OPTIMIZED EFFICIENT SOFT-DECISION VITERBI ARCHITECTURE FOR APPLICATION-SPECIFIC PROCESSORS

By

JOHN TOBOLA

Bachelor of Science in Electrical Engineering
Oklahoma State University
Stillwater, Oklahoma, United States of America
2017

Submitted to the Faculty of the
Graduate College of
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 2018

OPTIMIZED EFFICIENT SOFT-DECISION VITERBI
ARCHITECTURE FOR APPLICATION-SPECIFIC PROCESSORS

Thesis Approved:

Dr. James E. Stine, Jr.

Thesis Adviser

Dr. Sabit Ekin

Dr. Keith Teague

ACKNOWLEDGMENTS

I would like to sincerely thank my adviser, Dr. James E. Stine, Jr., for all of the time and effort he has spent to help guide me through this research.

I would like to thank Dr. Sabit Ekin and Dr. Keith Teague for serving as my committee members and providing advice throughout this work.

I would like to express my thanks and appreciation to my parents, David and Gail, for their love and support throughout my entire life.

I would like to express my love and appreciation to my fiancé, Amanda, for her unending support and encouragement in all aspects of my life.

Acknowledgements reflect the views of the author and are not endorsed by committee members or Oklahoma State University.

Name: John Tobola

Date of Degree: May, 2018

Title of Study: OPTIMIZED EFFICIENT SOFT-DECISION VITERBI ARCHITECTURE FOR APPLICATION-SPECIFIC PROCESSORS

Major Field: Electrical Engineering

Abstract: This thesis provides an efficient implementation of a soft-decision Viterbi decoder implemented in Global Foundries cmos32soi 32nm technology. This architecture utilizes an efficient branch metric (BM) and normalization architecture by using application specific squaring and comparator units. Results indicate a good trade off between area, delay, and power. Compared to a previous implementation, results indicate a significant decrease in area and delay while running in excess of 1 GHz. In addition, when compared with a soft-decision implementation using a traditional multiplier and comparator, significant reductions in area, delay, and power were observed. Results are given based on using ARM-based standard-cells, and energy/power results are based on Hardware-Descriptive Language implementation. Although this architecture uses more area than hard-decision branch metric implementations, soft-decision implementations can decrease the bit error rate two orders of magnitude thus reducing possible retransmit rates, resulting in both increased throughput and transmission rates.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The Viterbi algorithm is an important decoding algorithm used for covolutional codes [1]. Convolutional codes create a state transition structure called a trellis as information is encoded. Viterbi decoding works by finding the most-likely (ML) path of received data through this trellis. Hardware implementations of the Viterbi algorithm for decoding purposes are referred to as Viterbi decoders. Viterbi decoders are often used in communications to decrease the bit error rate (BER) of data as it travels across a noisy channel [2]. It is also used in signal processing and many wireless applications. In addition, it is particularly important for an ever-increasing use of Internet-of-Things (IoT) devices which cause a large increase in congestion.

Viterbi decoders are recursive by nature and cycle through the same hardware many times. Therefore, to implement an effective Viterbi decoder in terms of area, delay, and power, it is necessary to optimize the computations and flow of the Viterbi datapath. A Viterbi decoder can calculate the ML path by using either hard-decisions or soft-decisions, and optimizations are dependent on which of these architectures is used. Hard-decision decoders use the Hamming distance between received inputs and possible trellis output values to determine state transitions, and soft-decision decoders use the squared Euclidean distance between received inputs and possible trellis output values. Soft-decision decoders have been shown to exhibit a 3-dB coding gain increase compared to hard-decision decoders in an additive white Gaussian noise (AWGN) channel [2], making them a desirable design choice due to their increased accuracy of reception. However, soft-decision decoders can require a large amount of squaring

operations that can be costly in digital hardware.

Although the Viterbi algorithm is an efficient implementation in Very Large Scale Integration (VLSI) architectures, it tends to be shown in block diagrams. This thesis presents an implementation and optimization targeted at VLSI architectures for use in sequential soft-decision decoders. The architecture is implemented in Verilog and synthesized for use with high-performance multiple threshold voltage standard-cells. The fixed-point soft-decision Viterbi decoder is optimized by using an efficient dedicated squaring datapath and a logarithmic depth two's-complement comparator used to select the ML path and carry out modulo normalization. In addition, this thesis provides a straight-forward architecture for a shift-register based storage system that allows a designer to quickly bring up a Viterbi decoder without the need to use memory and memory-management hardware such as SRAM, address generators, and pointers.

This thesis is organized as follows. Chapter 2 presents background information. Chapter 3 presents optimizations and the final datapath design of the optimized soft-decision Viterbi decoder. Chapter 4 presents area, delay, and power estimates for the decoder design when implemented using topographical synthesis in a 32nm CMOS technology. Chapter 5 presents the conclusions.

CHAPTER 2

VITERBI DECODER

Viterbi decoders are used to process data that is encoded using convolutional codes. Convolutional codes are usually generated using a shift-register, such as the one seen in Figure 2.1.

As k input bits are read into the encoder serially, the input bits are added together with various bits in the shift-register to produce n output bits. Each of the n output bits is calculated according to a generator, g, which specifies which bits are added together. The generators of an encoder are often given as octal or binary values. When represented in binary form, a 1 in a generator value means that corresponding bit position is used to calculate the respective output bit. The number of registers in the shift-register plus the current input value is referred to as K, or the constraint length. It should be noted that K, the constraint length, is different from k, the number of input bits. The number of registers in the shift-register is then equal to

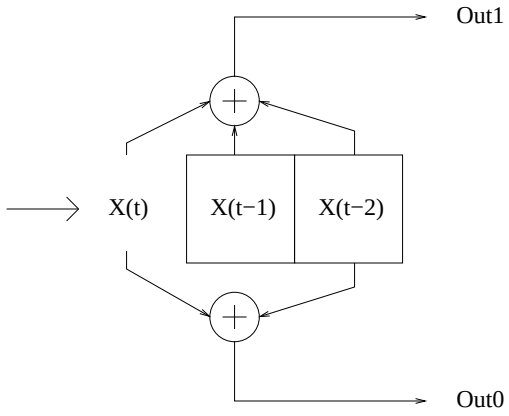

Figure 2.1: Convolutional Encoder for $g = \{111_2, 101_2\}$.

$K-1$, and the $K-1$ bits in the shift register are used as the state of the encoder. Therefore the encoder has a total of 2^{K-1} possible states. After the current input bits are used to calculate the output bits, the current input is then moved into the shift-register and the next input value is read in. This process is repeated until all inputs have been read into the encoder. The code rate of the encoder is equal to k/n, meaning for k input bits, there are n output bits. So for the common case where one bit is read in at a time to produce two output bits, the code rate is equal to 1/2. For example, the encoder in Fig 2.1 has $K = 3$, $k = 1$, states $= 2^{(3-1)} = 4$, $g = \{111_2, 101_2\}$ and code rate $= 1/2$.

