Unit (SPU).

  - Support for convolutional codes of various coding rates $R = k/n$ and transfer function matrixes $G(x)$.

- The Viterbi decoder will be integrated into the receiver chain of a WLAN 802.11a compliant transceiver developed in this research work to analyze its performance in terms of PER. The digital transceiver is functional up to the Medium Access Control (MAC) layer of the IEEE 802.11 standard, and will include complex functional blocks such as time and frequency synchronization and a channel equalization with phase offset tracking. In this analysis, the impact integrating the cost-effective Viterbi decoder has on the overall transceiver architecture will be discussed.

- A fast verification platform based in HiL simulations will be implemented to quickly obtain performance curves of the overall WLAN 802.11a transceiver under different noise and channel conditions. This platform will be further used in a parametrical analysis of the Viterbi decoder to find the best trade-off between area resource utilization and decoding capacity of the decoder and transceiver.

3.4 SCOPE OF THIS WORK

This research work focuses on obtaining a flexible and cost-effective Viterbi decoder architecture that fits in the decoding chain of any given communication system. The following assumptions have been taken:

- Portability of the source files.

  The source code of the Viterbi decoder has to be portable to platforms other than those used in this research work. Therefore, to facilitate its portability, no vendor specific Intellectual Properties (IPs) will be used.

- Final implementation.

  In this work a functional description of a WLAN 802.11a compliant transceiver will be obtained and analysed. The WLAN 802.11a transceiver is the result of the combined effort of the research group in which this research work has been developed. For this dissertation all the functional blocks of the digital transceiver will be designed

except the direct and inverse Fast Fourier Transform (FFT) module. No external IPs or code generation tools will be used for the transceiver implementation. The digital transceiver is functional up to the MAC layer of the IEEE 802.11 standard, and the system analysis and measurement includes synchronization, channel equalization and phase offset tracking.

- HiL verification platform.

  The interface between the physical and virtual elements is one of the most critical and complex elements of the HiL platform. *System Generator*, a tool provided by *Xilinx* to quickly integrate digital processing algorithms into a FPGA will serve as the base reference of the fast verification platform since it provides a seamless integration with *Matlab/Simulink*, allows the inclusion of custom Register Transfer Level (RTL) code into the design and permits the physical component of the HiL simulator to operate asynchronously with respect to the virtual component in a free-clock scenario.

CHAPTER 4

Viterbi decoder architecture

Contents

4.1 INTRODUCTION

This chapter presents the hardware description of the Viterbi decoder developed in this research work. A highly parameterizable decoder is desired, so that the decoder can easily fit into a complete transceiver architecture.

The chapter is structured as follows. First, a quick overview of the terminology used in the chapter is introduced. Then, the implemented building blocks of the Viterbi decoder are described. Finally, the concluding remarks of the chapter are summarized.

4.2 TOP LEVEL VITERBI DECODER ENTITY

Figure 4.1 depicts the top level view of the Viterbi decoder implemented in this research work. The description of the Viterbi decoder allows decoding any possible *feedforward* convolutional code. A *feedforward* code is defined by the triplet (n, k, ν) and its transfer function $G(x)$. The parameters that control the implementation of the Viterbi decoder are listed in table 4.1.

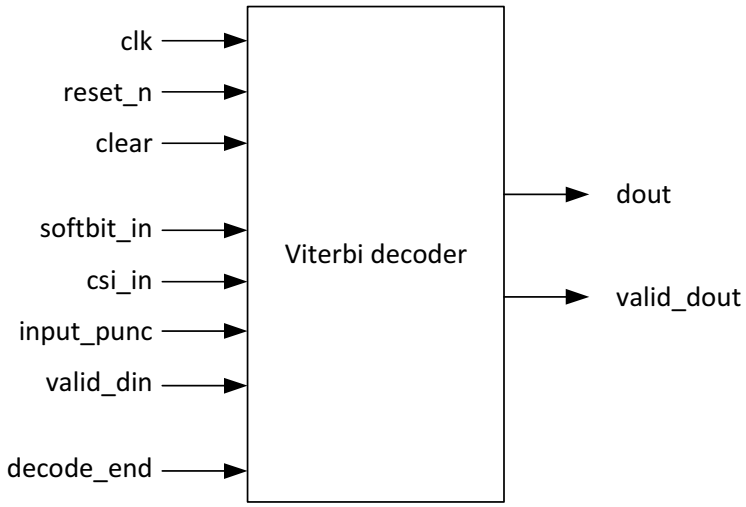


Figure 4.1: *Top level Viterbi decoder*

Table 4.2 lists and describes the input ports of the decoder description.

csi_in is an optional input port that contains the CSI values associated with the received vector **r**. For the Viterbi decoder developed in this research work the CSI value is normalized. The carrier strength of a subcarrier corresponds to the magnitude of the channel estimation for that carrier divided by the maximum magnitude of the channel estimation. Since the CSI values are encoded using csi_{bw} bits, this means that their fixed point representation belongs to the range $[0 \quad 2^{csi_{bw}-1}]$.

Table 4.3 lists and describes the output ports of the decoder description.

Name	Type	Description
$softbit_{bw}$	Integer	Bitwidth of the elements of the input array softbit_in. If $softbit_{bw} = 1$ hard decoding is implemented. If $softbit_{bw} > 1$, soft decoding is implemented.
csi_{bw}	Integer	Bitwidth of the elements of the input array csi_in. If $csi_{bw} \leq 1$, Carrier Strength Indicator (CSI) aware decoding is internally disabled.
$code_n$	Integer	Parameter n of the convolutional code.
$code_k$	Integer	Parameter k of the convolutional code.
$const\_len$	Integer	Constraint length ν of the convolutional code.
$code_{pol}$	Integer vector	Transfer function $G(x)$ of the convolutional code in octal form as described in (2.3).
acs_{xtr_bw}	Integer	Number of bits the Add-Compare-Select (ACS) accumulator registers are extended to avoid overflow.
$traceback\_depth$	Integer	Traceback depth τ of the convolutional code.
$compare\_tree_{pipe}$	Integer	Pipeline configuration of the comparator tree that identifies the survivor path with minimum accumulated metric.

Table 4.1: *Generics of the top level Viterbi decoder*

Name	Size (bits)	Description
clk	1	System clock.
reset_n	1	Asynchronous reset, active low.
clear	1	Synchronous clear, active high.
softbit_in	$[code_n \times softbit_{bw}]$	Array of binary words containing the elements of the received message **r**.
csi_in	$[code_n \times softbit_{bw}]$	Array of binary words containing the CSI values associated to the received message **r**.
input_punc	$code_n$	Vector that indicates which elements of the received message **r** contain dummy bits.
valid_din	1	On high validates the values of input ports softbit_in, csi_in and input_punc.
decode_end	1	Used on packet/burst communication schemes. On high indicates that a complete data packet has been received and that the decoder must identify and decode the data associated with the survivor path with minimum accumulated metric.

Table 4.2: *Input ports of the Viterbi decoder*

Name	Size (bits)	Description
dout	$code_k$	Binary representation of decoded message **x**.
valid_dout	1	On high validates the value of port dout.

Table 4.3: *Output ports of the Viterbi decoder*

Figure 4.2 depicts a basic view of the elements that form the Viterbi decoder architecture and their interconnection.

As can be seen, the Viterbi decoder consists of three main components: the Branch Metric Unit (BMU), the ACS cluster and the Survivor Path Unit (SPU) core. The BMU calculates the branch metrics between the received message **r** and all possible 2^{code_n} codewords. The data output of the BMU, branch_out, is therefore a vector with 2^{code_n} elements that contains those branch metrics. branch_out is the input to the next main component: the ACS cluster. The ACS cluster contains 2^{const_len} instances of ACS units. There is one ACS instance per state in the trellis in the ACS cluster, and each ACS is responsible for selecting the survivor path ending at its corresponding state. The ACS cluster generates two output vectors of 2^{const_len} elements: path_metric_out and prev_state. The former contains the path metrics of all survivor paths at that decoding iteration, and the latter is a set of flags that identify the survivor path selections. Both vectors are used by the last building block of the decoder, the SPU core. The BMU core is responsible for storing and updating the historic of survivor paths and calculating the decoded output data.

The complexity and interconnection of each component is very dependent on the convolutional code being implemented. Therefore, the example four state convolutional code depicted earlier in figure 2.2 will be used during the exposition in this chapter to better understand the architecture of each component of the decoder.

The next section will describe the architecture of each component in detail.

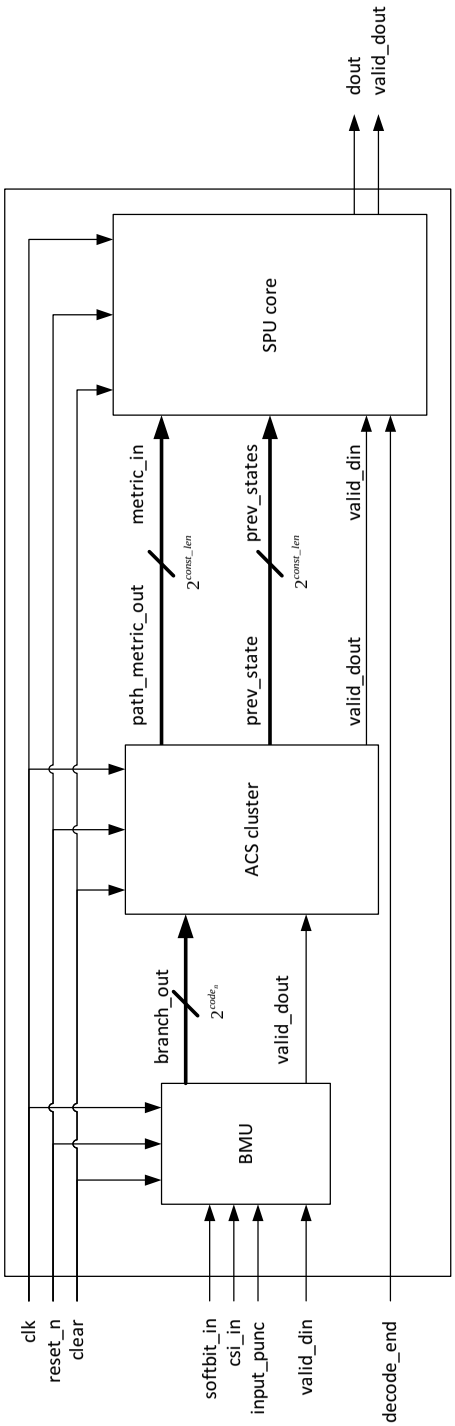


Figure 4.2: *Block diagram of the Viterbi decoder architecture*

4.3 VITERBI DECODER COMPONENTS

The global architecture of a Viterbi decoder is divided in three main functional blocks: the BMU, responsible for calculating the branch metrics at each iteration of the algorithm, the ACS units, which select the survivor paths among all the possible path ramifications in the trellis of the code, and the SPU, where the historic of survivor paths is stored and the decoded message is calculated. The following subsections describe the implementation of each building block in detail.

4.3.1 BRANCH METRIC UNIT (BMU)

The BMU of the Viterbi decoder calculates all the branch metrics due to the state transition given by the received message $\mathbf{r}$ at time instant t. A convolutional code with a coding rate $R = k/n$ can generate a total of 2^k different codewords $\mathbf{c}$ of length n bits. At the receiver, the Viterbi decoder compares the received sequence $\mathbf{r}$ with all possible 2^n codewords $\mathbf{c}$ and calculates the associated branch metric.

Table 4.4 lists the generics of the BMU unit. These generics are the same as the top level generics of table 4.1. Table 4.5 enumerates the input ports to the BMU. As before, these input ports are directly connected to the top level input ports of table 4.2. Finally, table 4.6 describes the output ports of the BMU.

The architecture of the BMU developed in this research work is shown in figure 4.3. The figure does not show the clk, reset_n and clear ports of the entity, as these are common signals that are routed to every register in the BMU.

As can be seen in the figure, the BMU consists of a total of n units that calculate the distance between each of the elements of softbit_in and the most confident representation of a binary one and zero. The distance between each element of softbit_in and the most confident representation of a bit is given by the quantity dX_i, where X is an integer in the range $[0 \quad code_n - 1]$ and i represents the logical value of the bit to which the distance is being calculated.

input_punc is an input vector used to implement depuncturing in the BMU. Each element of input_punc is associated to one of the n distance

Name	Type	Description
$softbit_{bw}$	Integer	Bitwidth of the elements of the input array softbit_in. If $softbit_{bw} = 1$ hard decoding is implemented. If $softbit_{bw} > 1$, soft decoding is implemented.
csi_{bw}	Integer	Bitwidth of the elements of the input array csi_in. If $csi_{bw} \leq 1$, CSI aware decoding is internally disabled.
$code_n$	Integer	Parameter n of the convolutional code.
$code_k$	Integer	Parameter k of the convolutional code.

Table 4.4: *Generics of the BMU*

Name	Size (bits)	Description
clk	1	System clock.
reset_n	1	Asynchronous reset, active low.
clear	1	Synchronous clear, active high.
softbit_in	$[code_n \times softbit_{bw}]$	Array of binary words containing the elements of the received message **r**.
csi_in	$[code_n \times softbit_{bw}]$	Array of binary words containing the CSI values associated to the received message **r**.
input_punc	$code_n$	Vector that indicates which elements of the received message **r** contain dummy bits.
valid_din	1	On high validates the values of input ports softbit_in, csi_in and input_punc.

Table 4.5: *Input ports of the BMU*

Name	Size (bits)	Description
branch_out	$[2^{code_k} \times (softbit_{bw}+$ $+log_2(code_n)+$ $+(csi_{bw}-1))]$	Array containing the branch metrics to the 2^{code_k} codewords of the convolutional code.
valid_dout	1	On high validates the value of port branch_out.

Table 4.6: *Output ports of the BMU*

codeword	partial metrics
00	$d0_0$ $d0_1$
01	$d0_0$ $d1_1$
10	$d1_0$ $d0_1$
11	$d1_0$ $d1_1$

Table 4.7: *Calculation of branch metrics in the BMU*

calculator units of the BMU. When the input_punc element of a distance calculator unit is set high, the input softbit to that distance unit is considered a dummy bit. When this condition is met the distance quantities dX_i generated by the distance calculator unit are all set to zero. This way punctured bits are discarded from the branch metric calculation.

The distance dX_i between each element of softbit_in and the most confident representation of a bit is given by $softbit_{bw}$ bits. If $softbit_{bw} = 1$, the BMU implements hard-decoding. Otherwise soft-decoding is implemented.

If $csi_{bw} > 1$, then the BMU implements a CSI-aware design. In this case, a set of $2n$ multipliers are instantiated. These multipliers weight the previously calculated dX_i distances with the CSI value associated to element X of the input vector csi_in. Since the values of the elements of csi_in fall in the range $[0 \quad 2^{csi_{bw}-1}]$, then, the weighted distances of dX_i are expressed with $softbit_{bw} + (csi_{bw} - 1)$ bits

The branch metric associated to a codeword can then be calculated by adding the appropriate values of the weighted dX_i together. For example, for a convolutional code with $n = 2$, table 4.7 summarizes how the different dX_i values are combined together to generate the branch metrics to all possible 2^n codewords.

To generate all branch metrics, the BMU instantiates a total of 2^{code_n} adders of $code_n$ inputs. The generated branch metric for a given codeword is then a binary word of $softbit_{bw} + log_2(code_n) + (csi_{bw} - 1)$ bits.

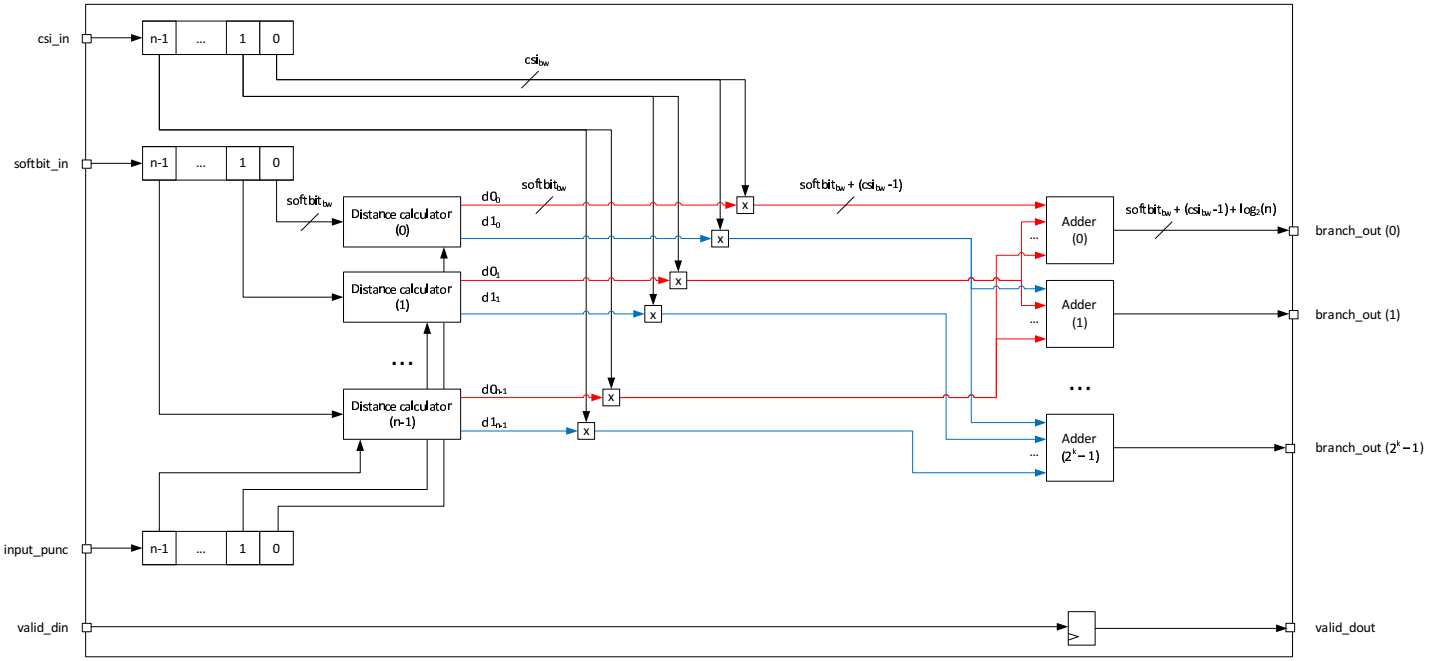


Figure 4.3: *Block diagram of the CSI aware BMU*

4.3.2 ADD-COMPARE-SELECT UNIT (ACSU)

During each iteration of the Viterbi algorithm the ACS units estimate the survivor paths ending at each of the convolutional code's states. Following the Viterbi algorithm, the operation of the ACS is divided in the following steps:

1. Extend the paths by adding their path metrics to the branch metrics due to the state transition.
2. Compare the accumulated metrics of all possible 2^{code_k} candidate paths
3. Select the path with best (lowest) accumulated metric.

Table 4.8 summarizes the generics of the ACS unit. Value $data_{bw}$ is the bitwidth of the accumulator of the ACS. It is given by $softbit_{bw} + (csi_{bw} - 1) + log_2(code_n) + acs_{xtr\_bw-1}$, and corresponds to the bitwidth of the branch metrics calculated by the BMU extended with acs_{xtr_bw} bits. This last element is a generic of the top level Viterbi decoder and was first introduced in table 4.1 in section 4.2.

The ACS developed in this research work implements a low complexity overflow avoiding mechanism on the accumulator registers of the ACSs. Value $overflow_{val}$ in table 4.8 is the minimum accumulated metric in the ACS unit that triggers this mechanism.

Table 4.9 lists the input ports of the ACS unit. Input port branch_metric is an array of 2^{code_k} vectors containing the branch metrics that converge in the ACS unit at a given time instant. The bitwidth $branch_{bw}$ of each element of the array is $softbit_{bw} + log_2(code_n) + (csi_{bw} - 1)$ bits.

As was mentioned during the explanation of the Viterbi algorithm in section 2.2.2, not all the states in the trellis are active during the decode process. At the beginning of the decode process only state 0 of the convolutional code is active, and the remaining states become active as the decoding process advances in time. Input port active_states_in indicates the ACS unit if its precursor states are active or not. It is necessary to prevent an inactive state with uninitialized metric to become part of the survivor path. The ACS unit ignores all metrics coming from inactive states. An ACS becomes active when any of its precursor states is active.

Name	Type	Description
$code_k$	Integer	Parameter k of the convolutional code.
$data_{bw}$	Integer	Bitwidth of the accumulator of the ACS.
$overflow_{val}$	Integer	Quantity that enables the overflow avoiding mechanism of the ACS.

Table 4.8: *Generics of the ACS unit*

Name	Size	Description
clk	1	System clock.
reset_n	1	Asynchronous reset, active low.
clear	1	Synchronous clear, active high.
path_metric_in	$[2^{code_k} \times data_{bw}]$	Array containing the path metrics of the survivor paths of the precursor states.
branch_metric	$[2^{code_k} \times branch_{bw}]$	Array containing the branch metrics for the current received message $\mathbf{r}$.
active_states_in	2^{code_k}	On high indicates that the precursor ACS unit is active in the trellis diagram.
overflow_flag_in	1	On high enables the overflow avoiding mechanism of the ACS.
valid_metric	1	On high validates the data input ports.

Table 4.9: *Input ports of the ACS unit*

Name	Size	Description
path_metric_out	$data_{bw}$	Accumulated metric of the survivor path.
active_state_out	1	On high indicates that the ACS unit is active in the trellis diagram.
prev_state	$code_k$	A pointer indicating the precursor state of the survivor path.
overflow_flag_out	1	Overflow flag of the ACS. On high indicates that the accumulated metric of the survivor path exceeds the generic value $overflow_{val}$.
valid_dout	1	On high validates the value of the output ports.

Table 4.10: *Output ports of the ACS unit*

overflow_flag_in is an indicator that the metrics of all ACS units exceed the quantity $overflow_{val}$. When this signal is asserted high, the ACS unit executes the overflow avoiding mechanism.

Table 4.10 lists the output ports of the ACS unit. active_state_out is an active high flag that indicates if the ACS is active on the decoding trellis or not. At the beginning of the decode process, only the ACS unit associated to state 0 of the convolutional code is active.

prev_state is an indicator that points to the precursor state that originates the survivor path and is needed by the SPU logic.

overflow_flag_out is an output flag that on high indicates that the accumulated metric of the survivor path of the ACS unit exceeds the quantity $overflow_{val}$.

Figure 4.4 depicts the basic block diagram of the ACS unit core.

The difference between the accumulated metric of survivor and non-survivor paths is more pronounced on transmission channels with low levels of noise than on transmission channels with higher levels of noise. Similarly, as decoding progresses, the accumulated metric of the survivor paths of all

ACS units grows slower on channels with low levels of noise than on channels with high levels of noise.

As the noise level of the transmission channel grows, the uncertainty on the survivor path selection increases because the difference between all candidates paths converging to an ACS unit at a certain instant of time becomes narrower. Consequently, the accumulated metrics of all ACS units grows faster.

As the accumulated metrics of the ACS units increases, noisy transmission channels lead to uncertain (high) branch metric values which can produce overflow errors during the calculation of the metrics of all candidate paths. When the metric of a candidate path overflows, its accumulated metric is reduced compared to the metric of all the remaining candidate paths, and therefore, with high probability, it will be erroneously chosen by the ACS as the survivor path.

To prevent this situation the branch metrics generated by the BMU unit are extended with acs_{xtr_bw} bits. The value of acs_{xtr_bw} is chosen to meet the noisy characteristics of the transmission channel.

An overflow compensation mechanism is still necessary to prevent the accumulated metrics of the ACS units from continuously growing during signal decoding. It mechanism operates by subtracting a constant amount $overflow_{val}$ every time the accumulators of all ACS units exceed the same quantity. For the decoder, the value $overflow_{val}$ is selected as the maximum value that can be generated on the BMU: $softbit_{wl} + log_2(code_n) + csi_{bw} - 1$.

The overflow avoiding mechanism is simplified by the fact that the value of $overflow_{val}$ is an integer value that is expressed as a power of two. When the state overflow flag of the ACS is being calculated, only the $data_{bw} - log_2(overflow_{val})$ Most Significant Bits (MSBs) of the accumulator have to be considered. Similarly, when the quantity $overflow_{val}$ is subtracted from the metrics of the survivor paths, only the $data_{bw} - log_2(overflow_{val})$ MSBs of the accumulator are considered.

The ACS unit only takes into account the metrics generated from active states in the trellis. Candidate metrics originated from inactive states are automatically discarded by the compare and select modules of the ACS.

At the beginning of the decode process (or after the Viterbi decoder has been cleared), only the ACS unit associated to state 0 is active. The remaining ACSs become active as they identify survivor paths originated from active ACSs.

The ACS unit concludes its operation in a single clock cycle. Therefore, the output validation port valid_dout is a registered version of the input validation port valid_din.

To maximize the decoding rate achievable by the architecture, the Viterbi decoder instantiates a total of 2^{ν} ACS units, one per state in the convolutional code. All ACS units are then connected according to the trellis diagram of the convolutional code. After this interconnection has occurred, the resulting set of ACS units is referred as a ACS cluster. For example, for the convolutional code introduced in section 2.2.2, the resulting ACS cluster is shown in figure 4.5. To simplify its layout , clock, reset and clear ports have been removed from the diagram in the figure.

The ACS cluster input is directly connected to the output of the BMU.

The overflow_flag_in input port for all ACS units is calculated as the AND function of all the overflow_flag_out output ports of all 2^{ν} ACS units. Figure 4.5 depicts how the different branch metrics are loaded into the various ACS units, how the overflow flag signal is generated in the decoder and how the metrics and active flags are propagated in the cluster according to the Trellis diagram of the code.

Note that only the output validation port of the ACS unit associated to state 0 is connected to the output validation port of the ACS cluster in figure 4.5. Since this is the only state that is active during the entire decode process, its output serves as the validation of the cluster output.

The output accumulated metrics and the previous state indicators of all ACS units in the ACS cluster are rearranged in data array structures that are the input to the final component of the Viterbi decoder architecture, the SPU.

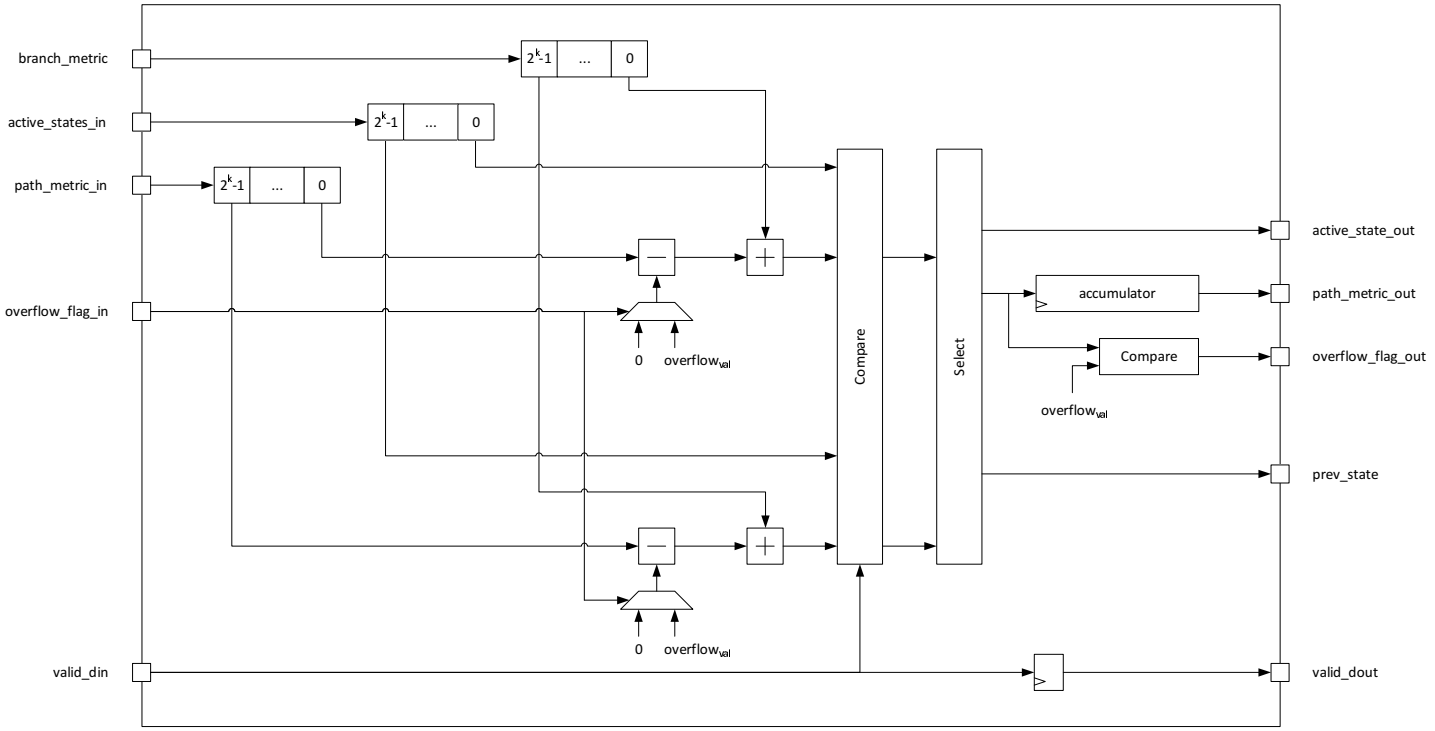


Figure 4.4: *Block diagram of the Add-Compare-Select unit*

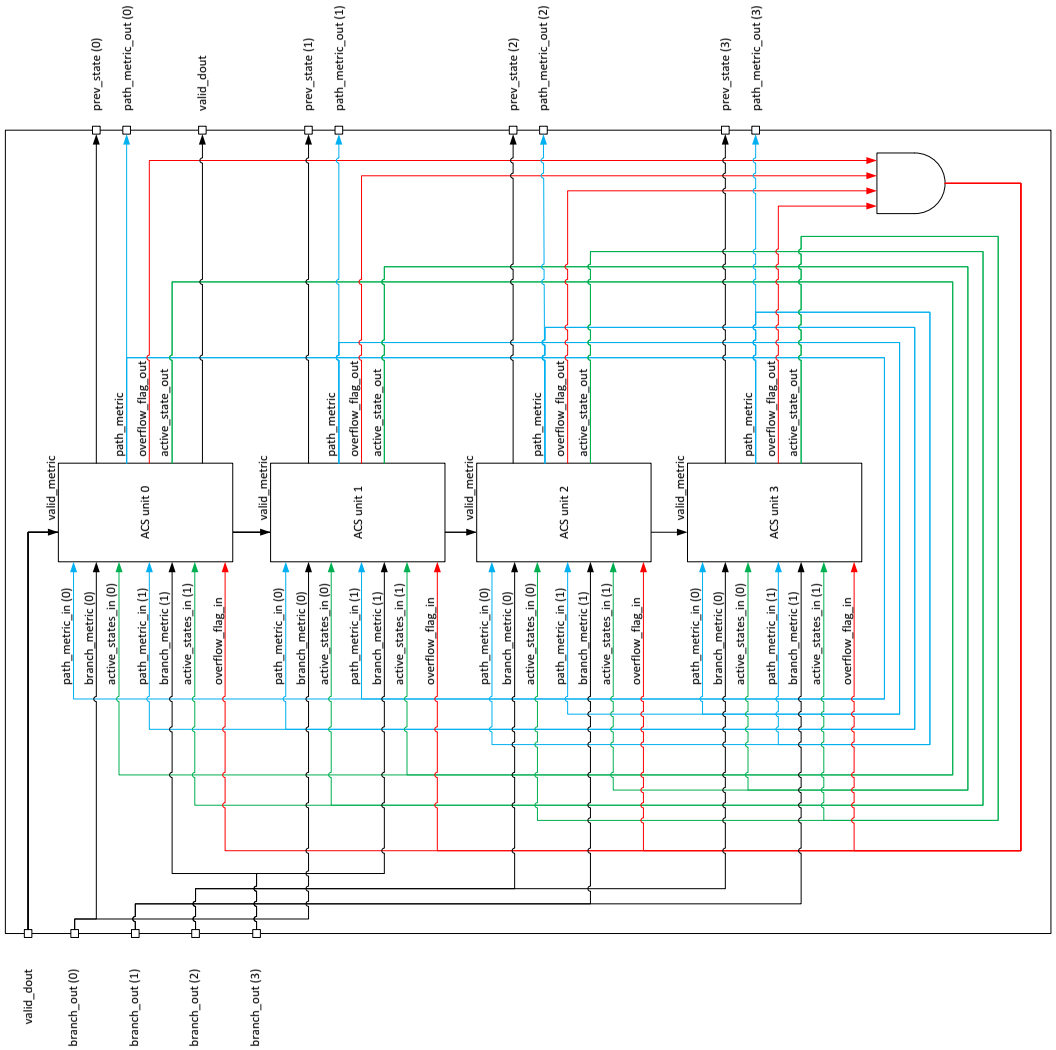


Figure 4.5: *ACS cluster. Interconnection of the different ACS units for the trellis diagram of figure 2.4*

4.3.3 SURVIVOR PATH UNIT (SPU)

The SPU is the unit responsible for storing the historic of survivor paths and decoding the output message $\mathbf{x}$ as new data is available by the ACS units. The SPU designed in this research work is suitable for burst communication schemes, where the length of the data to be decoded is constrained or known by the receiver, and for continuous flow communication schemes.

The top level view of the SPU is depicted in figure 4.6. Once again, clk, reset_n and clear ports have not been shown, as they are common signals for all registers in the unit. As can be seen, the SPU is divided into two main blocks: the SPU core itself, where the register exchange or the traceback algorithms are implemented, and the Minimum Path Unit (MPU), a block that assists the SPU operation by identifying the state with minimum accumulated path.

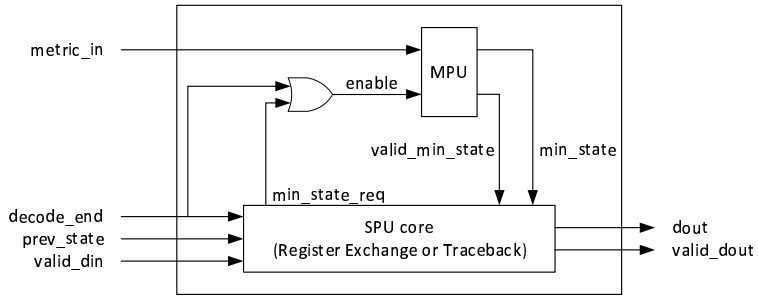


Figure 4.6: *SPU top level view*

Table 4.11 lists the generics of the SPU. The generics of the SPU are initialized with the generic values of the top entity of table 4.1 with the exception of data_bw, that takes the bitwidth of the accumulators of the ACS units: $softbit_{bw} + log_2(code_n) + (csi_{bw} - 1) + acs_{xtr\_bw}$.

Table 4.12 enumerates and describes the input ports of the SPU unit. metric_in, prev_state and valid_din are generated by the ACS cluster. The remaining input ports of the SPU are connected to the counterpart top level input ports of the Viterbi decoder in table 4.2.

Table 4.13 enumerates and describes the output ports of the SPU unit. These output ports correspond with the output ports of the Viterbi decoder

Name	Type	Description
$code_k$	Integer	Parameter k of the convolutional code.
$constr\_len$	Integer	Constraint length ν of the convolutional code.
$data_{bw}$	Integer	Bitwidth of the accumulators of the ACSs.
$traceback\_depth$	Integer	Traceback depth τ of the convolutional code.
$compare\_tree_{pipe}$	Integer	Pipeline configuration of the comparator tree that identifies the survivor path with minimum accumulated metric.

Table 4.11: *Generics of the SPU*

Name	Size (bits)	Description
clk	1	System clock.
reset_n	1	Asynchronous reset, active low.
clear	1	Synchronous clear, active high.
metric_in	$[2^{constr\_len} \times data_{bw}]$	Array of survivor path metrics given by the ACS units.
prev_state	$[2^{constr\_len} \times code_k]$	Array of pointers of the survivor path precursors given by the ACS units.
valid_din	1	On high validates the values of input ports metric_in, and prev_state.
decode_end	1	Used on packet/burst communication schemes. On high indicates that a complete data packet has been received and that the SPU must identify and decode the data associated with the survivor path with minimum accumulated metric.

Table 4.12: *Input ports of the SPU*

Name	Size (bits)	Description
dout	$code_k$	Binary representation of decoded message **x**.
valid_dout	1	On high validates the value of port dout.

Table 4.13: *Output ports of the SPU*

and, consequently, are directly connected to the top level output ports of table 4.3.

As has been mentioned, SPUs implementing Register Exchange and Traceback implementations have been written. Due to the innate nature of both implementations, all SPUs are governed by the Finite State Machine (FSM) depicted in figue 4.7.

Decoding starts at state *Idle*, and the SPU remains in this state until the block of ACSs generates the first set of survivor paths or after the input port clear is asserted high.

After the first decode iteration, the SPU transitions to state *Initial load*, and will remain there until the set of decoded survivor paths have a length equal to the traceback depth τ of the decoder. After τ decode iterations have occurred the SPU transitions to state *Normal decode*. In this state, every decoding cycle the SPU stores the survivor paths generated by the ACSs and estimates the decoded message **x** that occurred τ decoding cycles earlier.

On continuous flow communication data schemes the decoder remains in the *Normal decode* permanently. The continuous data input to the Viterbi decoder guarantees that input vectors **r** will be decoded after a quantity of decoding instant proportional to the traceback depth τ of the decoder have elapsed. If the decoder were to operate on data packets of limited size, the data flow entering the decoder would stop at a certain moment, and the SPU would have in memory the last τ state transitions of the survivor paths. Since no new data is loaded into the SPU, these last transitions would not be decoded, and the estimated vector **x** would be τ elements short.

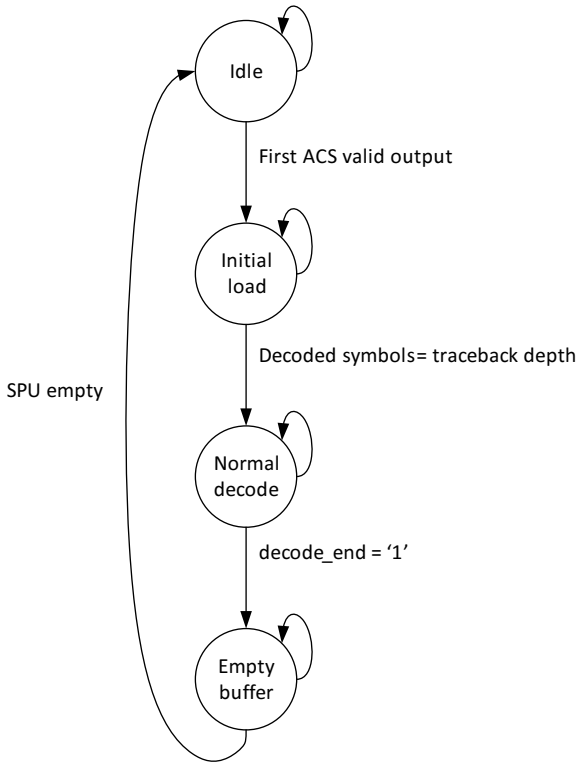


Figure 4.7: *Global FSM of the SPU*

Therefore, a mechanism that decodes the last τ state transitions of the convolutional code after the last bit is loaded into the decoder must be implemented. This mechanism is also necessary in case the decoder needs to decode transmitted messages $\mathbf{m}$ whose binary length is much shorter than its traceback depth τ.

