A HARDWARE IMPLEMENTATION OF PUNCTURED CONVOLUTIONAL CODES TO COMPLETE A VITERBI DECODER CORE

E. García, M. Guzmán & D. Torres

Centro de Investigación y Estudios Avanzados del IPN, Unidad Guadalajara
Prol. López Mateos Sur # 590 Guadalajara, Jalisco, México.
dtorres@gdl.cinvestav.mx

Received: October 14th, 2002. Accepted: May 30th, 2005

ABSTRACT

This paper presents a VLSI (Very Large Scale Integration) implementation of high punctured convolutional codes. We present a new circuit architecture that is capable of processing up to 10 convolutional codes rate (n-1)/n with the constraint length-7 derived by the puncturing technique from the basic rate-1/2. The present circuit was designed in order to complete an existing Viterbi decoder core, adding some *extra functionality* such as a *convolutional encoder, differential encoder/decoder, punctured convolutional encoder and symbol insertion to depuncture the received data. This extra functionality includes 10 different programmable coding rates without the need to add additional logic in the system implementation, while other existing coders need it to attain higher coding rates.* Therefore, *a single chip solution is presented.* The design was implemented in VHDL (*Very High Speed Integrated Circuit Hardware Description Language*) synthesized in *Synopsys* tool, and tested in a FPGA. Functional verification was done, by means of simulation, to ensure that the circuit implements intended functionality. Such simulations were executed using *Synopsys* and a *Sun Ultra Sparc 10 workstation.* Different bit error probability performance curves show an agreement between simulated and theoretical values.

RESUMEN

El presente trabajo presenta el diseño, la implementación y la verificación de un circuito con código convolucional perforado para completar un decodificador de Viterbi. A esta versión se le añadieron las siguientes funcionalidades: codificador convolucional, codificador/decodificador diferencial, codificador convolucional perforado con inserción de símbolos. Además, se incluyeron 10 diferentes razones de codificación, superando versiones existentes en el mercado. El circuito desarrollado no requiere lógica externa para realizar el perforado, por lo que puede implementarse en un solo chip. El diseño fue implementado en VHDL, sintetizado usando la herramienta Synopsys, y probado en un FPGA. Se empleó una estación de trabajo Ultra Sparc 10. La verificación funcional permitió probar que el diseño satisface los requerimientos formulados. Finalmente, se presentan diferentes curvas que demuestran una buena coincidencia de los valores obtenidos de la probabilidad de error con los teóricos.

KEYWORDS: Convolutional Coding, Viterbi Decoding, Punctured Codes.

INTRODUCTION

In a typical communication system, is a need to protect the information against the noise existing in every communication channel. Therefore, rate-½ convolutional coders have been widely used in many communication systems due to their moderate complexity and acceptable coding gain [1]. However, in many cases, it is desirable to have a flexible channel encoding scheme because the data to be transmitted over a channel need different levels of error protection since the channel can be time variant or the parameters of the channel are unknown. Given the availability of hardware to encode and decode rate-½ convolutional encoders, one can take advantage of this fact to obtain high rate codes derived from the rate-½ encoder. A technique known as *puncturing* provides a way to do this [2]. Puncturing means that some symbols of a code are deleted and they are not transmitted. For example, to obtain a rate-3/4 from a rate-1/2 it is necessary to delete periodically two out of six bits [3] produced by the convolutional encoder. However, it is necessary to have a proper pattern to delete such symbols in a proper way since puncturing reduces the minimum free distance of a code [4]. The search of good punctured codes is often based on trials and errors instead of a mathematical construction [2]. However, extensive research has been done in the past years and good punctured patterns for sequential and Viterbi decoding were obtained [4].

Nowadays, several companies such as Qualcomm, Altera, Xilinx, and Intel (among others) provide different hardware and software implementations for the Viterbi decoder: ASIC (application-specific integrated circuit), and digital cores, which are provided in a Hardware Description Language. However, some of the available hardware do not provide a single chip solution when different punctured patterns are required. Therefore, in order to use the most common punctured patterns (proposed by Y. Yasuda *et al.* [5]), extra circuitry is needed to attain the desired rate.

Two Viterbi decoders, presented in [6 & 7], were considered as a reference of what is needed in a Viterbi decoder. The design presented in this work adds extra functionality that other circuits do not implement, such as the availability to choose between the most used punctured patterns and custom punctured patterns in a single chip solution.

The main goals of the present work are:

• *To provide a circuit architecture (as a single chip solution) that can handle ten different coding rates derived from the rate-1/2 convolutional encoder.*

To obtain a circuit architecture suitable to puncture one symbol per clock cycle and also to be fully programmable.