As mentioned, the $K-1$ length shift-register is referred to as the state of the encoder, meaning that convolutional encoding can be represented as a state transition data structure. The most common state transfer representation for communications applications is called the trellis. In a trellis, the encoder is represented as a vertical grouping of all possible states at time instant t. This vertical grouping is then followed by another vertical grouping to the right, representing all possible states at time instant $t+1$. As bits are read into the encoder, they cause the state to change by appending a zero or one to the left side of the state values. The transitions of the states are represented on the trellis by drawing lines between states at different time instants to show the possible state transitions. As this is done, the bit that caused the transition and its respective encoder output are drawn beside the line.

This method is continued for all input bits. The resulting trellis shows all possible state transitions that can occur for a given starting state. For convolutional encoders, it is almost always assumed that the initial starting state is the zero state. In this case, transitions from other states at time instant t can be ignored. In addition, often $K-1$ zeros are appended to the original data to be sent in order to always bring the trellis back to its zero state after encoding a packet of data.

When these codes are read in at the receiver, the same trellis can be used to decode

the data. The received n encoded bits are compared with the possible n output bits of each state transition to find the possible output that is closest in value to the received bits. This difference measurement of each branch is known as the branch metric (BM). For an encoder that reads in one bit at a time, each state after time instant t will have two branches leaving from it and two branches coming into it. At time instant $t + 1$, each state in the trellis will compare the BMs of the branches coming into it and find the branch with the smallest BM. It will choose this branch and set it as its current path metric (PM). At each subsequent time instant, each state will take as input the two possible BMs as well as the PMs of the previous states for a given state transition. The state will add each BM and PM pair together and choose the minimum sum to set as its PM. This process is represented by the following equation,

$$PM[i]_{(t+1)} = \min(PM[k]_{(t)} + BM([k], [i])) \tag{2.1}$$

that represents the transition from state k to state i [3]. This process is repeated for all encoded inputs. Since this calculation adds the BM and PM together, compares them, and selects the minimum value, this calculation is known as add-compare-select

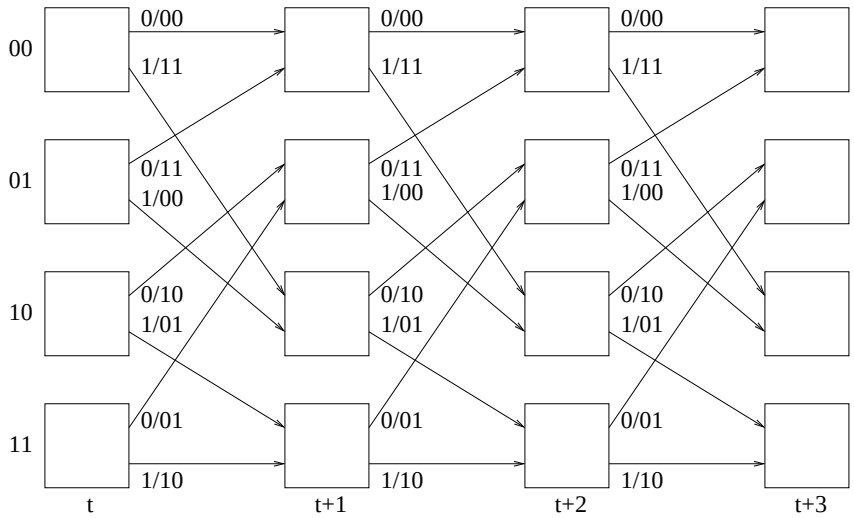

Figure 2.2: Trellis for Encoder in Figure 2.1.

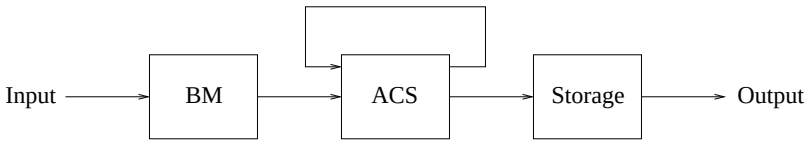

Figure 2.3: Viterbi Decoder Processing (Adapted from [4]).

(ACS). After each ACS operation, the state saves in some form of memory which path it chose as the minimum path. It will save a zero if the minimum path was at its zero input, and a one if the minimum path was at its one input. These saved values are called decision bits. This information is saved in order to trace back through the trellis and recover the original data later on. The overall function of the trellis is to keep track of what path was most likely taken to encode the original data, and a high level data flow can be seen in Figure 2.3 [4]. The final goal of decoding the received data is to find the path through the trellis with the minimum PM. According to the algorithm, the path with the minimum PM will be the ML path and therefore represent what the data is most likely to be according to the possible state transitions of the trellis.

By rearranging the trellis, it can be seen that it is made up of several butterfly structures, as shown in Figure 2.4. This regularity in the algorithm can be utilized to improve the VLSI implementation of the algorithm. This strategy will be discussed more in the following chapter.

Once the minimum PM and its corresponding state at the last time instant is found, the decoder uses the decision bits to trace back through the trellis to find the original data. The original data is equal to the decision bits for the minimum path through the trellis. Since the output is generated by traversing the trellis backwards however, the output will be in reverse order from the original data. Therefore the trace back output has to be reversed before it is finally given as an output.

There are two ways to calculate the BM of a state transition: hard-decisions or

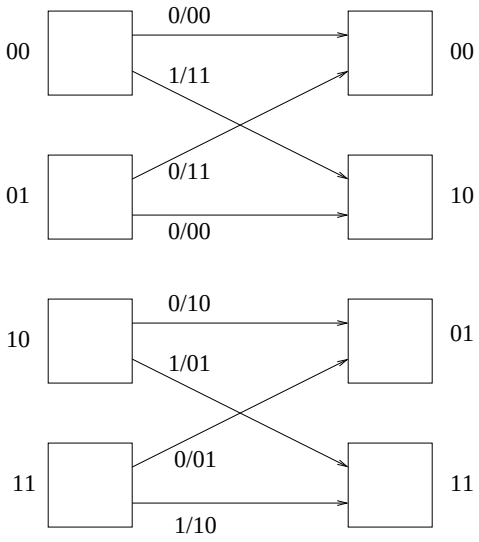

Figure 2.4: Butterfly Property of Convolutional Trellis.

soft-decisions. Hard-decisions use the Hamming distance between the coded received values and the possible trellis outputs to calculate the BM. The Hamming distance is the number of bits that are different for a given bit position between two binary values. For example the Hamming distance between 110 and 111 is

$$d(111, 110) = 1 \tag{2.2}$$

Soft-decisions use the squared Euclidean distance between the fixed-point received values and the theoretical fixed-point possible trellis outputs to calculate the BM [2]. The equation for the squared Euclidean distance can be seen below.