In our implementation, when the input port decode_end of the Viterbi decoder is asserted high, the FSM of the SPU transitions to a new state: *Empty buffer*. Here, the SPU identifies the survivor path with minimum accumulated metric and decodes the last τ state transitions associated to it. To identify the state with minimum accumulated metric a comparator tree of 2^{ν} elements is used. This comparator tree is referred as MPU.

Name	Type	Description
$data_{bw}$	Integer	Bitwidth of the accumulators of the ACS units.
$code_k$	Integer	Parameter k of the convolutional code.
$constr\_len$	Integer	Constraint length ν of the convolutional code.
$compare\_tree_{pipe}$	Integer	Pipeline configuration of the comparator tree of the MPU.

Table 4.14: *Generics of the MPU*

4.3.3.1 Minimum Path Unit

The MPU assists the SPU by identifying the state with minimum accumulated metric at certain decoding instants. An identification process is initialized whenever the input port decode_end of the SPU is asserted high or when the internal signal min_state_req is asserted high by the SPU core as shown in figure 4.6.

Table 4.14 lists the generics used by the MPU. All generics correspond with the top level generics of the Viterbi decoder shown in figure 4.1 with the sole exception of $data_{bw}$, which takes the same value as the homologous ACS unit generic in table 4.8.

Table 4.15 summarizes the input ports of the MPU. In the table, input port enable is the OR function between the decode_end input port of the top level Viterbi decoder of table 4.2 and the min_state_req output port of the SPU core.

Table 4.16 enumerates the output port of the MPU.

For a convolutional code with 2^ν states, the MPU can be seen as a comparator tree with ν levels and $\sum_{k=0}^{\nu} 2^k$ comparators. Figure 4.8 depicts the architecture of a MPU for a convolutional code with $\nu = 2$.

Name	Size (bits)	Description
clk	1	System clock.
reset_n	1	Asynchronous reset, active low.
clear	1	Synchronous clear, active high.
metric_in	$[2^{const\_bw} \times data_{bw}]$	Array of accumulated metrics of the ACS units.
enable	1	On high it request the identification of the state with minimum accumulated metric. This port can be triggered by the input port decode_end of the top level entity or by request of the SPU core.

Table 4.15: *Input ports of the MPU*

Name	Size (bits)	Description
min_state	$constr\_len$	Binary representation of the state with minimum accumulated metric.
valid_min_state	1	On high validates the value of port min_state.

Table 4.16: *Output ports of the SPU*

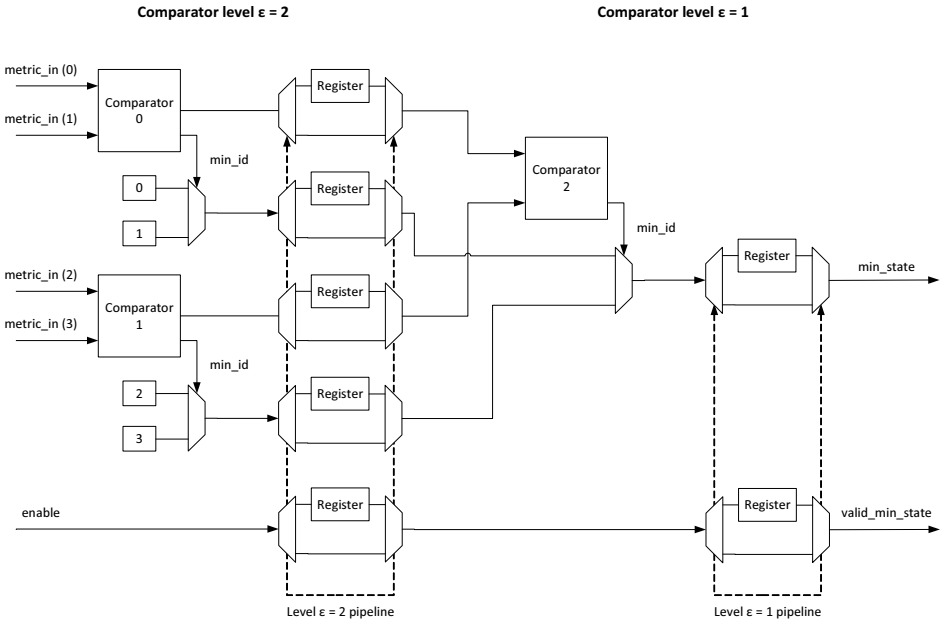


Figure 4.8: *Block diagram of the MPU for constr_len* $= 2$

The MPU is splitted into ν levels. At level ϵ, where ϵ belongs to the range [ν 1], there are a total of $2^{\epsilon-1}$ groups of comparators and multiplexers.

Each group of comparator and multiplexer are loaded with two sets of data. Comparators are fetched with path metrics of survivor paths, and the state identifiers of these survivor path metrics are the input to the multiplexers. The output of each comparator-multiplexer group consists of the metric of the survivor path with minimum accumulated metric and an indicator of its state.

The inputs to the last level comparators of the MPU ($\epsilon = \nu$) are the accumulated metrics of the survivor paths of the 2^{ν} ACS units, and the inputs to the multiplexers at this same level are constants identifying the states whose metrics are loaded into their associated comparators.

The register exchange and traceback implementation of the SPU do not require the knowledge of the metric of the survivor path with minimum

compare_trepipe	Binary value	Level $\epsilon = 2$	Level $\epsilon = 1$	l_{mpu}
0	00	Non-registered	Non-registered	0
1	01	Non-registered	Registered	1
2	10	Registered	Non-registered	1
3	11	Registered	Registered	2

Table 4.17: *Pipeline examples of the MPU for* $\nu = 2$

accumulated metric, and therefore, the output generated by the MPU corresponds with the state identifier given by the multiplexer at the last state of the MPU ($\epsilon = 1$).

After each level, an optional register stage can be placed at the output of all comparators and multiplexers of that level. These registers are used to minimize the impact the MPU has on the maximum frequency achievable by the SPU and the Viterbi decoder. However, adding extra pipeline stages increases the latency of the MPU and the SPU. Let l_{mpu} be the latency of the MPU.

The pipeline stages of the MPU are controlled by the value of the generic $compare\_tree_{pipe}$. If $compare\_tree_{pipe}$ is converted to its binary unsigned representation, then every comparator level at the positions where the unsigned representation of $compare\_tree_{pipe}$ have a logical one are registered. Table 4.17 shows the effects all possible $compare\_tree_{pipe}$ values for a decoder with $\nu = 2$.

The output validation port of the MPU, valid_min_state, corresponds to a delayed version of the logical value of its input port enable with the quantity l_{mpu}.

For a generic MPU implementation, the internal signals are arranged in arrays of length $2^{\nu+1} - 1$ elements inside the Register Transfer Level (RTL) code. For the comparator at location (ϵ, ζ), where ϵ is the comparator level and ζ is the identifier of the comparator within the level, the elements $(2^{constr\_len+1}-1)-(2^{\epsilon+1}-1)+2\zeta$ and $(2^{constr\_len+1}-1)-(2^{\epsilon+1}-1)+2\zeta+1$ of the array behave as inputs to the comparator, while signal $(2^{constr\_len+1} -$

Name	Type	Description
$code_k$	Integer	Parameter k of the convolutional code.
$constr\_len$	Integer	Constraint length ν of the convolutional code.
$traceback\_depth$	Integer	Traceback depth τ of the convolutional code.

Table 4.18: *Generics of the SPU core*

$1) - (2^{\epsilon} - 1) + \zeta$ behaves as the output of the comparator. The output of the MPU is obtained in the element position $(2^{constr\_len+1} - 1) - 1$.

4.3.3.2 SPU core

The SPU core is a top level wrapper that provides a common interface between the SPU unit and the specific implementation of the SPU algorithm (register exchange or traceback).

Table 4.18 lists the generics of the SPU core. As can be seen, the generics of the SPU core are a subset of those defined for the SPU in table 4.11.

The input ports of the SPU core are enumerated in table 4.19. Three sources can drive the input ports of the SPU core. Input ports clk, reset_n, clear and decode_end are driven by the homologous ports of the top level Viterbi decoder input ports of table 4.2. Inputs ports prev_state and valid_din are driven by the ACS cluster, and finally, input ports min_state and valid_min_state are driven by the homologous output ports of the MPU of table 4.16.

Table 4.20 describes the output ports of the SPU core. Output ports dout and valid_out are the decoded output of the Viterbi decoder, and thus, are directly connected to the homologous output ports of the top level Viterbi decoder output ports of table 4.3.

Name	Size (bits)	Description
clk	1	System clock.
reset_n	1	Asynchronous reset, active low.
clear	1	Synchronous clear, active high.
prev_state	$[2^{constr\_len} \times code_k]$	Array of pointers of the survivor path precursors given by the ACS units.
valid_din	1	On high validates the values of input ports metric_in.
decode_end	1	Used on packet/burst communication schemes. On high indicates that a complete data packet has been received and that the SPU must identify and decode the data associated with the survivor path with minimum accumulated metric.
min_state	$constr\_len$	Identifier given by the MPU that points to the survivor path with minimum accumulated metric.
valid_min_state	1	Flag given by the MPU. On high validates the value of port min_state.

Table 4.19: *Input ports of the SPU core*

Name	Size (bits)	Description
min_state_req	1	On high request the MPU to calculate which is the survivor path with minimum accumulated metric.
dout	$code_k$	Binary representation of decoded message $\mathbf{x}$.
valid_dout	1	On high validates the value of port dout.

Table 4.20: *Output ports of the SPU core*

The MPU can be activated in two ways: regularly in a controlled way through the min_state_req output port of the SPU core or externally through the decode_end port of the SPU. Output port min_state_req of the SPU core is asserted high every time the unit has processed traceback_depth new state transitions. On packet based communication systems, when the decoder has received all the coded bits of the packet input port decode_end of the top level unit is asserted high. When this occurs, it forces the MPU to calculate the survivor path with minimum accumulated metric and makes the SPU to validate any remaining data that is may have beginning from this state.

4.3.3.2.1 Register Exchange implementation

The register exchange implementation of the SPU is based on shift register. In this implementation, the data stored in the memory of the SPU is the input to the convolutional encoder that forces the state transitions estimated by the ACS units. Figure 4.9 depicts an example architecture of a register exchanged based SPU implementing the convolutional code of the Trellis diagram in figure 2.4 when $traceback\_depth = 5$.

As seen in figure 4.9, the register exchange implementation of the SPU core consists of 2^{constr_len} register rows of $traceback\_depth$ registers each. The first register in the row is connected to an input Look-up Table (LUT) or Read-Only Memory (ROM) that indicates the input message $\mathbf{m}$ that was loaded into the convolutional coder for the state transition indicated by the input port prev_state.

Before each of the following registers in the row a $(2^{code_k} + 1)$:1 multiplexer is instantiated. The first 2^{code_k} inputs of the multiplexer are used to store the input messages associated to the survivor path indicated by the ACS unit and to discard the historic of the remaining candidate paths. The last input to the multiplexer are used to perform the shift right operation as described in section 2.2.3.3.1.

The output multiplexer of the SPU core selects which shift register will be used as the output of the Viterbi decoder. This selection is governed by the minimum state indicated by the MPU. When the FSM shown in figure 4.7 is at state *Normal decode*, the SPU core selects the shift register associated to state 0 as the output of the decoder. When the FSM transitions to state *Empty Buffer*, first min_state_req is asserted high and the SPU core operation is freezed until the MPU sets its output port valid_min_state high. After the state with minimum state has been identified by the MPU, the SPU core begins extracting the contents of the shift register associated to the state pointed by signal min_state.

Output validation signal valid_dout is the logical value of input port valid_din delayed by a quantity of traceback_depth cycles.

4.3.3.2.2 Traceback implementation

The traceback implementation of the SPU is based on single port Random Access Memories (RAMs). Figure 4.10 depicts the block diagram of the module developed in this research work that implements the traceback algorithm.

The architecture of this module contains two main memory elements: a Last-In First-Out (LIFO) memory implemented in single port RAM to store the historic of state transitions and an output memory buffer implemented in registers that stores the decoded message.

prev_state is an input port containing pointers to the precursors of the survivor paths generated by the ACS cluster. This set of pointers is rearrange as a bit vector and stacked into the LIFO memory. The LIFO memory can store a maximum of traceback_depth vectors. When the LIFO memory has stored a quantity of vectors equal to traceback_depth, the output flag min_state_req is asserted high and the module stops its operation until the MPU identifies the state with minimum accumulated metric.

Once the MPU returns the state with minimum accumulated metric the contents of the LIFO memory is read sequentially. First, the state identifier is loaded in the *Previous State* register of figure 4.10. This register has two functions. First, it contains the data to be stored in the output memory buffer. Secondly, it serves as the pointer of the traceback algorithm. In the first clock cycle following the validation of the MPU output, the contents of the *Previous State* register is loaded into the output memory buffer and the LIFO memory is read backwards.

The vector read from the LIFO memory contains the state transitions of the survivor paths in the previous clock cycle. Using the contents of the *Previous State* register, the state transition identifier of the survivor path is extracted. This identifier, along with the *Previous State* register, is loaded into a ROM that contains precursor state transitions defined in the trellis diagram of figure 2.4. The output of the ROM becomes the state of the survivor path in the previous encode cycle. This new state is loaded into the *Previous State* register and its value is loaded into the output memory buffer.

This process is repeated until the LIFO memory is empty. Once the LIFO memory is empty, the contents of the output memory buffer are read backwards. Every time the output memory buffer is read, port *dout* takes the $code_k$ MSB of the stored state and output port valid_dout is asserted high.

The traceback element can not accept new data samples until the contents of its LIFO and output buffer memories are read, which is a drawback on continuous flow data transmission systems. To overcome this limitation, SPU implementing three traceback modules is suggested. Let us assume decoding a continuous stream of data as shown in figure 4.11.

Each traceback element of the SPU has four associated time windows. The time window depicted in blue in figure 4.11 is where the traceback element stores new precursor data given by the ACS units, and it has a length equal to τ clock cycles. After this time window has elapsed, the MPU unit is activated to identify the ACS with minimum accumulated metric. The time window in green depicts the latency l_{mpu} of the MPU. Ideally, l_{mpu} would be zero, but high speed implementations of the Viterbi decoder require pipelining the MPU logic, and consequently l_{mpu} will take values in the range $[0 \quad constr\_len]$. After the survivor path with minimum accumulated metric has been identified, the contents of the traceback element can

be traced back. This stage is depicted by the purple time window in figure 4.11 and has a length of τ cycles.

Consequently, it is not until time instant $t = 2\tau + l_{mpu}$ when the traceback element begins validating its output. Each traceback element validates its output for a time window of τ clock cycles (depicted in orange in the figure).

When three traceback elements are instantiated in parallel as shown in figure 4.11, the Viterbi decoder implementing a traceback based SPU is capable of decoding continuous flow data transmissions.

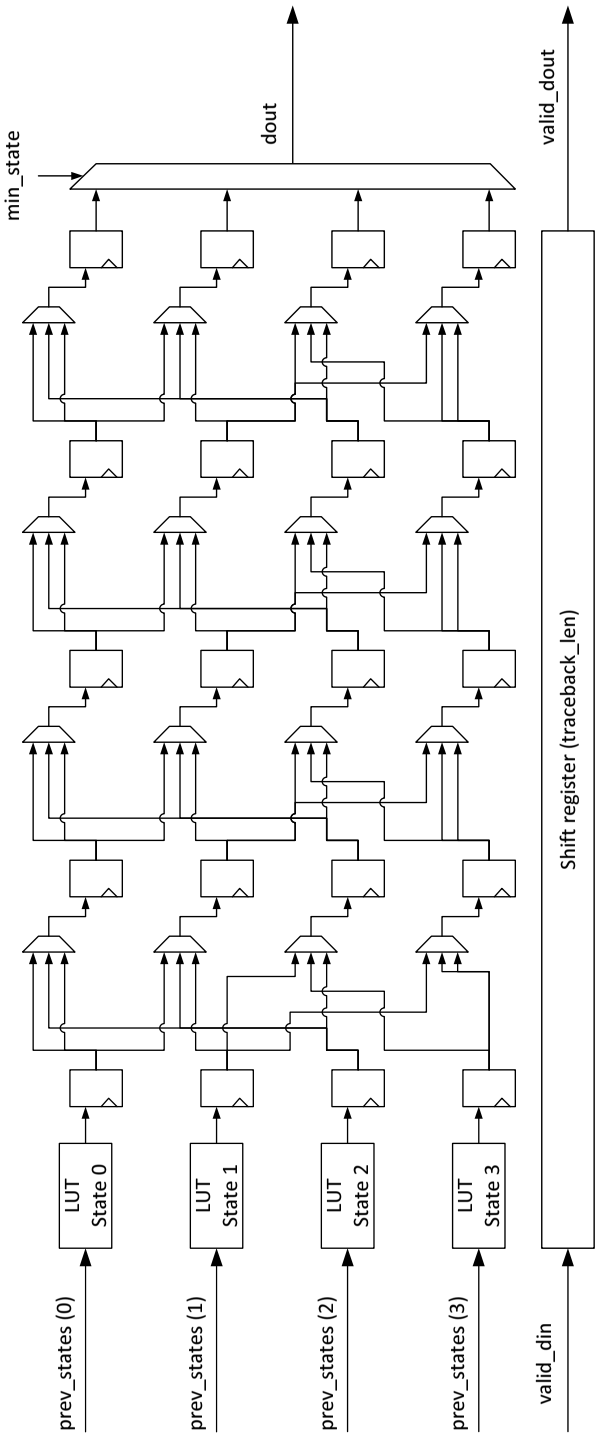


Figure 4.9: *Register exchange SPU for* $constr\_len = 2$ *and* $traceback\_depth = 5$

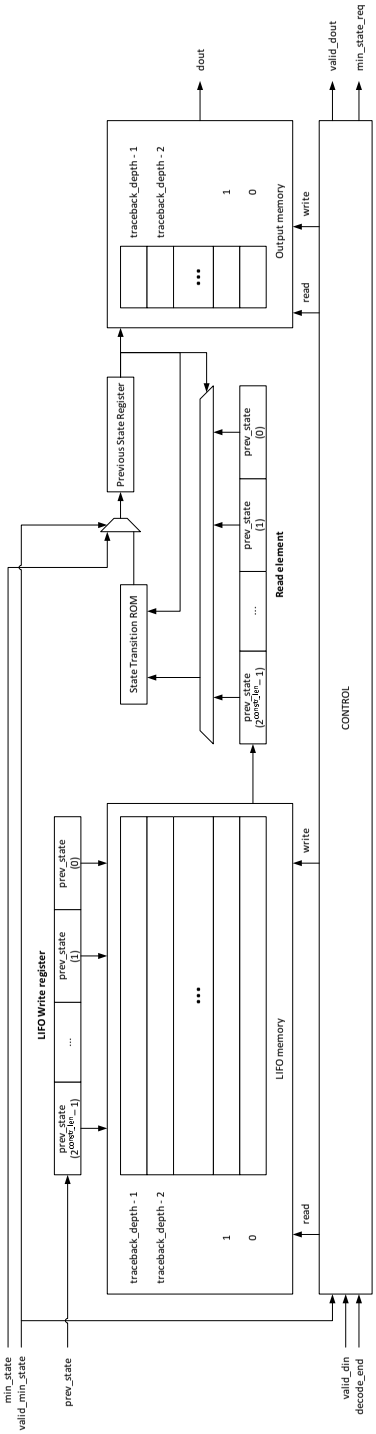


Figure 4.10: *Block diagram of an element implementing the traceback SPU*

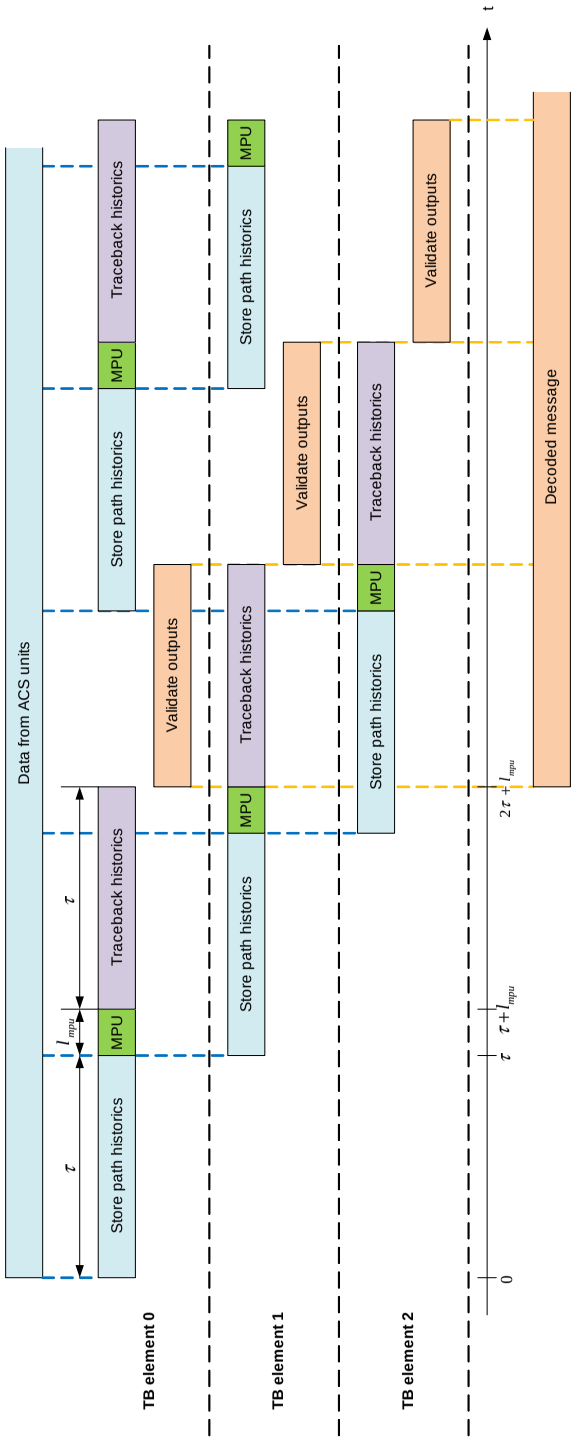


Figure 4.11: *Timing diagram of the traceback implementation of the SPU*

4.4 CONCLUDING REMARKS

In this chapter the hardware description of a flexible Viterbi decoder has been explained. The decoder architecture is highly parameterizable and can be adapted to decode any *feedforward* convolutional code. The core supports both register exchange and traceback based SPU implementations, and the set of generics of the unit gives the designer the opportunity to make trade offs between the complexity and area resource utilization of the core and its achievable system clock speed and decoding capacity.

The design has been described in RTL and contains no vendor specific Intellectual Properties (IPs), so that it can be easily ported to any Field Programmable Gate Array (FPGA) manufacturer or Application Specific Integrated Circuit (ASIC) technology.

The design is currently being sold as a commercial IP. To the author's knowledge, it has already been used in satellite Global Positioning System (GPS) and Global System for Mobile communications (GSM) applications.

CHAPTER 5

Hardware-in-the-Loop simulations

Contents

5.1 INTRODUCTION

In this chapter the Viterbi decoder designed in this research work is integrated into a Wireless Local Area Network (WLAN) 802.11a compliant transceiver. In order to evaluate the performance of the Viterbi and to evaluate the parameters of the Viterbi decoder have on the transceiver architecture, the overall transceiver is implemented in a Hardware-in-the-Loop (HiL) platform.

Opposed to other HiL platforms discussed in section 2.3.3 which focus only in reducing the simulation time necessary to characterize a Device Under Test (DUT), the HiL platform developed in this research work will be used for, first, perform a parametrical analysis of the Viterbi decoder inside the transceiver architecture to optimize its area utilization with respect to the Packet Error Rate (PER) figure of merit, and, second, quickly obtain PER curves of the optimized architecture under multipath frequency selective channels.

The chapter is structured as follows. First, the WLAN 802.11a transceiver is presented. Then, the HiL platform is introduced and the parametrical analysis is explained. Finally, the obtained results are compared to those found in the literature.

5.2 WLAN 802.11A TRANSCEIVER

WLAN 802.11a was one of the first standards to be built upon the now common Orthogonal Frequency Division Multiplexing (OFDM) scheme. OFDM distributes the information to be transmitted between a series of orthogonal carriers so that a single high-speed data stream is divided into multiple slower data-streams that are transmitted in parallel through a channel. The inverse and direct Fast Fourier Transform (FFT) algorithms have proven to be an efficient way to achieve the carrier orthogonality at the transmitter and receiver sides.

Figure 5.1 shows a simplified block diagram of the WLAN architecture. As it can be seen, the functionality of the system is divided into several modules. The Medium Access Control (MAC) layer is the highest abstraction layer of the WLAN, and provides a common interface to all 802.11 standards. Digital signal processing occurs in the Physical Medium De-

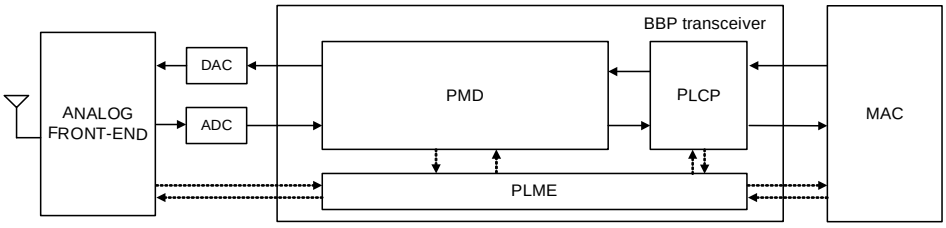


Figure 5.1: *Block view of the transceiver*

pendent (PMD) block, and it provides the means to transmit and receive information through the channel. Between the MAC and PMD layer a translation entity known as Physical Layer Convergence Protocol (PLCP) is instantiated. This element hides the specific implementation delays of the PMD and gives the MAC a common interface. Finally, the Physical Layer Management Entity (PLME) provides the means to the PMD, PLCP and MAC layers to communicate with the analog interfaces of the transceiver and provides, among others, the Received Signal Strength Indicator (RSSI) measurements.

The transceiver architecture developed in this research work is functional up to the MAC level of the standard [IEE99]. The top level entity labelled as Base Band Processor (BBP) in figure 5.1 includes all the building blocks of the PMD, PLCP and PLME developed in this research work.

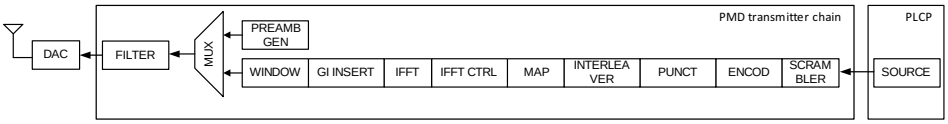


Figure 5.2: *Transmitter chain block diagram of the PMD*

The PMD can be further divided into the transmitter and receiver chains. Figure 5.2 and 5.3 depict, respectively, the simplified block diagram of the transmitter and receiver chains of the PMD.

The transmitter chain of the PMD operates as follows. Processing begins with the scrambler. Data coming from the MAC is randomized by means of a Pseudo Random Number Generator (PRNG) so that DC com-

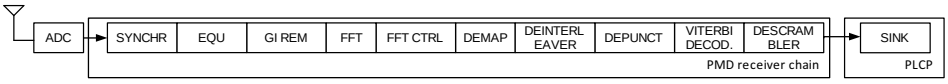


Figure 5.3: *Receiver chain block diagram of the PMD*

ponents are minimized in the modulated signal and the raw unprocessed source bits have better statistical characteristics. The scrambled signal is then encoded using a convolutional code. For WLAN 802.11a, a convolutional code with $k = 1$, $n = 2$, $d = 6$ and $G(x) = [191121]$ is used. Depending on the transmission rate indicated by the MAC, the codewords generated by the convolutional encoder can be punctured. After puncturing WLAN 802.11a supports convolutional codes with $R = 1/2$, $R = 2/3$ and $R = 4/3$. An interleaver shuffles the coded and punctured bits to protect the data against burst noise.

After interleaving has occurred, the binary data is mapped into complex constellation symbols. WLAN 802.11a supports BSPK, QPSK, 16-QAM and 64-QAM modulations. The complex constellation are loaded into a control unit that distributes them between the available data carriers in the transmitter. This unit also adds pilot tones for signal acquisition and null tones to prevent interference between adjacent channels. Data is then modulated applying the Inverse Fast Fourier Transform (IFFT) operation to the complex symbols. After the IFFToperation, a guard interval is appended to each OFDM symbol and the generated signal is filtered by a Root Raised Cosine (RRC) filter, so that the transmitted signal meets the spectral mask defined by the standard [IEE99].

The transmitted signal is prepended with a sequence of short and long training symbols, and the resulting signal is referred as PLCP Protocol Data Unit (PPDU). Each PPDU can carry between 1 and 4095 bytes of raw information, and its training sequence aids the receiver chain in signal detection, frequency and time offset error compensations and channel equalization.

The receiver chain of the PMD operates as follows. A synchronizer module is required to detect the beginning of the received signal. This module also compensates time and frequency offset that originate due to the mismatches in the sampling clocks of the transmitter Digital to Analog Converter (DAC) and receiver Analog to Digital Converter (ADC), and the

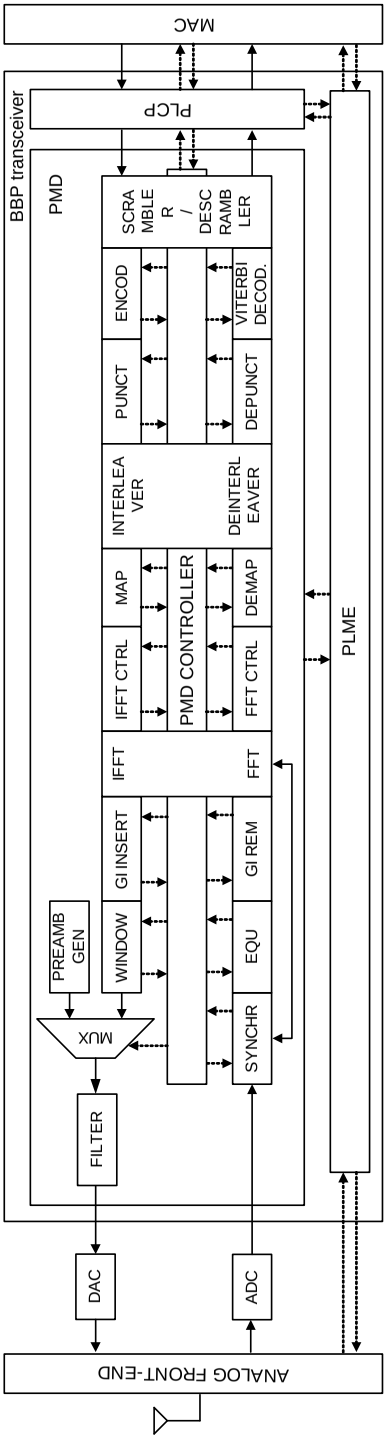


Figure 5.4: *Block diagram of the transceiver architecture*

Automatic Gain Control (AGC) amplifier in the analog front-end of the receiver. The received signal is then equalized. The equalizer developed in this research work is responsible for estimating the Carrier Strength Indicator (CSI) values for the decoder and also implements a phase tracking algorithm that compensates any residual frequency offset that may appear after signal synchronization.

The guard interval of the received signal is eliminated before it is demodulated using the FFT operation. A control unit then extracts the complex coefficients of the data carriers in the OFDM symbol, and a demapper generates s soft binary representation of the received constellation points. The binary sequence is then de-interleaved and dummy soft bits are introduced to compensate for the puncturing process. Data is then decoded using a Viterbi decoder, and the estimated received binary signal is then de-scrambled.

If the MAC layer detects that all of the bits of the PPDU have been correctly received it validates the contents of the PPDU. Otherwise it discards the entire PPDU and a data retransmission is triggered.

WLAN 802.11a operates in a half duplex configuration. These means that the transceiver can transmit and receive data, but both operations can not occur simultaneously. This gives the opportunity to share some processing elements between the transmitter and receiver chains of the PMD and, consequently, reduce the area requirements of the transceiver. Figure 5.4 gives a closer look at the architecture of the transceiver developed in this research work. As it can be seen, both the scrambler, interleaver and FFT cores are shared between both chains.

The FFT core is the most reused element of the design as it is used for signal modulation in the transmitter chain, signal demodulation in the receiver chain and supports the synchronizer during time offset estimation.

The following sections describe some of the receiver chain modules that have a significant impact on the overall decoding capacity of the receiver.

5.2.1 SYNCHRONIZER

WLAN 802.11a transmits and receives information in data packets known as as PPDUs. Besides the encoded data, each PPDU contains special headers for signal detection, synchronization, channel equalization and receiver

configuration. The signal detection and synchronization header consists of repetitions of short and long training symbols.

The synchronization module is based on the algorithms in [LL04]. It uses the short training sequence for a first raw estimation of the time and frequency offsets of the received signal and then further tunes the estimation with the use of the long training sequence. The estimation stages are mostly based on signal cross-correlation in the time and frequency domain, so the existing FFT used in the transmission and receiver chains of the PMD is employed here to reduce the silicon area.

The algorithm was originally presented with a pipeline structure: the received signal was consecutively corrected with the estimations at each stage of the synchronization algorithm and served as the input for the next estimation stage. This architecture has been modified to reduce the number of Coordinate Rotation Digital Computer (CORDIC) cores. Figure 5.5 shows the architecture of the synchronizer.

5.2.2 DEMAPPER

Regarding the output they produce, demappers can be classified into *hard* demappers and *soft* demappers. In a *hard* demapper the I-Q plane is divided into disjoint regions. Each region contains a single constellation point, and thus, it is assigned the binary representation of that constellation point. The demapping algorithm checks in which region each received coefficient falls and returns the binary representation of that region. From a hardware perspective, they are a simple solution since the output can be obtained after a number of logical comparisons.

The biggest drawback of this methodology is that it provides no information about the certainty of the decision. Since the regions of the I-Q plane are disjoint, the output of the demapper changes abruptly when a certain decision threshold is exceeded. This behaviour is depicted in figure 5.6.

In the figure, two constellation points are depicted, x_0 and x_1. Received positive symbols are demapped as a logical 1, and received negative symbols are demmaped as a logical 0. However, as the received symbol gets closer to zero, one would expect a reduced certainty in the demapping function, as

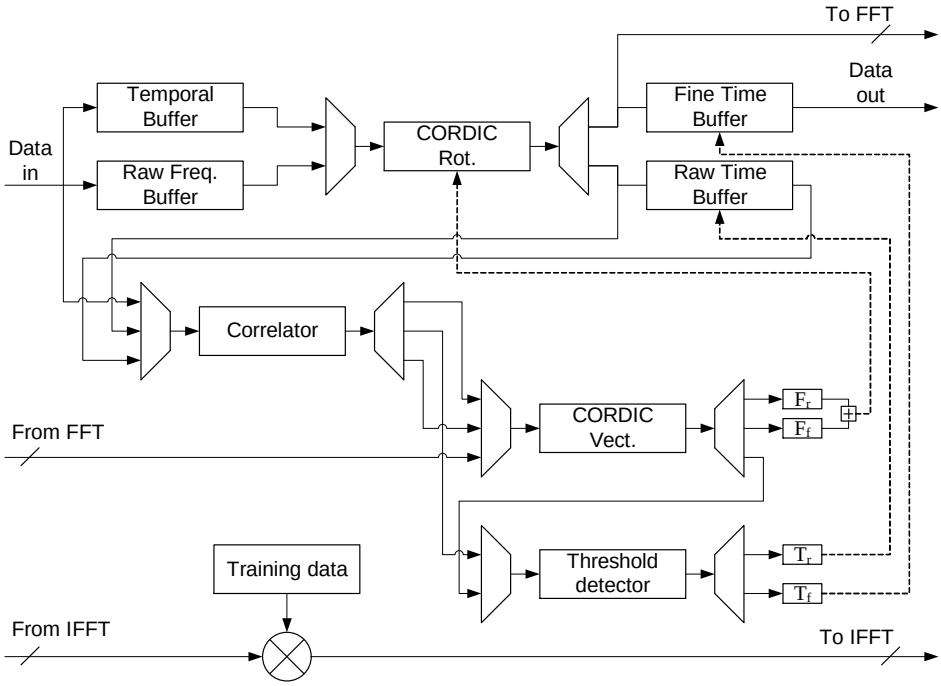


Figure 5.5: *Architecture of the synchronizer*

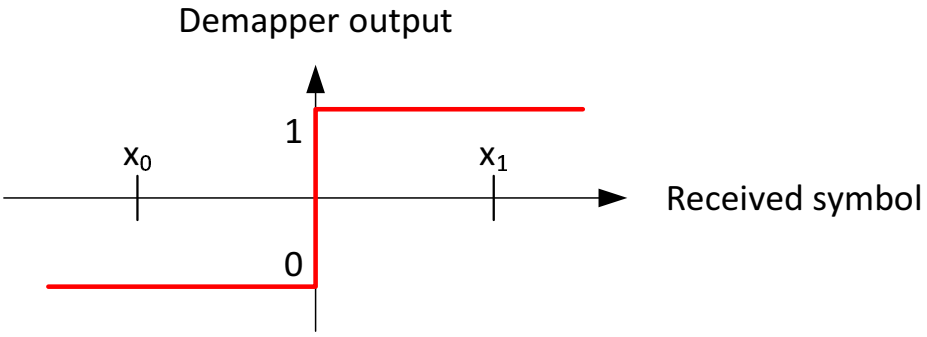


Figure 5.6: *Hard demapping*

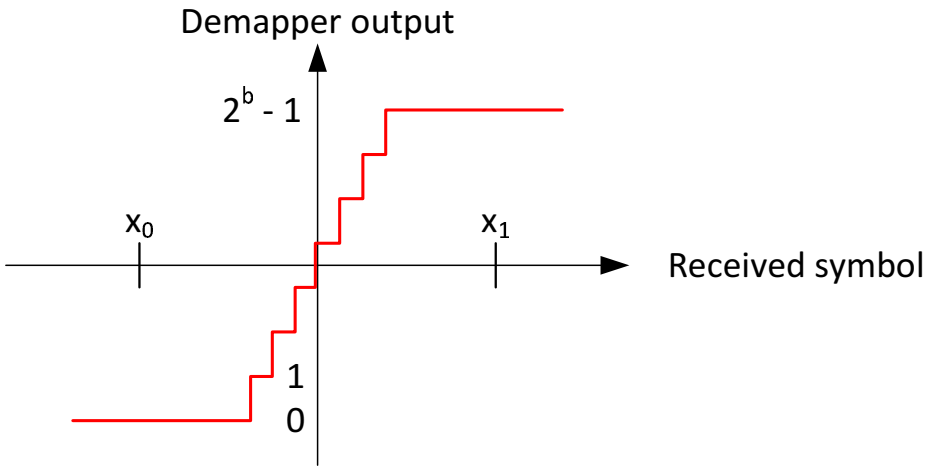


Figure 5.7: *Soft demapping*

small variations due to channel noise would toogle the logical representation of the demapped bit.

Information about the confidence in the decision is given by *soft* demappers. A *soft* demapper returns a sequence of b-bit wide integers known as soft-bits. All soft-bit values above $2^{b-1} - 1$ correspond to a logical '1', while the rest of values correspond to a logical '0'. The most certain representation of logical '1' is given by the value $2^b - 1$ while value 0 is the most confident logical '0'. Figure 5.7 summarizes the soft decoding process. It has been demonstrated [Moo05] that *soft* demappers improve the performance of decoding algorithms, but require more complex hardware architectures.

Defining the decision thresholds of the I-Q regions and obtaining a practical hardware architecture that implements them are the most challenging tasks when designing a *soft* demapper.