To build a set of reusable modules written in VHDL that can be synthesized in any FPGA or in an ASIC.

2. REQUIREMENTS FOR THE PRESENTED PUNCTURED CONVOLUTIONAL ENCODER

The requirements, denoted in general by φ_{SPEC}, for the punctured convolutional encoder are:

1. Rate-1/2, K=7 (K value is the constrained length of the code) convolutional encoder
2. Differential encoder/decoder
3. Punctured codes for the following rates: 2/3, 3/4, 4/5, 5/6, 6/7, 7/8, 11/12, 12/13, 15/16 and 16/17.
4. A programmable punctured pattern capability.
5. Punctured convolutional decoder.

3. ARCHITECTURE

Starting from the requirements, φ_{SPEC}, an architecture of the circuit is described in this section. The proposed architecture is divided in two major blocks: encoder and decoder. These blocks, and their associated components, were implemented using the synthesizable subset of VHDL (Very High Speed Hardware Description Language), see [8] for details.

In the next section, an explanation of the encoder/decoder architecture is shown. The general encoder scheme is shown in Figure 1 and it consists of the following components: *Differential Encoder, Convolutional Encoder and a Punctured Encoder.*

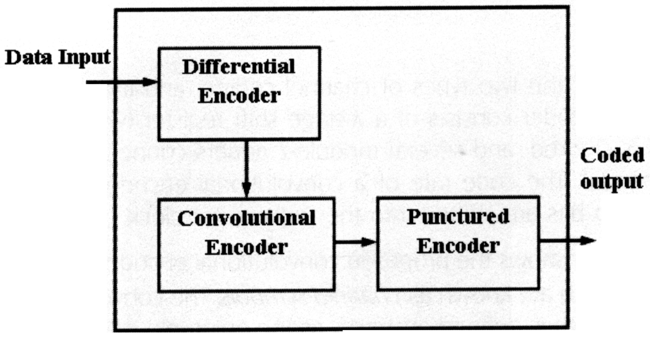

Figure 1. Architecture

3. Differential encoder

In a typical M-ary PSK (*Phase Shift Keying*) system, there is no absolute phase reference, so phase ambiguities in the modulated data may occur. The phase ambiguities must be resolved in order to properly decode the received data [9]. A differential encoder provides this function. Thus, the differential encoder transforms the input data stream into an indication of transitions rather than ones or zeros.

Encoder principle: Let a_k *an input sequence and* b_k *an output sequence of the differential encoder,* $k \in \{1, 2, ...\}$. *If* $a_k=0$ *for some k, the output remains unchanged. If* $a_k=1$ *the output will be a transition from zero to one or one to zero.* This is given by eq (1). Figure 2 shows the differential encoding process.

$$b_k = a_k \oplus b_{k-1}$$

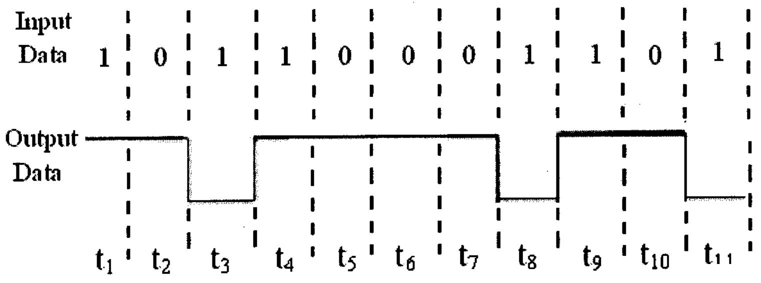

Figure 2. Differential encoding

A one-bit delay and a module-2 adder, as shown in Figure 3, are the principal elements of a differential encoder.