$$d^2 = \sum_{i=1}^{n} (x_i - y_i)^2 \tag{2.3}$$

It is easier to implement hard-decision BM modules in digital hardware, however the resulting BM is not as accurate as the soft-decision BM. A significant reduction in BER can occur by using soft-decision versus hard-decision as shown by Figure 2.5 [5]. Soft-decision calculations can be more difficult to implement, and some methods require multiplier hardware that causes an increase in delay, area, and power. Moreover, the utilization of traditional multipliers for computing Euclidean distances can

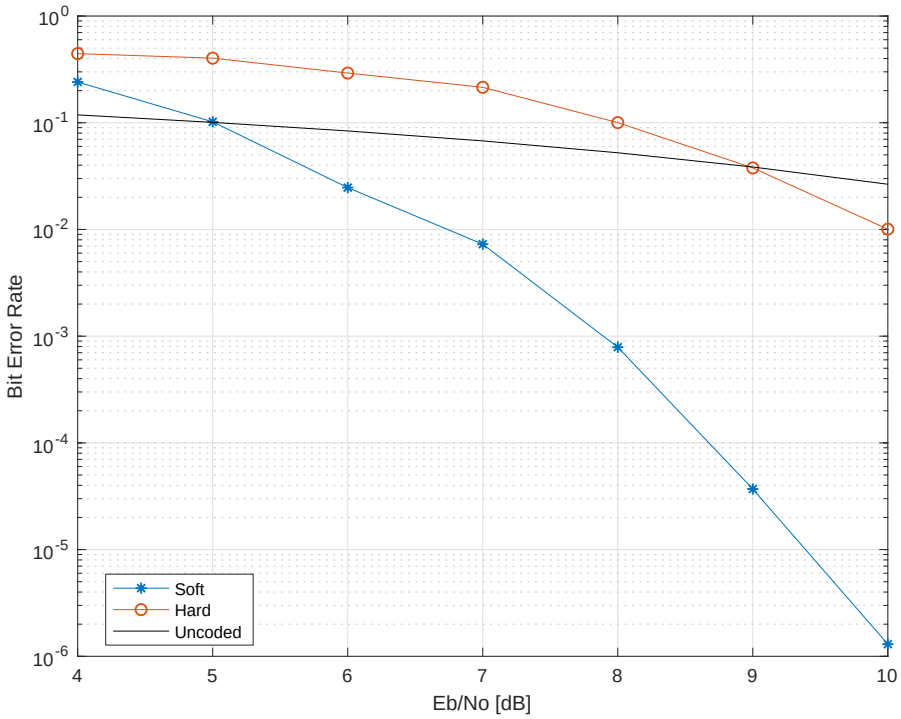

Figure 2.5: Bit Error Rate (BER) for Hard vs. Soft Decision Viterbi Decoders (Adapted from [5]).

be notoriously draining on energy consumption due to the high amounts of power dissipation consumed for multiplication [6]. However, it has been shown that using soft-decision BM calculations can cause a 3-dB BER reduction [2]. In addition, squaring architectures can be an effective method for reducing area, delay, and energy consumption where application-specific hardware is required. Therefore this paper works to demonstrate an efficient implementation of a soft-decision Viterbi decoder applying application-specific hardware to give an efficient implementation for soft-decision decoding.

CHAPTER 3

IMPLEMENTATION AND OPTIMIZATIONS

Before implementing the design, a communication system for the decoder was specified. The decoder was designed around the convolutional encoder specified in the IEEE 802.11 WLAN standard [7]. According to this standard, the system has:

- Constraint Length = $K = 7$
- States = $2^{(K-1)} = 2^6 = 64$
- Code Rate = $1/2$
- Generators = $\{1011011_2, 1111001_2\} = \{133_8, 171_8\}$

In addition, for this decoder it was assumed that the digital data is encoded using bipolar Non-Return-to-Zero-Level (NRZ-L) digital encoding with 0 = −1 and 1 = 1, and transmitted using binary phase shift keying (BPSK). The analog-to-digital converter of the receiver has a reference voltage of 1 V and converts the received signal into two's-complement (1.7) notation values using mid-tread quantization, where (1.7) is the notation used for fixed-point systems meaning the values are represented with one integer bit and seven fractional bits. The fixed-point value 1.000000 is used to represent -1, and 0.1111111 is used to represent 1. It should be noted 0.1111111 is slightly less than 1, but since it is the largest representable value for the (1.7) quantization, it is used to represent 1. Prior to being encoded, the original data is assumed to be in 32-bit packets, with $K - 1 = 6$ zero bits appended to the end of the data to bring the trellis state back to zero during encoding. Therefore, the encoder

takes in 38 bits at a time, and due to the 1/2 code rate, the decoder will therefore take in $2 \times 38 = 76$ bits at a time. After processing, the decoder will output the original 38 bits.

3.1 Soft-Decision Viterbi Decoder Implementation

The decoder implementation discussed in this paper is based on the architecture described in [3]. However, the proposed implementation utilizes a parallel structure with only one processing cycle.

The overall decoder architecture can be seen in Figure 3.1. It is made up of three main sections: the processing section, the memory section, and the decode section. The processing section is made out of the processing element (PE) array on the left side of the architecture. The PE array represents the trellis and this area is the main datapath for calculating all metrics and decisions needed for the Viterbi algorithm. The memory section is made up of the last-in-first-out (LIFO) modules in the middle of the architecture. These LIFOs store the decision bits from each PE. Finally, the decode stage is made up of all modules to the right of the LIFOs in the architecture. This section is where the decision bits are processed and turned back into the original data.

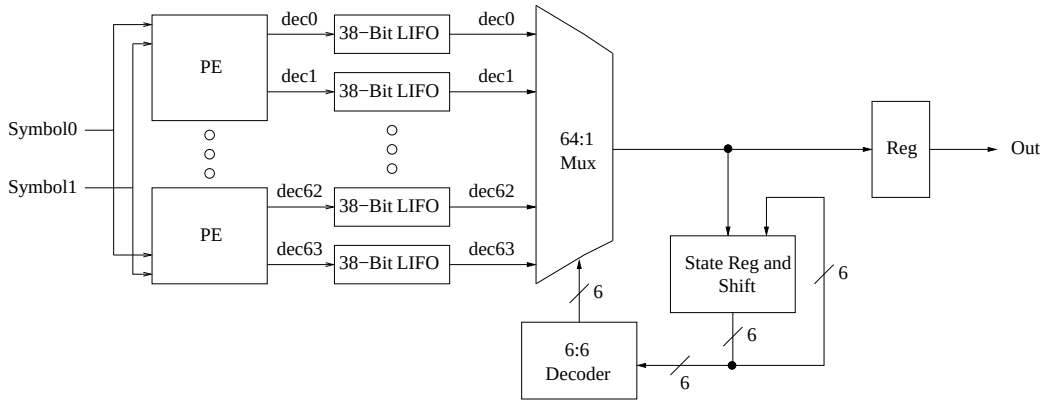

Figure 3.1: Overall System Architecture.