The decision regions of the demapper were obtained after simulating the dispersion that the WLAN multipath channel model [OP99] [Coo04] induces to the transmitted mapped symbols and, consequently, to the transmitted binary sequence. Matlab software models were used to simulate the behaviour of the channel and the transmission and reception chains of the

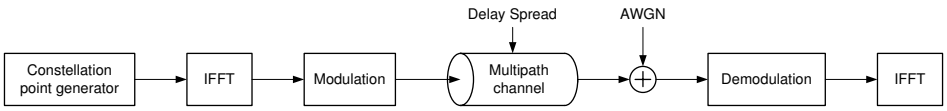


Figure 5.8: *Simulation set-up to analyze coefficient dispersion*

PMD. The parameters that control the behaviour of the channel model are the *delay spread* and the noise variance. The simulation set-up is depicted in figure 5.8.

A family of demapping functions has been defined after analyzing the distribution of the transmitted and received constellation points and the binary sequences they represent. These demapping functions return the probability of a soft-bit being a logical '1' for an input coefficient x_{in} and a channel configuration c_{chan} which includes its *delay spread* and noise levels. Mathematically this can be written as:

$$f_{demap}(x_{in}, c_{chan}) = p(bit_{out} = 1 | x_{in}, c_{chan}) \tag{5.1}$$

As an example, figure 5.9 shows different demapping functions for the 3rd mapped bit in a 64-QAM constellation for a channel with a Signal-to-Noise Ratio (SNR) level of 30dBs and different values of *delay spread*.

For an input symbol value to the demapper x_{in}, the demapper function returns the probability that that symbol is mapped as a logical one. Therefore, if this probability is quantized with b bits, the output of the operation becomes the output of the demapper.

The Gray encoding used in the mapping stage leads to the appearance of trapezoidal regions in the demapping functions. Storing the demapping functions defined in (5.1) with a sufficient precission is an impractical task from a memory perspective in an embedded system. Instead, our demapper uses first-order linear functions to approximate them as shown in green in figure 5.10. These linear approximation curves can be easily extended to support any M-QAM constellation. The linear functions have been chosen to minimize the mean square error.

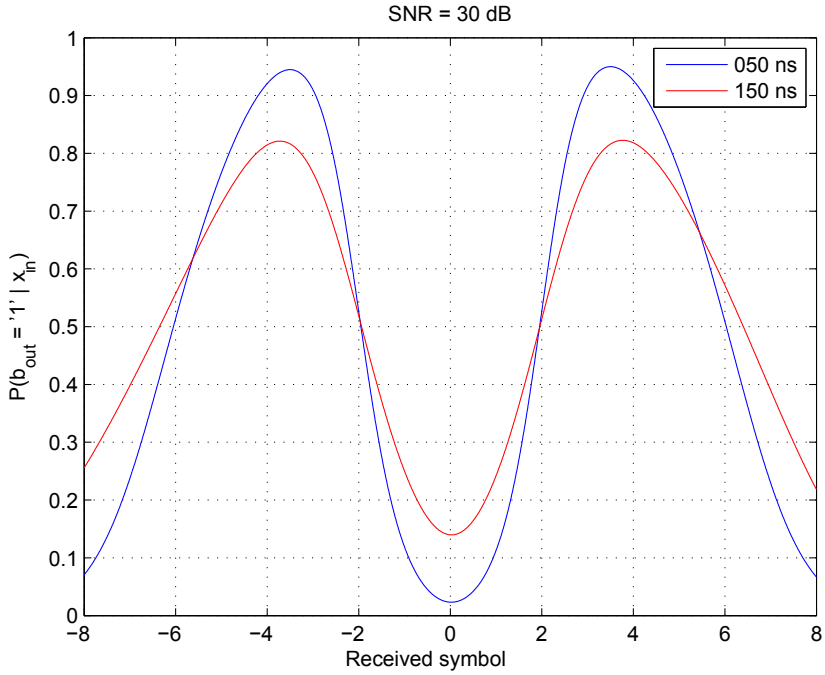


Figure 5.9: *Example of a demapping function*

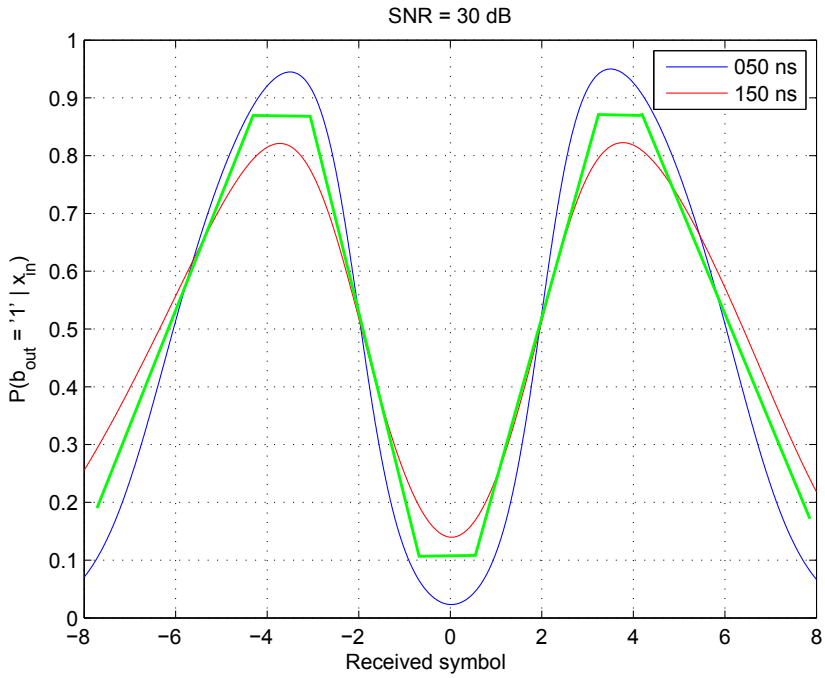


Figure 5.10: *Linear approximation of the demapping function in figure 5.9*

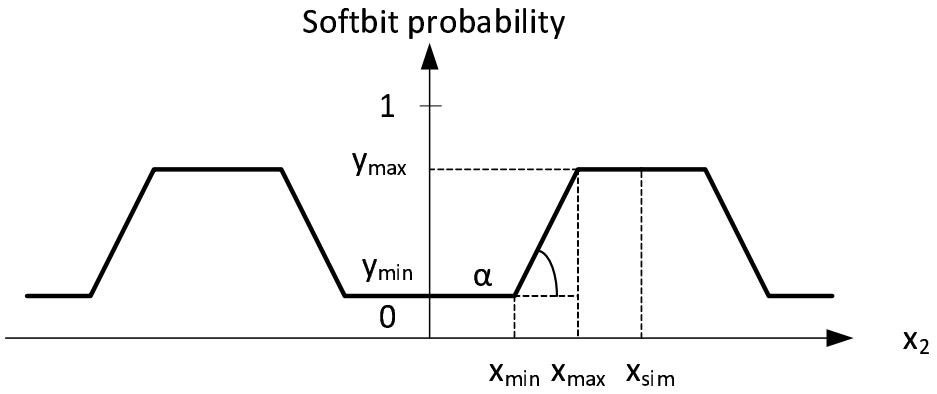


Figure 5.11: *Demapping parameters for softbit 3 and 6 on 64-QAM*

Figure 5.11 shows the parameters that define the linear approximation of the demapping function. These parameters are stored in a Read-Only Memory (ROM).

Due to the symmetries presented in the trapezoidal functions, they can be decomposed into a basic function that contains 3 unique intervals: an interval where the output is fixed to a minimum value, an interval where the output is fixed to a maximum value and an interval where the output value linearly varies from the minimum value to the maximum value. The linear interval is comprised between x_{min} and x_{max} in figure 5.11. Any input smaller than x_{min} is assigned an output value y_{min}, while any input greater than x_{max} is assigned an output value y_{max}.

Let x_0 be the I or Q component of the data carrier to be demapped. On a M-QAM constellation, where $M = 2^{2m}$, x_0 contains information about m mapped bits $b_0, b_1, \ldots, b_{m-1}$. For each bit there is a unique demapping function and a corresponding basic function. It is possible to transform x_0 so that it always falls in the region where the basic function is defined.

Equation (5.2) shows the transformation needed to adapt x_0 to demap bit b_1.

$$x_1 = \begin{cases} x_0 & x_0 < 0 \\ -x_0 & x_0 \geq 0 \end{cases} \tag{5.2}$$

Equation (5.3) shows the transformation needed to adapt x_0 to demap bit b_2.

$$x_2 = \begin{cases} x_1 & x_1 < x_{sim} \\ 2x_{sim} - x_1 & x_1 \geq x_{sim} \end{cases} \tag{5.3}$$

Similar equations can be obtained for constellations denser than 64-QAM.

Let $y_{out,i}$ be the output of the demapping function for bit b_i. $y_{out,i}$ can then be calculated as

$$y_{out,i} = \begin{cases} y_{min,i} & x_i < x_{min,i} \\ y_{min,i} + \alpha_i x_i & x_{min,i} \leq x_i < x_{max,i} \\ y_{max,i} & x_{max,i} \leq x_i \end{cases} \tag{5.4}$$

where $x_{min,i}$, $x_{max,i}$, $y_{min,i}$, $y_{max,i}$ and α_i are the demapping parameters associated to bit b_i. Since the demapping functions were defined so that they fall in the interval [0,1], then the b-bit wide softbit can be calculated as:

$$softbit_i = round(y_{out,i} \cdot 2^b) \tag{5.5}$$

5.3 HARDWARE-IN-THE-LOOP SIMULATOR

The fast verification platform is based on System Generator [Xil11a], a component of the Xilinx ISE Design Suite that enhances Matlab's Simulink platform. Simulink offers a powerful design environment for multi-domain simulation. System Generator adds a variety of new block libraries specially designed for digital signal processing.

Among its features, System Generator allows to import custom Intellectual Properties (IPs) in HDL and to compile designs to run them on real time under Xilinx Field Programmable Gate Arrays (FPGAs). Several evaluation boards are supported by default, but the environment allows any board that mets certain hardware requirements if proper configuration files are supplied by the user. System Generator also provides an Ethernet or Joint Test Action Group (JTAG) based interface for communication between the hardware and software layers of the simulation which is transparent to the user.

All these reasons make System Generator an interesting tool when designing hardware-in-the-loop simulations. Figure 5.12 depicts the block diagram of the platform that was designed to verify the performance of hardware transceiver implementation.

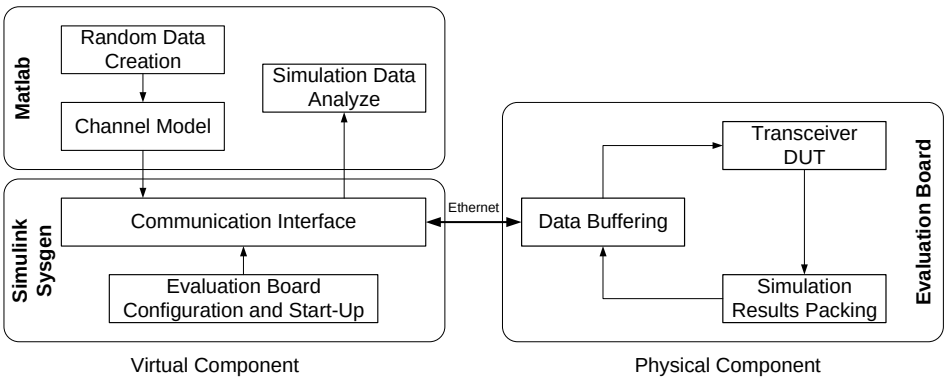


Figure 5.12: *Architecture of the fast simulator system*

The virtual component of the HiL simulator is run on software in a PC. It is responsible for creating PPDUs with random data and implementing the WLAN multipath channel with different values of noise and delay spread. The transceiver architecture is implemented in hardware on a FPGA evaluation board on the physical component of the HiL simulator. The FPGA decodes the received PPDU and counts the number of erroneous bits so that this overhead is released from the PC. It also returns information about the status of the transceiver (time and frequency offset estimations, channel estimation, ...) to the PC domain.

System Generator is an element of the virtual component of the HiL simulator that communicates the signal generation and processing on Matlab with the physical component of the FPGA using a fast ethernet link.

Given that the multipath channel model is implemented in software its statistical properties can get closer to a real channel than those obtained with embedded implementations. Moreover, it gives the possibility to evaluate the design in a more accurate scenario and not only on an Additive White Gaussian Noise (AWGN) channel like in [SRH+03]. The transceiver is also provided with an equalizer, so the design is self-sufficient and is capable of obtaining the channel state information unlike [AFC09].

Figure 5.13 gives more information about the underlying HiL platform.

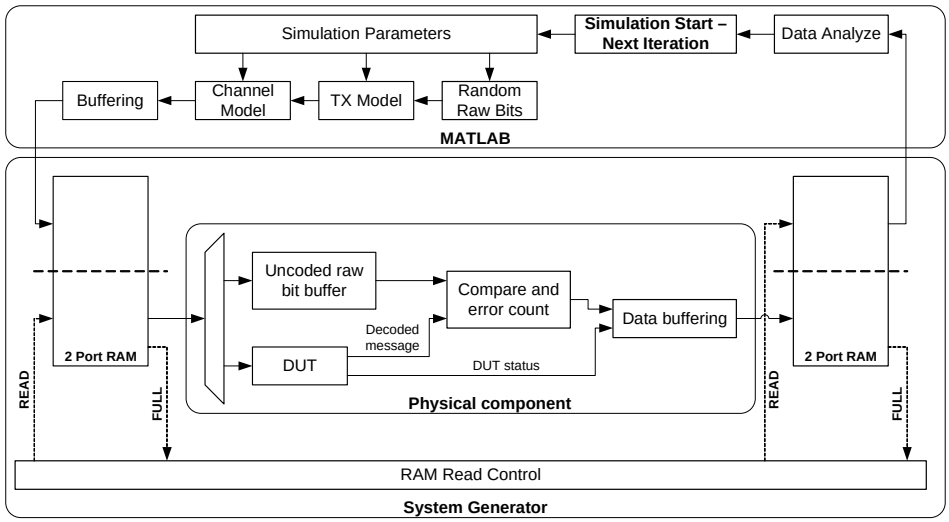


Figure 5.13: *Simulation flow dissection*

The simulation starts on the virtual component of the HiL simulator in Matlab. First, the simulation parameters (length of the PPDU, channel model, noise characteristics, WLAN link speed) of the simulation iteration are loaded into Matlab. Software models of the transmitter chain and communication channel are used to emulate the PPDU sample values as seen by the receiver. The PPDU samples are packed in 32 bit words: 12 bits are used for the I-component of the PPDU, 12 bits are used for the Q

component of the PPDU and the remaining 8 bits are used to represent the RSSI signal. Along with the PPDU, Matlab buffers a copy of the original uncoded binary message **m**.

After the PPDU data has been generated, Matlab launches System Generator. System Generator then programs the FPGA with the description of the WLAN transceiver to be tested. Data buffering between Matlab and System generator is performed using doble port Random Access Memories (RAMs) in the FPGAevaluation board. Each port of the RAM is controlled exclusively by the the virtual (System Generator) of physical (the DUT implemented on the FPGA) components of the HiL simulator. Blocking mechanisms have been provided so that a component of the HiL simulator can not gain access to the RAM while the component of the HiL simulator is using it.

At the beginning of the simulation, System Generator loads a copy of the simulation vector generated by Matlab into the double port RAM memory of the physical component. The transceiver implementation on the FPGA then stores the uncoded transmitted message **m** into a temporary buffer and decodes the received PPDU signal. The decoded bit stream sequence is compared on real time with that stored in the temporal buffer and the number of erroneous bits are counted. The error count along with other receiver parameters are buffered and loaded into the output dual port RAM. When this memory is full, System Generator sends a copy of its contents to Matlab, when PER analysis is executed. After this analysis, Matlab can launch a new simulation iteration.

Finally, figure 5.14 shows a picture of the HiL simulator under normal operation. The picture shows both the virtual (computer) and physical (Spartan-3A DSP Starter Platform [Xil09]) components of the simulator. The computer is connected to the evaluation board using two connections: the red box in the picture is the JTAG connection used to download the *.bit configuration file in the FPGA, and the red cable is the Ethernet connection between the physical and virtual components of the HiL platform.

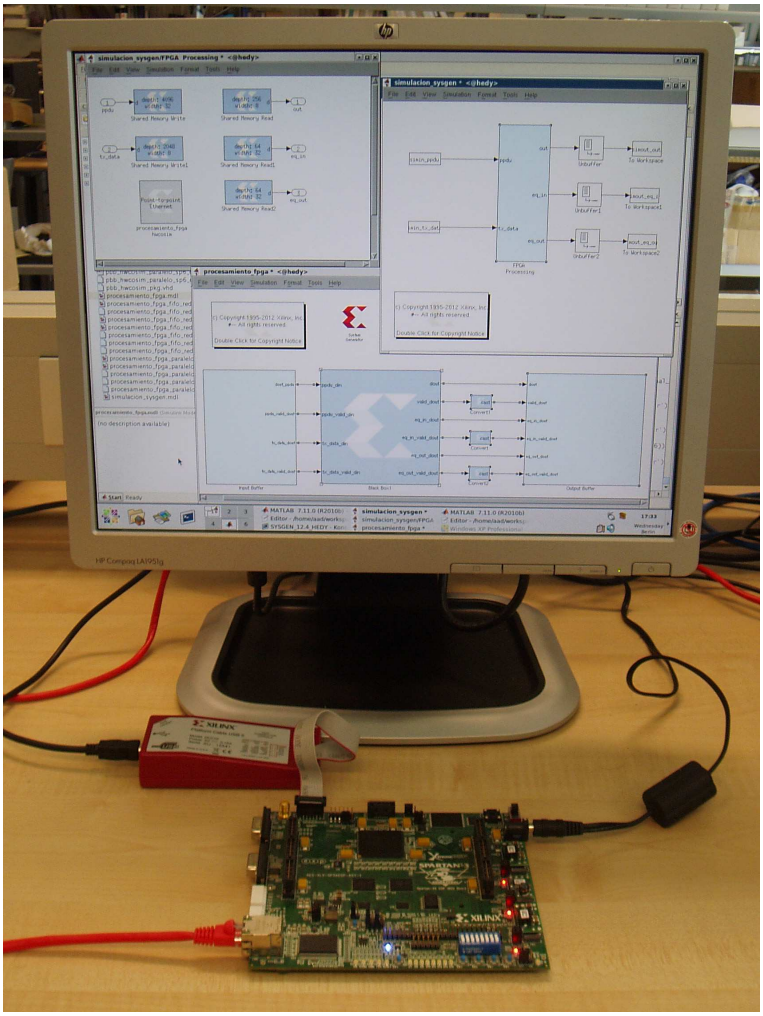


Figure 5.14: *Implementation of the HiL simulation platform*

5.4 PARAMETRICAL STUDY

Given a software model of the transceiver architecture, finding a balance between precision in the operations and area consumption is sometimes a time consuming task. Other times adjusting the size of internal accumulators depends on the working conditions of the device (channel characteristics and behaviour of different submodules of the entity). This section explores the use of the verification platform as a quantization effect analyzer: due to the time reduction hardware-in-the-loop simulations can obtain it seems logical to use them in tune-up stages.

An analysis of the quantization effects will be carried out at the following points in the Viterbi decoder:

1. Number of bits used to represent the CSI.
2. Number of bits the Add-Compare-Select (ACS) accumulator is extended.
3. Traceback depth of the decoder.

The first two parameters have a great impact on the logic consumed by the decoder as they control the size of several adders and comparators inside the design. The traceback depth parameters controls the memory requirements of the Survivor Path Unit (SPU) [ACS+08].

The HiL simulation platform will be implemented on a Xilinx Spartan-3A DSP 1800 evaluation board. This FPGA has limited block RAM resources that are necessary to buffer data between Matlab and the DUT. Large buffers are necessary to evaluate long PPDU packets. The longer the PPDU, the more data bits can be crammed on a single simulation iteration. On low speed transmission modes, large PPDU sequences are necessary to that the statistical characteristics of the decoded message $\mathbf{x}$ are representative of the communication model. Since register resources are more common in this FPGA, the SPU of the Viterbi decoder will be based on a register exchange implementation.

Taking into account the simulation flow introduced in figure 5.13 several System Generator models have been synthesized, one for each combination of parameters of the Viterbi decoder that is going to be analyzed. The

methodology employed has been as follows. First, the parameter configuration that maximizes the area consumption in the FPGA while keeping all parameters at reasonable high levels was chosen. For the Spartan-3A DSP 1800 used in this research work, the maximum values for the quantization of the CSI, ACS accumulator extension and traceback depth were chosen respectively 8, 7 and 60. The parameters were analyzed one by one, keeping the others constant.

The architectures were tested under the highest rate available on the 802.11a standard using a multipath channel with a delay spread of 150ns. The 20MHz baseband signal of the transmitter signal was oversampled by a factor of three and the noise level of the channel ranged from 18 to 30dBs. The frequency offset was randomly selected between $\pm 120ppm$ for a 20MHz bandwidth signal. Each SNR point was simulated 10^3 times. The same set of PPDUs were given to the different Viterbi decoder architectures being analyzed.

Only one parameter of the Viterbi decoder was analyzed at a time. During the analysis of a parameter, the remaining parameters were kept at a constant value.

The simulation results for the quantization of the CSI valued are depicted in figure 5.15.

Table 5.1 summarizes the hardware utilization of the different transceiver architectures

As shown in table 5.1 the hardware resources consumption of the decoder increases linearly with the number of bits used to represent the CSI. Adding an extra bit requires around 400 extra Look-up Tables (LUTs) and 100 flip-flops from the FPGA. Since no significative decoding capacity is achieved for CSI quantization levels above 5 bits this quantity has been used to represent the value of the CSI.

Simulations have shown that no performance boost is obtained if the registers of the ACS units are extended with more than 2 bits. Consequently the ACS register size has been increased with only 2 bits.

Finally figure 5.16 shows the influence of the traceback depth τ of the decoder on the performance of the transceiver. As it can be seen increasing

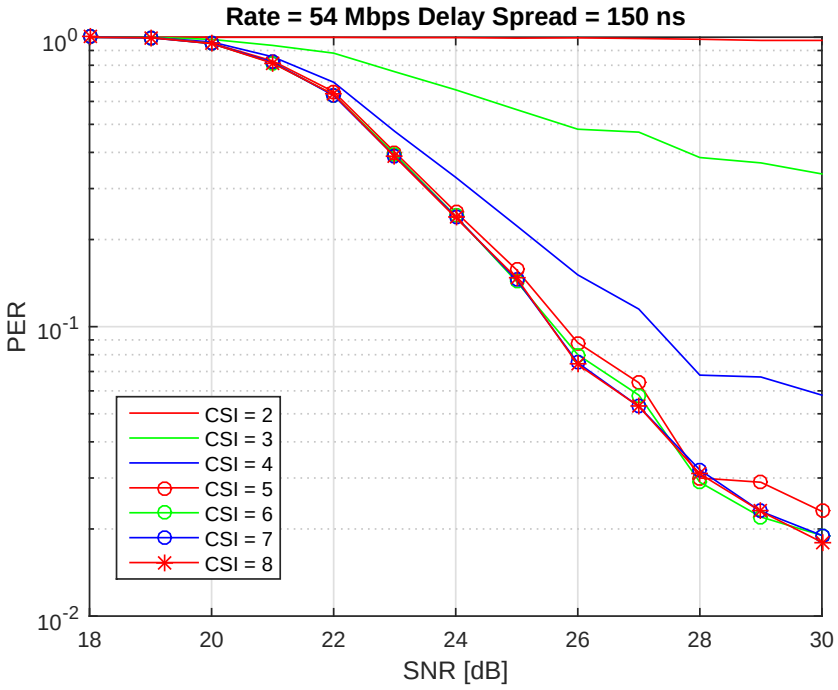


Figure 5.15: *Influence of the precision in CSI over the PER of the system*

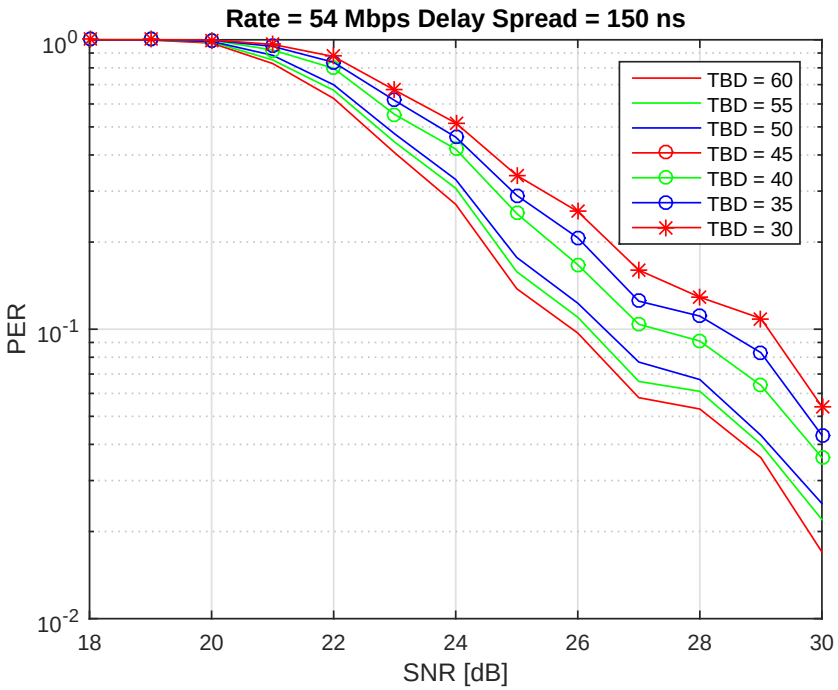


Figure 5.16: *Influence of the traceback depth of the decoder over the PER of the system*

the traceback length in 5 units supposes a gain in the decoding capacity of around 0.5 dB.

The influence of the traceback depth value τ on the decoder's hardware consumption is depicted in table 5.2. LUT and flip-flop requirements rise at a similar rate of around 65 extra components per increased unit in the traceback depth. Due to the impact this parameter has on the overall performance of the transceiver the traceback of the decoder has been kept to 60.

csi_{bw}	Viterbi decoder		WLAN transceiver		Clock Speed [MHz]
	Registers	LUTs	Registers	LUTs	
8	5824	11048	16237	24906	60.79
7	5667	10581	16081	24469	61.24
6	5573	10189	16046	24113	60.698
5	**5477**	**9797**	**15949**	**23716**	**61.002**
4	5395	9403	15866	23319	60.632
3	5278	8993	15751	22906	61.091
2	5184	8617	15656	22526	61.166

Table 5.1: *Hardware utilization of the transceiver architectures for various values of csi_{bw} with $\tau = 60$ and $acs_{xtr\_bw} = 7$*

τ	Viterbi decoder		WLAN transceiver		Clock Speed [MHz]
	Registers	LUTs	Registers	LUTs	
60	**5121**	**8037**	**15531**	**21845**	**60.787**
55	4805	7718	15213	21571	61.005
50	4480	7397	14888	21240	60.842
45	4160	7078	14568	20915	60.872
40	3840	6757	14248	20588	60.864
35	3518	6437	13926	20266	60.72
30	3198	6117	13606	19944	61.043

Table 5.2: *Hardware utilization of the transceiver architectures for various values of τ with $csi_{bw} = 5$ and $acs_{xtr\_bw} = 2$*

5.5 RESULTS

The transceiver architecture performance has been thoroughly analyzed after the optimum decoder parameters were obtained in the previous section. 6Mbps, 36Mbps and 54Mbps rates where simulated. The multipath channel had a random delay spread with a uniform distribution between 50 and 150ns. The frequency offset was also randomly selected from a uniform distribution between 0 and $\pm 120 ppm$ with a sampling clock speed of 20MHz. Each point in the PER curves of figure 5.17 was obtained after 10^5 simulations with PPDUs with 85 OFDM symbols, which results in PPDUs transmitting 2000 data bits in the 6Mbps transmission mode, 10880 data bits in the 36Mbps transmission mode and 18360 data bits in the 54MBps transmission mode.

The mean time to obtain a single PER point is about one hour. In contrast, a single Register Transfer Level (RTL) simulation of the transceiver required around fifty seconds using a Core2-Quad processor at 2.5Ghz with 2GB of RAM. Therefore, the fast verification platform can reduce the simulation time in a factor around 10^3.

The performance curves of the system have been compared to those found in the literature. Table 5.3 summarizes this comparison.

All authors that report the channel conditions in which their transceiver architectures have been simulated use a Rayleight fading channel [OP99] with different values of *delay spread*, which range from 75 to 100ns. In our case, our transceiver has been tested on a Rayleight fading channel with a delay spread of 150ns, which is much larger than any of the delay spreads reported in the literature. Consequently, with respect to the multipath fading channel our system has been tested under slightly more constraining characteristics.

The results in table 5.3 are not conclusive. First, the authors listed in the table give no clear indication about the bandwidth of the channel noise or how it was measured. Second, the models of the DACs and ADC, which can further degrade the quality of the transmitted and received signal, are not mentioned. Finally, some platforms in the table use equalizers with perfect channel knowledge, which can skew the results in their favor. However, we observe that the difference in the PER figures between our implementation and those found in the literature are normally no more

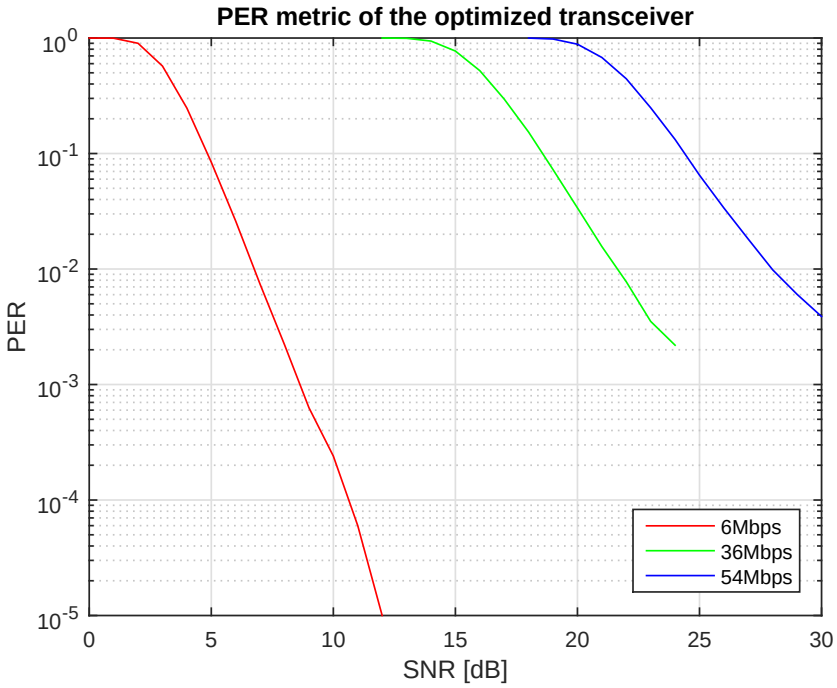


Figure 5.17: *PER of the optimized transceiver*

	[TCW05]	[Tro05]	[AA04]	[JNYK05]	[DAB$^+$01]	Proposed work
Rayleight fading channel	100ns	100ns	75ns	NA	NA	150ns
SNR for PER= 10^{-1} (54Mbps)	29dB	-	-	-	-	24dB
SNR for PER= 10^{-2} (6Mbps)	-	6.5dB	-	-	12dB	7dB
SNR for PER= 10^{-2} (36Mbps)	-	-	-	28dB	27dB	22dB
SNR for PER= 10^{-2} (54Mbps)	-	-	32dB	30dB	32dB	28dB

Table 5.3: *PER of different WLAN 802.11a transceivers*

than 5 dBs apart, which indicates that all implementations have a similar behaviour.

5.6 CONCLUDING REMARKS

In this chapter the Viterbi decoder description has been optimized with respect to the transceiver architecture in which it has been implemented.

The HiL simulation platform has reduced the time necessary to perform the parametrical analysis of the transceiver. Simulation results show that no significant improvements in the decoding capacity of the system is obtained when the decoder uses CSI word descriptions of more than 5 bits, and that increasing the traceback depth of the decoder in five units has an overall decoding capacity gain of 0.5dBs.

The optimized Viterbi decoder has been been analysed under Rayleight fading channels, and the PER curves of the resulting transceiver are in line with those found in the literature.

CHAPTER 6

Results

Contents

6.1 INTRODUCTION

In this chapter the implementation results of the Viterbi decoder architecture obtained in this research work are listed. To do so, the Viterbi decoder itself is compared with other implementations found in the literature in terms of area, speed (system clock and achievable data throughput) and the figure of merit Φ defined in section 3.2.

6.2 VITERBI DECODER IMPLEMENTATION RESULTS

The Viterbi decoder architecture presented in this chapter has been compared to other implementations found in the literature. To make a fair

comparison, the implementation of our Viterbi decoder must be as close as possible to the works found in the literature. This is feasible because our design is highly parameterizable and portable to any platform. Therefore, we can quickly adapt our description to meet the characteristics of the proposals found in the literature.

Most of the time, the transfer function $G(x)$ (code_pol in table 4.1) of the Viterbi decoders found in the literature is not given. If all the remaining parameters of a decoder are kept identical, changing the transfer function has a negligible impact on the area and timing results of the design. Indeed, the complexity of the design is governed by parameters such as the constraint length ν, the traceback depth τ and the coding rate $R = k/n$. The transfer function $G(x)$, however, only dictates the specific state transition of the Trellis diagram and is used to indicae how the available Add-Compare-Select (ACS) units are connected in the design. Therefore, we can safely use a generic transfer function $G(x)$ and make fear comparisons when this data is not proposed in the literature.

Some Field Programmable Gate Array (FPGA) implementations found in the literature discuss the FPGA family used to implement their works, but do not indicate the speed grade of the target devices. In this cases we have used the same target FPGA family with the slowest available speed grade. This way we use a worst case scenario when discussing the implementation results of our design. All our Xilinx FPGA implementation results will be obtained using PlanAhead at revision 14.7.

Some works found in the literature compare different Viterbi decoders in terms of their latency. For this research work we will define the latency of the Viterbi decoder as the number of clock cycles that are necessary to output the decoded message after the ACS cluster has calculated a set of survivor paths.

Table 6.1 summarizes the hardware resource utilization of different Viterbi decoders implementing a $R = 1/2$ convolutional code with a constraint length $\nu = 6$ and a traceback depth τ of 18 states. The decoders are implemented on a Xilinx *xa3s500ecpg132-4* FPGA [Xil11b].

Both the implementation in [MZMD13] and our solutions are based on a register exchange decoder, while the remaining implementations are based on a traceback decoder. Therefore, only [NBA13] and the Xilinx Intellectual Property (IP) decoder require block Random Access Memories (RAMs).

	[MZMD13]	[NBA13]	Xilinx IP [Xil11c]	Proposed work
Registers	2356	-	4581	1979
4 input LUTs	6261	3731	-	4631
Block RAMs	0	2	2	0
Maximum frequency [MHz]	118	97	126	113
Decoding throughput [Mbps]	118	48.5	126	113
Latency	$\tau + code_k$	1.5τ	4τ	τ
ISE version	10.1	13.4	13.1	14.7

Table 6.1: *Area and speed comparison of Viterbi decoder implementations with* $code_n = 2$, $code_k = 1$, $constr\_len = 6$ *and* $traceback\_depth = 18$ *on Xilinx xa3s500-ecpg132-4 FPGA*

From a hardware consumption perspective, our design consumes the least resources among all the solutions that disclose their requirements. The number of necessary Look-up Tables (LUTs) is reduced by around a 26% from the least demanding alternative. Similarly, the necessary register in our implementation is reduced by around a 16% from the least demanding implementation. Note that part of the registers consumed by our design are necessary to implement the register exchange logic of the Survivor Path Unit (SPU) that are embedded in the block RAMs necessary by the decoders implementing the traceback algorithm.

Our design also has the shortest latency between all the decoder implementations. On the other hand, the hardware savings come at the cost of a slight decrement of the maximum clock speed achievable by our design. As can be deduced, our design is a 10% slower than the fastest decoder implementation. However, our design is still 16% faster than the slowest implementation.

Table 6.2 summarizes the hardware resource consumption of different ACS units found in the literature. The ACS units are designed for a convolutional code with $R = 1/3$ and a 3-bit wide softbit word. All results are shown for a Xilinx xc3s500 FPGA.

The authors in [BSK+13] and [BSL11a] present two sets of results: the initial hardware consumption of a typical ACS unit and the resource utilization of their modified adaptive ACS unit. In the table we compare the implementation results of our ACS unit with their implementations.

The authors of [BSL11a] do not consider the path metric accumulator as part of the ACS unit. Therefore, their area results in table 6.2 show no registers. Taking this in consideration, our ACS implementation requires less registers than all other implementations in table 6.2. With respect to the 4-input LUT consumption, the less demanding module is the non-adaptive ACS approach in table 6.2. Our solution comes second, with an overall reduction of 27% over the next more demanding implementation.

According to the maximum clock speed achievable by the ACSs, our design is slightly slower than the adaptive ACS unit presented in [BSK+13]. Whereas the fastest ACS unit has a critical path of around 3-6ns, our proposal requires 7.3ns. The proposed architecture is, however, 2.5 times faster than the non-adaptive ACS unit in [BSK+13] .

	[BSK+13] ACS	[BSK+13] Adaptive ACS	[BSL11a] ACS	[BSL11a] Adaptive ACS	Proposed work
4 input LUTs	36	81	8	47	26
Registers	20	24	-	8	7
Critical path delay [ns]	18.45	3-6	-	-	7.3

Table 6.2: *Area and speed comparison of normal and adaptive ACSs with* $code_n = 2$, $code_k = 1$ *and* $softbit_{bw} = 3$ *in Xilinx xc3s50 FPGA*

	[HLS14]	This work	[HLS14]	This work
FPGA	xc3s200afg320-4		xc6vcx75tff484-1	
Slice	65	1537	45	595
Registers	57	1297	55	1103
LUTs	85	2246	113	1245
BRAMs	4	6	2	3
Clock [MHz]	67	100	165	201
Throughput [Mbps]	0,51	100	1,263	201

Table 6.3: *Data throughput comparison between Viterbi decoders with* $code_k = 1$, $code_n = 2$, $\nu = 6$, $softbit_{bw} = 1$ *and* $\tau = 35$ *on different Xilinx FPGA*

Table 6.3 compares the data throughput between an area optimized Viterbi decoder architecture [HLS14] and our solution. All solutions are configured to decode a convolutional code with $R = 1/2$, $\nu = 6$, a traceback depth τ of 35 states and are based on a hard-decoding receiver with traceback based SPUs. Implementation results are given for Xilinx Virtex6 xc6vx75tff484-2 and Spartan-3E xc3s200afg320-4 FPGAs.

The reduction in area resources presented in [HLS14] are obtained by minimizing the total number of ACS units in the design and by sharing the available ACS units between the different states in the trellis. Our solution, on the other hand, instantiates a total of 2^{ν} ACS units. The maximum clock speed achievable by the design in [HLS14] is 165MHz, but due to the reutilization of the ACS units, the maximum decoding throughput achievable by the core is reduced to 1.263Mbps. In other words, this implementation decodes one single bit every 130 clock cycles. Our solution, on the other hand, reaches a maximum clock speed of 214.5MHz on the same FPGA, and can decode an output bit per clock cycle, so the achievable decoding rate of our solution is 214.5Mbps (170 times that of [HLS14]). The penalty of our design in area is less than the penalty of the work in [HLS14] in decoding capacity reduction.