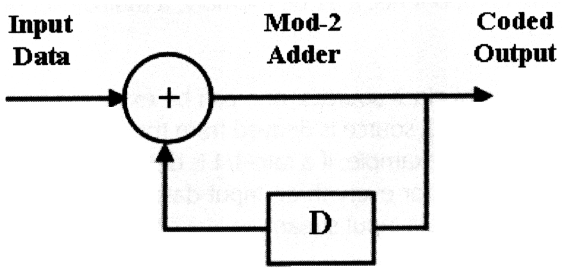

Figure 3. Differential encoder

3.2. Convolutional Encoder

The *convolutional coding* is one of the two types of channel coding, and it adds structured redundancy to a data sequence [10]. A convolutional encoder consists of a *K*-stage shift register (where K is the constraint length of the code), into which input data is clocked, and several module-2 adders connected to the stages of the shift register according to a chosen polynomial. The code rate of a convolutional encoder is given as a fraction *q/n*, where *q* determines how many input data bits are shifted into the register per clock cycle, while *n* is the number of coded symbols per input data bits. Figure 4 shows the proposed convolutional encoder, a *K*=7, rate-1/2, (113o, 155o[1]), where *U0* and *U1* are the coded data, which are known as *channel symbols*. The convolutional encoder was implemented in VHDL as a finite state machine, which is defined abstractly as the quintuple **<*S, I, O, f, h*>,** where ***S*** represents a *set of states*, including an initial state, ***I*** *set of inputs* (data input), ***O*** *set of outputs* (coded output), and ***f*** and ***h*** represent the *next state* and *output functions* respectively.

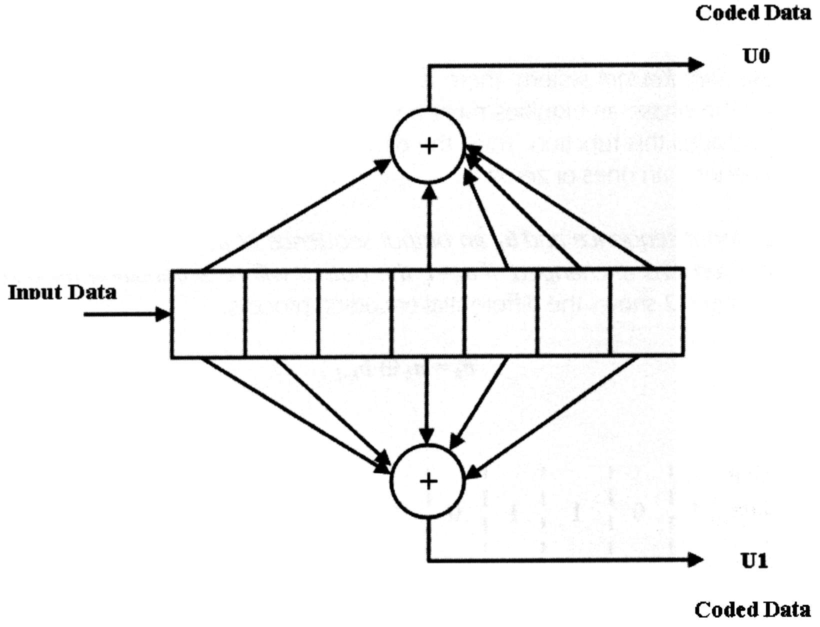

Figure 4. Convolutional encoder

3.3. Punctured convolutional encoder

The punctured convolutional coding is a good technique to obtain different rates derived from a rate-1/2 convolutional encoder as stated in section 1. Figure 5 shows the architecture of the punctured convolutional encoder (PCE), which consists of the following components: a 32 bit memory, a multiplexer (MUX), counters, pattern memory, and a control unit.

The PCE architecture assumes two different clock sources: one can be extracted from the input data stream, which is known as *input data clock*, and the other clock source is derived from the data clock frequency, and its value depends on the desired rate, known as *rate clock*. For example, if a rate-3/4 is desired, six encoded bits (three *U0* and *U1* pairs) are generated by the convolutional encoder for every three input data bits. These encoded bits are output from the *U0* and *U1* pins at 2/3 the frequency of the data input stream.

