A VHDL IMPLEMENTATION OF THE SOFT OUTPUT VITERBI ALGORITHM

Brett W. Werling
Department of Electrical Engineering & Computer Science
University of Kansas
Lawrence, KS 66045
bwerling@ittc.ku.edu

Faculty Advisor:
Erik Perrins

ABSTRACT

In this paper we present a VHDL implementation of the soft output Viterbi algorithm (SOVA). We discuss the usefulness of the SOVA in a serially concatenated convolutional code (SCCC) system. We explore various hardware design decisions along with their implications. Finally, we compare the simulated performance of the hardware implementation to a software reference model over an additive white Gaussian noise (AWGN) channel for several bit widths and traceback window lengths.

INTRODUCTION

In the communication theory world, bit error rate (BER) performance is typically improved by choosing longer and more complex codes. Unfortunately, they are not always implementable in the hardware available today. Forney first introduced the idea of concatenated codes in [1] as a way of dividing a long code into smaller, more manageable pieces. His system consisted of *inner* and *outer* codes, with the outer usually being longer and more complex than the inner.

The idea of concatenating multiple *convolutional* codes was introduced in [2], and was made possible in part by the the newly-developed soft output Viterbi algorithm (SOVA). An inner SOVA decoder provides a 1–2 dB gain in BER performance over an inner hard-decision Viterbi decoder [2]. The concatenation of convolutional codes was then examined further in [3], in which the name "serially concatenated convolutional code" (SCCC) came about.

SCCC systems are very useful in the telemetry world, as they are easily implemented and provide similar performance to that of some LDPC codes. Before an SCCC decoder can be built, however, the SOVA must be implemented in a portable way. In this paper, we introduce a VHDL implementation of the SOVA that can be used in several parts of an SCCC decoder. The following are our contributions:

- We examine the soft output Viterbi algorithm and its required data structures.
- We provide an overview of a particular implementation of the algorithm.
- We simulate the hardware implementation and compare its bit error rate (BER) performance with that of a software reference decoder.

Figure 1: Block diagram of the soft output Viterbi algorithm.

SOFT OUTPUT VITERBI ALGORITHM

The Viterbi algorithm (VA) is a widely-used method for maximum likelihood decoding, and its simple implementation makes it an easy choice in most practical applications. One of its shortcomings, however, is its inability to provide soft-decision outputs. Significant performance loss is encountered when concatenating two VA decoders, as opposed to soft-output decoders. An extension of the Viterbi algorithm, known as the soft output Viterbi algorithm (SOVA), was introduced in [4] partially as a way to address this issue. Instead of outputting decoded bits (i.e. hard decisions), the SOVA outputs a soft (or real) reliability value corresponding to each decoded bit. These soft outputs can then be used by other parts of a system (such as a serially concatenated convolutional code system) to improve overall performance.

The full SOVA makes use of two input streams and two output streams, as seen in Figure 1. One set of inputs and outputs corresponds to encoded bit stream, $\mathbf{C}$, while the other set corresponds to the message bit stream, $\mathbf{m}$. For simplicity, we assume that the $\mathbf{m}$ input is not present (i.e. zero) and the $\mathbf{C}$ output is not needed.

This remainder section describes the modified SOVA outlined in [5]. The goal of the SOVA was to leave the original VA as unchanged as possible, while extending its functionality to allow for soft outputs. For simplicity, we assume the use of a (5,7) convolutional code, an AWGN channel model, and a decoding traceback length of T.

Before we explore the algorithm itself, we explore a convolutional code's *trellis*, which is critical to the operation of the SOVA. Figure 2 shows a trellis diagram for a binary, recursive, systematic, rate $R = 1/2$ convolutional code. To organize the information contained in a trellis, we define the following labels:

- *The memory constraint length*: ν. The memory constraint length is the number of memory locations in the convolutional encoder.

- *The state index*: q. The states are labeled from top to bottom, with $q \in \{0, 1, \cdots, 2^{\nu} - 1\}$. These values are a direct mapping from the binary vector stored in an encoder's memory. Each stage in a trellis contains a column of starting states and a column of ending states, labeled q_{S} and q_{E}, respectively.