	[SV13]	Proposed work
Register	198	123
6-input LUTs	390	179
LUTs used as logic	370	179
LUTs used as memory	20	0
Number of 6-input LUT FF pairs	394	213
Maximum frequency [MHz]	394	487.3

Table 6.4: *Implementation results of a Viterbi decoder with* $code_k = 1$, $code_n = 2$, *constr_len* $= 2$ *and* $\tau = 32$ *in a Xilinx xc7vx330t-ffg1157-3 FPGA*

Table 6.4 compares the implementation results of the Viterbi decoder presented in [SV13] with our proposal. Both solutions are configured to decode a convolutional code with $R = 1/2$, $\nu = 2$, a traceback depth τ of 32 states and are based on hard-decoding receiver. Implementation results are given for a Xilinx xc7vx33t-ffg1157-3 FPGA

As it can be seen, our proposal consumes 37% less registers and around half the logic than the work in [SV13]. However, our design reaches a clock speed of 487.3MHz, a 23% more than the frequency achieved by [SV13]. Both architectures can decode a single bit per clock cycle of the system.

Table 6.5 compares our proposed Viterbi decoder architecture with various modified adaptive Viterbi decoders found in the literature. The synthesis tools available do not support the oldest FPGAs, and we have only been able to implement our design on the XCVLX300 FPGA. However, when we compare ourselves with [LDZL11a], the fastest design in the table, our design is 66.93% faster with equal constraint lengths ν, and 66.87% faster when we increase the constraint length of our design in a unit.

Table 6.6 compares our proposal with that found in [PA14b]. In all cases a convolutional code with $R = 1/2$, $\nu = 6$ and $\tau = 32$ has been implemented. All Branch Metric Units (BMUs) implement hard-decoding,

Work	ν	R	FPGA	Frequency [Mhz]	Troughput [Mbps]
[CC01]	6	1/2	XCV800	-	19.7
[STGB02b]	8	1/2	XC4036	40.5	0.333
[ZB03]	9	1/2	XCV300	101	12.5
[CV03]	8	1/2	Virtex-II	60.5	60.5
[TSR$^+$05b]	13	1/2	XCV1000	13	0.415
[BPBS06]	8	1/2	XC2V2000	32.26	2
[LDZL11a]	8	1/2	XC5VLX300	202.587	202.587
This work	8	1/2	XC5VLX300	338.181	338.181
This work	9	1/2	XC5VLX300	338.066	338.066

Table 6.5: *Throughput comparison of adaptive Viterbi decoders and the proposed implementation on different Xilinx FPGAs*

Work	FPGA	Area (LUTs)	Maximum Frequency [MHz]
[PA14b]	xcv5lx20tff3232-2	4379	127.474
[PA14b]	xc6vcx75tff484-2	4521	157.149
This work	xcv5lx20tff3232-2	3553	326.797
This work	xc6vcx75tff484-2	2598	342.936

Table 6.6: *Implementation results of a Viterbi decoder with* $code_k = 1$, $code_n = 2$, $constr\_len = 6$ *and* $softbit_{bw} = 1$ *in different Xilinx FPGAs*

and the Viterbi decoders have been implemented for the smallest Virtex-5 and Virtex-6 FPGAs.

As it can be seen, our design has an advantage of around 19% on area consumption in the Virtex-5 implementation, and the design can operate with clock frequencies 2.5 times that obtained in [PA14b]. The results improve when the FPGA is switched to the Virtex-6 family. Here, the area consumption in LUTs is reduced by a 42.5%, while the achievable maximum clock speed is increased by a factor of 2.18. This makes us assume that our design is much better suited for 6 input LUTs FPGAs as the Virtex-6 than the proposed work in [PA14b].

Tables 6.7, 6.8 and 6.9 compare the configurable Xilinx Viterbi decoder IP with the implementation obtained in this research work when a Virtex-6 xc6vlx75t-3 FPGA, a Spartan-6 xc6slx45t-2 FPGA and a Virtex-5 xc5vl30-3 FPGA are used respectively. All implementations of the Xilinx IP are based on a traceback SPU, and in all cases implementation results of our decoder based on both register exchange and traceback SPUs are given.

In general, our register exchange based Viterbi decoder requires more logic elements than Xilinx's reconfigurable IP because the SPU must be implemented in logic instead of RAM. However, by doing so, our design reaches clock speeds that are around 7% higher than those achieved by Xilinx's IP on Virtex-6 and Virtex-5 (tables 6.7 and 6.9) implementation. Also, our implementation has half the latency than that of the Xilinx IP.

	[Xil11c]	Proposed work [RE]	Proposed work [TB]
LUTs	2573	6604	2656
Registers	1863	6873	1969
BRAMs	2	-	3
Clock [MHz]	347	373.7	361.9

Table 6.7: *Implementation results of a Viterbi of the Xilinx reconfigurable Viterbi decoder with* $code_k = 1$, $code_n = 2$, $constr\_len = 6$, $\tau = 96$ *and* $softbit_{bw} = 3$ *in a Xilinx xc6vlx75t-3 FPGA*

	[Xil11c]	Proposed work [RE]	Proposed work [TB]
LUTs	2442	6708	2624
Registers	1980	6980	1886
BRAMs	2	-	3
Clock [MHz]	126	182.55	176.8

Table 6.8: *Implementation results of a Viterbi of the Xilinx reconfigurable Viterbi decoder with* $code_k = 1$, $code_n = 2$, $constr\_len = 6$, $\tau = 96$ *and* $softbit_{bw} = 3$ *in a Xilinx xc6slx45t-2 FPGA*

	[Xil11c]	Proposed work [RE]	Proposed work [TB]
LUTs	2457	9689	3652
Registers	1538	6867	2053
BRAMs	2	-	3
Clock [MHz]	272	291.3	313.7

Table 6.9: *Implementation results of a Viterbi of the Xilinx reconfigurable Viterbi decoder with* $code_k = 1$, $code_n = 2$, $constr\_len = 6$, $\tau = 96$ *and* $softbit_{bw} = 3$ *in a Xilinx xc5vl30-3 FPGA*

When our traceback based SPU decoder is compared to Xilinx's IP, implementation results are much more similar. In table 6.7 it is shown that our design requires 3% more LUTs and 5% more registers than Xilinx's implementation. However, our design achieves 4% higher clock speeds at the same latency. For the Spartan-6 in table 6.8, our design consumes 7.5% more LUTs and 4.7% less registers and achieves a 40% higher clock speed than Xilinx's IP. This advantage in the achievable clock speed can be explained by the logic distribution inside the FPGAs.

The top level logic element inside a Xilinx FPGA is the Configurable Logic Block (CLB). In series-6 FPGAs each CLB is divided in two slices, and each slice contains four 6 input LUTs and eight registers. Virtex-6 slices are classified as SLICEDs and SLICEMs. The LUTs of the former have standard logic reconfigurability, while the LUTs of the latter can be configured to produce shift registers and distributed RAM. On the Spartan-6 there is a third slice type, the SLICEX. Compared to the previous slices, SLICEX has more limited routing capabilities. On Spartan-6 FPGAs the availbale slices are classified as follows: 50% of the total are of type SLICEX, 25% of the total are of type SLICEM and the remaining 25% are of the SLICED. On Virtex-6 FPGAs, however, half of the slices are of type SLICEM and the other half are of type SLICED. Therefore, the routing resources are more limited on the cheaper Spartan-6. According to the data shown in table 6.7, our design is less routing resource intensive than the Xilinx IP. When both designs are ported to a more constrained

	[PN14]	Proposed work [RE]
Slices	17606	13445
Clock [MHz]	197.25	82.974

Table 6.10: *Implementation results of a reconfigurable Viterbi decoder with maximum parameters defined as* $code_k = 5$, $code_n = 6$, $constr\_len = 8$, $\tau = 96$ *and* $softbit_{bw} = 1$ *in a Xilinx xc5vlx330t-1 FPGA*

FPGA such as the Spartan-6, it is logical that our design takes a more reduced impact.

Finally, for the Virtex-5 implementation in table 6.9 our design reaches clock speeds 15% higher than those of the Xilinx's IP at the expense of nearly 50% more LUTs and 33% more registers.

The advantage in hardware resource utilization of the Xilinx IP over our implementation can be explained by the fact that our decoder uses general purpose Hardware Description Language (HDL) code and Xilinx's IP explicitly instantiates hardware resources on each FPGA.

Table 6.10 compares the implementation results of a reconfigurable Viterbi decoder for Software Defined Radio (SDR) with our proposed Viterbi decoder. Implementation results are given for the most demanding modes supported by the reconfigurable decoder ($k = 5$, $n = 6$, $\nu = 6$, $\tau = 96$ and hard-decoding) on a Xilinx Virtex-5 FPGA.

As can be seen in table 6.10 requires one fourth of the logic resources than those of [PN14] at the expense of a reduction in the achievable clock speed of around 60%. These clock speed results can be explained by the fact that in 6.10 it is only mentioned that a Xilinx xc5vlx330t FPGA is used, but no explicit mention of its speed grade is make, so for our comparison we have opted to analyse the worst case scenario and, therefore, we have implemented our design in the slowest grade FPGA with that part-name, the Xilinx xc5vlx330t-1.

Table 6.11 compares the implementation results of a Viterbi decoder aimed at Ultra Wide Band (UWB) applications with our proposed decoder

	[VS12]		Proposed work [RE]		Proposed work [TB]	
softbit	LUTs	Clock	LUTs	Clock	LUTs	Clock
3	2546	256	3891	156	2492	242.6
4	3214	252	4478	147.76	3618	230.097
5	3748	248	4666	150.852	4092	191

Table 6.11: *Implementation results of a Viterbi decoder with* $code_k = 1$, $code_n = 2$, $constr\_len = 6$, *and* $\tau = 120$ *on a Xilinx xc6vcx75t-1 FPGAs for different softbit widths*

in register exchange and traceback configurations. In all cases, a decoder code with $k = 1$, $n = 2$, $\nu = 6$ and $\tau = 120$ has been implemented on a Xilinx Virtex-6 xc6vcx75t-1 FPGA. Both decoders used a soft-decision BMU and the input soft-bits have been quantized with three, four and five bits.

As can be seen in table 6.11, for all softbit quantization levels all of our implementations achieve lower clock speeds than [VS12]. The traceback based implementation matches, however, more closely the clock speeds achieved by [VS12] than the register exchange based implementation, with clock speeds being at least the 92% of those in [VS12] when the input softbits are quantized with three or four bits. With respect to the logic resource utilization, our traceback based decoder has identical LUT requirements than [VS12] when the softbits are quantized with 3 bits, but as this level increases, our design requires around 10% more LUTs than [VS12].

Table 6.12 compares a Viterbi decoder aimed at 802.15.3c applications with our proposed implementation. All decoders are based on the traceback algorithm and are generated with a convolutional code with $k = 1$, $n = 2$, $\nu = 6$ and $\tau = 64$ on a Xilinx Virtex-6 xc6vcx75t-1 FPGA. Two different ACS units are considered in 6.12: a regular ACS core and a retimed CSA unit.

As can be seen in table 6.12, compared to the regular ACS in [VNS12b], our design requires 25% less LUTs and achieves clock speeds 10% higher.

	[VNS12b] Regular ACS	[VNS12b] CSA	Proposed work
LUTs	3110	2344	2362
BRAMs	1	1	3
Data throughput [Mbps]	226	272	247.52

Table 6.12: *Implementation results of a Viterbi decoder with* $code_k = 1$, $code_n = 2$, $constr\_len = 6$, *and* $\tau = 84$ *on a Xilinx xc6vcx75t-1 FPGAs 802.15.3c applications*

When compared with the CSA ACS, however, both implementation require the same amount of LUTs, but our decoder has a penalty of around 9% on its achievable clock speed. It is also remarkable the total number of RAMs required by both designs: whereas [VNS12b] requires only one dual port RAM our design uses up to three RAMs. The reduced number of RAMs in [VNS12b] is explained by the fact that the Viterbi decoder operates with packets of predefined size, whereas our design has been optimized for continuous data streams.

Finally, tables 6.13 and 6.14 compare our design with most of the decoders found in this section in terms of the metric Φ that was introduced in section 3.2.2. The flexibility of our description has allowed to quickly adapt our proposal to the specific characteristics of the proposals found in the literature.

The metric Φ indicates the average data throughput that can be achieved for a given complexity in the decoder in terms of LUTs and register. Higher values of Φ indicate that the design is more efficient. Consequently, only those decoders whose total LUT and register requirements and data throughput were known have been included in this discussion.

The metric Φ rewards those designs that do not share ACS among states in the trellis (and therefore, sacrifice data throughput for area resource utilization). This explains the penalty in terms of Φ of the area efficient design in [HLS14] in table 6.13. In general, the Φ normalization factor indicates that most Viterbi decoder implementations are similar in

		[MZMD13]	[NBA13]	[HLS14]	[SV13]	[PA14b]	[PA14b]	[HLS14]
Literature	FPGA	xa3s500-4	xa3s500-4	xc6vcx75-1	xc7vx330-3	xc5lx20t-2	xc6vcx75t-2	xc3s200-4
	ν	6	6	6	2	6	6	6
	LUTs	6261	3731	113	390	4379	4521	85
	Registers	2356	-	55	198	-	-	57
	Clock	118	97	165	394	127.474	157.149	67
	Throughput	118	48.5	1.263	394	127.474	157.149	0.51
	Φ	0.014	0.013	0.008	0.670	0.029	0.035	0.004
This work	LUTs	4631	4631	1703	179	3553	2598	2246
	Registers	1979	1979	1103	123	-	-	1297
	Clock	113	113	254.065	487.3	326.797	342.936	100
	Throughput	113	113	254.065	487.3	326.797	342.936	100
	Φ	0.017	0.017	0.090	1.614	0.092	0.132	0.028

Table 6.13: *Decoder comparison in terms of Φ comparison 1/2*

		[Xil11c]	[Xil11c]	[Xil11c]	[PN14]	[VNS12b]	[VNS12b]
Literature	FPGA	xc6vlx75t-3	xc6slx45t-2	xc5vl30-3	xc5vx300t-1	xc6vcx75t-1	xc6vcx75t-1
	ν	6	6	6	8	6	6
	LUTs	2573	2442	2457	17606	3110	2344
	Registers	1863	1980	1538	-	-	-
	Clock	347	126	272	197.25	226	272
	Throughput	347	126	272	197.25	226	272
	Φ	0.078	0.028	0.068	0.011	0.073	0.116
This work	LUTs	2656	2624	3652	13445	2362	2362
	Registers	1969	1886	2053	-	-	-
	Clock	361.9	176.8	313.7	82.974	247.52	247.52
	Throughput	361.9	176.8	313.7	82.974	247.52	247.52
	Φ	0.078	0.039	0.055	0.006	0.105	0.105

Table 6.14: *Decoder comparison in terms of Φ comparison 2/2*

terms of data throughput to logical resources utilization ratio, with the sole exceptions of [NBA13], [SV13], [PA14b] and [PA14b] in table 6.13, where our design shows a significant advantage over the implementations found in the literature, and [PN14] in table 6.13 where our design shows a worse behavior.

When comparing to the Xilinx reconfigurable IP [Xil11c], our design generally requires more hardware resources. However, all our register exchange and traceback based implementations obtain higher clocks (and, therefore, higher data decoding throughputs) than the Xilinx IP. In terms of Φ, this means that our design is competitive against the proposal of Xilinx, specially in designs where our architecture achieves much higher decoding throughputs such as the Spartan-6.

6.3 CONCLUDING REMARKS

In this chapter the Viterbi decoder developed in the research work has been analyzed and its performance has been compared to other implementations found in the literature.

The Φ metric, a value that gives an idea of the data throughput that can be achieved per LUT and register in the decoder, indicates that our implementation is competitive against other options found in the literature. In most cases the Φ parameter obtained in this work and other proposals fall in the same order of magnitude, with a slight advantage to our design when we obtain a design that achieves higher clock speeds. Compared to the other works with the literature, this indicates that our design, for a given value of ν, requires less area to obtain the same decoding rates. Therefore, ours is a cheaper implementation that offers identical performance to that of the competing decoders found in the literature.

Compared to the commercial Viterbi decoder IP by Xilinx, the Φ parameter rewards those designs that favor total system clock speed. This way, our design has similar performance to that of Xilinx when implementing on a Virtex-6 FPGA. When implemented on a Spartan-6 FPGA our design becomes more competitive due to the increase in the achievable system clock speed.

CHAPTER 7

Conclusions and areas for further research

Contents

7.1 CONCLUSIONS

The Viterbi decoder has a significant impact on the size and decoding capacity of transceivers based on convolutional codes. In this research work the hardware description of a flexible Viterbi decoder has been proposed and optimized taking into account the transceiver in which it will be embedded.

With respect to the Viterbi decoder description it has been concluded that:

- The design is highly configurable and offers a wide range of input parameters that can be specified by the designer. This way, the designer can set-up the decoder to comply with any given feedforward

convolutional code $(n, k, \nu, G(x))$. The Branch Metric Unit (BMU) can be configured so that it performs hard or soft decoding. In case soft decoding is selected, the bitwidth of the input softbits is a system parameter too.

- Depending on the technology and available hardware resources of the platform in which the decoder will be implemented, two different Survivor Path Unit (SPU) architectures have been implemented: a register-exchange based SPU is highly dependent on register and multiplexers, while a traceback based SPU relays fundamentally on single port Random Access Memory (RAM). The former has the shortest decoding delay possible on Viterbi decoders (τ clock cycles), while the latter relaxed the utilization of hardware resources at the expense of a higher decoding latency of 2τ cycles. The two SPU architectures have been optimized for both continuous data decoding flows and burst or packet communications, so that the decoder is fitted for applications such as Wireless Local Area Network (WLAN) and Code Division Multiple Access (CDMA) or Terrestrial Digital Video Broadcasting (DVB-T) and Ultra Wide Band (UWB). The traceback depth of both SPUs is a system parameter as well.

- The BMU can be optionally configured to take into account Carrier Strength Indicator (CSI) values of the received symbols $\mathbf{r}$ when calculating the branch metrics. CSI has been proven a cost-effective solution that, based on the channel estimation performed by an equalizer, can extract further information from the received symbol stream. By using CSI information on the BMU, the decoding capacity of the receiver has been improved on Rayleight fading multipath channels. The bitwidth of the optional CSI values is a system parameter as well.

- The decoder has comparable configuration parameters to other reconfigurable decoders found in the literature as the Viterbi decoder by Xilinx [Xil11c]. Compared to the solution of Xilinx, our proposed decoder:

  - Is not tied to Xilinx Field Programmable Gate Arrays (FPGAs) and can be implemented on any Application Specific Integrated Circuit (ASIC) technology or FPGA vendor.
  - Supports both register exchange and traceback SPUs.
  - Depending on the target FPGA where it is synthesized, both SPU implementations can be clocked with a system clock 7%

higher than that supported by the Xilinx solution. Consequently, our solution has a 7% higher data decoding throughput than the Xilinx solution.

  - The decoding delay of the register exchange based SPU is half that of the Xilinx solution

- The high flexibility of the decoder proposed in this research work will reduce design costs and the time to market of future products developed in the research group. Since the design is highly reconfigurable, by means of generics the Viterbi decoder can be set up to decode any given feedforward convolutional code.

- The Viterbi decoder is currently being sold as a commercial Intellectual Property (IP) and, to the author's knowledge, it has already successfully been implemented on Global System for Mobile communications (GSM) and Global Positioning System (GPS) satellite applications.

With respect to the transceiver designed to evaluate the behaviour of the Viterbi decoder it has been concluded that:

- A fully functional WLAN 802.11a compliant transceiver architecture has been obtained that is functional up to the Physical Layer Convergence Protocol (PLCP) layer.

- All building blocks of the transceiver have been implemented in-house in the research group and. With the exception of the Fast Fourier Transform (FFT) and Inverse Fast Fourier Transform (IFFT) cores, all modules of the transceiver have been specifically designed during this research work.

- To reduce area resource utilization, the FFT/IFFT core is reused between the modulator and demodulator of both the transmitter and receiver chains, and also supports in the synchronization algorithm.

- The synchronizer developed in this research work for the WLAN 802.11a transceiver performs time and frequency offset compensation. The error in the time offset estimation performed by the synchronizer is negligible for channels with a Signal-to-Noise Ratio (SNR) above 8 dBs, and the frequency offset estimation has a variance of around 1 kHz for channel SNRs above 14 dBs.

- The equalizer developed in this research work serves a triple purpose in the receiver chain. First, it performs channel equalization on the received complex symbols. Second, it performs a phase tracking algorithm that compensates the spurious frequency offset after synchronization by means of information on the pilot tones of the Orthogonal Frequency Division Multiplexing (OFDM) symbols. Third, it calculates the optional CSI values by normalizing the magnitude of the channel estimation with its maximum value. This way, CSI values always belong to the set $[0, 1]$. The hardware resource requirements of the equalizer are minimized by sharing the necessary Coordinate Rotation Digital Computer (CORDIC) core in rotational mode between the equalization and phase tracking algorithm and the hardware divider between the equalization process and the CSI value calculation.

- An analysis of the spread transmitted coefficients suffer due to multipath fading channels has been carried out. With that, a set of optimal demapping functions have been obtained and an iterative hardware architecture that implements them has been described in VSIC (Very High Speed Integrated Circuits) Hardware Description Language (VHDL) based on comparators and multipliers. The configurable size of the multipliers in the demapper controls the bitwidth of the softbits at its output, and the hardware architecture of the demmaper can easily be updated to support any given M-QAM constellation.

- When the transceiver supports CSI decoding, a delay line between the equalizer and the Viterbi decoder must be implemented so that the received sequence and estimated CSI values are moved along the receiver chain simultaneously.

  - If the transceiver implements interleaving, its functionality must be duplicated in the receiver chain so that received bits and CSI values are synchronized.
  - If multi-level Quadrature Amplitude Modulation (QAM) constellations are to be used, the demapper must replicate the CSI values accordingly.

- A method to increase the data throughput of the transceiver has been proposed in cases where the system is limited by the maximum achievable clock frequency of the Viterbi decoder. To do so, the

convolutional code has been parallelized and type-1 [BK13] radix-2^{pk} Add-Compare-Select (ACS) implementations have been obtained with $p = 1, 2, 3$. The entire WLAN transceiver has been synthesized on a 90nm TSMC mutivoltage process at 0.84 and 1 volts. The parallel Viterbi decoder operates at the power domain with the lowest voltage and is clocked by a signal that is 2^{p-1} times lower than the main system clock. Even though the hardware resource utilization of the parallel transceiver archietctures grows exponentially with p, power estimation carried out with Switching Activity Interface File (SAIF) files concludes that the power consumption on all cases remains in simular values.

With respect to the Hardware-in-the-Loop (HiL) verification it has been concluded that:

- The platform permits to quickly characterize the performance of different transceiver architectures in terms of their Packet Error Rate (PER).
- The platform reduces substantially the time requirements to simulate the reception of a data packet. For Joint Test Action Group (JTAG) based HiL simulators, the simulation time is reduced by a factor of 20 compared to purely based simulations. When Ethernet based HiL simulators are used, the simulation time is reduced by a factor of 1000.
- The reduction in simulation time allows the possibility of performing parametrical analysis on the Device Under Test (DUT). In this research work, the impact several parameters of the Viterbi decoder (namely, the bitwidth of the CSI values and the traceback depth of the decoder) have on the performance of the transceiver architecture has been analyzed. After the analysis, a cost effective transceiver architecture has been obtained in terms of its PER.
- By means of HiL simulations, a high confidence on the correctness of the VHDL sources can be obtained early on the design process, which in turn reduces the number of runs necessary to obtain a working final implementation of the DUT being developed.
- The increase in the simulation speed provided by the HiL platform can be used to evaluate multi-mode communication systems such as

WLAN 802.11a and detect, early on the development, rare parameter configuration that cause unexpected behaviour on the design.

- Apart from communication systems, the HiL simulation platform developed in this research work can be expanded to evaluate DUTs whose testing relies on the generation and comparison of input and output data vectors.

7.2 AREAS FOR FURTHER RESEARCH

The aim of this research work was to find and optimize the description of Viterbi decoders with respect to the transceiver architecture in which they will be integrated. During this work, several topics have been found that deserve further research:

Add more parametrization options to the Viterbi decoder

The Viterbi decoder description obtained in this research work has been optimized so that it decodes a block of k bits per clock tic. This decision maximizes the system data throughput, but increases the hardware resource requirements. Some of the proposals found in the literature reduce their area requirements by reducing or sharing ACSs between states of the convolutional code at the expense of a lower data decoding throughput.

In applications with limited area resources, adding the possibility of sharing ACS units among different states on the trellis can reduce the hardware resource utilization of our design. This, in turn, will make our flexible design more competitive against other implementations that have been designed for low area footprint goal.

Improve the traceback implementation of the SPU

The decoder architecture designed in this research work has been obtained with portability in mind, so that it can be easily implemented in any platform (ASIC or FPGA). In order to obtain this goal, our description does not make use of any proprietary IPs.

For traceback implementations of the SPU this means that we have opted for a standard single port RAM implementation that can be

easily ported to any ASIC technology of FPGA vendor. This, however, means that we do not make use of more advanced hardware resources found in FPGAs such as double port RAMs. Consequently, to achieve a continuous data decoding stream, our solution requires more RAM resource than other solutions found in the literature which do not pursuit portability.

Therefore, a new parameter could be added to our flexible design in a future revision of the decoder that indicates whether using dual port RAMs is allowed or not.

Obtain the description of a technology independent HiL platform

The HiL platform developed in this research work can implement DUTs described in Hardware Description Language (HDL). However, the HiL platform itself relies on libraries provided by Xilinx, so it is limited to FPGAs of the same vendor. This solution reduces the initial time necessary to obtain the HiL, but ties the entire platform to a specific vendor.

In future releases of the HiL verification system it would be interesting to obtain custom descriptions of its components so that it can be easily ported to any platform.

Distribute the HiL platform between multiple FPGAs

The HiL platform is limited to a single FPGA. This means that the logic of both the transceiver and the HiL platform must fit into a single FPGA. When the transceiver architecture grows in complexity, the available resources in a single FPGA can not be sufficient to fit the entire transceiver architecture in the physical layer of the simulator.

Therefore, it would be highly valuable if future revisions of the HiL platform could distribute its logic between an array of FPGAs.

Dynamic configuration of the HiL platform

The HiL platform developed in this research work evaluates the performance of a transceiver description by generating a data vector containing the received signal as seen from the analog front-end and by reading the decoded sequence generated by the transceiver.

During each simulation, there is no control mechanisms over the transceiver architecture itself. It would be interesting if future revisions of the HiL platform could provide mechanisms to have control over certain parameters of the DUT on real time.

Analysis of performance curves with ideal models

The PER performance curves shown in this research work include the effects of all final implementations of the modules that comprise the WLAN transceiver. In the future, it would be interesting to obtain the performance curves of the several iterations of the transceiver. The analysis would begin with a software model of the transceiver with perfect channel knowledge, and in each iteration each component would be substituted with its real implementation in the HiL platform. This way we would have a more precise understanding of the errors introduced by the different modules that comprise the transceiver and not the whole picture of the system.

ASIC implementation and power consumption of the decoder

The decoder sources developed in this research work make it easy to implement them in different platforms such as ASIC. Developing for ASIC technology has the advantage that a project is not constrained by area resource utilization as much as FPGAs (in theory, area resources can grow as much as necessary in ASIC, while in FPGAs they are limited to the available logic elements in the device).

Consequently, ASIC technology is an ideal solution to further continue with the parametrical study carried out in this research work and analyze, for example, the effects of parallelization of the ACSs has on overall area resource utilization and power consumption.

References

[AA04] E. Akay and E. Ayanoglu. High performance viterbi decoder for OFDM systems. In *Vehicular Technology Conference, 2004. VTC 2004-Spring. 2004 IEEE 59th*, volume 1, pages 323–327 Vol.1, May 2004. 137

[ABAAA15] H. Abbes, M. Ben Ayed, H. Abid, and M. Abid. Design verification based on hardware-in-the-loop simulation for photovoltaic system. In *Systems, Signals Devices (SSD), 2015 12th International Multi-Conference on*, pages 1–6, March 2015. 60, 61, 63, 64

[ACS+08] F. Angarita, M.J. Canet, T. Sansaloni, J. Valls, and V. Almenar. Architectures for the implementation of a OFDM-WLAN viterbi decoder. *Journal of Signal Processing Systems*, 52(1):35–44, 2008. 32, 129

[AFC09] A. Alimohammad, S.F. Fard, and B.F. Cockburn. FPGA-based accelerator for the verification of leading-edge wireless systems. In *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, pages 844–847, July 2009. 126

[AGBL+13] J. Arias-García, A. Braga, C. H. Llanos, M. Ayala-Rincón, R. Pezzuol Jacobi, and A. Foltran. FPGA HIL simulation of a linear system block for strongly coupled system applications. In *Industrial Technology (ICIT), 2013 IEEE International Conference on*, pages 1017–1022, Feb 2013. 62, 63, 64

[ALR11] Doug Amos, Austin Lesea, and René Richter. *FPGA-Based Prototyping Methodology Manual*. Synopsys Press, 2011. 57

[AMJ+14a] Ranjit Atwal, Kanae Maita, Annette Jump, Lillian Tay, Roberta Cozza, Tuong Huy Nguyen, Bruno Lakehal, Mikako Kitagawa, Tracy Tsai, Annette Zimmermann, Anshul Gupta, Atsuro Sato, CK Lu, and William Lutman. Forecast: Pcs, ultramobiles and mobile phones, worldwide, 2011-2018, 3q14 update. *Gartner*, pages 1–12, October 2014. 2

[AMJ+14b] Ranjit Atwal, Kanae Maita, Annette Jump, Lillian Tay, Roberta Cozza, Tuong Huy Nguyen, Bruno Lakehal, Mikako Kitagawa, Tracy Tsai, Annette Zimmermann, Anshul Gupta, Atsuro Sato, CK Lu, and William Lutman. Forecast: Pcs, ultramobiles and mobile phones, worldwide, 2011-2018, 3q14 update, October 2014. Available online at `http://www.gartner.com/newsroom/id/2875017`. 2

[ASA+12] M. W. Azhar, M. Själander, H. Ali, A. Vijayashekar, T. T. Hoang, K. K. Ansari, and P. Larsson-Edefors. Viterbi accelerator for embedded processor datapaths. In *2012 IEEE 23rd International Conference on Application-Specific Systems, Architectures and Processors*, pages 133–140, July 2012. 52

[BK13] W. Byun and J. H. Kim. High-speed radix-4 add-compare-select unit for next generation communication systems. In *SoC Design Conference (ISOCC), 2013 International*, pages 1–2, Nov 2013. xv, 47, 48, 49, 163

[BKM+15] C. Barker, M. Kwiatkowska, A. Mereacre, N. Paoletti, and A. Patane. Hardware-in-the-loop simulation and energy optimization of cardiac pacemakers. In *Engineering in Medicine and Biology Society (EMBC), 2015 37th Annual International Conference of the IEEE*, pages 7188–7191, Aug 2015. 61, 62, 63, 64

[BPBS06] L. Bissi, P. Placidi, G. Baruffa, and A. Scorzoni. A multi-standard reconfigurable viterbi decoder using embedded FPGA blocks. In *Digital System Design: Architectures, Methods and Tools, 2006. DSD 2006. 9th EUROMICRO Conference on*, pages 146–154, 2006. 44, 148

[BSK+13] N.D. Bobby, S.K. Srivatsa, L. Kishore, A. Rajiv, and S.S. Suresh. Comparison of fast radix 2 ACS with adaptive fast radix 2 ACS in viterbi decoder. In *Emerging Trends in VLSI, Embedded System, Nano Electronics and Telecommunication System (ICEVENT), 2013 International Conference on*, pages 1–5, Jan 2013. 26, 42, 43, 144, 145

[BSL11a] N. D. Bobby, S. K. Srivatsa, and Lalkishore. Implementation of radix2 acs in adaptive viterbi decoder. In *Nanoscience, Engineering and Technology (ICONSET), 2011 International Conference on*, pages 604–606, Nov 2011. 42, 43, 144, 145

[BSL11b] N.D. Bobby, S.K. Srivatsa, and Lalkishore. Implementation of radix2 ACS in adaptive viterbi decoder. In *Nanoscience, Engineering and Technology (ICONSET), 2011 International Conference on*, pages 604–606, Nov 2011. 26

[CC01] K. Chadha and J. R. Cavallaro. A reconfigurable viterbi decoder architecture. In *Signals, Systems and Computers, 2001. Conference Record of the Thirty-Fifth Asilomar Conference on*, volume 1, pages 66–71 vol.1, Nov 2001. 26, 148

[CHIW98] D. J. Costello, J. Hagenauer, H. Imai, and S. B. Wicker. Applications of error-control coding. *IEEE Transactions on Information Theory*, 44(6):2531–2560, Oct 1998. 2

[Chu14] Christopher Chute. U.S. SMB Tablet Adoption 2014-2018 Forecast: Primed for Content Creation at the Edge. *IDC*, pages 1–11, July 2014. 2

[Cis14] Cisco. *White Paper Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2013–2018.* Cisco, 2014. 2

[CMS14] Melissa Chau, Tom Mainelli, and Michael Shirer. A Future Fueled by Phablets – Worldwide Phablet Shipments to Surpass Portable PCs in 2014 and Tablets by 2015, September 2014. Available online at `http://www.idc.com/getdoc.jsp?containerId=prUS25077914`. 2

[Coo04] Todor Cooklev. *Wireless Communication Standards*. Standards Information Network IEEE Press, 2004. 119

[CRBD13] D. Chakraborty, P. Raha, A. Bhattacharya, and R. Dutta. Speed optimization of a FPGA based modified viterbi decoder. In *Computer Communication and Informatics (ICCCI), 2013 International Conference on*, pages 1–6, Jan 2013. 52

[CSN14] Y. H. Chen, M. L. Su, and Y. F. Ni. FPGA implementation of trellis coded modulation decode on sdr communication system. In *Information Science, Electronics and Electrical Engineering (ISEEE), 2014 International Conference on*, volume 1, pages 89–93, April 2014. 52

[CV03] J. R. Cavallaro and M. Vaya. Viturbo: a reconfigurable architecture for viterbi and turbo decoding. In *Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03). 2003 IEEE International Conference on*, volume 2, pages II–497–500 vol.2, April 2003. 45, 148

[DAB+01] A. Doufexi, S. Armour, M. Butler, A. Nix, and David Bull. A study of the performance of HIPERLAN/2 and IEEE 802.11a physical layers. In *Vehicular Technology Conference, 2001. VTC 2001 Spring. IEEE VTS 53rd*, volume 1, pages 668–672 vol.1, 2001. 137

[Doo10] Dave Doody. *Deep Space Craft: An Overview of Interplanetary Flight*. Springer Science and Business Media, 2010. 5

[DPH93] John R. Deller, Jr., John G. Proakis, and John H. Hansen. *Discrete Time Processing of Speech Signals*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 1993. 14

[EdDLSH+13] L. A. L. Espinosa, J. d. D. L. Sánchez, J. I. N. Hipolito, M. V. Briseño, A. E. P. Ramos, and S. V. Reyes. Viterbi decoders generation for FPGA platforms. In *Mechatronics, Electronics and Automotive Engineering (ICMEAE), 2013 International Conference on*, pages 211–215, Nov 2013. 52

[EDE02] D. A. F. Ei-Dib and M. I. Elmasry. Low-power register-exchange viterbi decoder for high-speed wireless communications. In *Circuits and Systems, 2002. ISCAS 2002. IEEE International Symposium on*, volume 5, pages V–737–V–740 vol.5, 2002. 52

[EDE04] D. A. El-Dib and M. I. Elmasry. Modified register-exchange viterbi decoder for low-power wireless communications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 51(2):371–378, Feb 2004. 52

[FBEM96] G. D. Forney, L. Brown, M. V. Eyuboglu, and J. L. Moran. The V.34 high speed modem standard. *IEEE Communications Magazine*, 34(12):28–33, Dec 1996. 2

[For72] G.D. Forney. Maximum-likelihood sequence estimation of digital sequences in the presence of intersymbol interference. *Information Theory, IEEE Transactions on*, 18(3):363–378, May 1972. 14

[For73] Jr. Forney, G.D. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, March 1973. 14

[For74] G. David Forney. Convolutional codes iii. sequential decoding. *Information and Control*, 25(3):267 – 297, 1974. 35

[GXC+13] Bo Gao, Zhenyu Xiao, Zhen Chen, Depeng Jin, and Lieguang Zeng. Multigigabit balanced add-select-register-compare viterbi decoders architecture in 60 ghz WPAN. In *Communications (APCC), 2013 19th Asia-Pacific Conference on*, pages 531–535, Aug 2013. 45, 46

[HC77] F. Hemmati and Jr. Costello, D.J. Truncation error probability in viterbi decoding. *Communications, IEEE Transactions on*, 25(5):530–532, May 1977. 35

[HJ71] J. Heller and I. Jacobs. Viterbi decoding for satellite and space communication. *Communication Technology, IEEE Transactions on*, 19(5):835–848, October 1971. 31, 52

[HLS14] M. Hiller, L.R. Lima, and G. Sigl. Seesaw: An area-optimized FPGA viterbi decoder for pufs. In *Digital System Design (DSD), 2014 17th Euromicro Conference on*, pages 387–393, Aug 2014. 50, 51, 52, 146, 154, 155

[HYS14] Shaowei Huang, Yuntao Yang, and Zhenquan Sun. Add-select-delay-compare viterbi decoder for UWB communications in electronic power systems. In *Communication Problem-Solving (ICCP), 2014 IEEE International Conference on*, pages 1–4, Dec 2014. 33

[IEE99] IEEE. IEEE Standard 802.11a-1999, "Supplement to Information Technology—Telecomm. and Information Exchange between Systems—Local and Metropolitan Area Networks-Specific Requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: High Speed Physical Layer(PHY) in the 5 GHz Band". *IEEE Std 802.11a-1999*, pages 1–102, Dec 1999. 69, 113, 114

[Int11] Intel. Intel identifies chipset design error, implementing solution, January 2011. Available online at `http://newsroom.intel.com/community/intel_newsroom/blog/2011/01/31/intel-identifies-chipset-design-error-implementing-solution`. 57

[JNYK05] Yunho Jung, Seungpyo Noh, Hongil Yoon, and Jaeseok Kim. Implementation of wireless LAN baseband processor based on space-frequency OFDM transmit diversity scheme. *Consumer Electronics, IEEE Transactions on*, 51(2):393–398, May 2005. 137

[Jr.05] G. David Forney Jr. The viterbi algorithm: A personal history. *CoRR*, abs/cs/0504020, 2005. 2

[JUB+11] O. Jiménez, I. Urriza, L.A. Barragan, D. Navarro, J.I. Artigas, and O. Lucia. Hardware-in-the-loop simulation of FPGA embedded processor based controls for power electronics. In *Industrial Electronics (ISIE), 2011 IEEE International Symposium on*, pages 1517–1522, June 2011. 60, 63, 64

[KBJR14] Y. V. P. Kumar, R. Bhimasingu, M. Jyothi, and B. Ramakrishna. Real time and high fidelity controller design for hardware in the loop (HIL) testing of flight attitude control. In *Control, Instrumentation, Communication and Computational Technologies (ICCICCT), 2014 International Conference on*, pages 1217–1222, July 2014. 61

[KMPR06] A. Kokrady, R. Mehrotra, T.J. Powell, and S. Ramakrishnan. Reducing design verification cycle time through testbench redundancy. In *VLSI Design, 2006. Held jointly with 5th International Conference on Embedded Systems and Design., 19th International Conference on*, pages 6 pp.–, Jan 2006. 57

[LDZL11a] R. Li, Y. Dou, J. Zhou, and G. Lei. A high-throughput reconfigurable viterbi decoder. In *Wireless Communications and Signal Processing (WCSP), 2011 International Conference on*, pages 1–6, Nov 2011. 26, 45, 147, 148