o means octal base

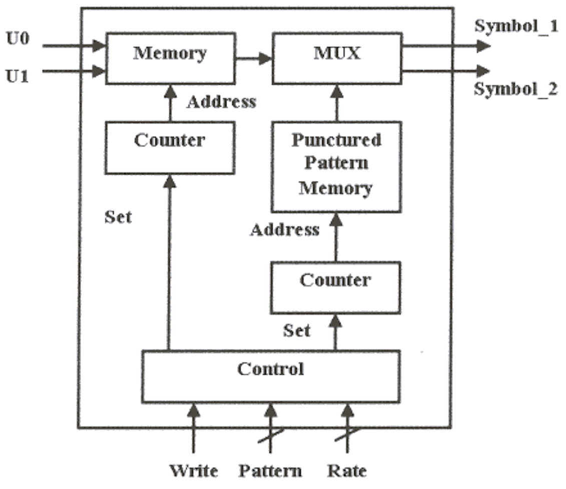

Figure 5. Punctured convolutional encoder

The PCE proposed in this work uses a 32-bit memory to store the coded data that the rate-1/2 convolutional encoder provides (*U0* and *U1* in Figure 5). A counter provides the address of the memory that is incremented after each *input data clock* cycle. In order to delete the required symbols, the memory shall have 2x*P* stored symbols, where *P* is the period of the punctured pattern. For example, if a rate-3/4 is desired, the memory will store 12 bits (2x3=6 pairs of *U0* and *U1*). Then, a MUX is needed to choose which symbols will be deleted according to the punctured pattern provided by the user. Such punctured pattern is stored in a memory which can be introduced in one of two ways: by providing a desired rate via the *RATE* pin in Figure 5, i.e. 3/4, and the circuit will choose the appropriate pattern; the second one is to introduce a specific sequence of bits that represents the desired punctured pattern. These bits shall be introduced via the *PATTERN* pin shown in Figure 5. Depending on the desired rate, the control block shown in Figure 5 will set the appropriate address of the punctured pattern as well as the needed configuration of the MUX and the blocks that compose the encoder.

3.4. Decoding

A code must be decodable in order to be useful, so there are different ways to decode a convolutional code: *sequential, feedback, Fano* [9], but there is an optimum decoder, known as *Viterbi decoding algorithm* [11]. This algorithm is optimum in the maximum likelihood sense. The complexity of the Viterbi algorithm increases in an exponential way when the coding rate is increased. However, the puncturing technique allows the use of the rate-1/2 Viterbi decoding. In order to be decodable by the Viterbi algorithm, the punctured bits are replaced with dummy data [2] at the receiver. In Figure 6, the structure of the decoder is shown. The decoder includes the following components: symbol inserter, Viterbi decoder, and differential decoder.

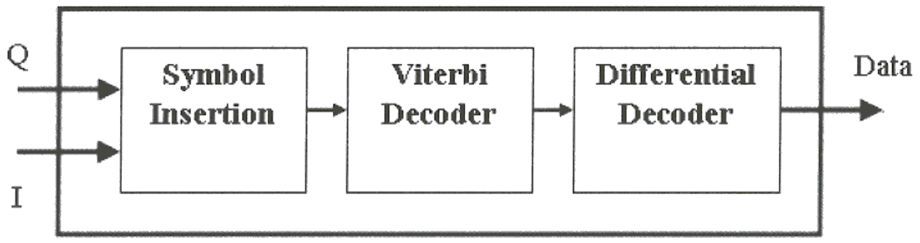

Figure 6. Decoder

The Viterbi block shown in Figure 6 was proposed in a Master Thesis at CINVESTAV [12].

3.5. Symbol Insertion

The symbol insertion is the inverse function of puncturing. Its main purpose is to insert dummy data into the positions corresponding to the deleted code symbols at the transmitter. This dummy data is, in general, a *weak one* or a *weak zero*. In a 3-bit soft decision scheme, a weak one is a 3-bit vector than can have the following values: *011*, *101*, and *110*, whereas a weak zero can have the following values: *001*, *010*, and *100*. The symbol insertion is shown in Figure 7.

As was stated before, *the symbol insertion is the inverse operation of puncturing*, so it works the same as the encoder with some brief modifications:

- The architecture assumes a QPSK (Quadrature Phase Shift Keying) modulation.
- Viterbi soft decision decoding.