- *The edge index*: e. Similar to the state index, the edge index is labeled from top to bottom according to its exit point from the column of starting states, and takes on the values $e \in \{0, 1, \cdots, 2^{\nu+1} - 1\}$. The edge index is perhaps the most important value in a trellis stage, as it is used to refer to all the other values.

- *Input and output*: $m(e)/\mathbf{c}(e)$. Each input into an encoder corresponds to a particular output vector. This is determined by the current state of the encoder, and also corresponds to a state transition.

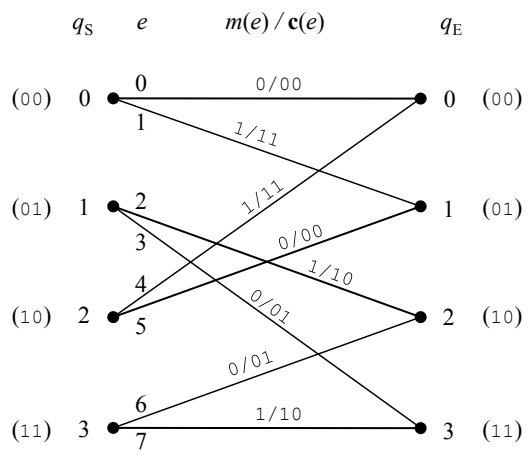

Figure 2: Trellis diagram of a binary, recursive, systematic, rate $R = 1/2$ convolutional code.

Therefore each edge has its own input and output information, which is determined by the structure of the code.

The SOVA makes use of many variables and structures to decode a convolutionally encoded sequence, which we now define:

- *The path metrics*: M_K. At each time step K, a path metric is stored for each state q in the trellis. The path metric is used to determine which state corresponds to the current maximum likelihood path through the trellis.
- *The edge metric increments*: γ_K. An edge metric increment is calculated for each edge in a trellis stage, and is combined with the appropriate path metric before arriving at the next set of states.
- *The winning edges*: W_K. The winning (or surviving) edges are those that maximize the merging metrics.
- *The delta values*: Δ_K. The delta values are the differences between competing path metrics.
- *The path decision values*: $\hat{\mathbf{m}}/\hat{\mathbf{C}}$. During the second pass through the trellis, decoded values are stored in the appropriate path decision stream.
- *The reliability values*: $\mathbf{y}/\mathbf{Z}$. Each path decision is the decoder's "best guess" at what the original transmitted bit was, and each reliability represents how confident we are that those path decisions are correct.

The algorithm makes two passes over the received sequence: one in the forward direction, and one in the backward direction. The forward pass calculates the path metrics according to the received values for each time step $K \in \{0, \cdots, N-1\}$, where N is the length of the received sequence. For simplicity, we assume an AWGN channel, BPSK modulation, and a coding rate $R = k/n$.

For each time step, edge metric increments are calculated as

$$\gamma_K(e) = \sum_{i=0}^{n-1} r_K[i] a(e)[i], \tag{1}$$

where $\mathbf{r}_K$ contains noisy BPSK-mapped received symbols and $\mathbf{a}(e)$ contains BPSK-mapped versions of $\mathbf{c}(e)$ for a particular edge in the trellis. We define a *path metric candidate*

$$M_K^{(i)}(q) = M_{K-1}(q_{\mathrm{S}}(e_i)) + \gamma_K(e_i), \tag{2}$$

where $i \in \{1,2\}$ and $q_{\mathrm{E}}(e_i) = q$. Therefore each state has two path metric candidates that are compared to select the updated path metric

$$M_K(q) = \max\left\{M_K^{(1)}(q),\, M_K^{(2)}(q)\right\}. \tag{3}$$

The value $i = 1$ is typically assigned to the winning candidate, with $i = 2$ being assigned to the losing candidate. Each time a path metric for state q is updated, the absolute difference between the winning and losing candidates merging at state q is stored, or

$$\Delta_K(q) = \left|M_K^{(1)}(q) \;-\; M_K^{(2)}(q)\right|. \tag{4}$$

Winning edges are calculated as

$$W_K(q) = \underset{e:q_{\mathrm{E}}(e)=q}{\arg\max}\left\{M_{K-1}(q_{\mathrm{S}}(e)) + \gamma_K(e)\right\}. \tag{5}$$