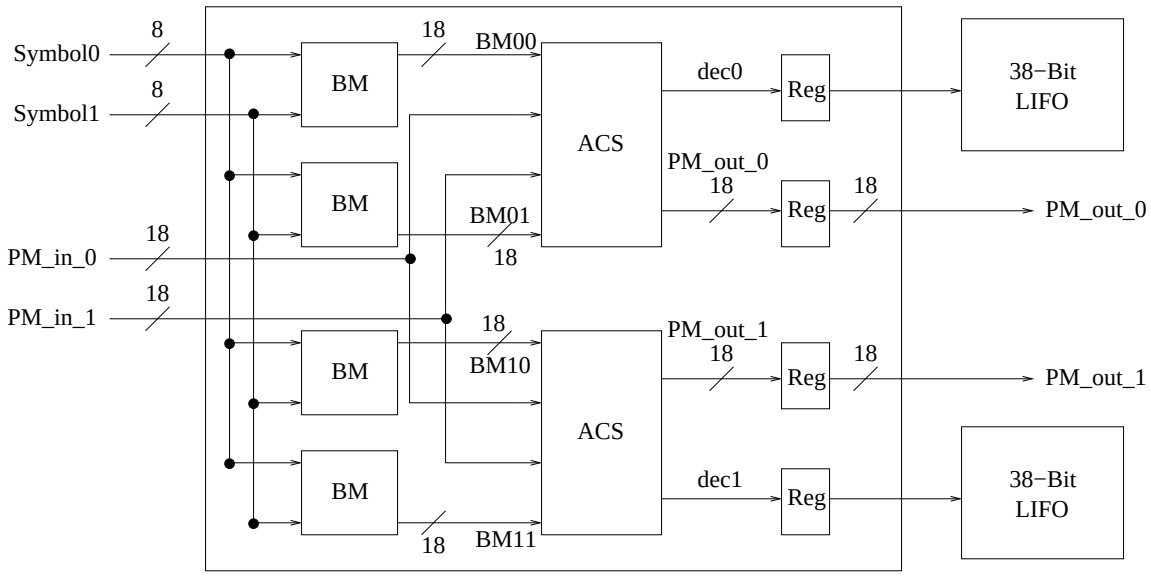

Figure 3.2: PE Architecture.

This architecture is implemented by using the fact that the trellis can be rearranged to form butterfly structures, as shown in Figure 2.4, which allow the Viterbi decoder to be implemented using the parallel array architecture of PEs shown in Figure 3.1. Each PE represents a butterfly from the trellis, and therefore each PE represents two different states. The PE consists of four BM modules (one for each branch of the butterfly) and two add-compare-select (ACS) modules (one for each state of the butterfly at the given time instant) as shown in Figure 3.2. The ACS modules pass the decision bits to two 38-bit LIFO modules after passing them through a register. Since there are sixty-four states and each PE handles two states, thirty-two PE modules need to be instantiated and connected according to the trellis.

In the implemented Viterbi decoder, there were four versions of the PE architecture developed. Each version differed only by what hard coded value the BM modules used to compare with the input symbols. When looking at the full trellis diagram for this encoder/decoder, it can be seen for each butterfly the top and bottom outputs are equal, and the two middle branch outputs are equal. This characteristic is demonstrated in Figure 2.4. There are four possible branch output orderings, and

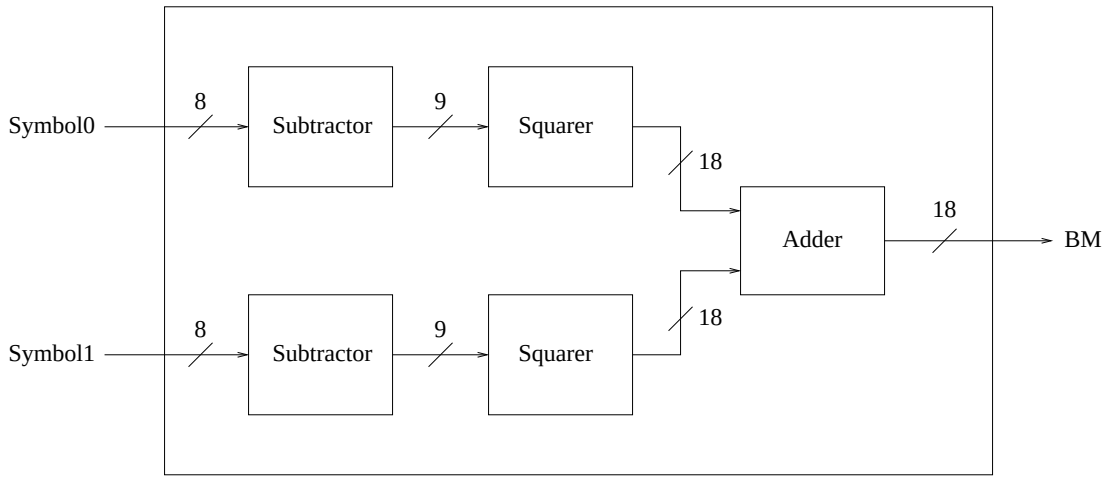

Figure 3.3: Branch Metric (BM) Architecture.

each PE version represents one of these orderings.

The BM module is a structural combination of a subtractor, squarer, and adder in order to implement Equation (2.3). The received Symbol0 is subtracted from the expected Symbol0 for the given BM module. This value is then squared and added to the value of the received Symbol1 subtracted from the expected Symbol1, squared. The BM architecture can be seen in Figure 3.3. Since a fixed-point system was utilized, great care was used to ensure the accuracy of results. In order to ensure accuracy in the BM module, an integer bit had to be added in the subtractor. This addition of a bit can be seen in Figure 3.3, where the subtractor takes in an 8 bit value but outputs a 9 bit value.

The ACS module is a structural combination of two two's-complement adders, a modified signed comparator, and a multiplexer, as described in [8] and [9]. The adders add the path metric (PM) of the previous state with the BM for both sets of ACS inputs. The sums are then sent to the signed comparator to find the smaller PM. The comparator then sends a select signal to the multiplexer to choose the corresponding PM with the smallest value. The ACS architecture can be seen in Figure 3.4.