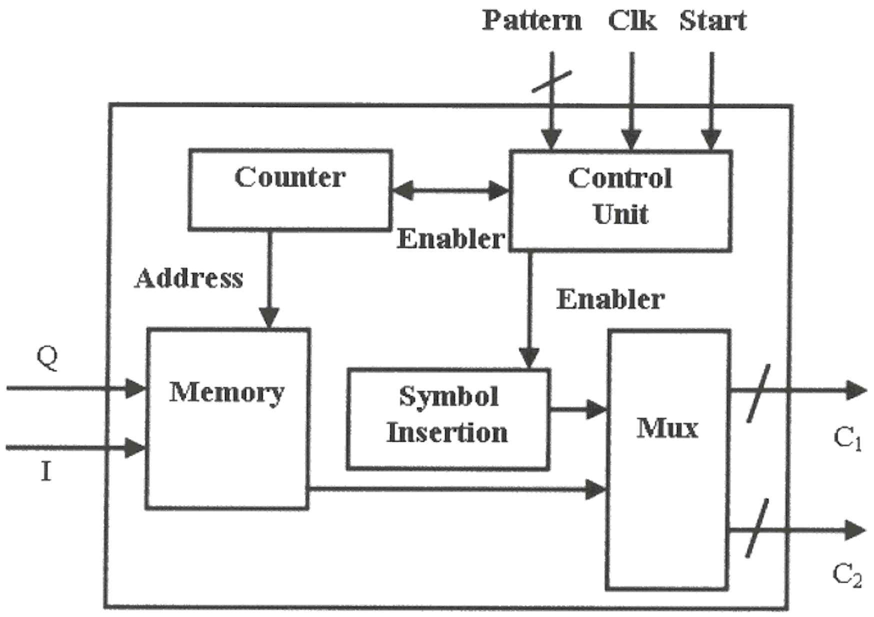

Figure 7. Symbol Inserter

As in the punctured encoder, the symbols shall be stored in the memory, where the address of the memory is provided by a counter that increases every *rate clock* cycle. The control unit block provides the proper configuration of the blocks to perform the symbol insertion according to the punctured pattern used at the transmitter.

3.6. Differential Decoding

The differential decoder is shown in Figure 8, and it is the inverse of the diagram shown in Figure 3. It is build by a one bit delav and a module-2 adder.

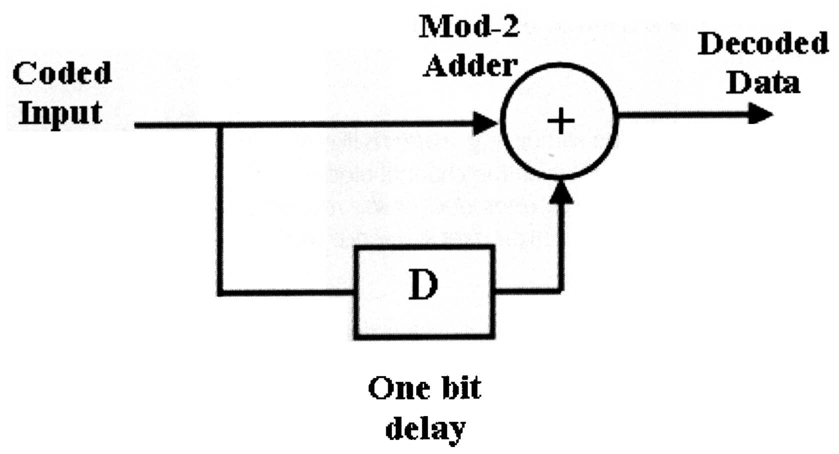

Figure 8. Differential decoder

4. VERIFICATION

Once the design process is finished, the design must be verified in order to check that the requirements specification has been followed. In other words, *functional verification is the proof* that a circuit *meets* or behaves according to *a given set of requirements*. This can be expressed by the following general statement: $\varphi_{IMP} \rightarrow \varphi_{SPEC}$.

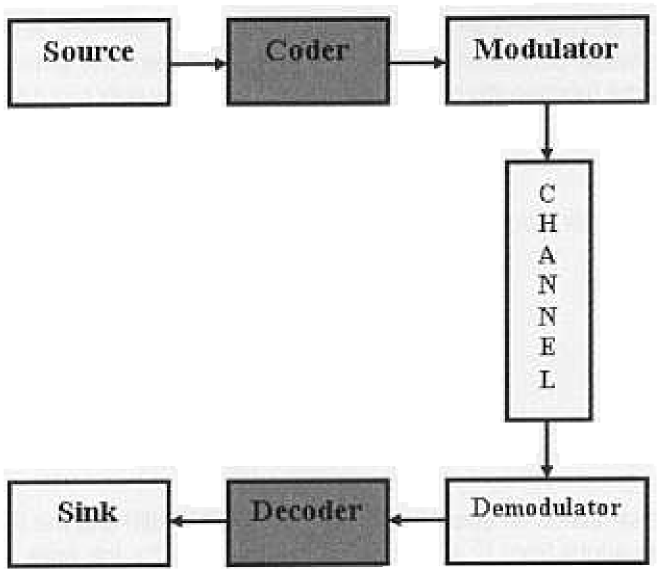

Figure 9. Verification Environment

It should be noted that the functional verification process consumes about the 70% of the design cycle [13] in terms of effort, computer resources and time consuming. Such amount of time is because of the need to verify at *unit level*, e.g. the convolutional encoder or the PCE, then the verification should be done at *block level*, e.g. the complete encoder, and finally a *system level* simulation (the whole design) should be done in order to assure that the circuit *meets* the required functionality. In order to carry out the verification process of the design presented in this work, the testbench, shown in Fig. 9, was coded in VHDL. A testbench can be defined as an executable model that instantiates the Design Under Verification (DUV) providing the capability to drive the DUV with a set of stimuli and comparing the outputs with expected results. The shadowed blocks in Fig 9 are the DUV, while the remaining blocks simulate a typical communication system.