[LDZL11b] R. Li, Y. Dou, J. Zhou, and G. Lei. A high-throughput reconfigurable viterbi decoder. In *Wireless Communications and Signal Processing (WCSP), 2011 International Conference on*, pages 1–6, Nov 2011. 52

[LGSB14] W. Li, L. A. Grégoire, S. Souvanlasy, and J. Bélanger. An FPGA-based real-time simulator for HIL testing of modular multilevel converter controller. In *2014 IEEE Energy Conversion Congress and Exposition (ECCE)*, pages 2088–2094, Sept 2014. 60, 62, 63, 64

[LL04] Jianhua Liu and Jian Li. Parameter Estimation and Error Reduction for OFDM-Based WLANs. *IEEE Transactions on Mobile Computing*, 3:152–163, 2004. 117

[LLHW08] Xiang Ling, Zhongqi Li, Jianhao Hu, and Shihong Wu. HW/SW co-simulation platforms for VLSI design. In *Circuits and Systems, 2008. APCCAS 2008. IEEE Asia Pacific Conference on*, pages 578–581, Nov 2008. 62, 63, 64

[MGJ04] K. Maharatna, E. Grass, and U. Jagdhold. A 64-point fourier transform chip for high-speed wireless LAN application using OFDM. *Solid-State Circuits, IEEE Journal of*, 39(3):484–493, March 2004. 32, 41, 67

[Min11] Yang Min. Design optimization of FPGA based viterbi decoder. In *Electric Information and Control Engineering (ICEICE), 2011 International Conference on*, pages 4129–4131, April 2011. 52

[MM15] A. J. Mandwale and A. O. Mulani. Different approaches for implementation of viterbi decoder on reconfigurable platform. In *Pervasive Computing (ICPC), 2015 International Conference on*, pages 1–4, Jan 2015. 52

[Moo65] G.E. Moore. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86(1):82–85, Jan 1965. 1

[Moo05] Todd K. Moon. *Error Correction Coding: Mathematical Methods and Algorithms*. Wiley-Interscience, 2005. 7, 16, 31, 119

[MS00] Todd K. Moon and Wynn C. Stirling. *Mathematical methods and algorithms for signal processing*. Prentice Hall, Upper Saddle River, NJ, 2000. 14

[MW96] D. W. Matolak and S. G. Wilson. Variable-complexity trellis decoding of binary convolutional codes. *IEEE Transactions on Communications*, 44(2):121–126, Feb 1996. 26

[MZMD13] B. A. Muhammad, M. A. Zanna, D. A. Mohammed, and D. Dajab Danjuma. Low complexity FPGA implementation of register exchange based viterbi decoder. In *Emerging Sustainable Technologies for Power ICT in a Developing Society (NIGERCON), 2013 IEEE International Conference on*, pages 21–25, Nov 2013. 37, 50, 51, 142, 143, 155

[NBA13] Milind Shah Nirmal Bhatt and Bhavesh Asodariya. FPGA implementation of power efficient low latency viterbi decoder. In *International Journal of Engineering Research and Technology*, pages 578–581, May 2013. 50, 51, 142, 143, 155, 157

[NG13] P. Narayanasamy and S. Gopalakrishnan. FPGA implementation of less area overhead radix4 Threshold Viterbi

decoder with trace forwarding for OFDM based cognitive radio. In *Emerging Trends and Applications in Computer Science (ICETACS), 2013 1st International Conference on*, pages 236–241, Sept 2013. 47

[OP99] Bob O'Hara and Al Petrick. *802.11 Handbook. A designer's Companion*. Standards Information Network. IEEE Press, 1999. 69, 119, 135

[OP05] Jingzhao Ou and V.K. Prasanna. MATLAB/simulink based hardware/software co-simulation for designing using FPGA configured soft processors. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pages 148b–148b, April 2005. 58, 59

[PA14a] R. V. W. Putra and T. Adiono. A configurable and low complexity hard-decision viterbi decoder in VLSI architecture. In *Information and Communication Technology (ICoICT), 2014 2nd International Conference on*, pages 182–186, May 2014. 52

[PA14b] R. V. W. Putra and T. Adiono. A configurable and low complexity hard-decision viterbi decoder in VLSI architecture. In *Information and Communication Technology (ICoICT), 2014 2nd International Conference on*, pages 182–186, May 2014. 147, 149, 155, 157

[Pad05] R. Padovani. Ten years of progress in CDMA. *Viterbi Conference, Univ. So. Calif., Los Angeles*, 2005. 2

[Par99] Keshab K. Parhi. *VLSI digital signal processing systems : design and implementation*. Wiley, New York, 1999. A Wiley-Interscience publication. 47

[PN14] A. K. Pradhan and S. K. Nandy. A reconfigurable viterbi decoder for SDR and mobile communications. In *High Performance Computing and Applications (ICHPCA), 2014 International Conference on*, pages 1–6, Dec 2014. 52, 152, 156, 157

[Roh04] Rohde & Schwarz. *802.11 Packet Error Rate Testing*, January 2004. Application Note. 69

[SI05] S.M. Shah and M. Irfan. Embedded hardware/software verification and validation using hardware-in-the-loop simulation. In *Emerging Technologies, 2005. Proceedings of the IEEE Symposium on*, pages 494–498, Sept 2005. 59

[SL97] S. Sjoholm and L. Lindh. The need for co-simulation in ASIC-verification. In *EUROMICRO 97. New Frontiers of Information Technology., Proceedings of the 23rd EUROMICRO Conference*, pages 331–335, Sept 1997. 57

[SOBM+15] H. Saad, T. Ould-Bachir, J. Mahseredjian, C. Dufour, S. Dennetière, and S. Nguefeu. Real-Time Simulation of MMCs Using CPU and FPGA. *IEEE Transactions on Power Electronics*, 30(1):259–267, Jan 2015. 60

[SRH+03] V. Singh, A. Root, E. Hemphill, N. Shirazi, and J. Hwang. Accelerating Bit Error Rate Testing Using a System Level Design Tool. *Proceedings of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2003. 126

[SSV14] N.V. Sugur, S.V. Siddamal, and S.S. Vemala. Design and implementation of high throughput and area efficient hard decision viterbi decoder in 65nm technology. In *VLSI Design and 2014 13th International Conference on Embedded Systems, 2014 27th International Conference on*, pages 353–358, Jan 2014. 46, 47

[STGB02a] Sriram Swaminathan, Russell Tessier, Dennis Goeckel, and Wayne Burleson. A dynamically reconfigurable adaptive viterbi decoder. In *Proceedings of the 2002 ACM/SIGDA Tenth International Symposium on Field-programmable Gate Arrays*, FPGA '02, pages 227–236, New York, NY, USA, 2002. ACM. 26

[STGB02b] Sriram Swaminathan, Russell Tessier, Dennis Goeckel, and Wayne Burleson. A dynamically reconfigurable adaptive viterbi decoder. In *Proceedings of the 2002 ACM/SIGDA Tenth International Symposium on Field-programmable Gate Arrays*, FPGA '02, pages 227–236, New York, NY, USA, 2002. ACM. 26, 44, 45, 148

[SV13] P. Singh and S. K. Vishvakarma. RTL level implementation of high speed-low power viterbi encoder amp; decoder. In *Information Science and Technology (ICIST), 2013 International Conference on*, pages 345–349, March 2013. 46, 50, 51, 70, 147, 155, 157

[SWJS15] K. Sengchuai, W. Wichakool, N. Jindapetch, and P. Smithmaitrie. FPGA-based hardware-in-the-loop verification of dual-stage HDD head position control. In *Micro and Nanoelectronics (RSM), 2015 IEEE Regional Symposium on*, pages 1–4, Aug 2015. 61, 63, 64

[TCW05] Wei-Hsiang Tseng, Ching-Chi Chang, and Chorng-Kuang Wang. Digital VLSI OFDM transceiver architecture for wireless SoC design. In *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, pages 5794–5797 Vol. 6, May 2005. 137

[TNHN99] K. Tsukano, T. Nishiya, T. Hirai, and T. Nara. Simplified EEPR viterbi detector based on a transformed radix-4 trellis for a disk drive. *IEEE Transactions on Magnetics*, 35(5):4387–4401, Sep 1999. 47

[Tro05] M. G. Troulis. A low complexity, fixed point channel estimator for 802.11a transceivers. In *Global Telecommunications Conference, 2005. GLOBECOM '05. IEEE*, volume 4, pages 5 pp.–2223, Dec 2005. 137

[TSR+05a] R. Tessier, S. Swaminathan, R. Ramaswamy, D. Goeckel, and W. Burleson. A reconfigurable, power-efficient adaptive viterbi decoder. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 13(4):484–488, April 2005. 26

[TSR+05b] R. Tessier, S. Swaminathan, R. Ramaswamy, D. Goeckel, and W. Burleson. A reconfigurable, power-efficient adaptive viterbi decoder. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 13(4):484–488, April 2005. 44, 45, 148

[Ver84] S. Verdu. *Optimum multi-user signal detection*. PhD thesis, University of Illinois, 1984. 14

[Vit67] Andrew James Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory, IEEE Transactions on*, 13(2):260–269, April 1967. 5, 14

[VNS12a] M. Véstias, H. Neto, and H. Sarmento. Design of High-Speed Viterbi Decoders on Virtex-6 FPGAs. In *Digital System Design (DSD), 2012 15th Euromicro Conference on*, pages 938–945, Sept 2012. xv, 49, 50

[VNS12b] M. Véstias, H. Neto, and H. Sarmento. Sliding block viterbi decoders in FPGA. In *22nd International Conference on Field Programmable Logic and Applications (FPL)*, pages 595–598, Aug 2012. 47, 153, 154, 156

[VPG13] M. Veshala, T. Padmaja, and K. Ghanta. FPGA based design and implementation of modified viterbi decoder for a Wi-Fi receiver. In *Information Communication Technologies (ICT), 2013 IEEE Conference on*, pages 525–529, April 2013. 26

[VS12] M. Véstias and H. Sarmento. Tradeoffs in the design of sliding block Viterbi decoders for MB-OFDM UWB systems. In *Consumer Electronics - Berlin (ICCE-Berlin), 2012 IEEE International Conference on*, pages 173–177, Sept 2012. 153

[Wic95] Stephen B. Wicker. *Error Control Systems for Digital Communication and Storage*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1995. 35

[Xil09] Xilinx. *Spartan-3A DSP Starter Platform User Guidew*, January 2009. User Guide. 127

[Xil11a] Xilinx. *System Generator for DSP. User Guide*. Xilinx, 2011. 124

[Xil11b] Xilinx. *XA Spartan-3A Automotive FPGA Family Data Sheet*, April 2011. User Guide. 142

[Xil11c] Xilinx Inc. *LogiCORE IP Viterbi Decoder v7.0*, March 2011. Product Specification. xvi, 46, 53, 55, 56, 65, 143, 150, 151, 156, 157, 160

[Xil15] Xilinx. *7 Series FPGAs Overview*, May 2015. Product Specification. 70

[XJMCZYD04] Qin Xiang-Ju, Zhu Ming-Cheng, Wei Zhong-Yi, and Chao Du. An adaptive viterbi decoder based on FPGA dynamic reconfiguration technology. In *Field-Programmable Technology, 2004. Proceedings. 2004 IEEE International Conference on*, pages 315–318, Dec 2004. 52

[YPAB14] N. Yousefpoor, B. Parkhideh, A. Azidehak, and S. Bhattacharya. Convertible static transmission controller (CSTC) system model validation by controller hardware-in-the-loop-simulation. In *2014 IEEE Energy Conversion Congress and Exposition (ECCE)*, pages 2960–2966, Sept 2014. 59, 61

[ZB03] Y. Zhu and M. Benaissa. Reconfigurable viterbi decoding using a new ACS pipelining technique. In *Application-Specific Systems, Architectures, and Processors, 2003. Proceedings. IEEE International Conference on*, pages 360–368, June 2003. 44, 148

APPENDIX A

Publications

Contents

Papers in conferences and journals results of the research time of the author are included in chronological order in this appendix.

A.1 INTERNATIONAL CONFERENCE PAPERS

- A. Alonso, A.Irizar, J.R. Martín, I. Vélez, A. Cortés and N. Arrue. Hardware Implementation of a Low-Complexity Synchronizer for a WLAN 802.11a Transceiver. *XXIV Conference on Design of Circuits and Integrated Systems*, Zaragoza, November 2009.

- A. Alonso, A. Irizar, A. Cortés and I. Vélez. Impact of algorithms over the PER of a WLAN 802.11a based transceiver. *XXV Conference on Design of Circuits and Integrated Systems*, Lanzarote, November 2010.

- A. Alonso, J.F. Sevillano and I. Vélez. Parallel Implementation of a Sample Rate Conversion and Pulse-Shaping Filter for High Speed Backhauling Networks. *XXIX Conference on Design of Circuits and Integrated Systems*, Madrid, November 2014.

- A. Alonso and A. Irizar. Fast hardware-in-the-loop verification platform: a case study for convolutional decoders. *XXIX Conference on Design of Circuits and Integrated Systems*, Madrid, November 2014.

- A. Rezola, A. Alonso, J.F. Sevillano, I. Gurutzeaga, R. Berenguer and I. Vélez. Implementation of a Zero-Second-IF Transmitter for Wide-Band Millimeter-Wave Links. *XXX Conference on Design of Circuits and Integrated Systems*, Estoril, Nov 2015

Hardware Implementation of a Low-Complexity Synchronizer for a WLAN 802.11a Transceiver
A. Alonso, A.Irizar, J.R. Martín, I. Vélez, A. Cortés and N. Arrue
XXIV Conference on Design of Circuits and Integrated Systems, Zaragoza, Nov 2009

Hardware Implementation of a Low-Complexity Synchronizer for a WLAN 802.11a Transceiver

Aritz Alonso, Andoni Irizar, Jose Ramón Martín, Igone Vélez, Ainhoa Cortés, Naiara Arrue
Department of Electronic & Communications
CEIT and Tecnun (University of Navarra)
Manuel de Lardizábal 15, 20018 San Sabastián, Spain
Email: {aadomingo, airizar, jrmartin, ivelez, acortes, narrue}@ceit.es

***Abstract*— In this paper we are presenting a hardware implementation of a synchronizer for a WLAN 802.11a compliant transceiver based on the algorithm proposed by J. Liu and J. Li. The architecture of the synchronizer has been designed with simplicity in mind and to reuse as many blocks as possible from the existing transceiver. The synchronizer has been described in VHDL.**

I. Introduction and IEEE 802.11a overview

IEEE WLAN 802.11 standards define the physical layer of communication systems based on OFDM. This set of standards have reached maturity over the years and are integrated in many common devices such as laptops, smart phones and gaming consoles.

The first revision of the standard, WLAN 802.11a, works in the band of 5 GHz and comprises data rates from 6 Mbps to 54 Mbps. IEEE WLAN 802.11 continues evolving nowadays with new drafts and proposals such as 802.11n and 802.11r that permit, respectively, theoretical data rates of 600 Mbps and the ability to perform hand overs between access points.

WLAN 802.11a [1] defines 12 channels of 20 MHz. In each channel an OFDM modulation scheme with 64 carriers is used. 48 out of the 64 carriers are used for data transmission and 4 carriers are used as pilot tones. A single OFDM symbol has a duration of 4μs. A guard interval of 0.8μs is cyclically extended to each OFDM symbol for robustness against multipath. WLAN 802.11a uses PPDU frames to send and receive data. PPDU frames have three fields: PLCP preamble, signal and data. Figure 1 shows a typical PPDU frame.

OFDM based communication systems are known for their sensitivity to frequency offsets. In [2] Pollet, Bladel and Moeneclaey suggest that the degration of the BER for OFDM signals due to small frequency offsets between the transmiter and the receiver with respect to the subcarrier spacing is aproximately

$$D \cong \frac{10}{3ln10}(\pi \triangle FT)^2 \frac{E_S}{N_o} \quad (1)$$

where T is the period of the OFDM symbol, $\triangle F$ is the frequency offset between the transmiter and the receiver and N_o/E_S is the variance of the noise.

From equation (1) is readily seen that high accuracy clocks are required to keep BER degradation to a minimum. Since the cost of this kind of clocks is high signal processing of the FFT input is performed to reduce the impact of the frequency offsets.

In this paper we present a hardware implementation of a synchronizer for a WLAN 802.11a compliant transceiver and its synthesis and place and route results for a Xilinx Spartan-3A DSP 3400A FPGA.

II. Synchronization Algorithm

Bibliography on synchronization for WLAN 802.11a is extensive. However, many available algorithms are based on auto-correlation and cross-correlation techniques that use the PLCP preamble, as first mentioned in [3].

The PLCP preamble consists of two training sequences. In the first sequence a repetition of ten short symbols is made while the second training sequence replicates two long training symbols. A short training symbol has a length of 16 samples while a long training symbol has a length of 64 samples. In the transition between training sequences a cyclical extension of the last 32 samples of a long training symbol is inserted. Therefore, the PLCP preamble has a length of 320 samples.

Many of the proposed algorithms estimate an specific parameter, such as the frequency offset ([4], [5], [6], [7]), the symbol timing ([8], [9], [10], [11]) and the channel response ([12]).

The hardware implementation of the synchronizer we present is based on the algorithm proposed by J. Liu and J. Li [13]. This algorithm has been chosen because of its good performance, the simplicity of its operations and because it is capable of estimating the time and frequency offsets and the channel response sequentially.

The frequency offset estimation compensates the mismatch between the transmiter and the receiver sampling clock frequencies and the time offset estimation validates the data samples in the PPDU after the first long training symbol. Each parameter is obtained after a coarse and a fine estimation. Data samples between stages are corrected with the partial estimations. The estimations are obtained in the following order:

A. Coarse Frequency Estimation

Let ϵ be the frequency offset between the transmiter and the receiver normalized with respect to the sampling frequency. The short symbol training sequence has a periodicity of $N_S =$

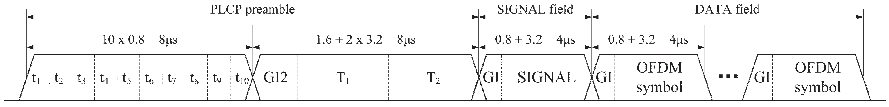


Fig. 1. PPDU

16 samples, so for every data sample $z(l)$ before the last short training symbol we have

$$z(l+N_S) = z(l)e^{j2\pi\epsilon N_S} \tag{2}$$

The correlation P_S of two consecutive short training symbols in the abscense of noise can be written as

$$\begin{aligned} P_S &= \sum_{l=k}^{k+N_S-1} z(l)(z(l+N_S))^* \\ &= e^{-j2\pi\epsilon N_S} \sum_{l=k}^{k+N_S-1} |z(l)|^2 = Pe^{-j2\pi\epsilon N_S} \end{aligned} \tag{3}$$

In the presence of noise we have that

$$P_S = Pe^{-j2\pi\epsilon N_S} + e_n \tag{4}$$

where e_n is due to the noise. The coarse frequency offset estimation $\hat{e}_c$ can then be written as

$$\hat{e}_c = -\frac{1}{2\pi N_S}\angle P_S \tag{5}$$

This parameter can be used to correct the incoming data samples as follows:

$$z_{ec}(l) = z(l)e^{-j2\pi l\hat{e}_c} \tag{6}$$

B. Coarse Timing Estimation

This stage determines the transition from the short training symbols to the guard interval of the long training symbols. A real-valued correlation sequence P_R is generated as follows

$$\begin{aligned} P_R(v+1) &= P_R(v) + Re\{z_{ec}(v+N_S)z^*_{ec}(v+2N_S) \\ &\quad - z_{ec}(v)z^*_{ec}(v+N_S)\} \end{aligned} \tag{7}$$

with $P_R(v=0) = Re\{P_S\}$.

Equation (2) guarantees that the value of P_R will maintain constant, with fluctuations depending on the noise level, as long as the two sliding windows $z(v+N_S)$ and $z(v+2N_S)$ contain data samples of the short training sequence. Once the first sliding window $z(l+2N_S)$ starts to store samples from the long symbols' guard interval the level of P_R drops quickly. In an optimum scenario in the absence of noise P_R will drop to half its maximum level when the first sliding window has half of its registers with data from the guard interval samples.

Let T_P be the first sampling index for which P_R has dropped below half of its maximum value. The coarse timing estimation T_C is meant to point to the middle of the long symbols' guard interval and is calculated as follows:

$$T_C = T_P + \frac{3}{2}N_S + N_S \tag{8}$$

C. Fine Frequency Estimation

The correlation P_L of two consecutive long training symbols of length $N_L = 64$ samples can be written as

$$P_L = \sum_{l=T_C}^{T_C+N_L-1} z_{ec}(l+T_C)z^*_{ec}(l+T_C+N_S) \tag{9}$$

Similar to (5), the fine frequency offset estimation $\hat{e}_f$ can be written as

$$\hat{e}_f = -\frac{1}{2N_L\pi}\angle P_L \tag{10}$$

and the incoming data samples are corrected using

$$z_{ef}(l) = z_{ec}(l)e^{-j2\pi l\hat{e}_f} \tag{11}$$

D. Fine Timing Estimation

This last stage correlates the received data stream in the frequency domain with the data of a long training symbol. The resulting output will present peaks in the data samples where a long training symbol begins.

A small set $\{y[k]\}$ of 64 data samples is only needed during this estimation. $\{y[k]\}$ is defined as follows

$$\{y[k]\} = \{z_{ef}(l)\}_{l=T_C+3N_S}^{T_C+3N_S+N_L-1}$$

The indexes have been chosen to improve the probability of $\{y[h]\}$ having samples of both the first and the second long training symbols.

The correlation in the frequency domain $h[n]$ between $\{y[k]\}$ and a long training symbol is expressed as:

$$h[n] = IFFT\{X[k]^*Y[k]\}_{64} \tag{12}$$

where $\{Y[k]\}$ is the Fourier Transform of $\{y[k]\}$ and $\{X[k]\}$ are the data samples of a long training symbol.

Since $\{X[k]\}$ is a real sequence, $\{X[k]^*\} = \{X[k]\}$. Therefore, equation (12) can be seen as the channel response. In absence of noise, $h[n]$ is a real sequence. Noise and sampling clock frequency offsets lead the channel response being complex, so the modulus of $h[n]$ is used for timing estimation.

Let T_I be the first element of $|h[n]|$ that it is above one third of its maximum. The fine timing estimation T_F can be expressed as:

$$T_F = T_C + T_I - N_S - 3 \quad (13)$$

where the third term compensates the delay of N_S samples introduced in (8) and the last term minimizes the probability of T_C falling after the first long training symbol.

III. Synchronizer Architecture

Figure 2 shows the BBP (baseband processor) transceiver for WLAN 802.11a designed in our department. Solid lines represent data paths while dashed lines represent control paths. Data coming from or going to the converters has a length of 12 bits with a 2.10 format for both the real and imaginary components of the signal at a rate of 20 Msps. The BBP transceiver is clocked at 60 MHz. The FFT/IFFT core is shared between the transmission and reception chains in order to reduce area.

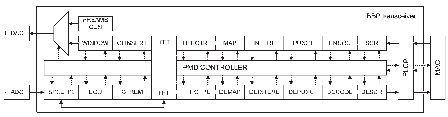


Fig. 2. Block diagraman of the WLAN 802.11a compliant transceiver

As mentioned in section II-D, fine timing estimation output $h[n]$ can be used for channel equalization. However, our BBP transceiver included an equalizer so the synchronizer architecture we propose does not include this functionality.

The correlation in the frequency domain requires the calculation of both direct and inverse Fourier transforms. The statistical properties of data input to the FFT core are different when calculating correlations or sending or receiving OFDM symbols. The FFT core used in our BBP transceiver was originally designed only for OFDM symbol transmision and reception. Therefore, several tests have been run to the FFT core to ensure it can correlate data sequences correctly.

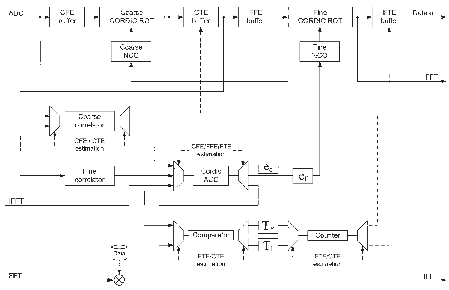


Fig. 3. Block diagraman of the synchronizer

Figure 3 shows the proposed architecture for our synchronizer. Two correlators have been designed to perform the coarse estimations and the fine frequency estimation. The *coarse correlator* calculates the value P_S and the sequence $P_R(v)$ from (4) and (7). The *fine correlator* calculates the value P_L from (9). Both correlators share a similar architecture: they only differ in the size of their sliding windows and the fact that the *coarse correlator* has an extra sliding window to correctly compute the sequence $P_R(v)$.

Trigonometric operations have been implemented with two different Cordic (COrdinate Rotation DIgital Computer) cores: Cordic in *rotation configuration* or Cordic ROT and Cordic in *vector mode* or Cordic ACC.

Cordic ROTs can rotate a complex input word a certain phase. They are used to calculate values $z_{ec}(l)$ and $z_{ef}(l)$ from equations (6) and (11). The sequential nature of the synchronization algorithm forces to use two Cordic ROT entities in our design.

Cordic ACCs rotate a complex input word until its phase equals to zero. Therefore they can be used to calculate a complex input word's phase and magnitude. Since only one synchronization parameter is estimated at a time a single Cordic ACC unit can be used in the architecture. This unit calculates the angles of P_S and P_L in equations (5) and (10) and the magnitude of the channel response $h[n]$ as described in (12). Preliminary simulations on Matlab have shown that the only synchronization parameter that is sensitive to phase quantization is the mean frequency error after synchronization. As shown in figure 4 the error in the frequency estimation is reduced by a factor of two for every extra bit in the expresions of the angles of P_S and P_L. The precission of all the Cordic units has been decided to be kept at 18 bits as it guarantees an average error of 50 Hz while it enables to use the internal 18 by 18 bit hardware multipliers of the target FPGA.

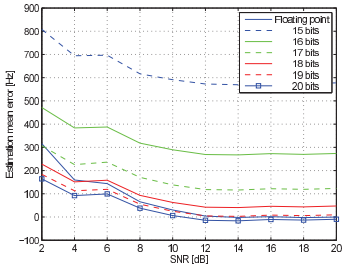


Fig. 4. Effects of the phase quantization in the frequency estimation

The comparator detects the sampling index where an incoming data sequence has fallen to a programable factor of its maximum value. This sampling index is connected with the values T_P and T_I in (8) and (13). These estimations set a counter that validates the data stored in the temporary CTE and FTE buffers.

TABLE I
MAXIMUM TIME DELAY INTRODUCED IN EACH SYNCHRONIZATION STAGE

Synchronization stage	Associated parameter	Maximum completion time [clock cycles]
CFE	$\hat{e}_c$	38
CTE	T_P	193
FFE	$\hat{e}_f$	134
FTE	T_I	584

An analysis of the time delay introduced by each synchronization stage has been performed. This analysis has been performed in the worst scenario possible where the digital converters send the whole PPDU frame to the synchronizer. Table I sumarizes the conclussions of this analysis.

IV. Simulation and Synthesis

The VHDL implementation of the synchronizer has been simulated and its performance has been compared with the output of a Matlab floating point model. The input for each simulation consists of the PLCP preamble field of the PPDU degraded with AWGN and frequency and time offsets.

The frequency offset f_{offset} has been chosen randomly from a uniform distribution between ± 232 kHz. The time offset t_{offset} sample index has been chosen from 1 to 121 in steps of 30 samples. The effects of the time and frequency offsets can be modeled as follows:

$$z_{input}(l) = \begin{cases} 0 & l < t_{offset} \\ z(l)e^{j2\pi l f_{offset}}(l) & l \geq t_{offset} \end{cases}$$

Finally the PLCP preamble is degraded with AWGN. The SNR of the resulting signal has been set from 2 to 20 dBs in steps of 2 dBs.

For each [SNR - t_{offset}] pair 10^4 simulations have been completed. Figures form 5 to 8 show the performance of the synchronizer averaged with respect of all the values of t_{offset}.

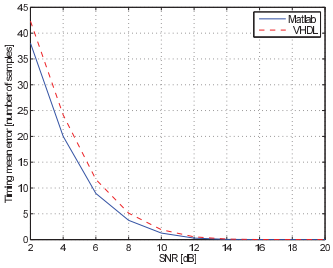


Fig. 5. Timing estimation mean error

As it can be seen in figure 5 our synchronizer estimates correctly the beginng of the first long training symbol for SNR values above 12 dB. However, the inclusion of a cyclical extension of $N_S = 16$ samples on each OFDM symbol allows to accept timing estimations that have a maximum error of 16 samples. If T_F falls between 177 and 193 the resulting inputs to the FFT become a cyclical shift of the perfectly timed data sequence. Under this condition the output of the FFT could be written as follows:

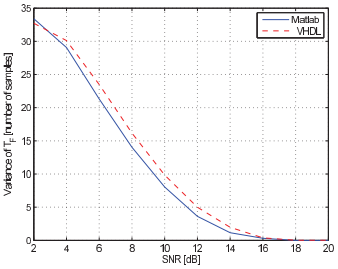


Fig. 6. Variance of the timing estimation

$$X_{mismatch}[k] = e^{-j2\pi k(193-T_F)/64} X[k] \qquad (14)$$

where X[k] is the espected output of the FFT with perfect synchronization. The quantity $e^{-j2\pi k(193-T_F)/64}$ is a constant complex magnitude that the equalizer can easily compensate.

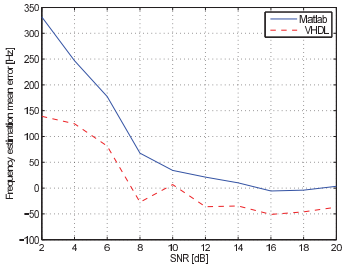


Fig. 7. Frequency offset estimation mean error

The frequency offset estimation presents an average error of 50 Hz at high values of SNR. As it was mentioned in section III this error is due to the Cordic cores having been limited to a 18 bit resolution.

Figure 8 shows that the frequency estimations have a considerable deviation at low levels of SNR. This is due to the noise inducing wrong T_C estimation and therefore, the fine frequency estimation stage uses incorrectly both short and long training symbols to estimate $\hat{e}_f$.

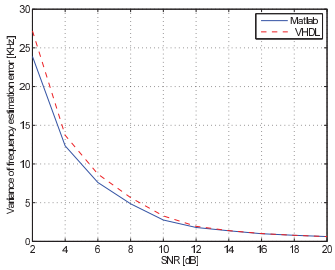


Fig. 8. Variance of the frequency offset estimation error

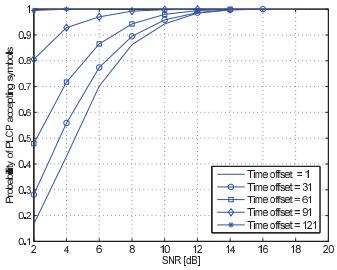


Fig. 10. Influence of the original time offset over the synchronized symbol acceptance probability

The PLCP module accepts the PPDUs whose SIGNAL symbol has been correctly decoded. As mentioned before, timing errors above 16 samples makes decoding the PPDU impossible. Figure 9 shows the probability of the PLCP accepting the output of the synchronizer.

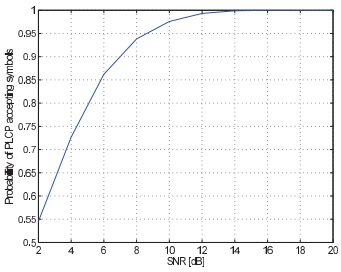


Fig. 9. Probability of the PLCP accepting symbols after synchronization

As it can be seen, the probability of accepting PPDUs is above 90% for values of SNR above 7 dB.

Figure 10 shows the influence that the original time offset has on the PLCP accepting the output of the synchronizer.

It is clearly seen that higher time offsets improve drastically the probability of the PLCP accepting synchronized PPDUs at low levels of SNR.

Finally, the frequency estimation performance has been compared at the outputs of the synchronizer and the PLCP.

As it can be seen in figures 11 and 12 the symbols at the output of the PLCP improve their frequency estimation since the PLCP rejects the PPDUs with incorrect timing estimations.

The VHDL description of the synchronizer has been synthesized on a Xilinx Spartan-3A DSP 3400A FPGA using high effort speed optimization. Table II summarizes the resources consumed. Note that this implementation includes extra logic

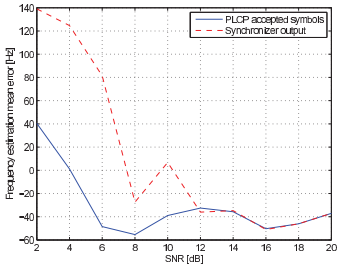


Fig. 11. Comparison of the frequency estimation mean error of the symbols at the output of the synchronizer and the PLCP

to correctly empty the internal buffers of the synchronizer once the PPDU has been received.

The device reaches a maximum clock speed of 63.784 MHz and power estimation tools have shown that it requires a dynamic power of 98mW to operate on the mentioned FPGA.

V. Conclusions

In this paper we have presented the hardware implementation for a WLAN 802.11a compliant synchronizer. First, the physical layer of WLAN 802.11a has been overviewed and the synchronization algorithm proposed by J.Liu and J. Li has been presented. The algorithm has been chosen because of its good performance, for giving a means of simple hardware

TABLE II
Resource consumption of the implementation

	Consumed resources	Available resources
Slices	4656	23872
Block RAM	9	126
Dedicated multipliers	10	126

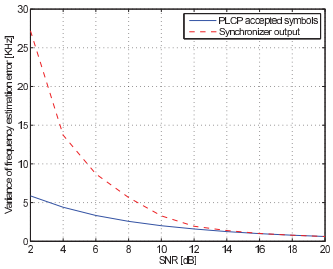


Fig. 12. Comparison of the frequency estimation variance of the symbols at the output of the synchronizer and the PLCP

implementation and because of the posibility of reusing previously existing modules in the WLAN transceiver so that the hardware resource consumption is kept to a minimum. Then the proposed hardware architecture of the synchronizer has been described. Finally the simulation, synthesis and place and route results have been reported.

The synchronizer we have presented produces valid timing estimations for SNR ≥ 6 dB and the average frequency offset after synchronization is of just 50 Hz for SNR ≥ 8 dB. On a Xilinx Spartan-3A DSP 3400A FPGA the hardware implementation of the synchronizer requires 4656 slices and reaches a maximum clock speed of 63.784 MHz.

The synchronizer is meant to become an IP core and its interface has been studied so that it is easily interchangeable with other modules implementing different synchronization algorithms. This fact will provide an easy way to further study on synchronization algorithms for OFDM based systems.

REFERENCES

[1] *IEEE Standard 802.11a-1999, "Supplement to Information Technology—Telecomm. and Information Exchange between Systems—Local and Metropolitan Are Networks-Specific Requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer(PHY) Specifications: High Speed Physical Layer(PHY) in the 5 GHz Band"*.

[2] M. v. B. Pollet. T. and M. Moeneclaey, "Ber Sensitivity of OFDM Systems to Carrier Frequency Offset and Wiener Phase Noise," *IEEE Transactions on COmmunications, Vol. 43, No. 2/3/4, pp. 191-193*, 1995.

[3] D. U. Fort A. Weijers JW, "A performance and complexity comparison of auto-correlation and cross-correlation for OFDM burst synchronization," *International Conference on Acoustics Speech and Signal Processing (ICASSP), pp. 341-344*, 2003.

[4] P. Moore, "A Technique for Orthogonal Frequency Division Multiplexing Frequency Offset Correction," *IEEE Transactions on Communications, Vol. 42, pp. 2908-2914*, October 1994.

[5] U. M. M. Morelli, "An Improved Frequency Offset Estimator for OFDM Applications," *IEEE Communication Letters, Vol. 3, pp. 75-77*, March 1999.

[6] I. S. Y.H. Kim, "An efficient Frequency Offset Estimator for OFDM Systems and Its Performance Characteristics," *IEEE Transactions on Vehicular Technology, Vol. 50, pp. 1307-1312*, September 2001.

[7] G. G. J. Li, G. Liu, "Carrier Frequency Offset Estimation for OFDM Based WLANs," *IEEE Signal Processing Letters, Vol. 8, pp. 80-82*, March 2001.

[8] C. J. Gadhiok M, "Preamble-based symbol timing estimation for wireless OFDM systems," *CONFERENCE RECORD OF THE FORTY-FIRST ASILOMAR CONFERENCE ON SIGNALS, SYSTEMS & COMPUTERS, pp. 1791-1794*, 2007.

[9] L. G. Wen JH, Lee SH, "Timing and delay spread estimation scheme in OFDM systems," *IEEE TRANSACTIONS ON CONSUMER ELECTRONICS, Vol.54, pp. 316-320*, May 2008.

[10] K. L. B. Yang, "Timing Recovery for OFDM Transmission," *IEEE Journal on Selected Areas in Communications, Vol. 18, pp. 2278-2291*, November 2000.

[11] C. J. Gadhiok M, "A new symbol timing synchronization design for OFDM WLAN systems," *10th International Conference on Communication Technology, pp. 767-770*, 2006.

[12] J. C. R. Neigi, "Pilot Tone Selection for Channel Estimation in a Mobile OFDM System," *IEEE Transactions on Consumer Electronics, Vol. 44, pp. 1122-1128*, August 1998.

[13] J. Liu and J. Li, "Parameter Estimation and Error Reduction for OFDM-Based WLANs," *IEEE Transactions on Mobile Computing*, vol. 3, pp. 152–163, 2004.

Impact of algorithms over the PER of a WLAN 802.11a based transceiver
A. Alonso, A. Irizar, A. Cortés and I. Vélez
XXV Conference on Design of Circuits and Integrated Systems, Lanzarote, Nov 2010

Impact of algorithms over the PER of a WLAN 802.11a based transceiver

Aritz Alonso, Andoni Irizar, Ainhoa Cortés, Igone Vélez
Department of Electronic & Communications
CEIT and Tecnun (University of Navarra)
Manuel de Lardizábal 15, 20018 San Sebastián, Spain
Email: {aadomingo, airizar, acortes, ivelez}@ceit.es

***Abstract*— Simulations performed over a WLAN 802.11a compliant transceiver developed on our department has lead to identify some sources of error that were not considered in the original model and that reduce considerably the system's performance, packet error rate (PER) being the most significant figure of merit. In this paper we present and analyze hardware architectures that compensate the influence of this new sources of error.**

I. Introduction and IEEE 802.11a overview

In the last couple of years the IEEE WLAN 802.11 family of standards has become a huge success in the deployment of high-speed wireless communications. IEEE 802.11 deals with the data and physical layers of an OSI based communication system. The most notable entities defined by the standards are the MAC (*Medium Access Control*), which encapsulates the data link layer functionality, and the PMD (*Physical Medium Dependence*), which defines and provides the means to transmit and receive data in the wireless channel. Most of the rate improvements that newer IEEE 802.11 standards achieve are due to changes performed over the PMD, so a PLCP (*Physical Layer Convergence Procedure*) is defined to give the MAC further independence over the PMD implementation.

WLAN 802.11a [1] operates in the band of 5 GHz and uses OFDM for data modulation. Data is transmited in packets known as PPDUs or *PLCP Protocol Data Units*. Each PPDU consists of a known training sequence for signal detection, channel estimation and frequency and time offset correction, a SIGNAL symbol which carriers information necessary for correct signal demodulation and DATA symbols where the transmited information is coded. Each OFDM symbol consists of 64 subcarriers, 48 being data carriers and 4 being pilot tones. WLAN 802.11a reaches transmission speeds of up to 54 Mbps and many of its functionality has been the base for the newly adopted 802.11n standard [2].

Figure 1 shows the architecture of the WLAN 802.11a compliant base band processor that has been developed on our department. As it can be seen the FFT, interleaving and scrambling cores are shared among the transmission and reception chains to reduce the area consumption.