At the end of the received sequence, the decoder makes a pass through the information gathered in the forward pass. This second pass, known as the "traceback" (TB) loop, is where the decoded information is obtained. We initialize the traceback loop with the state corresponding to the maximum likelihood path metric at time $N-1$, or

$$q_{\mathrm{TB},N-1} = \underset{q\in\{0,1,\cdots,2^{\nu}-1\}}{\arg\min}\left\{M_{N-1}(q)\right\}. \tag{6}$$

We use the time index K again, but this time in reverse order. It is initialized as

$$K = N-1. \tag{7}$$

The steps of the traceback loop are as follows:

1. Find the winning edge whose ending state is $q_{\mathrm{TB},K}$, or

$$e_{\mathrm{TB},K} = W_K(q_{\mathrm{TB},K}), \tag{8}$$

   where $W_K(q_{\mathrm{TB},K})$ returns the winning edge with the ending state $q_{\mathrm{TB},K}$.

2. Find the path decisions corresponding to $e_{\mathrm{TB},K}$, or

$$\hat{m}_K = m(e_{\mathrm{TB},K}) \tag{9}$$
$$\hat{\mathbf{c}}_K = \mathbf{c}(e_{\mathrm{TB},K}), \tag{10}$$

   where $m(e_{\mathrm{TB},K})$ and $\mathbf{c}(e_{\mathrm{TB},K})$ return the input and output information contained on the edge $e_{\mathrm{TB},K}$.

3. Update the reliability values according to the method described in [5]. (We only show the updating of $\mathbf{y}$, but the update process of $\mathbf{Z}$ follows the same steps.)

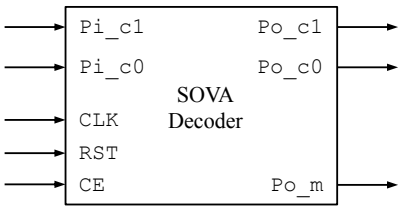

Figure 3: A black box view of the SOVA decoder.

- Set the current reliabilities equal to Δ, or $y_K(q) = \Delta_k(q)$.
- Let J loop on the remaining reliabilities, or $J \in \{K-T+1, \cdots, K-1\}$.
- If $\hat{m}_J^{(1)}(q) \neq \hat{m}_J^{(2)}(q)$, then we update $y_J(q)$ using the equation

$$y_J(q) = \min\left\{\Delta_K(q),\, y_J^{(1)}(q)\right\} \tag{11}$$

- If $\hat{m}_J^{(1)}(q) = \hat{m}_J^{(2)}(q)$, then we update $y_J(q)$ using the equation

$$y_J(q) = \min\left\{\Delta_K(q) + y_J^{(2)}(q),\, y_J^{(1)}(q)\right\} \tag{12}$$

4. Move to the starting state of the edge $e_{\mathrm{TB},K}$, or

$$q_{\mathrm{TB},K-1} = q_{\mathrm{S}}(e_{\mathrm{TB},K}). \tag{13}$$

5. If $K = 0$, traceback is done. Otherwise, decrease K by 1 and go back to Step 1.

HARDWARE IMPLEMENTATION

Our SOVA decoder design begins with a look at the inputs and outputs that are encountered by the algorithm. The information inputs and outputs are soft, real probabilities in the form of log-likelihood ratios (LLRs). This design assumes quantization using a universal bit width, B, across all elements in the decoder.

- *The information inputs*: `Pi_c1` and `Pi_c0`. These are the quantized versions of $r_K[1]$ and $r_K[0]$, which are the *a priori* LLRs at time K.
- *The information outputs*: `Po_c1`, `Po_c0`, and `Po_m`. These are the updated *a posteriori* output LLRs.

As with most digital hardware, our decoder has a concept of time, the ability to clear its contents, and the ability to be enabled/disabled. This is accomplished with the following control signals, which are common among time-sensitive digital hardware:

- *The clock signal*: `CLK`. Perhaps the most important control signal in digital hardware, `CLK` provides a common time reference to all relevant circuits. All components in this design that require `CLK` are sensitive to its rising edge.