The source block produces the mentioned stimuli: e.g. *patterns* like AAh[2], (55h), and pseudorandom data sequences. The modulator block *simulates a QPSK modulator*; the channel block is *configurable* in order to simulate different SNR (Signal-to-Noise Ratio). The demodulator block *demodulates the received data* and provides a *soft decision input* to the decoder. The sink block *compares* the *received data sequence* with the original data sequence in order to detect transmission errors.

In order to check the general statement about the verification process, $\varphi_{\mathbf{IMP}} \rightarrow \varphi_{\mathbf{SPEC}}$, the VHDL implementation was exercised with several stimuli to verify the following functionalities at system level:

- Noise free environment and a AAh data sequence
- Rate-½ with different noise levels and a AAh data sequence

  Rate-½ with different noise levels and a pseudorandom data sequence
- System in a noise free environment and using different punctured patterns and AAh data sequence

  System with different levels of noise and using different punctured patterns with a pseudorandom data sequence
- System using a "user programmable" punctured pattern under different noise condition and AAh data pattern. The programmable punctured pattern used in this test were the same as in the previous tests, the purpose was to verify the functionality of this feature and not a new punctured pattern.

5. RESULTS

The theoretical bit error probability for a punctured convolutional code derived from a rate-½ code is given by

$$P_B \leq \frac{1}{l}\sum_{k=d}^{\infty} C_k P_k = \sum_{k=d}^{\infty} \tilde{C}_k P_k$$

where d is the minimum free distance of the punctured code, C_k is the total number of error bits and P_k is the probability that one incorrect path is selected in the Viterbi decoding process, see [5] for details.

The comparison between the simulated bit error probability versus E_b/N_0 (dB) and the theoretical bit error probability versus E_b/N_0 calculated by equations from [2 & 4], is given in Figs. 10-12 for the rates 2/3, 3/4 and 4/5 respectively. E_b/N_0 is the measure of signal to noise ratio for a digital communication system. It is measured at the input to the receiver and is used as the basic measure of how strong the signal is. Good coincidence between simulated and theoretical curves is observed. For simulated curves above R=4/5, huge amount of computer resources are needed. For that reason, *unit level verification was done for every rate.* Critical parts of the circuit are: its internal memory, and Finite State Machines. Therefore, interesting test-cases to assure that the *memory can handle the required amount of bits for each punctured pattern, and the FSM controls the data puncturing process,* were developed. Although not enough data were available to generate all curves, *system level tests and critical cases* prove that the circuit at higher rates behaves similar to the results shown in Figs. 10-12.