II. Sources of error in reception

The baseband processor introduced in section I offers basic mechanisms to compensate the impairment of the received signal due to the transmission channel:

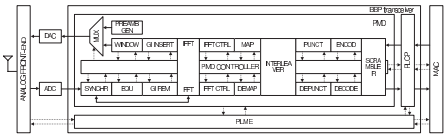


Fig. 1. Architecture of the WLAN 802.11a compliant transceiver

- A synchronizer estimates the begining of the PPDU and compensates the frequency offset between the transmiter and receiver sampling clocks.
- An equalizer flattens the frequency response of the transmission channel.
- A Viterbi Decoder decodes the convolutionaly coded bit stream and provides FEC (Forward Error Correction).

These mechanisms have proven to be insufficient in order to obtain a reliable receiver. At its simplest configuration, the base band processor requires considerable SNR values to reach PERs of about 10%.

In the following subsections we will analyze our processor's architecture to find potential flaws that decrease its performance

A. Residual frequency offset after synchronization

The synchronizer implemented in our transceiver is based on the algorithms proposed by J.Li and J.Liu for time and frequency offset estimation [3]. Figure 2 shows the frequency estimation error our synchronizer achieves for various levels of SNR. As is can be seen, the effects of the frequency offset can be mitigated but not completely compensated. As a result, even after synchronization the constellation points will continue to rotate causing decoding errors. In each OFDM symbol this rotation can be seen as a phase offset.

In order to overcome this limitation a mechanism to track the phase offset accumulated in each OFDM symbol must be provided. The same authors [3] suggest an algorithm based on the pilot tones of the OFDM symbol to estimate and compensate the accumulated phase offset on each OFDM symbol.

Let N_P be the number of pilot tones in an OFDM symbol, $\hat{h}$ a vector of length N_P containing the channel estimations

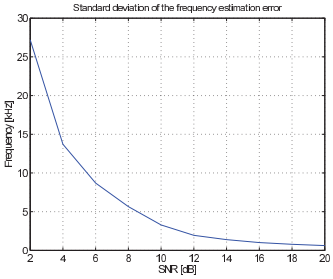


Fig. 2. Frequency estimation error after synchronization

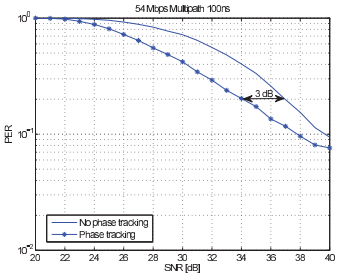


Fig. 3. Improvement in the PER due to the inclusion of the phase tracking algorithm

for the pilot tones, y_k a vector of length N_P containing the received pilot tones for the kth symbol and P_k a vector of length N_P that constains the data transmited on the pilot tones for the kth symbol. The accumulated phase offset on the kth OFDM symbol ϕ_k can be estimated with

$$\widehat{\phi_k} = \angle \left\{ \sum_{i=1}^{N_p} \hat{h}_i^* P_{i,k}^* y_{i,k} \right\} \quad (1)$$

The parameter $\widehat{\phi_k}$ can be used to compensate the accumulated phase offset in the time domain or in the frequency domain by calculating

$$z_k = y_k e^{-j \cdot \widehat{\phi_k}} \quad (2)$$

where y_k is the incoming time or frequential data sequence corresponding to the kth OFDM symbol and z_k is the compensated output data sequence.

Figure 3 shows the simulation results of the impact the phase tracking algorithm has over our transceiver for the fastest transmission mode. However, the impact of the phase tracking algorithm is more significant on lower rate transmission modes: faster transmission modes use more complex constellations that requiere higher SNR levels to correctly operate. As a result the synchronizer will provide more accurate frequency estimations on these faster modes and thus the rotation of the constellation due to the frequency mismatch will be reduced. Therefore, the inclussion of the phase tracking algorithm suposes a greater improvement at 6Mbps transmission modes.

B. Multipath channels

So far now the transceiver has been tested over AWGN channels. This channels show a flat fading pattern, that is, the spectral characteristics of the transmitted signal are preserved at the receiver [4]. Wireless communications are known however for taking place in multipath environments. Under this circumstances the received signal can be seen as the addition of several copies of the transmitted signal faded and delayed in time. Time dispersion induces frequency selectivity: some carriers have greater gains than others, and as a result, the channel provoques intersymbol interference (ISI).

The multipath channel model implemented on our models is identical to the one adopted by the IEEE 802.11 Working Group [5] [6]. Some authors [7] [8] show that the knowledge of the gains the channel induces to each of subcarrier in the OFDM symbol improves the performance of the Viterbi decoder. This technique, known as *Channel State Information* (CSI), basically adjusts the metrics at the BMU in the Viterbi Decoder so that it takes into account the SNR relative to the carrier the soft-bit was demaped from. As a result, the contribution to the final metric of high SNR subcarriers is much greater than that of low SNR subcarriers.

Figure 4 shows the benefits of including CSI aware Viterbi decoding in our transceiver under multipath channels with different values of *delay spread*.

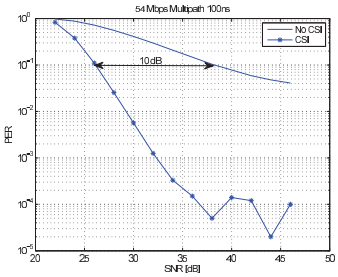


Fig. 4. Improvement in the PER due to the inclusion of the CSI

C. Demapping algorithm

At present our transceiver uses a 3-bit soft-demaper at the output of the equalizer. The demapping algorithm measures the distance between the received coefficient and the decission

threshold between logical '0's and '1's and returns a weighted soft-bit. The implementation is mainly based on shift-registers to keep its complexity to a minimum, but the inclusion of multipath channels leads us to consider more efficient demapping algorithms.

Simulations have been run to study the dispersion data carriers suffer from multipath channles. Figure 5 shows the simulation set-up that has been used. As it can be seen the variables that have been taken into acount have been the *delay spread* of the channel and its SNR level.

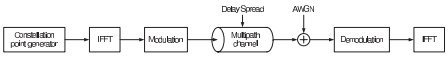


Fig. 5. Simulation set-up to analyze coeficient dispersion

After analyzing the statistical properties of the input and output of the simulation a new family of demapping functions has been defined. These demapping functions return the probability of a soft-bit being a logical '1' for an input coefficient x_{in} and a channel configuration c_{chan}. Mathematically this can be writeen as

$$f_{demap}(x_{in}, c_{chan}) = p(bit_{out} = \text{'1'} | x_{in}, c_{chan}) \quad (3)$$

As an example, figure 6 shows different demapping functions for the 3rd mapped bit in a 64-QAM constellation for a channel with a SNR level of 30dBs and different values of *delay spread*.

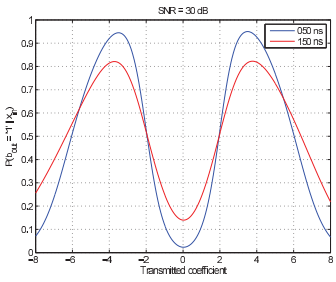


Fig. 6. Example of a demapping function

Simulations show that demapping functions with the average of the influence of the *delay spread* leads to improved performance, so our soft-demapper implements demapping functions that take into account the noise level of the channel and the input coefficients. Figure 7 shows that this new implementation leads to an improvement of around 2 dBs with respect to the original one on 54Mbps rates.

In order to estimate the SNR value we suggest using the two long training symbols available in the PPDU. Let $R_{xx}(\Delta)$ be the autocorrelation of the channel's noise, which can be

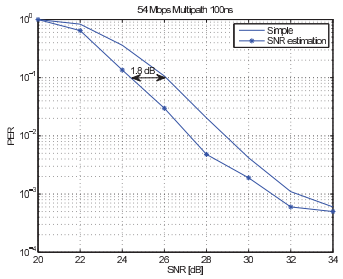


Fig. 7. Improvement in the PER due to the modified demapping algorithm in 64-QAM constellation

expressed as:

$$R_{xx}(\Delta) = N_0 \delta(\Delta) \quad (4)$$

The power of a long training symbol P_S can be expressed as:

$$\begin{aligned} P_S &= \frac{1}{N} \sum_{l=0}^{N-1} y_l \cdot (y_l)^* \\ &= \frac{1}{N} \sum_{l=0}^{N-1} (x_l + n_l) \cdot (x_l + n_l)^* = P + N_0 \end{aligned} \quad (5)$$

where N is the number of data points in a long training symbol and x_l are the noise-free data samples of the long training symbols. Therefore, the power of each long training symbol can be expressed as

$$S_i = P + N_i = P + N_0 \quad (6)$$

where i is the number of the long training symbol.

The power of the noise level of the channel can be calculated by means of the difference between the two consecutive long training symbols LS_1 and LS_2

$$\begin{aligned} x_{dif} &= LS_1 - LS_2 \\ &= (x(l) + n_1(l)) - (x(l) + n_2(l)) = n_1(l) - n_2(l) \end{aligned} \quad (7)$$

Let $R_{x_{dif}}[\Delta]$ be the autocorrelation of x_{dif}. The power P_{dif} of x_{dif} can be obtained evaluating $R_{x_{dif}}[\Delta]$ when $\Delta = 0$

$$\begin{aligned} P_{dif} = R_{x_{dif}}[0] &= \sum_{l=0}^{N-1} x_{dif}[l] \cdot x_{dif}^*[l] \\ &= E[x_{dif} \cdot x_{dif}^*] = 2N_0 \end{aligned} \quad (8)$$

where $E[\cdot]$ is the expectation.

Consequently, the channel's noise power is equal to half the power of the difference of the two consecutive long training symbols. As a result, the estimation of the SNR can be expressed as

$$\widehat{SNR} = \frac{S_1 - p_{dif}/2}{p_{dif}/2} \quad (9)$$

Figure 8 shows the performance of the estimator given in equation (9)

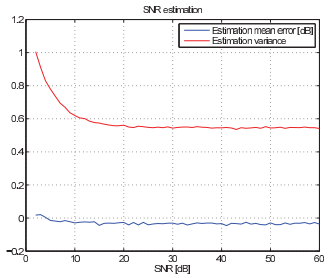


Fig. 8. SNR estimator performance

III. Proposed architectures

This section shows the architectures that have been designed to implement the proposals of section II.

A. Equalizer

The equalizer has been chosen to implement the phase tracking algorithm and provide the CSI information to the Viterbi decoder due to the convenience of the data it stores. Figure 9 shows the proposed architecture for the revised equalizer.

An initial process classifies the incoming coefficient sequence into long training symbols and SIGNAL or DATA PPDU symbols. The coefficients of the SIGNAL and DATA symbols are further classified into data or pilot tones. The mean of the long training sequence is used to estimate the channel response. A Cordic core in rotation configuration is used in conjunction with a multiplier to equalize the data carriers. In order to do so, the phase and inverse of the magnitude of the channel estimation must be calculated. A Cordic core in vector configuration estimates the phase and magnitude of the mean of the two long training symbols, and a divider calculates the inverse of the channel magnitude estimation.

A parallel process calculates the accumulated phase offset with the pilot tones. The Cordic core in vector mode is reused to estimate the angle defined in equation (1). The value of $\hat{\phi_k}$ is added to the phase estimation of each subcarrier before performing the equalization.

At the output of the equalizer a process provides the channel magnitude of the equalized symbol to provide CSI information to the Viterbi decoder.

An analysis has been carried our to establish the size of the different data buffers of the equalizer. In order to do so, several PPDU transmissions have been simulated. At the receiver, the incoming sequence has been normalized to have a maximum absolute real or imaginary part that varies from 0.7 to 1. The resulting incomming signal has then been quantized to signed 12 bit length numbers in 2.10 format. Table I shows the results of the analysis.

TABLE I
Register size study

Parameter	Minimum value	Maximum value
$H[k]$	-12	12
$\lvert H[k]\rvert$	0	13
$1/\lvert H[k]\rvert$	0	436
$\widehat{\phi_k}$	-136	139

TABLE II
Resource consumption of the equalizer

Resource name	Consumed	Available
DSP48	6	84
Block RAM	4	84
Slice	2832	16640

The equalizer we present has been described in VHDL, and its synthesis and *place and route* has been performed for a Xilinx Spartan-3A DSP 1800 FPGA. Table II shows the hardware consumption of the module. The description reaches a maximum clock speed of 66.934MHz.

B. Viterbi Decoder

Adding CSI functionality to a soft-bit Viterbi decoder is conceptually an easy task. Basically the *Branch Metric Unit* or BMU has to slightly be tuned up and register lengths must be redimensionated in order to prevent overflow.

Figure 10 shows the architecture of a general CSI capable BMU unit of a soft-bit Viterbi decoder. Since WLAN 802.11a uses a $k/n = 1/2$ convolutional coder, the BMU will have two soft-bit inputs and four output adders.

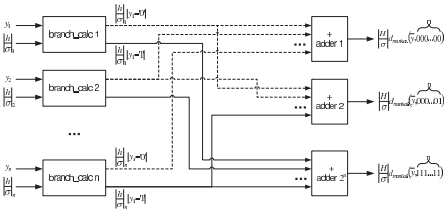


Fig. 10. BMU unit

Each branch_calc unit calculates the Manhattan distance between an input soft-bit and a logical '1 and '0 and multiplies this measure with the CSI information. Figure 11 shows the architecture of each branch_calc unit.

The branch_calc units are connected to the 2^n adders so that the output of the BMU is the Manhattan distance between the input soft-bit sequence and all the possible n-bit length output words at the convolutional coder.

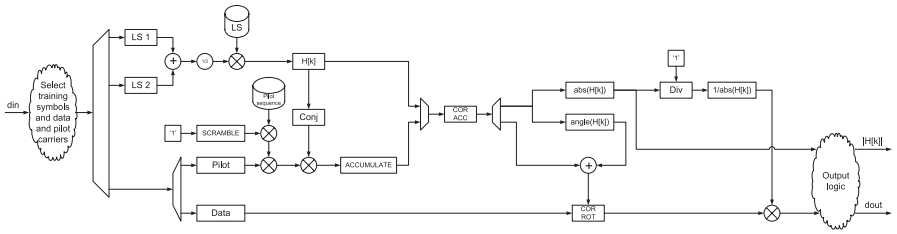


Fig. 9. Equalizer architecture

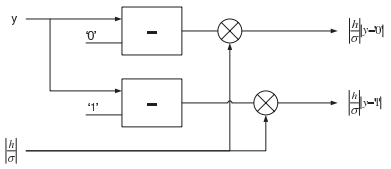


Fig. 11. branch_calc unit

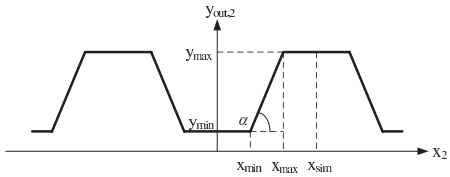


Fig. 12. Demapping parameters for softbit 3 and 6 on 64-QAM

C. Demapping

The Gray encoding used in the WLAN 802.11a mapper leads to the appearance of trapezoidal regions in the demapping functions. Storing the demapping functions defined in (3) with a sufficient precission is an impractical task from a memory perspective in an embedded system. Instead, our demapper uses first-order linear functions to approximate them.

The standard uses 4 different constellations that can map up to 3 bits in the I or Q component of each data carrier. For each constellation component I or Q let $b_{i,j}$ be the ith mapped bit in constellation j. It is readilly seen that the demmaping functions associated to bit b_i have similar shapes in all constellations. The demapping function associated to the first mapped bit in the I component of all constellations shows a postive slope shape. The second demapping function, if present, has a trapezoidal shape centered in the origin. Finally, the third demapping function, if present, has two trapezoidal shapes centered in the origin.

Figure 12 shows the parameters that characterize each trapezoid. x_{sim} is the symmetry point of each trapezoid and is used to demap bit 3 and 6 on 64-QAM only. These parameters are stored in a ROM memory.

Let x_0 be the I or Q component of the data carrier to be demapped. Due to the symmetries presented in the demapping functions, it is always possible to transform x_0 so that it falls in the interval $[x_{min}, x_{sim}]$. Equation (10) shows the transformation needed to addapt x_0 to demap bit b_1.

$$x_1 \begin{cases} x_0 & x_0 < 0 \\ -x_0 & x_0 \geq 0 \end{cases} \tag{10}$$

Equation (11) shows the transformation needed to addapt x_0 to demap bit b_2.

$$x_2 \begin{cases} x_1 & x_1 < x_{sim} \\ 2x_{sim} - x_1 & x_1 \geq x_{sim} \end{cases} \tag{11}$$

Let $y_{out,i}$ be the output of the demapping function for bit b_i. $y_{out,i}$ can the be calculated as

$$y_{out,i} \begin{cases} y_{min,i} & x_i < x_{min,i} \\ y_{min,i} + \alpha_i x_i & x_{min,i} \leq x_i < x_{max,i} \\ y_{max,i} & x_{max,i} \leq x_i \end{cases} \tag{12}$$

where $x_{min,i}$, $x_{max,i}$, $y_{min,i}$ and $y_{max,i}$ are the demapping parameters associated to bit b_i. If the demapping functions have been defined so that $0 \leq y_{out,i} \leq 1$, then the n-bit softbit can be calculated as

$$softbit_i = round(y_{out,i} \cdot 2^n) \tag{13}$$

IV. Conclusions

In this paper we have identified sources of error that degrade considerably the performance of a WLAN 802.11a compliant transceiver. Several algorithms have been proposed to overcome these limitations and architectures for implementing them have been described. Finally the synthesis and *place and route* results for some architectures have been presented.

With the new modifications, our transceiver is capable of obtaining packet error rates below 10^{-2} for datarates of 54Mbps under multipath channles for SNR values of 27dBs. Table III summarizes the impact each algorithm supposes to the PER of the receiver at 54Mbps rates.

TABLE III
PER IMPROVEMENTS ON 54MBPS

Algorithm	dB gain at PER = 10^{-1}
Phase tracking	3
CSI	10
Demapping	1.8

REFERENCES

[1] *IEEE Standard 802.11a-1999, "Supplement to Information Technology—Telecomm. and Information Exchange between Systems—Local and Metropolitan Area Networks-Specific Requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer(PHY) Specifications: High Speed Physical Layer(PHY) in the 5 GHz Band"*.

[2] *IEEE Standard 802.11n, "IEEE Standard for Information Technology – Telecommunications and Information Exchange between Systems—Local and Metropolitan Area Networks-Specific Requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer(PHY) Specifications. Amendment 5: Enhacements for Higher Throughput"*.

[3] J. Liu and J. Li, "Parameter Estimation and Error Reduction for OFDM-Based WLANs," *IEEE Transactions on Mobile Computing*, vol. 3, pp. 152–163, 2004.

[4] T. S. Rappaport, *Wireless Communications. Principles and Practice*. Prentice Hall, 1996.

[5] A. P. Bob O'Hara, *802.11 Handbook. A Designer's Companion*. Standards Information Network IEEE Press, 1999.

[6] T. Cooklev, *Wireless Communication Standards*. Standards Information Network IEEE Press, 2004.

[7] M. Butler, S. Armour, P. Fletcher, A. Nix, and D. Bull, "Viterbi Decoding Strategies for 5GHz Wireless LAN Systems," *IEEE Vehicular Technology Conference, Vol. 1, pp. 77-81*, 2001.

[8] F. Angarita, M. Canet, T. Sansaloni, and J. Valls, "Architectures for the Implementation of a OFDM-WLAN Viterbi Decoder," *Journal of Signal Processing Systems, Vol. 52, pp. 35-44*, 2008.

Parallel Implementation of a Sample Rate Conversion and Pulse-Shaping Filter for High Speed Backhauling Networks

A. Alonso, J.F. Sevillano and I. Vélez

XXIX Conference on Design of Circuits and Integrated Systems, Madrid, Nov 2014

Parallel Implementation of a Sample Rate Conversion and Pulse-Shaping Filter for High Speed Backhauling Networks

Aritz Alonso, Juan Francisco Sevillano and Igone Vélez
Department of Electronics & Communications
CEIT and Tecnun (University of Navarra)
Manuel de Lardizábal 15, 20018 San Sebastián, Spain
Email: {aadomingo, jfsevillano, ivelez}@ceit.es

***Abstract*—This paper considers the design of a parallel sample rate interpolation filter for the backhaul of the future mobile networks. These future networks must provide Gigabit data rates, which relay on the use of high spectral efficiency, high bandwidth baseband signals. Parallel signal processing becomes a necessity since state of the art technology is incapable of serially generating and converting these high bandwidth signals. Moreover, signal generation and digital-to-analog conversion are performed under incommensurate clock domains. Therefore, interpolation becomes a necessity when connecting signal generation to signal conversion. The paper analyses the interpolation equation and discusses several techniques to achieve a parallel implementation. The designed interpolation filter has been tested to adapt an incoming signal data stream at a rate of 1.7 giga-symbols per second into a stream of interpolants at a rate of 2.8 giga-interpolants per second under two incommensurate clock domains.**

***Index Terms*—Sample Rate Conversion, Interpolation, High Speed, Filtering, Parallel implementation**

I. Introduction

The growing demand for ubiquitous broadband communication has motivated the deployment of ultra high-speed communication systems. Optical fiber has been the preferred technology for backhaul networks. However, optical fiber has some important drawbacks such as high cost, long deployment times and low flexibility. In recent years, point-to-point wireless communication systems or Fixed Radio Systems (FRS) have been proposed as a suitable alternative to fiber optic. To be competitive against fiber optic based backhaul networks, these communication systems must achieve very high data rates.

The regulation of the E-band (71 to 76 and 81 to 86GHz) that is being carried out by the European Telecommunications Standards Institute (ETSI) facilitates the deployment of high-speed communication systems [1] [2]. At present, commercial off-the-shelf communication systems operating in the E-band support data rates up to 2.5GBit/s. However, backhauling networks for new wireless communication systems demand significantly higher data rates, in the order of 10Gbit/s or more, which require both wide-band and high-order modulations to utilize the spectrum efficiently.

978-1-4799-5743-9/14/$31.00

Pulse shaping is a fundamental step in all digital transmitters: they reduce the bandwidth requirements of the transmitted signal and minimize inter symbol interference (ISI) between samples. The high data rates involved in backhaul networks make serial architectures of these components infeasible, even for first tier FPGAs. Consequently, parallel implementations are required to fulfil the throughput requirements of these networks.

It is not uncommon for high speed communication systems that signal generation and digital-to-analog conversion are done under independent, incommensurate clock domains. This introduces an extra degree of complexity to the parallel interpolator and pulse shaping filter design, since it requires that it performs a *fractional delay* [3] or *sample clock conversion* to adapt the signal between the two clock domains.

Many research work has been carried out to obtain fractional ratio $r = A/B$ interpolators, such as in [4] and [5]. These techniques involve an interpolation by an integer factor A of the coming samples and then decimating the outcome by an integer factor of B and are, therefore, limited to fractional ratio interpolations. In this work we want to address the more general case where the two clock domains have incommensurate frequencies.

Some other authors suggest making use of signal properties in the frequency domain to obtain a fractional interpolation in the time domain. [6] presents algorithms in the frequency domain to interpolate or decimate signals in the time domain. However, these algorithms are also limited to fractional ratio conversions and require direct and inverse DFT cores with a number of points at least one order of magnitude greater than the order of the pulse shaping filter to obtain a neglectable *mean square error* in the conversion.

Another common technique consists in upsampling the incoming sample signal with a FIR filter and then interpolating the intermediate samples using well-known functions such as linear interpolators [7] or b-splines [8]. These interpolators are based on optimized Farrow structures [9] to minimize the hardware requirements, but due to the already wide bandwidth

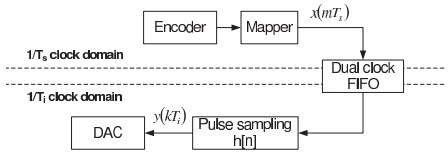


Fig. 1. Serial transmitter model

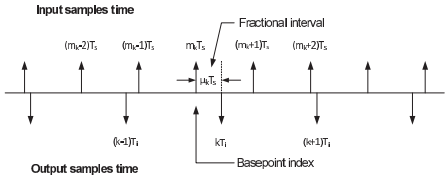


Fig. 2. Sample time relations

of the backhauling network signal, upsampling would require increasing even more the degree of parallelism, which in turn will augment the hardware requirements and complexity of the design.

In this paper we present the architecture of a parallel pulse shaping interpolator filter based solely on a FIR filter structure that can convert between irrational ratio clock domains. The architecture has been tested to pulse-shape an incoming 1.7 giga-symbol per second (*Gsps*) data stream into an output 2.8 giga-interpolants per second (*Gips*) data stream. Implemented on a Virtex7 FPGA, the design reaches clock speeds above 350MHz.

The paper is structured as follows. Section II introduces the basis of serial interpolation. Section III summarizes the parallel transceiver model and presents the parallel pulse shaping filter interpolator architecture. Section IV shows the implementation results of the parallel interpolator. Finally, Section V presents the conclusions.

II. Serial Interpolation

The mathematical analysis for the serial sampling rate converter in this section is derived from the works presented in [10] and [11]. Figure 1 depicts the serial model of a digital transmitter. The digital encoder produces new symbols at a rate $1/T_s$. These symbols are then mapped to complex constellation points according to the modulation of the transmission. Let $x(mT_s)$ be such mapped symbols.

The output of the mapper is sent to an upsampler and digital pulse shaping filter with impulse response $h[n]$. This digital filter is driven by a clock signal with a period T_i. The output of this filter can then be written as

$$y(kT_i) = \sum_m x(mT_s)h(kT_i - mT_s) \tag{1}$$

Note that, in general, the ratio T_i/T_s is irrational. A dual clock FIFO memory is used to isolate the two clock domains in the design. Finally, samples $y(kT_i)$ are sent to a DAC for digital to analog conversion.

Index m in (1) is a signal index for samples $x(mT_s)$. A more useful notation for (1) can be obtained rearranging the indexes as follows. First, the *filter index* is defined as

$$p = \lfloor kT_i/T_s \rfloor - m \tag{2}$$

where $\lfloor x \rfloor$ denotes the largest integer not exceeding x. Next, the *basepoint index* is defined as

$$m_k = \lfloor kT_i/T_s \rfloor \tag{3}$$

and the *fractional interval* is given by

$$\mu_k = kT_i/T_s - m_k \tag{4}$$

where $0 \leq \mu_k < 1$. Figure 2 depicts relations between the different timing indexes.

Taking these new indexes, (1) can be written as

$$\begin{aligned} y(kT_i) &= y\left((m_k + \mu_k)T_s\right) \\ &= \sum_{p=0}^{N-1} x[(m_k - p)T_s]h[(p + \mu_k)T_s] \end{aligned} \tag{5}$$

If the *interpolating filter* has a finite impulse response (FIR), then the digital interpolation filter has a finite number of taps N. In order to produce the k-th output sample $y(kT_i)$, the value used for the p-th tap in (5) corresponds to the value of the filter impulse response at time $(p + \mu_k)T_s$. This can be easily implemented using a *look-up-table* (LUT) for each tap with μ_k as address. The content of the LUTs are the samples of the impulse response of the pulse shaping filter sampled at T_s/O, $h_O[n]$, where O is the number of bits used for the address in the LUT.

A serial sample rate converter performs the calculations in (5). Therefore, N adjacent $x[m]$ signal samples and N samples of the impulse response h_O are required. The *basepoint index* m_k points to the correct set of signal samples $x(m)$ that has to be fetched from the FIFO memory, while the *fractional interval* μ_k points to the correct set of impulse response samples.

Indexes m_k and μ_k are obtained from an accumulator. The accumulator is clocked at a rate of $1/T_i$ Hz. Every clock cycle the adder is increased a quantity

$$\phi = \frac{T_i}{T_s} \tag{6}$$

On average, the adder increases one integer unit per T_s seconds.

If the adder value at time instant k is given by R_k, then

$$m_k = \lfloor R_k \rfloor \tag{7}$$

and

$$\mu_k = R_k - m_k \tag{8}$$

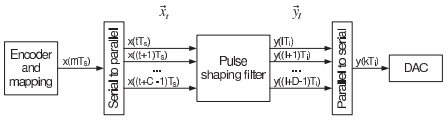


Fig. 3. Parallel transmitter model

III. Parallel Sample Rate Converter Architecture

The sampling rates involved in backhauling networks surpass the clock frequency range of state-of-the-art FPGAs. Parallelization of the transmission algorithms is therefore required in order to obtain communication systems with data rates in the order of Giga-bits per second (*Gbps*).

Figure 3 depicts a model of a parallel pulse shaping filter. Without any loss of generality, we can assume that the mapped $x(mT_s)$ symbols are parallelized just before the pulse shaping filter. A *serial to parallel* unit takes the high speed $x(mT_s)$ symbol stream and produces vectors $\vec{x}_t$ of length C at a rate $1/(CT_s)$. The index t in $\vec{x}_t$ indicates that the sample indexes of the encoded symbol samples in the vector belong to the set $m \in [tCT_s, (C(t+1)-1)T_s]$.

Similarly, the pulse shaping filter produces a new vector $\vec{y}_l$ of D interpolants at a rate $1/DT_i$. The time index l in $\vec{y}_l$ indicates that the sample indexes of the interpolants in the vector belong to the set $k \in [lDT_i, (D(l+1)-1)T_i]$. Vectors $\vec{y}_l$ are then sent to a parallel-to-serial unit (typically a *Serializer/Deserializer* or SerDes core) every DT_i seconds. The parallel-to-serial unit creates then a high speed interpolant stream $y(kT_i)$ at a rate $1/T_i$ which is sent to the digital to analog converter. Note that the parallelization factors C and D are not required to be the same.

To obtain a parallel pulse shaping filter structure, Equation 5 must be modified so that it takes C consecutive input symbols and produces a vector with D consecutive *interpolants*.

As was previously mentioned, each *basepoint index* m_k and *fractional interval* μ_k pair points to a unique set of symbol samples of $x[m]$ and impulse response samples of $h_O[n]$ to be used in (5). Consequently, each of the D *interpolants* in $\vec{y}_l$ can be calculated in parallel with D instances of a hardware architecture that computes (5) provided we know their associated m_k and μ_k.

Let $\vec{m}_l$ and $\vec{\mu}_l$ be vectors containing the *basepoint indexes* and *fractional intervals* for all the D *interpolants* in $\vec{y}_l$ at time instant l. Figure 4 depicts a simplified block diagram of the proposed architecture for the interpolator filter. As it can be seen, the filter consists of a control unit responsible, among others, of calculating $\vec{m}_l$ and $\vec{\mu}_l$ and a series of sub-filters responsible of calculating the elements of $\vec{y}_l$. The filter is complemented with C instances of dual-clock FIFOs that are used to isolate the two clock domains ($1/CT_s$ and $1/DT_i$) of the design and to provide vectors $\vec{x}_t$ as commanded by the control unit.

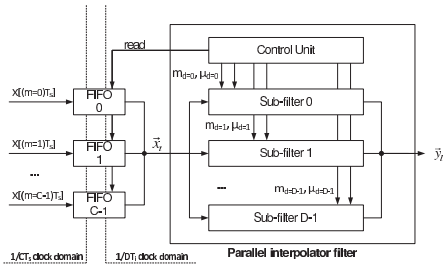


Fig. 4. Simplified block diagram of the parallel interpolation filter

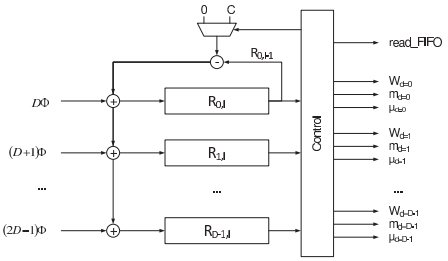


Fig. 5. Control Unit block diagram

The following sections describe each interpolator filter component in detail.

A. Control unit

The control unit is responsible for providing vectors $\vec{m}_l$ and $\vec{\mu}_l$ to the sub-filter units and triggering new symbol sample vector $\vec{x}_t$ reads from the external FIFOs. Figure 5 depicts the block diagram of the control unit.

Time index l in $\vec{m}_l$ and $\vec{\mu}_l$ indicates that the elements in both vectors correspond to time intervals in the range $k \in [lDT_i, (D(l+1)-1)T_i]$. Values $m_{d,l}$ and $\mu_{d,l}$ (with $d \in [0, D-1]$) correspond to the dth elements of vectors $\vec{m}_l$ and $\vec{\mu}_l$ respectively.

The control unit consists of a series of D accumulators similar to the one described in Section II. Let $R_{d,l}$ be the accumulated value in the dth accumulator at time instant l. Then, at time interval l each accumulator should be increased as follows:

$$R_{d,l} = R_{0,l-1} + (D+d)\phi \tag{9}$$

where time interval $l-1$ denotes the previous time interval to l. Note that the registered value in accumulator R_0 at time interval $l-1$ is used to update the register values of all accumulators at time interval l. This ensures that on a free running scenario all accumulators are synchronized to the same reference.

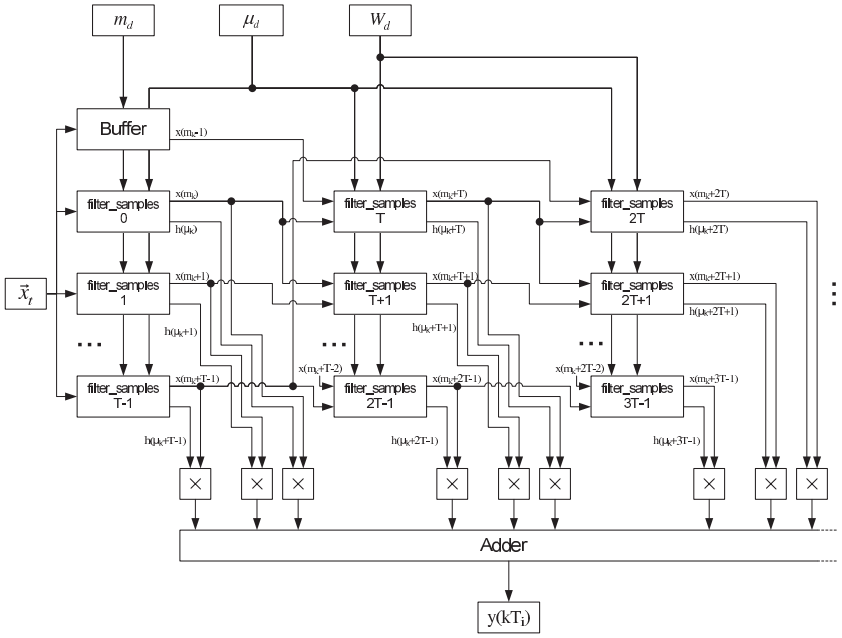


Fig. 6. Block diagram of the sub-filter unit

Finally, let $\vec{R}_l$ be a vector that contains the D values of all accumulators at time instant l. Then, from (9), vectors $\vec{m}_l$ and $\vec{\mu}_l$ can be obtained as

$$\vec{m}_l = \lfloor \vec{R}_l \rfloor \tag{10}$$

and

$$\vec{\mu}_l = \vec{R}_l - \vec{m}_l \tag{11}$$

Observe that the difference in the D elements of $\vec{m}$ between time indexes l and $l+1$ can take the values $\lfloor D \cdot T_i/T_s \rfloor$ or $\lceil D \cdot T_i/T_s \rceil$ only, where $\lceil x \rceil$ denotes the smallest integer greater than x. This property is vital to reduce the memory requirements of the sub-filter components of the interpolator. Consequently, the control unit provides a new vector $\vec{W}_l$ that indicates in which elements of vector $\vec{m}_l$ the condition $m_{d,l} - m_{d,l-1} = \lceil D \cdot T_i/T_s \rceil$ holds true.

Real implementations of the interpolation filter can not relay on infinite precision accumulators as those in (9) and the designer is limited to finite word length accumulators only. Consequently, overflow problems have to be considered when designing such adders. The following mechanism is proposed:

$$R_{d,l} = \begin{cases} R_{0,l-1} + (D+d)\phi, & \text{if } R_{0,l-1} < C \\ R_{0,l-1} - C + (D+d)\phi, & \text{otherwise} \end{cases} \tag{12}$$

This mechanism ensures that value of the last accumulator R_{D-1} is always less than $C + \lceil D\phi \rceil$, and thus, only $\lceil log_2(C + \lceil D\phi \rceil) \rceil$ bits are required to extract values m_d from the adders.

Every time the condition $R_{0,l} \geq C$ holds true, the following steps are followed:

- The control Unit requests a read to the external FIFOs.
- All the sub-filter units in the interpolation filter are given a new symbol vector $\vec{x}_t$ at time interval $l+1$.
- The read pointer of every *buffer* of the interpolation filter is increased by one unit. Refer to Section III-C for further details.

B. Sub-filter unit

Figure 6 depicts the architecture of the sub-filters. Each sub-filter contains N instances of components labelled *filter_samples* that are used to extract all signal and impulse response samples $x(m_d)$ and $h_O(\mu_d)$ for (5) per clock cycle.

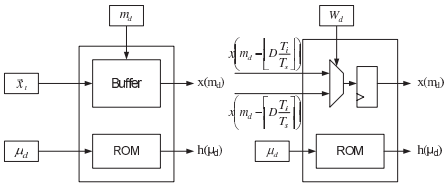


Fig. 7. Block diagram of the different *filter_samples* components

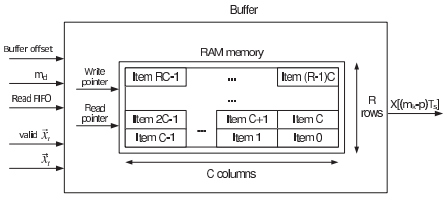


Fig. 8. Simplified view of the *buffer*

As was mentioned in Section III-A, signal samples at time instant l will only be shifted $\lfloor D \cdot T_i/T_s \rfloor$ or $\lceil D \cdot T_i/T_s \rceil$ positions at time instant $l+1$. That is to say: at time instant l, the possible signal samples for the tap $N-1$ of the FIR filter can only be the signal samples of the taps $N-1-\lfloor D \cdot T_i/T_s \rfloor$ or $N-1-\lceil D \cdot T_i/T_s \rceil$ at time instant $l-1$. Similarly, the possible signal samples for the tap $N-2$ at time instant l can only be the signal samples in taps $N-2-\lfloor D \cdot T_i/T_s \rfloor$ or $N-2-\lceil D \cdot T_i/T_s \rceil$ at time instant $l-1$, and so on.

If the FIR filter architecture is back-traced to its first taps, we will notice that the first $T = \lceil D * T_i/T_s \rceil$ taps require some sort of *buffering* because the filter architecture provides no *previous time interval* signal samples to them.

A more detailed block diagram of the architectures of the *filter_samples* component is show in Figure 7. The architecture on the left of Figure 7 corresponds to the first T *filter_samples* components of the sub-filter. These *filter_samples* use a *buffer* to extract their signal sample. The architecture of these *buffers* will be explained in detail in Section III-C. The architecture on the right of Figure 7 corresponds to the remaining *filter_samples* components of the sub-filter. As it can be seen, in these *filter_samples* the *buffers* have been replaced with multiplexers that behaves as the $\lfloor D * T_i/T_s \rfloor$ or $\lceil D * T_i/T_s \rceil$ position shift register.

Every *filter_samples* component contains a 2^M deep ROM memory to store the impulse response samples. The M most significant bits of the *fractional interval* μ_d are used as the read address for all ROM memories.

Finally, N multipliers and an add-tree are used to implement the taps of the FIR pulse shaping filter in (5).

C. Buffer

Figure 8 depicts the block diagram of the *buffer* unit.

The *buffer* is a data pool organized in R rows. Taking into account the overflow mechanism in (12), the value R is chosen large enough so that the *buffer* always contains sufficient symbol samples for the interpolation filter. Each row stores C symbol samples.