The signed comparator is modified to implement a technique known as modulo

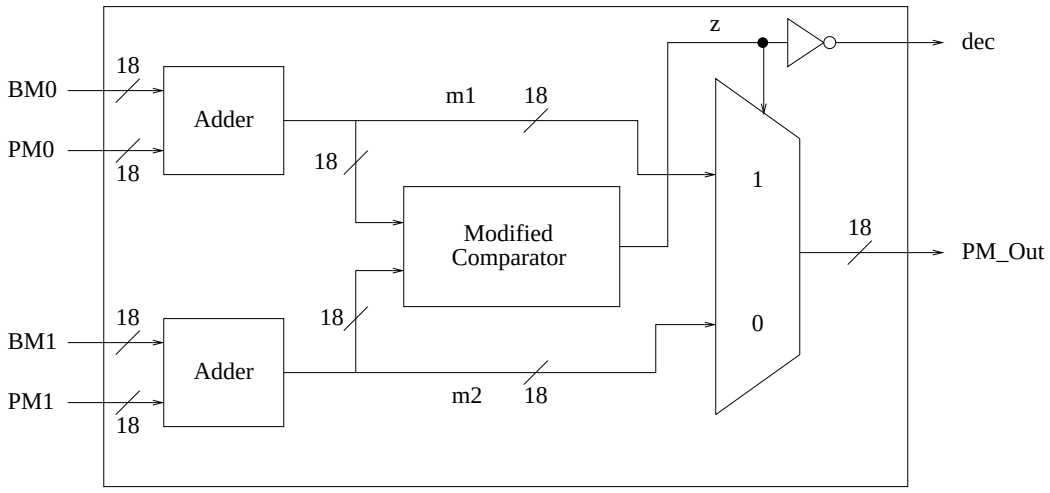

Figure 3.4: Add-Compare-Select (ACS) Architecture.

normalization, again as described in [8] and [9]. As the decoder is running, it is inevitable that eventually the PM will exceed its bit limit and overflow. In order to handle this issue, modulo normalization utilizes the fact that the difference in the PM values is bounded in order to control overflow in a way that preserves the signed comparison. To implement modulo normalization, the comparator must be of the modified form shown in Figure 3.5.

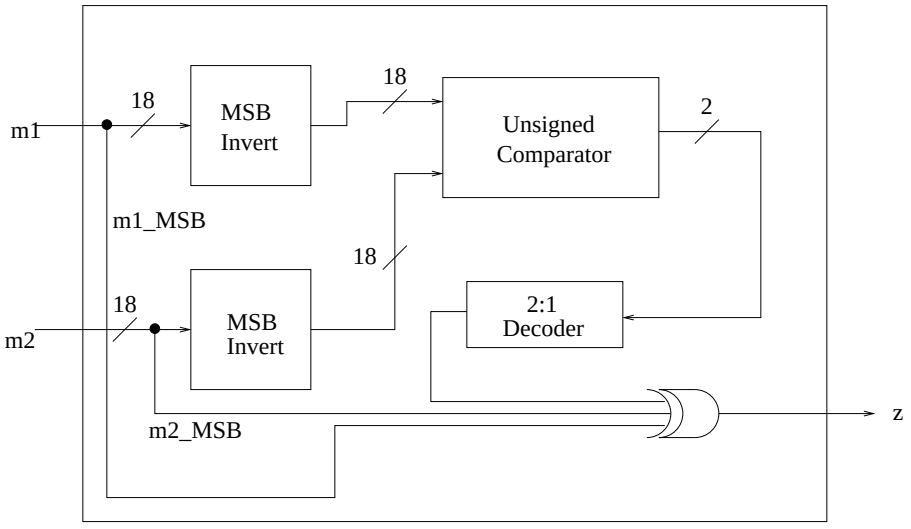

Figure 3.5: Modified Comparator for Modulo Normalization.

Modulo normalization works by having the modified comparator generate a select signal, z, that is the result of performing the XOR operation on the MSB of m1, the MSB of m2, and the result of an unsigned comparison of m1 and m2. For the unsigned comparison, the output equals zero if m1 is greater than m2 and one if m1 is less than m2. Therefore, z is equal to

$$\mathrm{z} = \mathrm{m1}_{MSB} \oplus \mathrm{m2}_{MSB} \oplus \mathrm{y(m1,\ m2)} \tag{3.1}$$

where y is equal to the unsigned comparison of m1 and m2 [8]. The MSB invert modules allow two's-complement values to be compared in an unsigned comparator by inverting the MSBs before comparison, as described in [10]. By making this inversion, negative two's-complement values will always appear smaller than non-negative two's-complement values. The 2:1 decoder module takes the two-bit output of the unsigned comparator and decodes it to a one-bit representation, where one means m1 is less than m2, and zero means m1 is greater than m2.

If m1 and m2 are equal, the logic in the 2:1 decoder can be designed to choose either m1 or m2 as the smallest sum. It does not matter which value is chosen since the PM sums are equal at that time instant and therefore have the same likelihood of being the ML path.

Since the decision bits are generated while traversing forward through the trellis, but decoding traces back through the trellis, the decision bits need to be stored in a LIFO. Since each ACS will generate a decision bit each clock cycle, each ACS needs its own LIFO. Each LIFO should be able to hold thirty-eight bits: thirty-two for the original data and six for the appended zeros to return the trellis to its zero state.

After all input bits have been read in and all decision bits have been stored in the LIFO, the decision bits can be read out of the LIFO into the trace back decode stage. The architecture for this decode stage is shown in Figure 3.6.

To read the correct decision bit each clock cycle while reading the values from the LIFO modules, a 6-bit state register must be used to keep track of the ML path

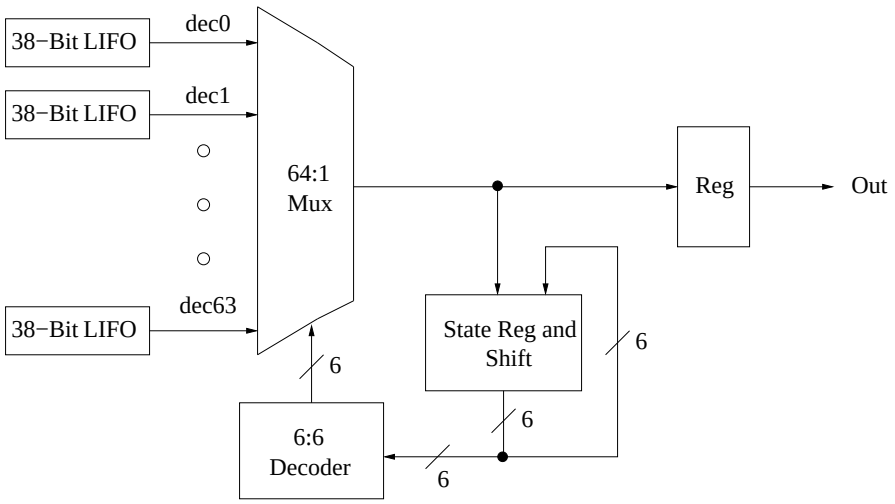

Figure 3.6: Trace-Back Decode Architecture.

as the trellis is traversed. Since it is known that the trellis was brought back to the zero state, the state register should be set to start in the zero state. This state value goes to the 6:6 decoder, where the non-sequential butterfly state mapping is mapped to sequential values to act as a select signal for the 64:1 multiplexer. At each clock cycle, the multiplexer will read the decision bit corresponding to the current state. This decision bit will be sent to the output register, and it will also be appended to the right-side of the state in the state register to provide the value for the next state. This process repeats thirty-eight times until the state register is back to the zero state. The values coming out of the output register are the original data values before encoding, only in reverse order. The zeros appended to the original data are also output, but these values can be removed in further processing.