- *The reset signal*: `RST`. This design assumes that `RST` is an active-high, synchronous reset, although that is not required for a SOVA decoder in general. All registers in this design are set to `0` when `RST` is activated.

- *The clock enable signal*: `CE`. Each register in this design is controlled by a common, active-high `CE`. When disabled, registers do not allow new values to be stored, while maintaining their current values. This signal is required by outside components to control the flow of information.

In order to manage all of the information needed during the decoding process, we break the design down into three individual pieces, each responsible for a separate task:

- *Metric manager*. The metric manager (MM) provides information about the path metrics to all other units in the decoder. This includes the current winning state, the winning paths at each time step, and the differences between path metric candidates (Δ).

- *Hard-decision traceback unit*. The hard-decision traceback unit (HTU) accepts information about the winning paths and updates the path decision vectors as defined by the trellis. It outputs the oldest path decision in the traceback window, as well as comparisons of path decision candidates.

- *Reliability traceback unit*. The reliability traceback unit (RTU) is similar to the HTU in that it accepts information about the winning paths, but it also accepts the four Δ values being output by the MM. It uses these two information sources to update the reliabilities for the current path decision vectors, while it outputs the oldest reliabilities in the traceback window.

These pieces are collectively controlled by the control signals. All addition operations are overflow-protected in the positive and negative directions by using a minimum and maximum value. A block diagram of the connected system is shown in Figure 4.

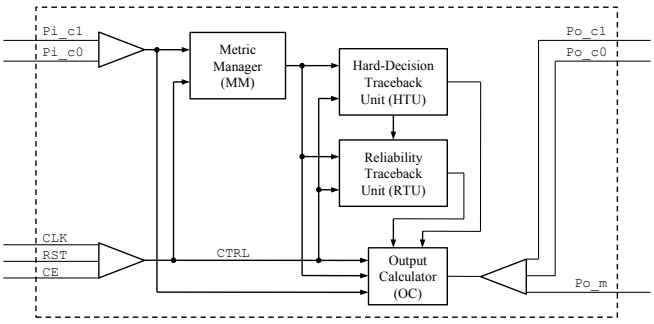

Figure 4: The internal structure of the SOVA decoder.

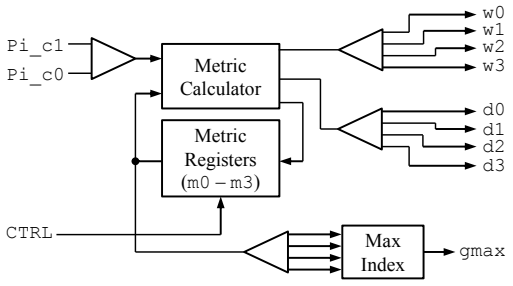

Figure 5: A block diagram of the path metric manager.

A. *Metric Manager*

The metric manager (MM) is responsible for updating, comparing, and storing the path metrics. It outputs information about the winning paths at each time step, as well as the state index corresponding to the current maximum likelihood path. It accomplishes this with a set of registers that store the metrics, while using a metric calculator (MC) to determine what the next set of metrics will be. The MC also determines winning edges as well as the values of Δ. A block diagram of the MM is shown in Figure 5.

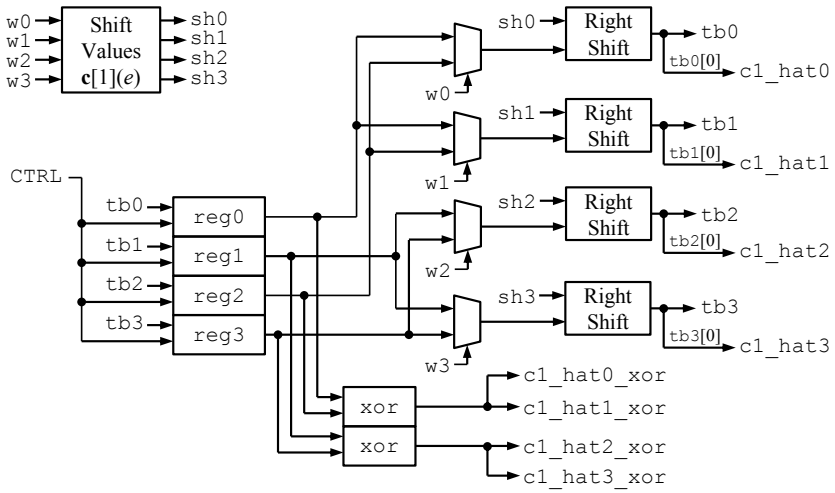

Figure 6: Block diagram of the traceback unit corresponding to $\hat{\mathbf{c}}[1]$.