The *buffer* instantiates a RAM memory and, as with FIFO memories, contains two pointers: one read pointer and one write pointer. The write pointer functions as in a FIFO memory: whenever a new vector $\vec{x}_l$ is received in the sub-filter unit, it is stored in the row pointed by the write pointer. The write pointer value is then incremented by one unit. As it can be seen from Figure 8, the newest signal samples in $\vec{x}_l$ occupy the leftmost positions in the row, while the oldest samples occupy the rightmost positions.

The read pointer, on the other hand, has a different behaviour: in conjunction with the port *Buffer offset* and the *basepoint index* m_k it is used to calculate the coordinate (row and column) where the symbol sample required by the filter tap is stored.

Each of the $T+1$ *buffers* in the sub-filter unit is given a unique *Buffer offset* value: in the architecture depicted in Figure 6, the *Buffer* component is given a *Buffer offset* of 0, the *buffer* inside the first *filter_sample* is given a *Buffer offset* of 1, and so on. Finally, the *buffer* inside the *filter_samples* $T-1$ is given a *Buffer offset* of T.

Given the triplet $m_{d,l}$, *Buffer offset* and read pointer, the coordinates of the symbol sample $x[(m_k-p)T_s]$ in (5) are calculated as

$$Column = (\text{Buffer offset} + m_{d,l} \bmod C) \qquad (13)$$

and

$$Row = (\text{Read pointer} + (Column \bmod C) \bmod R) \qquad (14)$$

Every time the control unit of the interpolator filter asserts true the condition $R_{0,l} \geq C$ in (12) the read pointer of all buffers is incremented by one unit.

The increment of the read pointer makes the stored signal samples below it not accessible by the signal sample extraction algorithm in (13) and (14) and thus, frees those memory locations so the write pointer can overwrite them with new signal samples. Observe that the read pointer increment and the overflow avoiding mechanism for the $\boldsymbol{R}_D$ adders of the control unit described in (12) in Section III-A occur simultaneously.

IV. Results

The parallel interpolation filter described in section III has been synthesized and implemented on a Xilinx Virtex7 XC7VX485T-2FFG1761C FPGA with Vivado 2013.4 at its default configuration with the set of parameters described in table I

To reduce the number of dedicated DSP units required by the design, the multipliers and adder tree shown in Figure 6 have been implemented in cascaded DSP48E1 elements configured as MACs. Since all *filter_samples* units provide the

Parameter	Value
C, D	8
$1/(T_s)$	1700MHz
$1/(CT_s)$	212.5MHz
$1/(T_i)$	2800MHz
$1/(DT_i)$	350MHz
N	30
bitwidth m_k	5 bits
bitwidth μ_k	25 bits
bitwidth M	7 bits
bitwidth $x(mT_s)$	12 bits
bitwidth $y(kT_i)$	14 bits

TABLE I
PARAMETER SUMMARY OF THE IMPLEMENTED INTERPOLATION FILTER

Resource	Used	Available	%
Slice	11840	75900	15.60
LUT	25826	303600	8.51
FF	44547	607200	7.34
RAMB18	560	2060	27.18
RAMB36	80	1030	7.77
DSP48E1	480	2800	17.14

TABLE II
RESOURCE UTILIZATION OF THE IMPLEMENTED INTERPOLATION FILTER

N signal and filter impulse response samples for (5) in one single clock cycle, the input to each of the cascaded DSP48E1 has to be delayed accordingly so that the arithmetic operation in (5) is preserved. Consequently, shift registers have been implemented before the tap multipliers in Figure 6.

Implemented for a clock frequency of 350MHz, the architecture obtains a worst negative slack of 0.05ns. Logic consumes the 11.9% of the timing while routing represents the 88.1% of the timing. The critical path is found in the routing through the shift registers that connect the output of the *filter_samples* cores to the input of the tap multipliers in figure 6. The reported dynamic power consumed by the design is of 7.44W.

Table II summarizes the resource utilization of the design: most of the hardware resources are dedicated to implement the multipliers of the sub-filters and the shift-registers necessary to synchronize the symbol and impulse response samples in the adder trees.

To the authors's knowledge, no work has been reported in the literature that simultaneously address the implementation of a parallel pulse shaping and irrational interpolation filter for high speed backhauling networks.

From a data bandwidth perspective, the work in [12] presents parallel sampling rate conversion algorithms for a multiband OFDM communication system with identical throughput to ours (10Gbps), but does not try to achieve an implemented design.

From the sample rate conversion and pulse shaping implementation perspective, authors in [13] present a combination of parallel decimators and interpolators based on FIR Farrow structures to achieve fractional (not *irrational*) sample rate conversions. The authors report an achievable clock speed of 211MHz when the parallelization of the sample rate converter is of $D = 4$ interpolants per clock cycle. The last stage parallel interpolator comes at a cost of up to $D \cdot Y$ extra multipliers not required by our design, where Y is the order of the polynomial interpolator. On the other hand, the tap coefficients of their filters are constant, so our architecture can be seen a tradeoff between multipliers and memory requirements to store the values of $h_O[n]$.

V. CONCLUSIONS

In this paper a parallel pulse shaping and interpolation filter architecture has been presented that translates signals samples between clock domains with incommensurate ratios. First, the serial interpolation (5) has been studied and the parallel transmitter model and hardware architecture have been proposed. Then, the parallel interpolation filter has been implemented on an FPGA and finally area and speed reports have been shown. The implemented architecture interpolates an incoming symbol sequence at a rate of 1.7Gsps into an output sequence at 2.8Gips working with a 350MHz clock.

.

REFERENCES

[1] "Fixed radio systems; characteristics and requirements for point-to-point equipment and antennas; part 1: Overview and system independent common characteristics," European Telecommunications Standards Institute, ETSI EN 302 217-1, September 2012.

[2] "Fixed radio systems; characteristics and requirements for point-to-point equipment and antennas; part 2-2: Digital systems operating in frequency bands where frequency co-ordination is applied; harmonized en covering the essential requirements of article 3.2 of the r&tte directive," European Telecommunications Standards Institute, ETSI 302 217-2-2, September 2012.

[3] T. I. Laakso, V. Valimaki, M. Karjalainen, and U. K. Laine, "Splitting the Unit Delay," *IEEE Signal Processing Magazine*, vol. 1, pp. 30–60, January 1996.

[4] M. Blok, "Fractional Delay Filter Design for Sample Rate Conversion," *Proceedings of the Federated Conference on Computer Science and Information Systems*, vol. 1, pp. 701–706, 2012.

[5] A. Senst, G. Fock, and H. Meyr, "Rate Conversion for Arbitrary Sampling Rates in the Transmit Path of a Digital Transceiver," *Global Telecommunications Conference*, vol. 1, pp. 3658–3662, 2001.

[6] G. Bi and S. K. Mitra, "Sampling Rate Conversion in the Frequency Domain," *IEEE Signal Processing Magazine*, vol. 1, pp. 140–144, May 2011.

[7] D. Babic, J. Vesma, and M. Renfors, "Decimation by Irrational Factor using CIC Filter and Linear Interpolation," *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, vol. 1, pp. 3677–3680, 2001.

[8] C.-C. Lin and H.-F. Chi, "A low-Complexity B-spline Based Digital Sample Rate Conversion Circuit Architecture," *International Symposium on Signals, Circuits and Systems*, vol. 1, pp. 505–508, 2005.

[9] C.W.Farrow, "A Continuously Variable Digital Delay Element," *IEEE International Symposium on Circuits and Systems*, vol. 1, pp. 2641–2645, 1988.

[10] F. M. Gardner, "Interpolation in Digital Modems - Part I: Fundamentals," *IEEE Transactions on Communications*, vol. 41, pp. 501–507, 1993.

[11] J. Ketola, J. Vankka, and K. Holonen, "Synchronization of Fractional Interval Counter in Non-Integer Ratio Sample Rate Converters," *Proceedings of the International Symposium on Circuits and Systems*, vol. 2, pp. 89–92, 2003.

[12] X. Huand, J. Joseph, J. A. Zhang, and Y. H. Guo, "Sample Rate Conversion with Parallel Processing for High Speed Multiband OFDM Systems," *IEEE Wireless Communications and Networking Conference*, vol. 1, pp. 2754–2759, April 2013.

[13] J. P. Long and J. A. Torres, "High Throughput Farrow Re-Samplers Utilizing Reduced Complexity FIR Filters," *Military Communications Conference*, vol. 1, pp. 1–6, October 2012.

Fast hardware-in-the-loop verification platform: a case study for convolutional decoders.
A. Alonso and A. Irizar
XXIX Conference on Design of Circuits and Integrated Systems, Madrid, Nov 2014

Fast hardware-in-the-loop verification platform: a case study for convolutional decoders

Aritz Alonso and Andoni Irizar
Department of Electronics & Communications
CEIT and Tecnun (University of Navarra)
Manuel de Lardizábal 15, 20018 San Sebastián, Spain
Email: {aadomingo, airizar}@ceit.es

***Abstract*—The time required for system analysis and verification has invariably become the bottleneck of the development process as designs become more complex. Methodologies involving configurable hardware have proven to be the only ones to break the inverse relationship between accuracy in simulation and performance in verification. In this work we present a *Hardware-in-the-Loop* based platform that substantially reduces the time required for system verification and parameter fine tuning. The verification platform has been used to evaluate the performance of the physical layer of a WLAN 802.11a compliant transceiver. Compared to a software RTL simulation this platform reduces simulation time by a factor of at least 10^3.**

***Index Terms*—OFDM, co-simulation, verification.**

I. Introduction

In 1965 Gordon E. Moore published [1] his well-known observation that states that the number of transistors that can be fitted inexpensively on a silicon device doubles every two years. This observation, which was introduced in the early days of the semiconductor technology, continues to hold true nowadays. Consequently, circuit designs are expected to continuously grow in complexity and include more advanced and accurate functions.

The increase in the complexity of the designs has some severe drawbacks as well: bigger projects require longer and more meticulous design processes, implementation time increases and demands a thorough verification stage. Moreover, the execution time of optimization stages where a number of parameters (memory and cache sizes, ...) has to be determined can become unacceptable [2].

With the improvement of the synthesis tools, the verification process has become the bottleneck of design and can involve around the 60% or the 70% of the development time [3]. Any reduction in the verification stage can lead to a considerable shorter time to market.

Many techniques have arisen to speed up simulation and reduce verification time. Among them, methodologies involving configurable hardware have proven to be the only ones to break the inverse relationship between accuracy in simulation and performance in verification stages [4]. These methodologies, also known as as *hardware-in-the-loop* simulations, lie in the principle of executing the most complex and time consuming tasks of the design on dedicated hardware while the lightest ones are executed in software.

978-1-4799-5743-9/14/$31.00 ©

Hardware-in-the-loop simulations could theoretically be executed at real-time speeds. However, communication between the hardware and software layers have shown to be the limiting factor of the simulation acceleration the platform can achieve [5].

Hardware-software co-simulation has been used for a variety of purposes such as hardware model verification [6] and algorithm calculation acceleration [7]. Obtaining the *packet error rate* (PER) curves of a transceiver is a time consuming task which could be improved significantly with the use of *hardware-in-the-loop* simulations.

In this work we present a *System Generator* based hardware-software platform for fast transceiver verification. The platform uses an Ethernet connection which reaches communication speeds of up to 1 Gbps, much faster than what serial connections RS232 [8] or USB [9] can achieve. The platform is suitable for medium-size projects and requires less effort to configure and control the simulation than those of proposed works [10] [11].

The proposed hardware-software verification platform has been used to analyze the performance of a WLAN 802.11a compliant transceiver [12] measuring its PER curves under real communication channels. Specifically, the platform has also been used to quickly study the impact several design parameters of the convolutional decoder have in the transceiver performance and size.

The paper is structured as follows: section II briefly describes the transceiver architecture and details the Viterbi decoder that will be analyzed with the verification tool . Section III presents the hardware-software verification platform and Section IV shows the fast parametrical study that was carried out. Finally, the conclusions of this work are presented in Section V.

II. Transceiver architecture

WLAN 802.11a was one of the first standards to be built upon the now common *orthogonal frequency division multiplexing* (OFDM) scheme. OFDM distributes the information to be transmitted between a series of orthogonal carriers so that a single high-speed data stream is divided into multiple slower data-streams that are transmitted in parallel through a channel. The inverse and direct FFT algorithms have proven

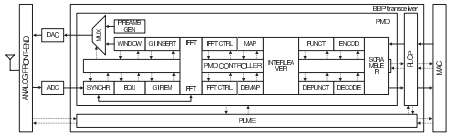


Fig. 1. Block diagram of the transceiver architecture

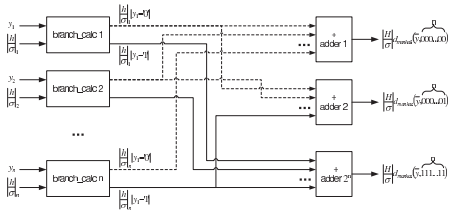


Fig. 2. BMU architecture

to be an efficient way to achieve the carrier orthogonality at the transmitter and receiver sides.

Figure 1 shows the architecture of the WLAN 802.11a compliant transceiver to be verified. As it can be seen several submodules are multiplexed in the transmission and reception chains to reduce the resource consumption. The FFT core is the most versatile submodule for it implements the inverse FFT algorithm in transmission mode and the direct FFT algorithm in reception mode, as well as support the synchronization stage.

The Viterbi decoder provides forward-error correcting to the WLAN 802.11a transceiver. The decoder implemented in the transceiver is a soft-decission decoder. It has been demonstrated [13] that *soft* decoders have better performance than *hard* decoders. However, from a hardware perspective, *soft* decoders require more complex architectures and thus, a balance between hardware complexity and decoding gain should be met.

The b bit wide softbits at the input of the soft decoder represent the logical value of the demapped symbol ('0' or '1') and its certainty. All values above $2^{b-1}-1$ correspond to a logical '1' with 2^b-1 being its most certain representation, while the rest of values correspond to a logical '0' with value 0 the most certain decision.

The encoding of the certainty, albeit practical for the demapper arithmetic, has an important disadvantage: since any binary word can represent an even number of values, neither of them correspond to an equal degree of uncertainty towards logical values '1' and '0'.

Many communication schemes achieve different data rates with a single encoder with the use of puncturing. At the receiver dummy bits are introduced in the positions where puncturing was performed. Since the dummy bits were not transmitted, they are seen as a maximum uncertainty by the forward error correcting algorithm.

Since no softbit value represents an equal uncertainty in the decision, soft-demapping can introduce a bias in the decision when puncturing is performed. A softbit transformation is proposed to avoid this problem. Let sb_{in} be the value of the input softbit to the forward error correcting algorithm. If b is the bitwidth of the softbit representation, then the transformed softbit sb^* is calculated as

$$sb^* \begin{cases} sb_{in} - (2^{b-1}-1) & sb_{in} \geq 2^{b-1} \\ sb_{in} - 2^{b-1} & sb_{in} < 2^{b-1} \end{cases} \quad (1)$$

After the transformation, the most certain values for logical '1' and '0' are given by $\pm 2^{b-1}$, and value 0 is reserved for punctured symbols. As it can be seen, the uncertainty in the decision grows as the softbit values tend to 0. Therefore an equally uncertain representation for both logical values is obtained at the expense of an extra bit to cover negative softbit values.

It has been proven [14] that the knowledge of the gains the channel induces to each subcarrier in the OFDM symbol improves the performance of the Viterbi decoder. This information is known as *channel state information* or CSI. The idea is to provide further information to the Branch Metric Unit (BMU) in the Viterbi decoder so that it takes into account the gains that have to be applied to each subcarrier to flatten the channel response. As a result, softbits that have been demapped from carriers that require higher gains (in other words, carriers that have been attenuated more) contribute less to the metrics in the decoder than those who require lower gains.

The BMU unit in the Viterbi decoder is provided with softbits and the values of the channel estimation for the subcarrier they were demapped from. The unit operates as follows:

- Each input softbit is transformed according to equation (1)
- A multiplication between each transformed softbit and corresponding channel estimation value is performed. The absolute value of the channel estimation is represented with c bits and is normalized so that it falls in the closed interval $[0,1]$. That is to say, the allowed values of the *CSI* port range from 0 to 2^c-1. Then the most certain values for logical '1' and '0' are given by $\pm 2^{b+c-1}$.
- A default *CSI* value of 2^c-1 is given to all the punctured symbols.
- For k/n rate codes, $2n$ multiplications are taken and compared with the 2^n possible codewords that the convolutional coder can obtain at a given moment. The maximum number of bits required to express the distance between the codewords and the received symbols is $b+c+n-1$.

Figure 2 depicts the architecture of the improved BMU unit in the Viterbi decoder.

In the Viterbi decoding algorithm the *add-compare-select*

unit (ACSU) is responsible for obtaining the minimum metric path that reaches each state in the convolutional coder at a given time moment. When the Viterbi decoder is built to maximize the decoding data throughput, a quantity equal as states in the convolutional code of ACSUs are implemented in the decoder, all of them working in parallel at the same time. The ACSU is the limiting factor of the maximum clock speed the Viterbi decoder can achieve. The path metric at a given moment depends on the path metrics in the previous iteration and on the branch metrics given by the BMU at that time, so there is no way of pipelining its architecture. Consequently, the ACSU must finish its calculations before new data arrives to the decoder. Typically the decoder is given new data each clock cycle, so the architecture of the ACSU should have a latency smaller than the clock period of the system.

In a k/n rate code, each state in the coder can be reached from 2^k different states. That is to say, each ACSU has to perform 2^k additions and select the minimum result per clock cycle. The addends of each sum are the accumulated metric path to the previous state and the metric path: the transition cost of going from the previous state to the current state.

The ACSU can be seen as a fixed precision accumulator. Consequently overflow problems can occur after several iterations. A first mechanism to avoid this problem consists in extending the word length of the path metrics with t bits. When the message to be decoded is long or the decoder works in stream mode the value needed of t to avoid overflow rises so fast that using this mechanism alone becomes unfeasible.

Another possibility would be to identify the ACSU with the minimum accumulated path metric and subtract this quantity to all survivor metrics in the next decoding iteration. As mentioned before, there is no possibility to pipeline the ACSU architecture, so the minimum metric selection and ACSU operation should be performed along the normal ACSU operation in a single clock cycle, making the overall decoder very slow.

In our implementation we compare the accumulated path metric in the ACSU with a constant value $L = 2^l$. If the accumulated value is greater than this fixed quantity a flag is set. Then, if all ACSU modules have set their flag, in the next decoding iteration each ACSU subtracts L to its accumulated path metric. We have set $l = b+c+n-1$, for it is the greatest possible value at the output of the BMU.

Figure 3 shows the architecture of the ACSU module.

In sum, the parameters that can modify the performance of the Viterbi decoder are the number of bits used to represent the CSI, the number of bits the ACSU extends the path metric to avoid overflows and the traceback depth of the algorithm.

III. Fast Verification Platform

The fast verification platform is based on System Generator [15], a component of the Xilinx ISE Design Suite that enhances Matlab's Simulink capabilities. Simulink offers a powerful design environment for multidomain simulation. System Generator adds a variety of new block libraries specially designed for digital signal processing.

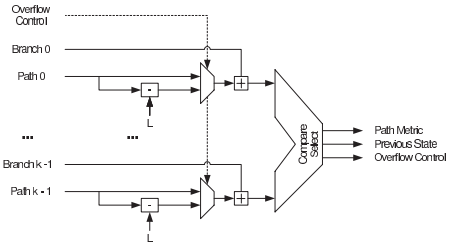


Fig. 3. ACSU architecture

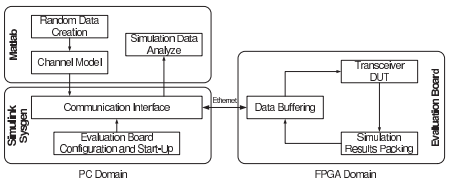


Fig. 4. Architecture of the fast prototyping system

Among its features, System Generator allows to import custom IPs in HDL and to compile designs to run them on real time under Xilinx FPGAs. Several evaluation boards are supported by default, but the environment allows any board that meets certain hardware requirements if proper configuration files are supplied by the user. System Generator also provides an Ethernet or JTAG based interface for communication between the hardware and software layers of the simulation which is transparent to the user.

All this reasons make System Generator an interesting tool when designing hardware-in-the-loop simulations. Figure 4 depicts the block diagram of the platform that was designed to verify the performance of hardware transceiver implementations.

The software component in the simulation is responsible for creating PPDUs with random data and implementing the WLAN multipath channel with different values of noise and delay spread as the model found in [16]. The whole WLAN 802.11a transceiver is implemented in the hardware component of the simulation. It is responsible for decoding the received PPDU and counters the number of erroneous bits to release overhead to the software layer. It also returns information about the status of the transceiver (time and frequency offset estimations, ...) to the software layer. System Generator is responsible for waking up and configuring the evaluation board and synchronizing the software and hardware components of the simulation.

Given that the multipath channel model is implemented in software its statistical properties can get closer to a real channel than those obtained with embedded implementations.

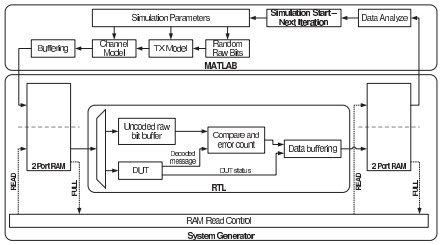


Fig. 5. Simulation flow dissection

Moreover, it gives the possibility to evaluate the design in a more accurate scenario and not only on an additive white gaussian noise (AWGN) channel like in [17]. The transceiver is also provided with an equalizer, so unlike [6], the design is self-sufficient and is capable of obtaining the channel state information.

Figure 5 gives more information about the underlying hardware layer.

Data buffering between the software and hardware layers is executed by means of double port RAMs available in the FPGA. Each port of the RAM is controlled exclusively by the software or the hardware layer, and blocking mechanisms have been provided so that no layer can gain access to the RAM while the other layer is using it.

Matlab/Simulink provide a data stream that contains the source random bits and the transmitted PPDU signal with the *received signal strength indicator* (RSSI). 12 bits are used to represent the I and Q components of the PPDU. The RSSI level is transmitted using 8 bits.

The RTL description of the hardware layer stores the source random bits in a temporal buffer and sends the PPDU and RSSI signals to the transceiver. The decoded bit stream sequence is compared on real time with that stored in the temporal buffer and the number of erroneous bits are counted. The error count along with other receiver parameters are stored in a specific format and sent back to Matlab with the output double port RAM buffer.

IV. Transceiver parameter study

Given a software model of the transceiver architecture, finding a balance between precision in the operations and area consumption is sometimes a time consuming task. Other times adjusting the size of internal accumulators depends on the working conditions of the device (channel characteristics and behaviour of different submodules of the entity). This section explores the use of the verification platform as a quantization effect analyzer: due to the time reduction hardware-in-the-loop simulations can obtain it seems logical to use them in tune-up stages.

An analysis of the quantization effects will be carried out at the following points in the Viterbi decoder:

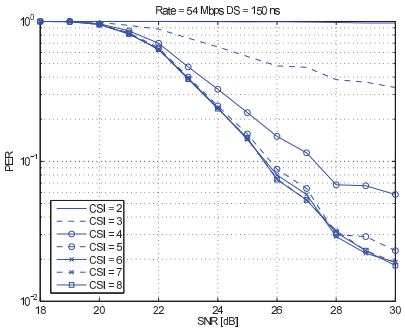


Fig. 6. Influence of the precission in CSI over the PER of the system

1) Number of bits used to express the channel state information (CSI)
2) Number of bits the ACSU accumulator is extended
3) Traceback depth of the decoder

The first two parameters have a greater impact on the logic consumed by the decoder as they control the size of several adders and comparators inside the design. The traceback depth parameters controls the memory requirements of the *Survivor Management Unit* (SMU) [14]. Since a *register-exchange* architecture has been used, the traceback depth of the decoder will significantly modify the quantity of flip-flops and multiplexers required by the SMU.

Taking into account the simulation flow introduced in figure 5 several System Generator models have been synthesized, one for each combination of parameters in the transceiver that are going to be analyzed. The methodology employed has been as follows. First, the decoder configuration that maximized the area consumption of the transceiver in the FPGA was chosen. For the Spartan-3A DSP 1800 used in this work, the maximum values for the CSI quantization, ACSU accumulator extension and traceback depth were respectively 8, 7 and 60. The parameters were analyzed one by one, keeping the others constant.

The architectures were tested under the highest rate available on the 802.11a standard using a multipath channel with a delay spread of 150ns. The signal to noise level was ranged from 18 to 30dBs and the frequency offset was randomly selected between $\pm 120ppm$ with a 20MHz clock. Each SNR point was simulated 10^3 times and the PPDUs for a given channel configuration were given to all architectures that were being tested.

The simulation results for the quantization of the CSI valued are depicted in figure 6.

As shown in figure 7 the hardware resources consumption of the decoder increases linearly with the number of bits used to represent the CSI. Adding an extra bit requires around 400 extra LUTs and 100 flip-flops from the FPGA. Since no

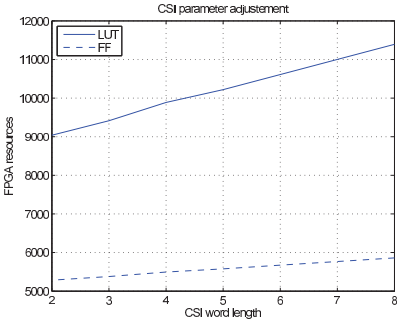


Fig. 7. Hardware resource consumption due to quantization in the CSI

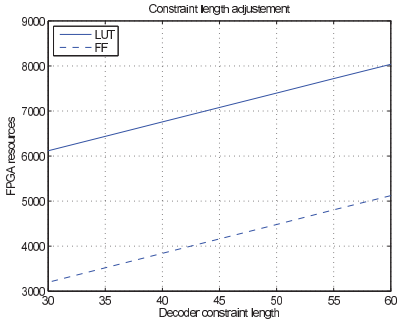


Fig. 9. Hardware resource consumption due to the traceback length

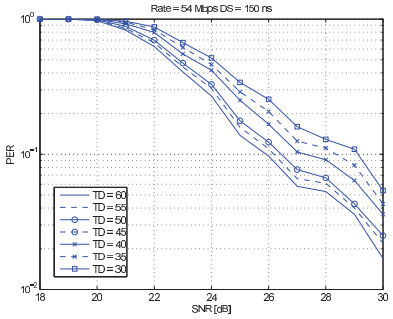


Fig. 8. Influence of the traceback depth of the decoder over the PER of the system

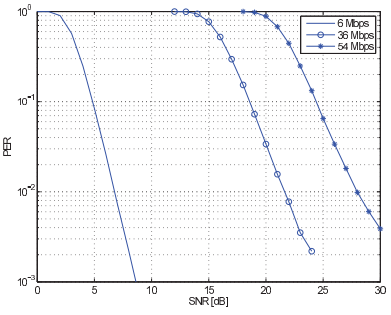


Fig. 10. PER of the transceiver

significative decoding gain is achieved for CSI quantization levels above 5 bits this quantity has been used to represent the value of the CSI.

Simulations have shown that no performance boost is obtained if the registers of the ACSUs are extended with more than 2 bits. Consequently the ACSU register size has been increased with only 2 bits.

Finally figure 8 shows the influence of the traceback depth of the decoder on the performance of the transceiver. As it can be seen increasing the traceback length in 5 units supposes a decoding gain of around 0.5 dB.

The influence of the traceback depth value on the decoder's hardware consumption is depicted in figure 9. LUT and flip-flop requirements rise at a rate of around 65 extra components per increased unit in the traceback depth. Due to the impact this parameter has on the overall performance of the transceiver the traceback of the decoder has been kept to 60.

V. Conclusions

The transceiver architecture performance has been thoroughly analyzed after the optimum decoder parameters were obtained in section IV. 6Mbps, 36Mbps and 54Mbps rates where simulated. The multipath channel had a random delay spread with a uniform distribution between 50 and 150ns. The frequency offset was also randomly selected from a uniform distribution between 0 and $\pm 120 ppm$ with a sampling clock speed of 20MHz. Each point in the PER curves of figure 10 was obtained after 10^5 simulations with PPDUs with 2000 data bits.

The PER of the transceiver architecture presented in this paper is around 4dBs better than those reported in [18] and in [19].

The mean time to run 10^5 simulations and obtain a single PER point varied depending on the number of OFDM symbols that contained each PPDU. 6Mbps data rates required 80 minutes to obtain a PER point, 36Mbps data rates required

65 minutes and 54Mbps data rates required 60 minutes. These times cover the PPDU generation and channel modelling in software, hardware-in-the-loop execution in real time, and data analysis in Matlab. In contrast, a single RTL simulation of a 54Mbps data reception in Modelsim required around 50 seconds using a Core2-Quad processor at 2.5Ghz with 2GB of RAM, which covers the data vector generation in Matlab, execution time in Modelsim and testbench result analysis in Matlab. Therefore, the fast verification platform can reduce the simulation time in a factor of at least 10^3.

REFERENCES

[1] G. E. Moore, "Cramming More Components Onto Integrated Circuits," *Proceedings of the IEEE*, 1965.

[2] S. Niar and N. Inglart, "Rapid Performance and Power Consumption Estimation Methods for Embedded System Design," *Processdings of the Seventeenth IEEE International Workshop on Rapid System Prototyping*, 2006.

[3] S. Sjoholn and L. Lindh, "The need for Co-Simulation in ASIC-verification," *EUROMICRO 97. 'New Frontiers of Information Technology', Proceedings of the 23rd EUROMICRO Conference*, 1997.

[4] D. Amos, A. Lesea, and R. Richter, *FPGA-Based Prototyping Methodology Manual*. Synopsys, 2011.

[5] J. Ou and V. K. Prasanna, "MATLAB/Simulink Based Hardware/Software Co-Simulation for Designing Using FPGA Configured Soft Processors," *Processdings of the 19th IEEE International Parallel and Distributed Processing Symposium*, 2005.

[6] A. Alimohannad, S. F. Fard, and B. F. Cockburn, "FPGA-based Accelerator for the Verification of Leading-Edge Wireless Systems," *46th ACM/IEEE Design Automation Conference*, 2009.

[7] J. Xing, S. Liu, and W. Zhao, "FPGA-Accelerated Real-time Colume Rendering for 3D Medical Image," *3rd International Conference on Biomedical Engineering and Informatics*, 2010.

[8] F. Jun and L. Shaobin, "Application of FPGA to accelerate plasma FDTD Algorithm," *3rd IEEE International Symposium on Microwave, Antenna, Propagation and EMC Technologies for Wireless Communications*, 2009.

[9] Y. A. Chapuis, L. Zhou, D. Casner, H. Ai, and Y. Hervé, "FPGA-in-the-Loop for Control Emulation of Distributed MEMS Simulation using VHDL-AMS," *First Workshop on Hardware and Software Implementation and Control of Distributed MEMS*, 2010.

[10] Y. Guo and D. McCain, "Compact Hardware Accelerator for Functional Verification and Rapid Prototyping of 4G Wireless Communication Systems," *Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers*, 2004.

[11] X. Ling, Z. Li, J. Hu, and S. Wu, "HW/SW Co-Simulation Platforms for VLSI Design," *IEEE Asia Pacific Conference on Circuits and Systems*, 2008.

[12] *IEEE Standard 802.11a-1999, "Supplement to Information Technology—Telecomm. and Information Exchange between Systems—Local and Metropolitan Area Networks-Specific Requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: High Speed Physical Layer(PHY) in the 5 GHz Band"*.

[13] T. K. Moon, *Error Correction Coding: Mathematical Methods and Algorithms*. Wiley-Interscience, 2005.

[14] F. Angarita, M. Canet, T. Sansaloni, and J. Valls, "Architectures for the Implementation of a OFDM-WLAN Viterbi Decoder," *Journal of Signal Processing Systems, Vol. 52, pp. 35-44*, 2008.

[15] *System Generator for DSP. User Guide.*

[16] A. P. Bob O'Hara, *802.11 Handbook. A Designer's Companion*. Standards Information Network IEEE Press, 1999.

[17] V. Singh, A. Root, E. Hemphill, N. Shirazi, and J. Hwang, "Accelerating Bit Error Rate Testing Using a System Level Design Tool," *Proceedings of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2003.

[18] Y. Jung, S. Noh, H. Yoon, and Jaeseok, "Implementation of Wireless LAN Baseband Processor Based on Space-Frequency OFDM Transmit Diversity Scheme," *IEEE Transactions on Consumer Electronics*, 2005.

[19] A. Doufexi, S. Armour, M. Butler, A. Nix, and D. Bull, "A Study of the Performance of HIPERLAN/2 and IEEE 802.11a Physical Layers," *2001 Vehicular Technology Conference*, 2001.

Implementation of a Zero-Second-IF Transmitter for Wide-Band Millimeter-Wave Links

A. Rezola, A. Alonso, J.F. Sevillano, I. Gurutzeaga, R. Berenguer and I. Vélez

XXX Conference on Design of Circuits and Integrated Systems, Estoril, Nov 2015

Implementation of a Zero-Second-IF Transmitter for Wide-Band Millimeter-Wave Links

Ainhoa Rezola, Aritz Alonso, J.F. Sevillano, Iñaki Gurutzeaga, Roc Berenguer and Igone Vélez
Electronics and Communications Department,
Centro de Estudios e Investigaciones Técnicas (CEIT), 20018 San Sebastián, SPAIN
Email: {argarciandia,aadomingo,jfsevillano,iguruceaga,rberenguer,ivelez}@ceit.es
Electrical, Electronic and Control Engineering Department,
Technology Campus of the University of Navarra (TECNUN), 20018 San Sebastián, SPAIN

***Abstract*—The growing demand for ubiquitous broadband communication has motivated the deployment of ultra high-speed communication systems. In order to achieve Gigabit data rates, the underlying backhauling network infrastructure demands wideband and high-order modulations in the E-band. This paper considers the design of a transceiver able to provide a data rate of 10Gbps for the backhaul of the future mobile network, with a signal bandwidth of 2GHz and 64-QAM modulation. The article describes the hardware implementation of both the analog front-end and the digital base-band processing of a modulator as part of an E-Band transceiver that is able to achieve the required capacity.**

***Keywords*—Mobile backhaul, millimeter-wave, transceiver, wideband**

I. Introduction

With the emergence of high speed wireless devices, such as smart phones and tablet computers, network subscribers demand that their network operators make it possible for them to use their handsets in the same way they use their Internet connection at home or in the office. This translates into a demand for high and peaky bandwidth, which is stressing current network infrastructures. Network operators are trying to cope with this demand for bandwidth in two ways [1]. On the one hand, they are transitioning from 3G to 4G/LTE to achieve higher spectral efficiency. On the other hand, they are increasing the density of the cells, which leads to smaller cells where interference among cells needs to be properly handled.

Higher spectral efficiency requires more capacity in the backhaul network. The increase in the density of the cells will result in an increasing number of sites that need to be connected, where short links will be the most common scenario. Furthermore, new network architectures are being considered with centralized base-band units for efficient and dynamic multi-cell operation. In this centralized radio access network (C-RAN) approach, the base-band processing unit (BBU) of the base-station is separated from the radio unit (Remote Radio Head, RRH) and the communication between RRH and BBU is a digital interface (CPRI or OBSAI), requiring capacities of up to 10Gbps [2]. This kind of architecture poses big challenges to the backhaul network for capacity and latency.

Microwave links are widely used as mobile backhaul solutions, as they lower the overall capital (CAPEX) and operating (OPEX) expenditures [3], [2]. The E-Band (71-76GHz, 81-86GHz) offers enough bandwidth to support the Gbps capacity required for the future backhauling network. This has motivated the research into and development of transceivers in the E-Band. Nowadays, commercial equipment is available that achieves data rates in the 1Gbps to 2.5Gbps range (e.g.: [4], [5], [6]). In order to achieve those data rates, two options are possible: either take advantage of the available wide-bandwidth (approx. 5GHz for each sub-band) and transmit low order modulated signals (typically BPSK or QPSK), or increase the modulation order and reduce the occupied bandwidth, thereby increasing the spectral efficiency. Most of the approaches found in the literature for increasing link capacities achieve spectral efficiencies not higher than 2bits/s/Hz (e.g.: [7], [8], [9]). In [10], a data-rate of 6Gbps with a spectral efficiency of 2.4bit/s/Hz was demonstrated using 8-PSK modulation over a 2.5GHz bandwidth. The approach to attaining this data rate relies on a frequency-domain channel multiplexing technique where four sub-bands of smaller bandwidth are transmitted in parallel. Similarly, [11] achieves a capacity of 10Gbps with a spectral efficiency of 3.2bit/s/Hz by using 8 parallel channels of smaller bandwidth. Nevertheless, to the best of the authors' knowledge, there is no work in the literature that deals with the highly challenging objective of implementing a 10Gbps backhaul transceiver in the E-Band and thus achieving high spectral efficiency. A communication data rate at 10Gbps implies an improvement that is 8 times over the average communication data rate (1.25Gbps) of commercial links operating in the E-Band.

This paper considers the design of a transceiver using a bandwidth of 2GHz. This bandwidth enables network operators to easily fit two full-duplex links in the same site. In order to achieve the required capacity of 10Gbps, a spectral efficiency of at least 5bit/s/Hz is needed, which calls for a 64-QAM modulation. Designing transceivers that employ higher-order modulations over a wide bandwidth to achieve 10Gbps is very challenging. Higher-order modulation requires digital processing in the base-band to provide enough throughput to perform all the codec and modem functions. Digital-to-analog (DAC) and analog-to-digital (ADC) converters that are able to provide enough sampling rate are needed. In this context, zero-second-IF architectures are attractive because they enable the lowest sampling rate.

In general, higher-order modulations require higher performance (better linearity, less phase-noise, reduced noisefigure, better image rejection, etc.) from the analog front-end components. Therefore, the different impairments introduced in the

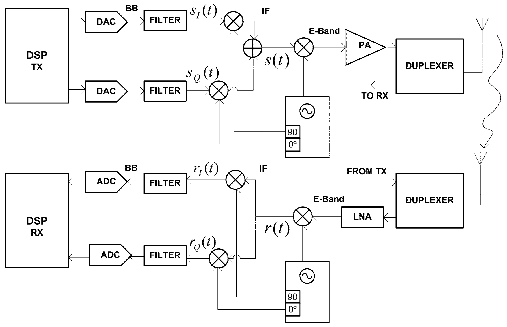


Fig. 1. Architecture of the point-to-point communication system. The upper part shows the block diagram of the transmitter part of the transceiver and the lower part shows the block diagrams of the receiver part of the transceiver.

analog front-end need to be carefully analyzed.

If the digital signal processing of the transceiver is to be implemented in a Field Programmable Gate Array (FPGA), the required throughput poses a big challenge for the design. The sampling rate to/from the RF front-end will be above 2GHz. However, current FPGAs are not able to work at this data rate [12]. Therefore, digital base-band processing needs to be implemented using parallel architectures. The use of parallel architectures not only increases the use of resources, but it also complicates the data flow. Despite the fact thath specific resources of the FPGA can be clocked at near 400MHz, making configurable logic work at frequencies close to those speeds requires a very careful design. Moreover, making a crowded FPGA work with a very high frequency clock and duty cycle results in very high power consumption.

The remainder of this paper is structured as follows. Section II presents the description of the wideband transceiver architecture. The implementation of the analog front-end is described in Section III, while Section IV refers to the digital base-band of the transceiver. In Section V the implementation setup for the transceiver system and the corresponding performance results are presented and analyzed. Finally, Section VI summarizes the conclusions.

II. System design

In order to address new applications for the future back-hauling networks, a point-to-point microwave link in the E-Band using a 64-QAM modulation with a signal bandwidth of 2GHz is considered. Figure 1 shows the proposed transceiver (TRx) architecture for a point-to-point microwave link in the E-Band.