3.2 Optimization using Squaring and Comparison

In order to optimize the Viterbi decoder, changes were made to the squaring module, the modified comparator used for the ACS modules and modulo normalization, and the LIFO design.

To make the squaring operation as efficient as possible, a squaring architecture

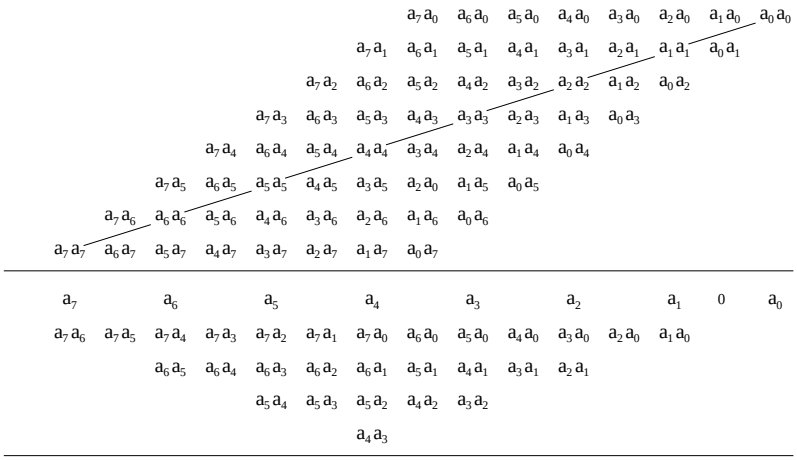

Figure 3.7: Partial Product Generation Reduction for Squaring Matrices.

described in [11] was used. This architecture is a hierarchical combinatorial method used to reduce the size of the partial-product-matrix (PPM). Squaring is more optimal than parallel multiplier architectures in that it has a significant reduction in partial products due to symmetry of partial products that get generated. This is illustrated in Figure 3.7 where simplification of the partial product matrix can happen along the diagonal of the PPM. First, since a value is multiplied with itself when squared, the multiplication of the same bit position of the multiplicand and multiplier is equal to the bit value in that bit position. For example, if the value $a_2a_1a_0$ is multiplied with itself, when multiplying the bit a_0 with a_0 the result is $a_0 \cdot a_0 = a_0$. Also, squaring has a commutative property which can be used to reduce the PPM. Since multiplication itself is commutative, a bit multiplication such as $a_0 \cdot a_1$ will equal $a_1 \cdot a_0$. Therefore, when adding values such as these in the PPM, the values can be simplified to $a_0 \cdot a_1 + a_1 \cdot a_0 = 2 \cdot a_1 \cdot a_0$, which is equivalent to a left shift of the value $a_1 \cdot a_0$. Both of these symmetry reductions can be seen in Figure 3.7. In addition, further optimization can happen within the PPM by using some Boolean logic simplifications, where i represents specific weight positions for a given input

operand x of size n-bits:

$$\begin{aligned}(x_i \cdot x_{i-1} + x_i) \cdot 2^{2 \cdot i - 2 \cdot n} &= x_i \cdot x_{i-1} \cdot 2^{2 \cdot i + 1 - 2 \cdot n} + \\ & \quad x_i \cdot \overline{x_{i-1}} \cdot 2^{2 \cdot i - 2 \cdot n} \end{aligned} \tag{3.2}$$

An additional benefit to using this squaring architecture is that a recursive divide-and-conquer architecture can be formed to reduce the PPM [11]. Using this method, the input operand x is divided into two parts $a = 0.x_{2m-1} \ldots x_m$ and $b = 0.x_{m-1} \ldots x_0$, so each part contains m-bits, and the final result p can be represented as

$$\begin{aligned} x &= a + b \cdot 2^{-m} \\ p &= x^2 \\ &= (a + b \cdot 2^{-m})^2 \\ &= a^2 + 2 \cdot a \cdot b \cdot 2^{-m} + b^2 \cdot 2^{-2 \cdot m} \end{aligned} \tag{3.3}$$

The PPM is modified to handle two's-complement numbers, and a Dadda-reduction [12] scheme is utilized for the reduction of the PPM. Once the PPM is reduced from the squaring architecture, a fast carry-propagate adder (CPA) is then utilized to finalize the product. Since squaring is one of the most used functions in the soft-decision Viterbi datapath, the use of an efficient squaring module will improve both delay and power costs.

To improve the performance of the heavily used ACS modules, an efficient two's-complement comparator is used in the modified comparator. The architecture is based on the design discussed in the paper [10], and can be seen in Figure 3.8. The comparator uses a logarithmic depth design to provide an efficient, parallel comparison. The logarithmic depth design is achieved by breaking up the input words into two-bit subwords and comparing them. The results are then sent to other two-bit word comparators until the final result is obtained. The logarithmic depth architecture reduces delay compared with a traditional comparator architecture [10]. In addition, the unit

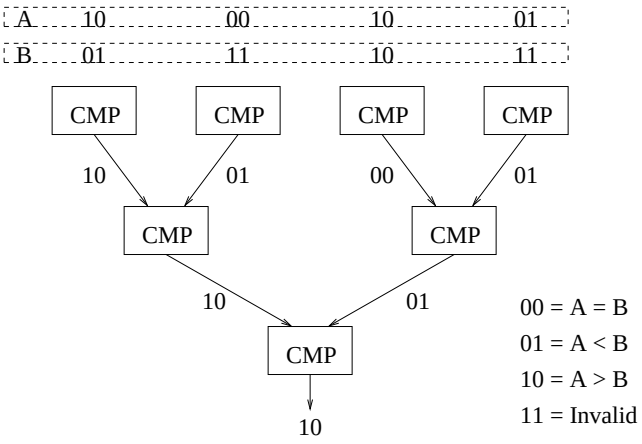

Figure 3.8: Comparator with Logarithmic Depth Architecture.

implements a two's-complement optimization by inverting the MSBs of each input. This inversion allows two's-complement values to be compared in the logarithmic depth unsigned comparator without having to be converted out of two's-complement before the comparison, and back into two's-complement after.

For this implementation, memory hardware was developed using flip-flop (FF) based shift-registers. This design choice provides an alternative to using SRAM, addressing modules, and pointers. This design approach is advantageous for those without access to memory designs or those in need of a quick design time. Figure 3.9 shows a three-bit implementation of the FF shift-register architecture used for the LIFO storage modules.