B. Hard-Decision Traceback Unit

The hard-decision traceback unit (HTU) is responsible for managing the path decisions that occur during the decoding process. It accomplishes this using a method of traceback called "register exchange," with a traceback window length of T. The typical HTU of a SOVA decoder contains traceback registers for each of the path decision streams. Our design works around a systematic code, therefore removing the need for traceback registers corresponding to the information bits. A block diagram of the traceback unit for $\hat{\mathbf{c}}[1]$ is shown in Figure 6.

C. Reliability Traceback Unit

The reliability traceback unit (RTU) is responsible for storing and updating the reliabilities of the path decisions. The RTU uses the same traceback window length as the HTU, although instead of maintaining four length-T registers, the RTU keeps track of four length-T arrays of length-B registers. In other words, there are four arrays, each with T locations. Each location in an array contains a register with B bits. Since the HTU only keeps track of path decisions $\hat{\mathbf{c}}[1]$ and $\hat{\mathbf{c}}[0]$, the RTU only maintains the reliabilities $\mathbf{z}[1]$ and $\mathbf{z}[0]$. A block diagram of the traceback unit for $\mathbf{z}[1]$ is shown in Figure 7, and the block diagram for the update process for that unit is shown in Figure 8.

PERFORMANCE RESULTS

The decoder presented in this paper was implemented in VHDL, and was compared with a given reference decoder written in MATLAB, which was known to be accurate. MATLAB was used to generate noisy, encoded streams of data, which were then quantized using a common bit width, B. A bit error rate (BER) comparison was done using only the information bit decisions (i.e. $\hat{\mathbf{m}}$ or $\hat{\mathbf{c}}[1]$), meaning the LLR outputs of each decoder were mapped to `0`'s and `1`'s.

Each simulation was run over a minimum of 1,000,000 transmitted bits, with a requirement of at least 100 bit errors after decoding. While these values are on the low end of what can be used to generate an

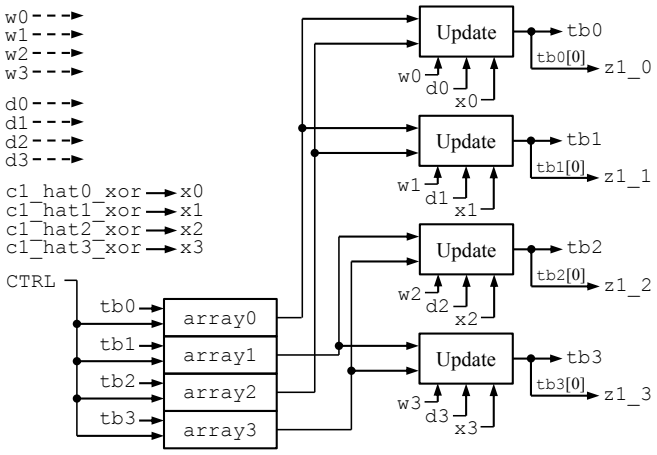

Figure 7: Block diagram of the reliability traceback unit corresponding to $\mathbf{z}[1]$.