[2] h means hexadecimal

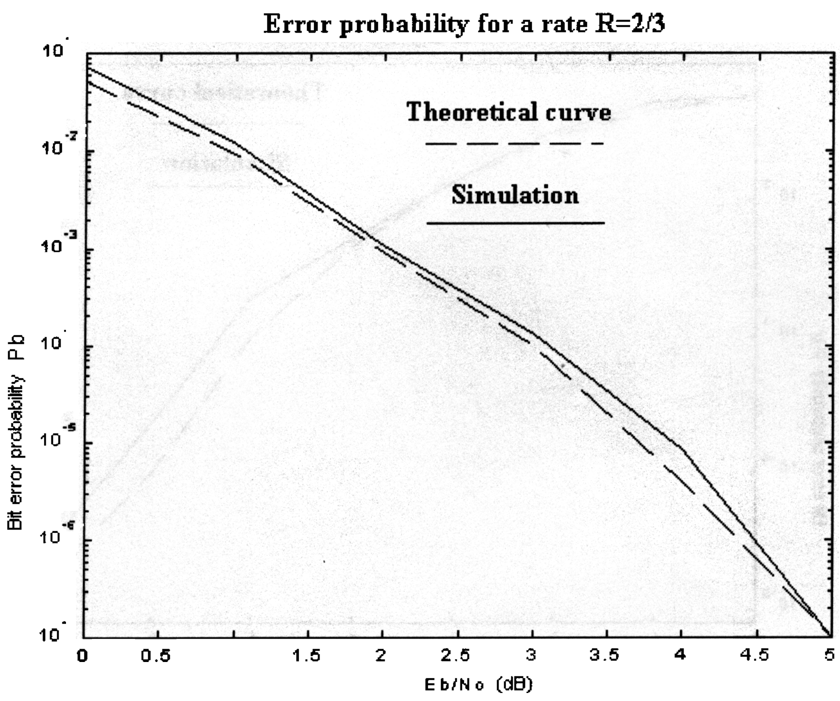

Figure 10. Bit error probability versus E_b/N_o for rate 2/3

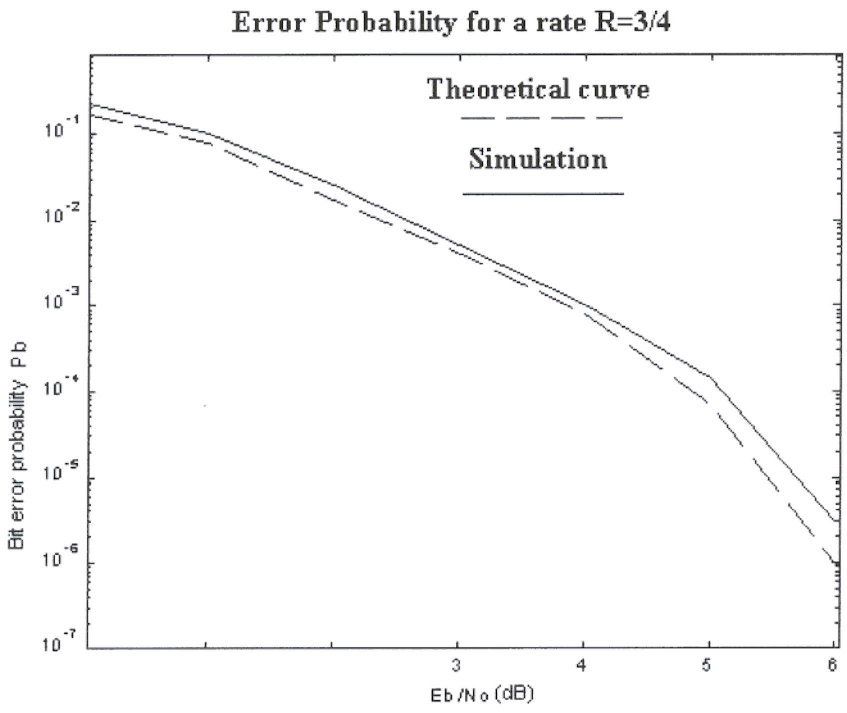

Figure 11. Bit error probability versus E_b/N_o for rate ¾

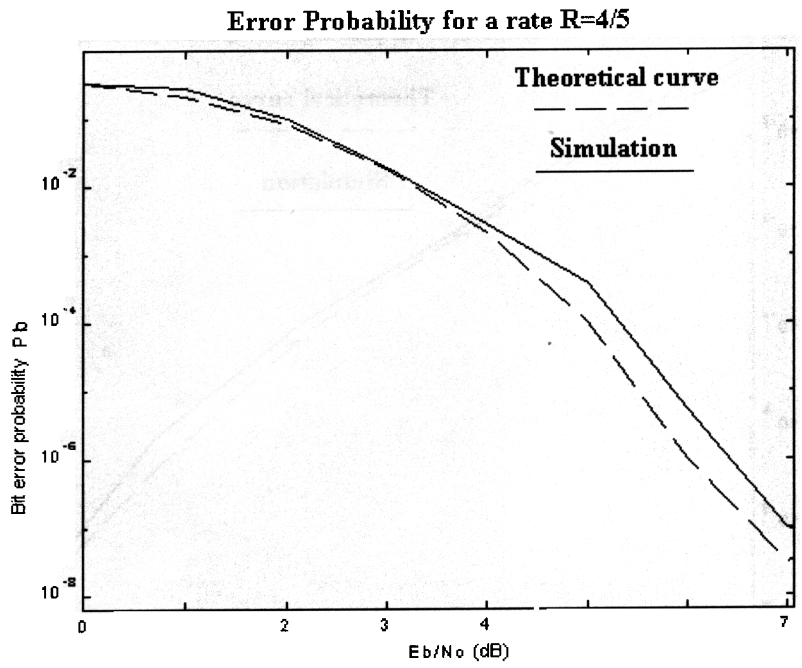

Figure 12. Bit error probability versus E_b/N_o for rate 4/5

After the verification process, the design was synthesized using a Flex10K200 FPGA from Altera and the results were compared to similar circuits in the market. The device usage is shown in Table I and the comparison with other circuits are shown in table II and Table III.