To use this LIFO, initially writeClkEn should be set low, Load should be set low, and readClkEn should be set high. These values should be held until all values are read in and the bottom read shift-register is full. Next, writeClkEn should be set high, load should be set high, and readClkEn should be set low. This will bring the word in the bottom read shift-register to the top write shift-register on the next clock cycle. After this clock cycle, load should be set low. The clock can then be run until all bits are out of the write shift-register. This operation will allow the LIFO to store decision bits as they are produced and then read them out in reverse order,

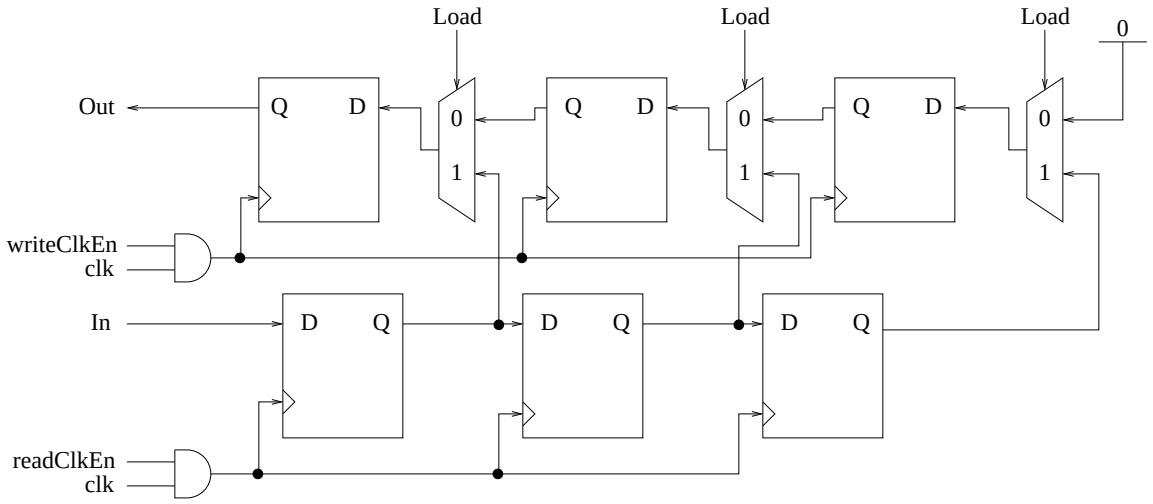

Figure 3.9: FF Shift-Register LIFO Architecture.

as desired. In order to improve performance, the LIFO can be pipelined so that it is both reading and writing values at the same time.

CHAPTER 4

RESULTS

Verification of hardware is sometimes difficult for Viterbi decoders as most implementations only have hard-decision architectures. However, it has been shown good signal-to-noise ratio (SNR) on a given channel using soft-decision decoding can improve the BER by several orders of magnitude [2]. This can be important for good transmission rates. For example, assuming a hard-decision mapping has a BER of 10^{-1} and a soft-decision mapping has a BER of 10^{-3}, this can provide significant savings in avoiding re-transmission of bits that are lost due to using hard-decision mapping (i.e., errors of 1 in 1000 instead of 1 in 100). Therefore, soft-decision, although more hardware intensive, has obvious savings if the area, delay, and power can be reduced significantly.

The proposed decoder was implemented in RTL-compliant Verilog and then synthesized in an ARM 32nm CMOS library in Global Foundries (GF) cmos32soi technology. The ARM standard-cell library utilizes multiple values of V_T to aid in synthesis (i.e., MTCMOS). Synthesis was optimized for delay utilizing Synopsys® Design Compiler™ (DC) in topographical mode using a PVT process at 25° C using TT corners. The average power estimation was achieved by running the simulation on $1,000$ random test vectors. The synthesis scripts are synthesized for delay using a 1ns clock (1 GHz) and a $5\times$ loading of a nominal flip-flop.

There were five different threshold voltage libraries used during synthesis: HVT, MVT, RVT, SVT, and UVT. HVT signifies "high" threshold voltage, MVT signifies "mezzanine" threshold voltage, RVT signifies "regular" threshold voltage, SVT signi-

fies "super-high" threshold voltage, and UVT signifies "ultra-high" threshold voltage [13]. Due to the higher threshold voltage, the HVT, SVT, and UVT libraries have a lower static power but higher dynamic power. The lower threshold voltage MVT and RVT libraries have a lower dynamic power but a slightly higher static power. By synthesizing the designs to different combinations of these libraries, the set of libraries that optimizes the proposed design in terms of lowering area, delay, and power was determined experimentally.

In order to evaluate the proposed design, it was compared with the same architecture using a traditional multiplier to carry out squaring and a traditional comparator to carry out the the ACS function and normalization. Also, a pipelined radix 4 soft-decision decoder described in [14] was used as a comparison. It should be noted that the pipelined radix 4 design was implemented using 130nm standard cells. The comparison to [14] is difficult however as the results do not include synthesis constraints, operating conditions, loading, or actual physical numbers (i.e., they are only plotted on a Figure). Again, comparisons are difficult because of the limited availability of soft-decision mapping Viterbi decoding architectures. In addition, comparisons are often not run with energy, and the code utilized is not available. For additional comparison, several hard-decision mapping implementations were synthesized using the same design constraints.

The hard-decision comparisons come from two sources. First, MATLAB™ has a Viterbi decoder generator that utilizes HDL Coder [15]™. It was made to be a "soft-decision" decoder, but it converts input data into a range of eight categories, then performs hard-decision decoding. Since MATLAB™ left off the logic to convert the input to one of the eight category values, it is only a hard-decision decoder. Second, an open-source hard-decision Viterbi decoder generator is utilized as a comparison [16]. This version also utilizes a module called `virtual_mem` that is left-off synthesis, because it is generated for SRAM implementations. Therefore, its area

RVT		#Cells/Area						Power [mW]			
Version	Type	#Comb	#Seq	#Hier	#Total	Area [mm^2]	Delay [ps]	Internal	Static	Dynamic	Total
MATLAB™ [15]	hard	12,628	4,904	102	17,634	0.0234	721.1	24.41	7.47	5.92	37.79
Open-Source [16]	hard	20,948	9,219	87	30,254	0.0376	467.32	39.22	13.36	2.04	54.62
Pipelined Radix 4 [14]	soft	–	–	–	≈ 179,000	2.5160	≈ 1,000.00	–	–	–	–
Proposed - No Opt.	soft	133,654	7,112	75,940	216,706	0.1087	1,086.88	96.47	44.87	72.90	214.24
Proposed - Squarer	soft	100,073	6,216	37,924	144,213	0.0882	879.47	73.47	35.89	58.72	168.08
Proposed - Opt Comp.	soft	136,206	7,112	77,220	220,538	0.1097	1,044.52	97.29	45.38	72.33	215.00
Proposed	soft	101,579	6,216	39,204	146,999	0.0891	880.59	74.44	36.16	59.96	170.56

Table 4.1: 32nm Synopsys® DC™ Synthesized Topographical Results using GF ARM-based RVT standard-cells (Note: no details other than a 130nm CMOS implementation in [14] is given, therefore, the comparison is difficult).