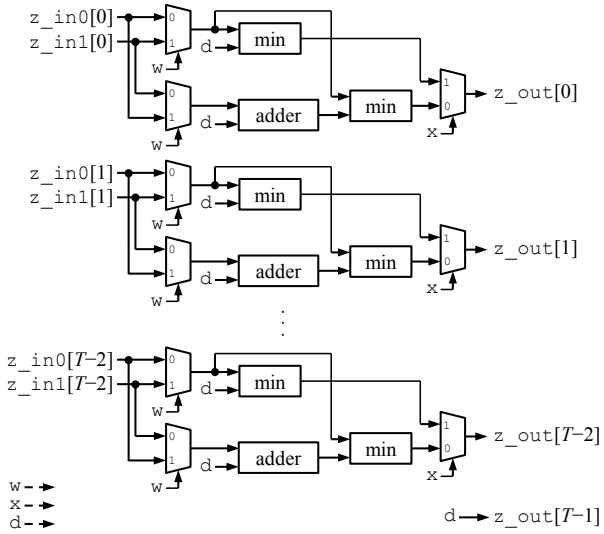

Figure 8: Block diagram of a reliability update unit.

Table 1: Hardware metrics of the SOVA decoder for the bit width $B = 8$.

T	Slices Occupied (%)	Maximum Clock Frequency (MHz)
8	6.111	143.369
16	11.817	143.390

"accurate" BER plot, they are sufficient for determining whether or not the VHDL decoder is performing as expected, since both decoders operate on the same noisy data. Figure 9 shows the results of simulations over various bit widths with a traceback length of $T = 8$. Figure 10 shows the results of simulations over various bit widths with a traceback length of $T = 16$.

The VHDL used to define the SOVA decoder was synthesized, mapped, and routed for use on the XC5VLX110T FPGA, which is a member of Xilinx's Virtex-5 family. The processing was done using the Xilinx ISE design tools. Table 1 shows the results of the building of this design with the traceback lengths $T \in \{8,\ 16\}$.

CONCLUSIONS / FUTURE WORK

The results of the simulations indicate that the VHDL decoder presented in this paper successfully performs the soft output Viterbi algorithm. Our design provides the framework for a general purpose SOVA decoder that can be used in a serially concatenated convolutional code (SCCC) system.

Our decoder implementation was built using a software-based approach. Work can be done in the future to improve the speed of the decoder, however doing so would require the design to be less portable. In the future, we plan to integrate this design with an SOQPSK-TG demodulator to create a SCCC decoder. The resulting system, when run over many iterations, obtains performance approaching the Shannon Limit.

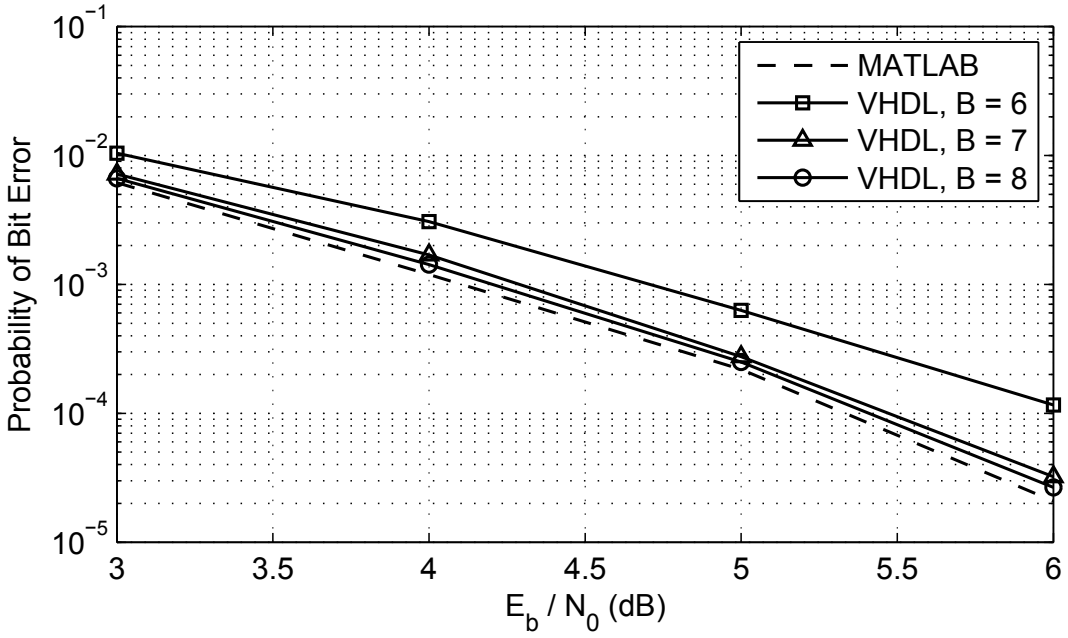

Figure 9: Plot of simulated bit error rate for $T = 8$.