As shown, the transmitter (Tx) front-end consists of an IQ modulator that up-converts the base-band I and Q channels to an intermediate frequency (IF) of 17.5GHz. After combining the I and Q channels, the IF signal is up-converted to the E-Band by means of the millimeter-wave (mmW) mixer. Finally, the wideband mmW power amplifier (PA) is used to amplify and transmit the mmW signal. The receiver (Rx) front-end consists of a wideband Low Noise Amplifier (LNA), which receives and amplifies the signal at the E-Band. After the LNA, a first mixer down-converts the mmW signal to the same IF as in the Tx. This way, the same PLL can be re-used for the Tx and the Rx. Finally, an IQ demodulator down-converts the IF signal to 0-Hz.

This architecture presents a good balance between different design aspects, and it enables the minimization of the sampling frequency of the DAC and ADC converters. Nowadays we can find commercial DACs and ADCs able to provide sampling rates in the range of 2.5Gsps and even higher, which is sufficient for practical implementation of the zero-second-IF architecture [13].

Due to the high channel bandwidth, the architecture depicted in Figure 1 presents a good balance between the DAC and the ADC requirements and the transceiver complexity. The use of other architectures, such as non-zero-second-IF, would require very high performance ADCs or DACs, as well as highly selective reconstruction filters to achieve a practical implementation of base-band and image rejection filters in the analog front-end.

However, the use of a zero-second-IF architecture presents well-known issues that should be addressed in order to avoid degrading the performance of the transceiver. This architecture is subject to DC offsets due to the self-mixing of the LO leakage, corruption of the signal close to DC due to flicker noise, and AC coupling between different base-band components, which lead to a high pass filtering of the signal. Another important source of signal corruption is the IQ imbalance both in the Tx modulator and the Rx demodulator. This issue has been a major obstacle in discrete designs, but it tends to decrease with higher levels of integration.

The first problem was addressed in [13]. In order to avoid placing useful signal near DC, the solution adopted in [13]

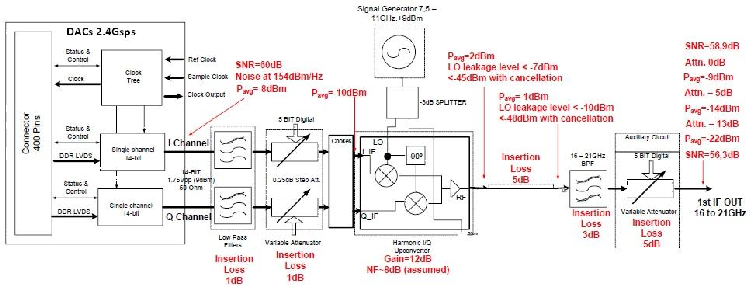


Fig. 2. Block diagram of the implemented I/Q up-converter circuit with expected power levels and signal-to-noise ratio values after each block.

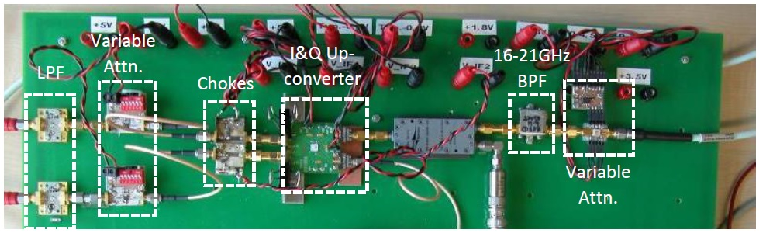


Fig. 3. Picture of the implemented I/Q up-converter circuit.

consists of splitting the signal to be transmitted in two sub-bands. One sub-band (DS0) is IQ modulated using a signal bandwidth of 1 GHz and centered at 500 MHz, and the other sub-band (DS1) uses another signal with 1 GHz of bandwidth and centered at -500 MHz. Both sub-bands have a 64-QAM data modulation and a raised-cosine pulse shaping with 15% roll-off. A symbol rate of 869.56MHz is used, yielding a total signal with a bandwidth of 2GHz and no information bearing components near DC. This solution relaxed the requirements near DC for the analog components, but at the expense of increasing the complexity of the base-band processing.

The second issue, IQ imbalance, was investigated in [14] and [15]. This impairment is caused by mismatches in the amplitude and phase responses of the I and Q signal paths. Instead of tightening the specifications and tolerances of the analog circuitry, a cost-effective solution proposed in these articles was to compensate the IQ imbalance in the digital domain by using signal processing algorithms. The solution involves using novel signal processing methods to automatically compensate for the aforementioned issues.

In the following sections both the analog front-end and the transmitter digital base-band of the transceiver are described, analyzing how they have been implemented in hardware.

III. Analog Front-End Hardware Implementation

This section presents the I/Q up-converter prototype implemented for a wideband E-Band transmitter. Fig. 2 shows the block diagram of the prototype with the expected power levels and signal-to-noise ratio values after each block, while Fig. 3 shows a picture of the fabricated prototype.

As shown in Fig. 2 the I and Q channels coming from the DACs go through a Low Pass Filter (LPF) with a passband bandwidth of 1GHz. The measured insertion loss of the filter was lower than 1dB and the filter rejection at 1.6GHz was higher than 30dB. Four samples of each filter were measured. No significant differences were observed between the different filters in terms of amplitude and phase, which is important in order to estimate the gain and phase imbalance of the signal path in the IQ modulator.

The next component in the Tx analog signal chain is the base-band attenuator. By selecting the attenuation value in each

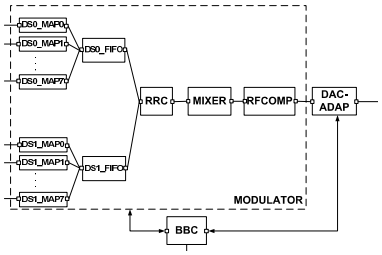


Fig. 4. Modulator block diagram

path (I/Q), non-frequency selective amplitude imbalances can be corrected in steps of 0.25dB. It will also be possible to adjust the signal level in the Tx chain. The 3-dB bandwidth of the attenuator is well above 1GHz.

The next component in the Tx analog signal chain is the I/Q modulator. The biasing voltages and current consumption are: Vgg = -0.8V and VDD1 = VDD2 = VDD3 = 5V. Total Idd is approximately 250mA. To power the up-converter it is important to follow the standard typical sequence of MESFET devices. The output measured for 2dB bandwidth centered at 18.5GHz is bigger than 6GHz, which is sufficient for the application under consideration. The measured gain is approximately 10.5dB and the 2xLO-RF isolation is around 15dB. These measurements were taken without the input hybrid for broadband operation. A 90 hybrid is used to measure the image rejection response of the I/Q modulator. The hybrid generates a 90 phase shift from 62.5 to 125MHz. The image rejection response modulated by the periodic behavior of the hybrid shows an image rejection (IMRR) value of 20dBc.

Next an inter-digital band-pass filter from 16GHz to 21GHz is implemented on a low loss microwave substrate. Measurements show a good rejection around 20dB up to a frequency of 50GHz. Finally, an attenuator after the band-pass filter is used to fix the signal level for the mmW part while keeping a high SNR. A broadband variable attenuator from DC to 30GHz, mounted on an evaluation board, is used. An auxiliary circuit is employed to generate two complementary DC voltages, in order to enhance the linearity and return losses of the attenuator. Measurements show that the attenuation range is quite linear and extends from -3dB to -43dB with DC control signals adequately configured.

IV. Digital base-band hardware implementation

Fig. 4 shows the architecture of the transmitter digital base-band (TX-DBB). TX-DBB receives already encoded bits to be transmitted in eight parallel buses. Each bus has two parts; one indicates the modulation order, while the other contains the bits to be mapped. This feature is attractive in that it enables different modulations in the preamble signaling field, as well as in the payload of a frame.

The digital base-band modulator produces the samples of the signal to be transmitted. The DAC-ADAP block handles all the clocking and data synchronization issues between the modulator and the DACs. The base-band controller (BBC) block, which consists of an external microcontroller, receives control commands and configuration information and, according to what is received, it generates different internal signals to send and retrieve information to/from the other blocks.

The modulator is composed of eight mappers in parallel, a root raised cosine (RRC) filter, a mixer and a RF compensator (RFcomp). A brief description of each each block follows.

A. Mapper

The mappers perform the mapping according to the required modulation. Supported modulations are QPSK, 8-PSK, 16-QAM, 32-QAM and 64-QAM. The mappers are implemented by means of access to RAM memories. This access is fast and minimum logic and control are needed. Moreover, it avoids the need for additional resources and the power rail of the RAMs is different from the rest of the logic.

At the output of the mappers, there are a couple of FIFOs that handle the fact that bits may arrive in bursts. This makes the IQ symbols at the output of the mapper also arrive in bursts. These FIFOs buffer the IQ symbols and make them available at the input of the next component when a request is made. The FIFOs also prevent the situation where data arrives and cannot be buffered.

B. RRC

The RRC filters receive IQ symbols and produce digital samples of the pulse-shaped base-band signal of each digital sub-band with a roll-off of 15%. The digital samples are produced with a sampling frequency equal to $F_{s@DAC}$ and are they sent to the next module at a rate of 8 samples per clock cycle of $F_{s@MOD}$, where

$$F_{s@MOD} = \frac{F_{s@DAC}}{8}. \quad (1)$$

It is common for high speed communication systems that signal generation and digital-to-analog conversion are done under independent, incommensurate clock domains, which introduces an extra degree of complexity to the design of this filter, since interpolation becomes a necessity. [16] describes more thoroughly how this issue has been handled.

C. Mixer

The mixer receives the pulse-shaped base-band signal from each digital sub-band. This block modulates each digital sub-band to subcarriers at 500MHz and -500MHz and adds the resulting signal to produce the base-band signal to be transmitted. The samples of the output signal of this block maintain the rate of the samples.

The implementation of this block consists of eight mixers in parallel, each one working at a sampling rate of $F_{s@MOD}$. Each mixer has its own numerically controlled oscillator (NCO), so that each sample is a certain phase offset away from the others. This way, the mixer outputs samples at $F_{s@DAC}$.

D. RFcomp

The RFcomp block predistorts the signal received from the mixer to compensate for the RF impairments. This block modifies the signal to be transmitted to account for part of the impairments introduced by subsequent processing in the DACs and the analog front-end. The samples of the base-band predistorted signal are output, maintaining the rate of the samples.

In [14] it was concluded that one of the performance-critical effects of interest in zero-second-IF transceiver architectures is IQ imbalance, which is caused by mismatches in the amplitude and phase responses of the I and Q signal paths.

The RFcomp block consists of a digital FIR filter compensator following the structure proposed by [17].

V. Performance evaluation

The measurement setup shown in Fig. 5 consists of an IF board, a FPGA prototyping board, a high speed DAC board, a power spectrum analyzer (PSA) and software running on a PC.

The digital base-band processing consists of a Virtex-7 FPGA prototyping board (VC707) [12]. The FPGA implements the digital base-band processor described in Section IV. Additionally, a data generator was implemented. This data generator is based on RAMs and some control hardware. The RAMs store tuples of data bits and the modulation order, which are read cyclically and fed to the base-band processor in Section IV.

Control of TX-DBB is performed externally, in software run on a PC. The software is responsible for producing the appropriate control and configuration signals based on the data collected from the analog front-end and sending them to TX-DBB.

At the output of the FPGA, a couple of high speed DACs are responsible for converting the complex base-band signal to the analog domain. The DAC board used in the prototype consists of an FMC230 board. The FMC230 has some programmable registers in order to enable or disable the DACs, as well as select an internal or external clock for them. The FMC provides information about several power supply voltages on the board as well as the temperature. This information is received at TX-DBB via I2C bus. The DACs work at a sampling rate $F_{s@DAC}$ of 2.4 GHz.

The outputs of the DACs are connected to the IF board. At the analog front-end the signal is first passed through the analog reconstruction filters to remove the signal replicas at multiples of the digital sampling rate. Finally, the IQ modulator translates the base-band signal to 17.5GHz, as explained in Section III.

At the output of the analog front-end the IF signal can be monitored in the PSA.

Fig. 6 shows the power spectrum of the signal at IF captured from the PSA. The two digital sub-bands are clearly visible in the spectrum. Each sub-band has a bandwidth of 1GHz and the relative signal levels are in agreement with the specification, since the signal fits the mask given by the standard ETSI EN 302 217-2-2 [18], [19].

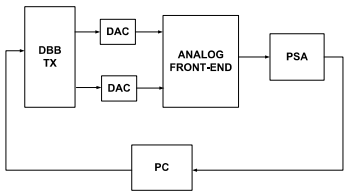


Fig. 5. Implementation setup

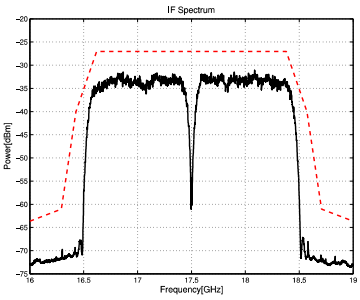


Fig. 6. IF power spectrum of the transmitted signal and its mask.

To the best of the authors knowledge, no work that is able to achieve a data rate of 10Gsps using a signal bandwidth of 2GHz and 64-QAM modulation has been reported. The approaches found in the literature ([7], [8], [9], [10]) use either a narrower bandwidth or a lower-order modulation to achieve high data rates.

The FPGA resource utilization is summarized in Table I, while Table II reflects the different consumptions measured from it. In order to measure the consumptions, Texas Instruments Fusion Digital Power Designer is used, which tests the FPGA, was used. This instrument validates whether the power conversion falls within acceptable Vout and Iout limits in a graphical user interface, configuring common operating characteristics and warning and fault thresholds.

VI. Conclusion

The design of an E-band transceiver able to provide 10Gbps within a bandwidth of 2GHz in E-band was considered in this paper as one of the building blocks of the future mobile back-

FPGA resource utilization		
Resource	**Utilized**	
Slices	23944	31.54%
Block RAMs	320	31.06%
DSP48E1	1328	47.42%

TABLE I. FPGA RESOURCE UTILIZATION

FPGA consumption		
Symbol	**Description**	**Power [W]**
VCCINT	Internal supply voltage	17.34
VCCAUX	Auxiliary supply voltage	0.45
VCC3V3	DAC supply voltage	5.82
VADJ	FPGA-DAC interface supply voltage	0.76
VCCBRAM	Supply voltage for the block RAM memories	0.25

TABLE II. FPGA CONSUMPTION

haul network. A zero-second-IF architecture was described to provide a good balance between the requirements of the digital and analog parts of the transceiver. The design of a transceiver with such features constitutes major progress beyond the state-of-the-art and, therefore, faces serious challenges, such as the implementation of the analogue RF front-end and the digital base-band processing.

An analog IQ Modulator prototype using discrete components was implemented. The IQ channels were up-converted from base-band to a channel located at 17.5 GHz. The 2dB bandwidth of the modulator was higher than 5GHz and the IMRR was 20dBc. These characteristics allowed the analog modulator to be used as part of the E-Band transceiver, allowing a data rate of 10Gsps.

The digital base-band was then implemented in a prototyping FPGA board using commercially available DAC boards. The digital base-band processing was implemented using parallelization. It was shown that parallel signal process-ing is necessary, since the state of the art technology is not able to serially generate and convert the signal with the required bandwidth, 2GHz, and sampling rate. The system's operation was illustrated in the article.

In summary, the article presented the design and implementation of a transceiver that has the features required to meet the necessities of future network backhauling infrastructures. Such features include high spectral efficiency, high data rate, cost effectiveness, fully implemented digital base-band processor and analog front-end, among others.

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Communitys FP7/2007-2013 Framework Programme under grant agreement no. 317957. Consortium: CEIT, Fraunhofer IIS, Alcatel-Lucent, CEA-Leti, IXYS, Silicon Radar, ST, Sivers IMA, OTE.

REFERENCES

[1] M. G. Frecassetti, "Mobile backhaul network evolution," presented at the XXVIII Conference on Design of Circuits and Integrated Systems (DCIS), 2013.

[2] J. Segel and M. Weldon, "Lightradio whitepaper 1: Technical overview," Alcatel-Lucent, Tech. Rep., 2011.

[3] D. Mavrakis, C. White, and F. Benlamlih, "Last mile backhaul options for west european mobile operators," Informa Telecoms & Media, Tech. Rep., 2010.

[4] "L2710 gigabit radio," Loea Corporation.

[5] "Flex4g," BridgeWave Communications, Inc.

[6] "Full outdoor alfoplus80 series brochure," SIAE MICROELETTRONICA S.p.A.

[7] Z. He, J. Chen, Y. Li, and H. Zirath, "A novel fpga-based 2.5gbps d-qpsk modem for high capacity microwave radios," in *Communications (ICC), 2010 IEEE International Conference on*, 2010, pp. 1–4.

[8] I. Sarkas, S. Nicolson, A. Tomkins, E. Laskin, P. Chevalier, B. Sautreuil, and S. Voinigescu, "An 18-gb/s, direct qpsk modulation sige bicmos transceiver for last mile links in the 70-80 ghz band," *Solid-State Circuits, IEEE Journal of*, vol. 45, no. 10, pp. 1968–1980, 2010.

[9] J. Chen, Z. He, L. Bao, C. Svensson, Y. Li, S. Gunnarsson, C. Stoij, and H. Zirath, "10 gbps 16qam transmission over a 70/80 ghz (e-band) radio test-bed," in *Microwave Integrated Circuits Conference (EuMIC), 2012 7th European*, 2012, pp. 556–559.

[10] V. Dyadyuk, J. Bunton, J. Pathikulangara, R. Kendall, O. Sevimli, L. Stokes, and D. Abbott, "Mobile backhaul network evolution," *Microwave Theory and Techniques, IEEE Transactions on*, vol. 55, no. 12, pp. 2813–2821, 2007.

[11] M.-S. Kang, B.-S. Kim, K. S. Kim, W.-J. Byun, and H. C. Park, "16-qam-based highly spectral-efficient e-band communication system with bit rate up to 10 gbps," *ETRI Journal*, vol. 34, no. 5, pp. 649–654, 2012.

[12] *VC707 Evaluation Board for the Virtex-7 FPGA*, Xilinx, 4 2015, v1.6.

[13] A. Rezola, D. Del Rio, J. F. Sevillano, M. Leyh, R. Berenguer, and I. Velez, "Impact of ac coupling on zero-if architectures for wide-band millimeter-wave gigabit transmitters," in *European Wireless 2014; 20th European Wireless Conference; Proceedings of*, May 2014, pp. 1–6.

[14] A. Rezola, J. Sevillano, R. Berenguer, I.Velez, M. Leyh, M. Lorenzo, and A. Vargas, "Non-frequency-selective i/q imbalance in zero-if transceivers for wide-band mmw links," in *The Tenth International Conference on Wireless and Mobile Communications ICWMC*, 2014.

[15] A. Rezola, J. Sevillano, M. Leyhz, M. Lorenzoz, R. Berenguer, A. Vargas, and I. Velez, "Iq imbalance in heterodyne transceivers with zero-second-if for wide-band mmw links," *IARIA Journal*, 2015.

[16] A. Alonso, J. Sevillano, and I. Velez, "Parallel implementation of a sample rate conversion and pulse-shaping filter for high speed backhauling networks," in *Design of Circuits and Integrated Circuits (DCIS), 2014 Conference on*, Nov 2014, pp. 1–6.

[17] M. Valkama, M. Renfors, and V. Koivunen, "Advanced methods for i/q imbalance compensation in communication receivers," *Signal Processing, IEEE Transactions on*, vol. 49, no. 10, pp. 2335–2344, Oct 2001.

[18] *Fixed Radio Systems; Characteristics and requirements for point-to-point equipment and antennas; Part 1: Overview and system-independent common characteristics*, ETSI EN 302 217-1, Sept. 2012.

[19] *Fixed Radio Systems; Characteristics and requirements for point-to-point equipment and antennas; Part 2-2: Digital systems operating in frequency bands where frequency co-ordination is applied; Harmonized EN covering the essential requirements of article 3.2 of the R&TTE Directive*, ETSI EN 302 217-2-2, Sept. 2012.

A.2 NATIONAL CONFERENCE PAPERS

- A. Alonso, A.Irizar, A. Cortés, J.A. Paredes and M. Turrillas. Hardware-in-the-Loop based fast system verification and parameter fine tuning platform. *50TH ANNIVERSARY CONFERENCE - ENGINEERING: SCIENCE AND TECHNOLOGY*, Donostia-San Sebastián, May 2012.

- A. Alonso and A. Irizar. An FPGA based OFDM baseband processor platform for fast high and low level system receiver algorithm prototyping: a case study for 802.11a. *Celebration of the* 500^{th} *PhD Dissertations of TECNUN*, Donostia-San Sebastián, November 2013.

Hardware-in-the-Loop based fast system verification and parameter fine tuning platform.
A. Alonso, A.Irizar, A. Cortés, J.A. Paredes and M. Turrillas
50TH ANNIVERSARY CONFERENCE - ENGINEERING: SCIENCE AND TECHNOLOGY, Donostia-San Sebastián, May 2012

Hardware-in-the-Loop based fast system verification and parameter fine tuning platform

Aritz Alonso[1, a], Andoni Irizar[1, b], Ainhoa Cortés[1, c], Jorge Alberto Paredes [1, d], Marta Turrillas[1, e]

[1]Department of Electronics & Comunications

CEIT and Tecnun (University of Navarra)

Manuel de Lardizábal 15, 20018 San Sebastián, Spain

[a]aadomingo@ceit.es, [b]airizar@ceit.es, [c]acortes@ceit.es, [d]jparedes@ceit.es, [e]mturrillas@ceit.es

Keywords: OFDM, Hardware-in-the-Loop, co-simulation, verification.

Abstract. The time required for system analysis and verification has invariably become the bottleneck of the development process as designs become more complex. Methodologies involving configurable hardware have proven to be the only ones to break the inverse relationship between accuracy in simulation and performance in verification. In this work we present a Hardware-in-the-Loop based platform that substantially reduces the time required for system verification and parameter fine tuning. The verification platform has been used to evaluate the performance of the physical layer of a WLAN 802.11a compliant transceiver. Compared to a software RTL simulation this platform reduces simulation time by a factor of at least 10^3.

Introduction

In 1965 Gordon E. Moore published [1] his well-known observation that states that the number of transistors that can be fitted inexpensively on a silicon device doubles every two years. This observation, which was introduced in the early days of the semiconductor technology, continues to hold true nowadays. Consequently, circuit designs are expected to continuously grow in complexity and include more advanced and accurate functions.

The increase in the complexity of the designs has some severe drawbacks as well: bigger projects require longer and more meticulous design processes; implementation time increases and demands a thorough verification stage. Moreover, the execution time of optimization stages where a number of parameters (memory and cache sizes …) has to be determined can become unacceptable [2].

With the improvement of the synthesis tools, the verification process has become the design bottleneck and can suppose about the 60% or the 70% of the development time [3]. Any reduction in the verification stage can lead to a considerable shorter time to market.

Many techniques have arisen to speed up simulation and reduce verification time. Among them, methodologies involving configurable hardware have proven to be the only ones to break the inverse relationship between accuracy in simulation and performance in verification stages [4]. These methodologies, also known as as *hardware-in-the-loop* simulations, lie in the principle of executing the most complex and time consuming tasks of the design on dedicated hardware while the lightest ones are executed in software.

Hardware-in-the-loop simulations could theoretically be executed at real-time speeds. However, communication between the hardware and software layers has shown to be the limiting factor of the simulation acceleration the platform can achieve [5].

Hardware-software co-simulation has been used for a variety of purposes such as hardware model verification [6] and algorithm calculation acceleration [7]. Obtaining the *packet error rate* (PER) curves of a transceiver is a time consuming task wich could be improved significantly with the use of *hardware-in-the-loop* simulations.

In this work we present a *System Generator* based hardware-software platform for fast transceiver verification. The platform uses an Ethernet connection that reaches communication speeds of up to 1 Gbps, much faster than what series connections RS232 [8] or USB 2.0 [9] can achieve. The platform is suitable for medium-size projects and requires less effort to configure and control the simulation than those of proposed works [10] [11].

The proposed hardware-software verification platform has been used to analyse the performance of a WLAN 802.11a compliant transceiver [12] measuring its PER curves under real communication channels. The platform has also been used to quickly study the impact that critical parameters have on the transceiver size and performance.

Transceiver architecture

WLAN 802.11a was one of the first standards to be built upon the now common *orthogonal frequency division multiplexing* (OFDM) scheme. OFDM distributes the information to be transmitted between a series of orthogonal carriers so that a single high-speed data stream is divided into multiple slower data-streams that are transmitted in parallel through a channel. The inverse and direct FFT algorithms have proven to be an efficient way to achieve the carrier orthogonality at the transmitter and receiver sides.

Figure 1 shows the architecture of the WLAN 802.11a compliant transceiver to be verified. As it can be seen several submodules are multiplexed in the transmission and reception chains to reduce hardware resource consumption. The FFT core is the most versatile submodule for it implements the inverse FFT algorithm in transmission mode and the direct FFT algorithm in reception mode, as well as supporting the synchronization stage.

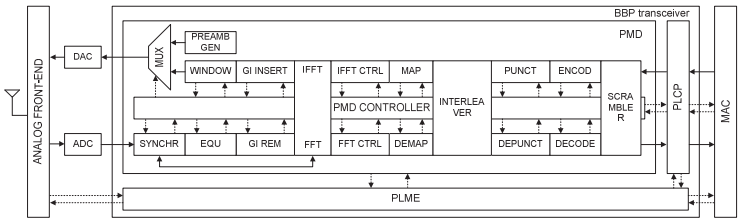


Fig. 1 - Block diagram of the transceiver architecture

Fast Verification Platform

The fast verification platform is based on System Generator [13], a component of the Xilinx ISE Design Suite that enhances Matlab's Simulink platform. Simulink offers a powerful design environment for multidomain simulation. System Generator adds a variety of new block libraries specially designed for digital signal processing.

Among its features, System Generator allows to import custom IPs in HDL and to compile designs to run them on real time under Xilinx FPGAs. Several evaluation boards are supported by default, but the environment allows any board if proper configuration files are supplied by the user and the board meets certain hardware requirements. System Generator also provides an Ethernet or JTAG based interface for communication between the hardware and software layers of the simulation which is transparent to the user.

All this reasons make System Generator an interesting tool when designing hardware-in-the-loop simulations. Figure 2 depicts the block diagram of the platform that was designed to verify the performance of hardware transceiver implementations.

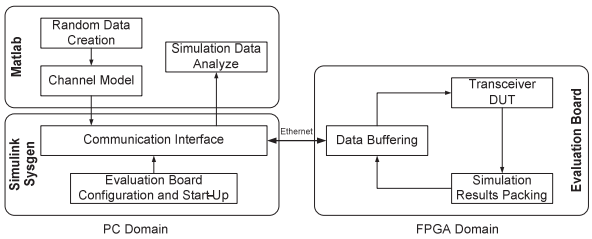


Fig. 2 - Architecture of the fast prototyping sytem

The software component in the simulation is responsible for creating *PLCP Protocol Data Units* (PPDU) with random data and implementing the WLAN multipath channel with different values of noise and delay spread. The transceiver architecture is implemented in the hardware component of the simulation. It is responsible for decoding the received PPDU and counters the number of erroneous bits to release overhead to the software layer. It also returns information about the status of the transceiver (time and frequency offset estimations …) to the software layer for statistical analysis. System Generator is responsible for waking up and configuring the evaluation board and synchronizing the software and hardware components of the simulation.

Given that the multipath channel model is implemented in software its statistical properties can get closer to a real channel than those obtained with embedded implementations. Moreover, it gives the possibility to evaluate the design in a more accurate scenario and not only on an Additive White Gaussian Noise (AWGN) channel like in [14]. The transceiver is also provided with an equalizer, so the design is self-sufficient and is capable of obtaining the channel state information as opposed to [6].

Figure 3 gives more information about the underlying hardware layer.

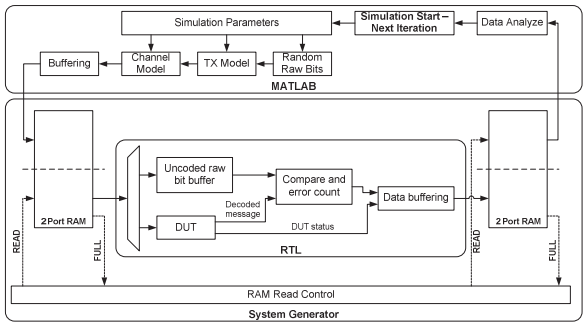


Fig. 3 - Simulation flow dissection

Data buffering between the software and hardware layers is executed by means of double port RAMs available in the FPGA. Each port of the RAM is controlled exclusively by the software or the hardware layer, and blocking mechanisms have been provided so that no layer can gain access to the RAM while the other layer is using it.

Matlab and Simulink provide a data flow that contains the source random bits and the transmitted PPDU signal with the *received signal strength indicator* (RSSI). 12 bits are used to represent the I and Q components of the PPDU and the RSSI level is transmitted using 8 bits.

The RTL description of the hardware layer stores the source random bits in a temporal buffer and sends the PPDU and RSSI signals to the transceiver. The decoded bit stream sequence is compared on real time with that stored in the temporal buffer and the number of erroneous bits is

obtained. The error count along with other receiver parameters are stored in a specific format and sent back to Matlab with the output double port RAM buffer.

Transceiver parameter study

Given a software model of the transceiver architecture, finding a balance between precision in the operations and area consumption is sometimes a time consuming task. Other times adjusting the size of internal accumulators depends on the working conditions of the device (channel characteristics and behaviour of different submodules of the entity). This section explores the use of the verification platform as a quantization effect analyser: due to the time reduction hardware-in-the-loop simulations can achieve it seems logical to use them in tune-up stages.

The parametric study was carried out to minimize the size of the Viterbi decoder available in the transceiver while at the same time keeping its decoding performance as high as possible. The parameters that were considered for the optimization were the number of bits used to represent the *channel state information* (CSI), the number of bits the *add-compare-select unit's* (ACSU) registers are extended in order to prevent overflow and the traceback depth of the decoder.

Taking into account the simulation flow introduced in figure 3 several System Generator models have been synthesized, one for each combination of parameters in the transceiver that are going to be analysed. The methodology employed has been as follows. First, an initial parameter configuration for the transceiver was chosen that at the same time could fit in the target FPGA and maximized the parameter values. For the Spartan-3A DSP 1800 used in this work, the maximum values for the CSI quantization, ACSU accumulator extension and traceback depth were 8, 7 and 60 respectively. Then the parameters were optimized one by one. When the influence of one parameter was being studied, the rest of the parameters were kept fixed to their initial value (or their optimum value if the corresponding analysis was already performed).

The architectures were tested under the highest rate available on the 802.11a standard using a multipath channel with a delay spread of 150ns. The signal to noise level was ranged from 18 to 30dBs and the frequency offset was randomly selected between ±120*ppm* with a 20MHz clock. Each SNR point was simulated 10^3 times. The same set of data was provided to all architectures under test.

Conclusions

The transceiver architecture performance has been thoroughly analysed after the optimum decoder parameters were obtained in the previous section. 6Mbps, 36Mbps and 54Mbps rates where simulated. The multipath channel had a random delay spread with a uniform distribution between 50 and 150ns. The frequency offset was also randomly selected from a uniform distribution between 0 and ±120*ppm* with a sampling clock speed of 20MHz. Each point in the PER curves of figure 4 was obtained after 10^5 simulations with PPDUs transmitting 2000 data bits.

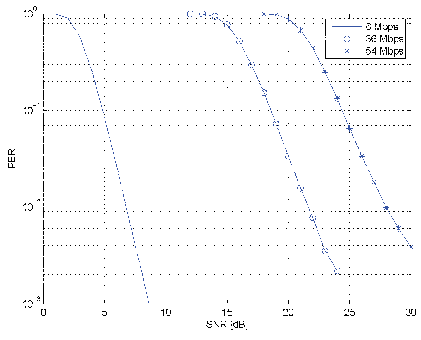


Fig. 4 - PER of the transceiver

The PER of the transceiver architecture presented in this paper is around 4dBs better than those of the works in [15] and [16].

The mean time to obtain a single PER point varied depending on the number of OFDM symbols that contained each PPDU. 6Mbps data rates required 80 minutes to obtain a PER point, 36Mbps data rates required 65 minutes and 54Mbps data rates required 60 minutes. In contrast, a single RTL simulation of the transceiver required around 50 seconds using a Core2-Quad processor at 2.5Ghz with 2GB of RAM. Therefore, the fast verification platform can reduce the simulation time in a factor of at least 10^3.

References

[1] G.E.Moore: Cramming More Components Onto Integrated Circuits (Proceeding of the IEEE, 1965)

[2] S.Niar, N.Inglart: Rapid Performance and Power Consumption Estimation Methods for Embedded System Design (Proceedings of the Seventeenth IEEE International Workshop on Rapid System Prototyping, 2006)

[3] S.Sjoholn, L.Lindh: The need for Co-Simulation in ASIC verification (Proceedings of the 23rd EUROMICRO Conference, 1997)

[4] D.Amos, A.Lesea, R.Richter: FPGA-Based Prototyping Methodology Manual (Synopsys 2011)

[5] J.Ou, V.K.Prasanna: MATLAB/Simulink Based Hardware/Software Co-Simulation for Designing Using FPGA Configured Soft Processors (Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, 2005)

[6] A.Alimohannad, S.F.Fard, B.F.Cockburn: FPGA-based Accelerator for the Verification of Leading-Edge Wireless Systems (46th ACM/IEEE Design Automation Conference, 2009)

[7] J.Xing, S.Liu, W.Zhao: FPGA-Accelerated Real-time Colume Rendering for 3D Medical Image (3rd International Conference on Biomedical Engineering and Informatics, 2010)

[8] F.Jun, L.Shaobin: Application of FPGA to accelerate plasma FDTD Algorithm (3rd IEEE International Symposium on Microwave, Antenna, Propagation and EMC Technologies for Wireless Communications, 2009)

[9] Y.A.Chapuis, L.Zhou, D.Casner, H.Ai, Y.Hervé: FPGA-in-the-Loop for Control Emulation of Distributed MEMS Simulation usign VHDL-AMS (First Workshop on Hardware and Software Implementation and Control of Distributed MEMS, 2010)

[10] Y.Guo, D.McCain: Compact Hardware Accelerator for Functional Verification and Rapid Prototyping of 4G Wireless Communication Systems (Conference Record of the Thirty-Eight Asilomar Conference on Signals, Systems and Computers, 2004)

[11] X.Ling, Z.Li, J.Hu, S.Wu: HW/SW Co-Simulation Platforms for VLSI Design (IEEE Asia Pacific Conference on Circuits and Systems, 2008)

[12] IEEE Standard 802.11a-1999, "Supplement to Information Technology—Telecomm. and Information Exchange between Systems - Local and Metropolitan Area Networks-Specific Requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: High Speed Physical Layer(PHY) in the 5 GHz Band"

[13] System Generator for DSP. User Guide

[14] V.Singh, A.Root, E.Hemphill, N.Shirazi, J.Hwang: Accelerating Bit Error Rate Testing Using a System Level Design Tool (Proceedings of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2003)

[15] Y.Jung, S.Noh, H.Yoon, H.Jaeseok: Implementation of Wireless LAN Baseband Processor Based on Space-Frequency OFDM Transmit Diversity Scheme (IEEE Transactions on Consumer Electronics, 2005)

[16] A.Doufexi, S.Armour, M.Butler, A.Nix, D.Bull: A Study of the Performance of HIPERLAN/2 and IEEE 802.11a Physical Layers (2001 Vehicular Technology Conference)

.

An FPGA based OFDM baseband processor platform for fast high and low level system receiver algorithm prototyping: a case study for 802.11a.
A. Alonso and A. Irizar
Celebration of the 500^{th} *PhD Dissertations of TECNUN*, Donostia-San Sebastián, November 2013

An FPGA based OFDM baseband processor platform for fast high and low level system receiver algorithm prototyping: a case study for 802.11a

Aritz Alonso Domingo, Andoni Irizar
Electronics and Communications Department

CONTEXT OF THIS WORK

High speed communication systems are achieving a growing interest thanks to the combined efforts of international organizations, manufacturers and end-users. Many of those systems chose Orthogonal Frequency Division Multiplexing (**OFDM**) as their modulation scheme due to its high spectral efficiency and robustness against multipath channels.

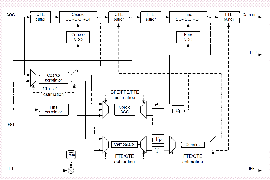
However, modern OFDM system development is being decelerated by two major limiting factors: the increase in the complexity of OFDM communication schemes and the dilation of the verification time required by such systems. Therefore a mechanism is required to reduce the development time and decrease the time to market.

OBJECTIVES

The aim of this research work is to create a set of configurable, low complexity **IP cores** that can be easily integrated into a given OFDM system and to develop an **FPGA based hardware-in-the-loop verification system** that will substantially reduce post-implementation simulation time, enable **parametrical analysis** and accelerate platform verification stages.

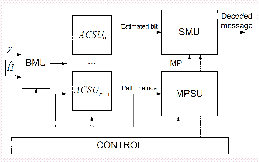
PROPOSED HARDWARE ARCHITECTURES

OFDM SIGNAL SYNCHRONIZER



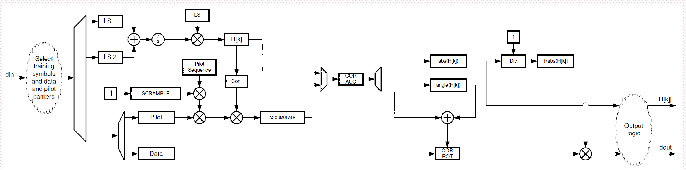
- Frequency and time offset correction
- Re-use of the transceiver FFT (**reduced hardware impact**)

VITERBI SOFT-DECODER



- Configurable Radix
- Multi-voltage/multi-frequency support
- Use of Carrier Strength Indicator (**improved decoding capacity**)
- **Comercial IP**

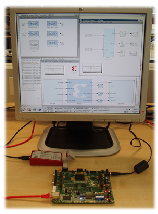
EQUALIZER



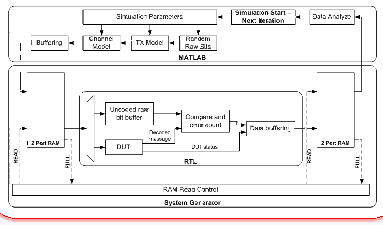
- Phase-tracking and correction on each OFDM symbol
- Carrier Strength Indicator estimation for improved Forward Error Correction performance

FPGA BASED HARDWARE-IN-THE-LOOP VERIFICATION PLATFORM

System implementation

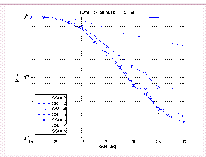


Platform architecture

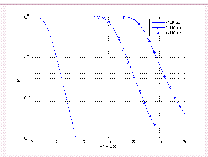


- A software layer creates PLCP Protocol Data Units and implements various channel models
- The OFDM transceiver, **implemented on an FPGA**, decodes the PPDU in **real time.**
- The hardware platform send the BER estimation along with transceiver status to the software for further analysis.

RESULTS



- A variety of low complexity and configurable IPs have been developed that can be easily integrated on an OFDM communication system
  - Some of the IPs are being offered as **commercial products.**
- An FPGA based hardware-in-the-loop verification system has been developed.
  - The entire system or only portions of it can be accelerated .
  - **Reduced simulation time by a factor of 1000** when compared to pure RTL software simulations.
  - The platform allows parametrical analysis of the DUT.



Celebration of 500th PhD Dissertations of Tecnun.
November 25, 2013, Donostia – San Sebastián.