Table I. Device Area

Memory used (bits)	% memory usage	Logic cells
14,400	14%	4919

Table II. Comparison of our device with others respect to coding rate

Architecture	Coding Rate and Constraint Length	Fabrication Process
Our device	*½ K=7*	*Altera Flex10K200*
STEL-2030C	*½ K=7*	*ASIC Unknown*
NOVA	*½ K=7* *¾ K=9*	*Altera Flex10K250*
Qualcom	*½ K=9*	*ASIC Unknown*

In the following lines, a summary of the attained results are shown:

The design was simulated and synthesized using Synopsys tools. A Flex10K200 FPGA from Altera was used

The synthesis results show that the design presented in this paper uses less than 5k logic cells and less than 16Kbits of memory.

A complete top level of the design includes: Convolutional encoder, differential encoder, punctured convolutional encoder, symbol insertion, Viterbi decoder [12] and BER (Bit Error Rate) monitor.

- The top level, including the Viterbi Decoder core, uses less than 115k gates.

Table III. Comparison of our device with others respect to punctured rates

Architecture	Punctured rates	Scrambler and descrambler V.35	Diferential Encoder/Decoder
Our device	*2/3, 3/4, 4/5, 5/6, 6/7, 7/8, 11/12, 12/13, 15/16, 16/17*	*No*	*Yes*
STEL-2030C	*2/3, 3/4*	*Yes*	*Yes*
NOVA	*3/4, 7/8*	*No*	*No*
Qualcomn	*3/4, 7/8*	*Yes*	*Yes*

6. CONCLUSIONS

A fully programmable device was designed, implemented and verified. It has the following features:

While other chips have just two or three different coding rates besides the 1/2-rate (2/3, 3/4, 7/8), the design presented in this work has 10 different programmable coding rates derived from the rate-1/2 convolutional encoder in a single chip. The rates are the following: 2/3, 3/4, 4/5, 5/6, 6/7, 7/8, 11/12, 12/13, 15/16. This is the main contribution of this work.

Rate-1/2, K=7 convolutional encoder.

Differential encoder/decoder.

Symbol insertion (depuncturing).

Symbol Encoding/Decoding per clock cycle, functionality that just a few decoders in the market have [6].

Reasonable amount of logic gates, 115kgates, which permits a single chip solution.

Searches done by the authors reveals the absence of new developed circuits with more features than this one, and that the coding gain remains in the same order as in major publications in this area. An additional advantage of our design is that it can be implemented, for research and commercial purposes, in an ASIC or in many of the available FPGAs from Altera, Xilinx, and other vendors.

7. REFERENCES

[1] Wilson S. G., Digital Modulation and Coding, Prentice Hall, New Jersey, 1996.

[2] Haccoun D. and Begin G., High-Rate Punctured Convolutional Codes for Viterbi and Sequential Decoding, IEEE Trans. Communications, vol 37, No 11, November, pp. 1113-1125, 1989.

[3] Lee C., Convolutional Coding: Fundamentals and Applications, Artech House, INC, Norwood, MA., 1997.

[4] Cain J.B, Clark G.C, and Geist J., Punctured Convolutional Codes of Rate (n-1)/n and simplified maximum likelihood decoding. IEEE Trans. Inform. Theory, vol. IT-25, Jan., pp. 97-100, 1979.

[5] Yasuda, Y., Kashiki, K., and Hirata, Y., High-Rate Punctured Convolutional Codes for Soft Decision Viterbi Decoding. IEEE Trans. Communications, vol. COM-32, March, pp. 315-319, 1984.

[6] Qualcomm Corp., Q1900 Viterbi Encoder/Decoder Data Sheet, 1998.

[7] Intel Corp., Stel 2030C Convolutional Encoder Viterbi Decoder Data Sheet, January, 2000.

[8] Sjoholm S. and Lindh L., VHDL for Designers. Prentice Hall, 1997.

[9] Proakis J.G., Digital Communications, Mc Graw Hill, Third Edition, 1995.

[10] Forney G.D Jr., Convolutional Codes I: Algebraic Structure. IEEE Trans. on Information Theory, Vol IT-16, No. 6, November, pp. 720-738, 1970.

[11] Forney G.D Jr., The Viterbi Algorithm. IEEE Proceedings, IT-61(3), March, pp:268-273, 1973.

[12] Bazdresch L.M., A VLSI Implementation of the Viterbi Algorithm. CINVESTAV Unidad Guadalajara, Mexico, July, 2000.

[13] Bergeron, J. Writing testbenches, functional verification of HDL models. Kluwer Academic Publishers, 2000.