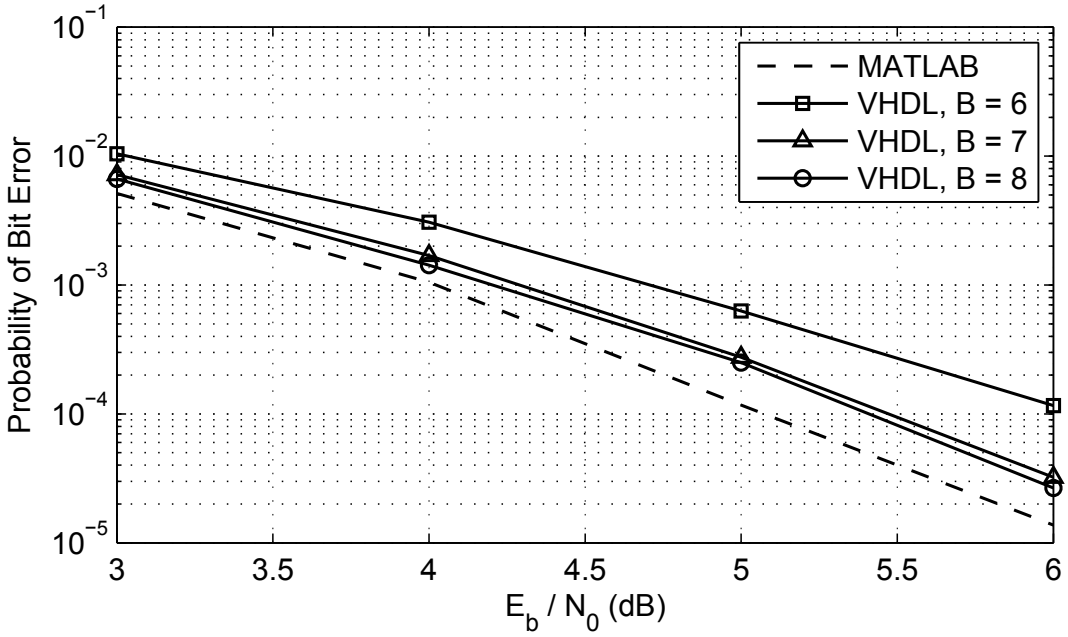

Figure 10: Plot of simulated bit error rate for $T = 16$.

ACKNOWLEDGEMENTS

The authors would like to thank the Test Resource Management Center (TRMC) Test and Evaluation/Science and Technology (T&E/S&T) Program for their support. This work was funded by the T&E/S&T Program through the U.S. Army Program Executive Office for Simulation, Training and Instrumentation (PEO STRI), contract number W900KK-09-C-0018 for High-Rate High-Speed Forward Error Correction Architectures for Aeronautical Telemetry (HFEC).

REFERENCES

[1] G. D. Forney, *Concatenated Codes*. Cambridge: M.I.T. Press, 1966.

[2] J. Hagenauer and P. Hoeher, "Concatenated Vitebi Decoding," in *Proceedings of the International Workshop on Information Theory*, 1989.

[3] S. Benedetto and G. Montorsi, "Serial concatenation of block and convolutional codes," *Electronics Letters*, vol. 32, pp. 887 – 888, September 1996.

[4] J. Hagenauer and P. Hoeher, "A Viterbi Algorithm with Soft-Decision Outputs and its Applications," in *Global Telecommunications Conference, 1989, and Exhibition. Communications Technology for the 1990s and Beyond. GLOBECOM '89., IEEE*, vol. 3, pp. 1680 – 1686, November 1989.

[5] M. P. C. Fossorier, F. Burkert, S. Lin, and J. Hagenauer, "On the Equivalence Between SOVA and Max-Log-MAP Decodings," *IEEE Communications Letters*, vol. 2, pp. 137–139, May 1998.