HVT, MVT, RVT		#Cells/Area						Power [mW]			
Version	Type	#Comb	#Seq	#Hier	#Total	Area [mm^2]	Delay [ps]	Internal	Static	Dynamic	Total
MATLAB™ [15]	hard	12,508	4,904	102	17,514	0.0233	901.8	21.38	1.04	5.89	28.32
Open-Source [16]	hard	21,055	9,219	87	30,361	0.0377	605.45	36.90	2.05	1.96	40.90
Pipelined Radix 4 [14]	soft	–	–	–	≈ 179,000	2.5160	≈ 1,000.00	–	–	–	–
Proposed - No Opt.	soft	135,608	7,112	75,940	218,660	0.1220	1,125.63	102.11	8.25	74.44	184.81
Proposed - Squarer	soft	98,151	6,216	37,924	142,291	0.0875	1,017.98	66.42	4.97	57.08	128.48
Proposed - Opt. Comp	soft	139,524	7,112	77,220	223,856	0.1265	1,061.81	106.86	9.063	80.64	196.56
Proposed	soft	100,818	6,217	39,204	146,239	0.0888	1099.70	67.71	5.06	58.85	131.62

Table 4.2: 32nm Synopsys® DC™ Synthesized Topographical Results using GF ARM-based HVT, MVT, RVT standard-cells (Note: no details other than a 130nm CMOS implementation in [14] is given, therefore, the comparison is difficult).

may be substantially increased due to the need for an SRAM instantiation.

Results are shown in Table 4.1 and Table 4.2. As mentioned, the set of libraries which optimized the proposed design in terms of lowering area, delay, and power was determined experimentally. It was found that using the RVT library alone and using the HVT, MVT, and RVT libraries together provided the optimal results. Since the results from both of these groupings were similar, both results have been presented. Cell counts are given for combinational and sequential devices.

During synthesis, Synopsys® DC™ looks for logic common to several design

paths. When this common logic is found, the synthesis engine will hierarchically abstract it in a way that allows multiple paths to utilize the same logic. This process reduces the total number of cells required and thus optimizes the design in terms of area. Synopsys® DC™ categorizes the cells used in this optimized logic as *hierarchical* cells. The number of these hierarchical cells can be found by subtracting the sum of the combinational and sequential cells from the total number of cells used in the top level design.

To verify the proposed design, test values were generated in MATLAB™ using convolutional encoder scripts found in [17]. As energy/power is input dependent, random vectors were generated through a value change dump (VCD) file and a switching activity intergchange format (SAIF) file and then used to obtain the power numbers in Table 4.1 and Table 4.2.

As seen in Table 4.1, the proposed design synthesized using RVT cells has a significant reduction in area, power, and delay compared to the implementation with a traditional multiplier and comparator (no optimizations). The use of the proposed architecture provides a reduction in the number of total standard cells (-24%), which contributes to the observed reduction in total power (-20%) and area (-18%). It also simplifies the squaring calculation and speeds up comparison, providing a shorter delay (-19%). As expected, the soft-decision architecture has more area than the hard-decision architectures. However, soft-decision architectures provide the significant benefit of a 2-4 order of improvement in BER over hard-decision architectures, and due to nanometer technologies providing the ability to incorporate more logic into the design, the proposed design only incurs $3\times$ more area than the optimized hard-decision version generated from MATLAB™ with about the same delay. In addition, compared to [14] the proposed design has considerable savings in cell count (-18%), area (-96%), and delay (-12%). It should be noted however, that the drastic difference in area is largely due to the proposed design's use of a 32nm technology versus

the 130nm technology used in [14]. Moreover, the 130nm implementation in [14] uses a pipelined array multiplier that would expend more area than the application-specific squarer in this paper. The proposed implementation is also memory-less and can be implemented easily with standard-cells or within custom-logic without using SRAM. Also, the hierarchical cell counts can be reduced with changes in loading.

Table 4.2 shows similar results when the proposed design is synthesized using HVT, RVT, and MVT cells. When compared to the implementation using a traditional multiplier and comparator (no optimizations), the squaring architecture provides a slightly more significant reduction in the number of total standard cells (-33%) than when the design is synthesized to RVT cells alone, which also leads to a more significant reduction in total power (-29%) and area (-27%). However, these more significant reductions come at the cost of only decreasing the delay by (-2%). When compared with [14], again the proposed design has considerable savings in cell count (-18%) and area (-96%), however there is a (10%) increase in delay.

The data with the squarer or optimized comparator individually added to the proposed architecture is also provided in Table 4.1 and Table 4.2. It can be seen that using the optimized comparator without the squarer causes a slight increase in area and power, but decreases the overall delay compared to using no optimizations. In addition, when the optimized comparator and squarer are used together, the comparator causes a slight increase in area, delay, and power compared to the design when only the squarer is used.

It can be assumed for the case where there are no optimizations the synthesizer tried to optimize the design in terms of area and power in exchange for a higher delay. In the case where only the squarer is used, it seems the synthesizer can optimize the traditional comparator in a way that leads to a decreased area, delay, and power when compared to the case when both the squarer and optimized comparator were used.

The optimized comparator is well defined structurally and does not provide as much room for the synthesizer to optimize the design.

While it may appear better to not use the optimized comparator, it is an advantageous design to use in two main cases. First, it is advantageous when a traditional multiplier is being used and delay is the critical metric to be optimized. In addition, it is advantageous when the architecture of the comparator needs to be known instead of being left up to the synthesizer to decide the best design. Otherwise, the results show that it could be advantageous to allow the synthesizer to optimize the traditional comparator.

CHAPTER 5

CONCLUSIONS

An efficient soft-decision Viterbi architecture is presented. It demonstates a significant reduction in area, delay, and power compared to the soft-decision architecture using a traditional multiplier and comparator, and a significant reduction in area and delay compared to a previous soft-decision decoder. Although the proposed implementation incurs more area, delay, and power than hard-decision architectures, the savings in transmission rates makes this implementation a good choice for communication protocols, such as IEEE 802.11. In addition, the implementation can easily be incorporated on digital signal processing architectures since it is memory-less as well as self-contained. This paper also provides results for both soft- and hard-decision architectures for a System on Chip (SoC) implementation.

Synthesis results using RVT standard cells show the proposed design reduces total standard cell count by 25%, total power by 20%, area by 18%, and delay by 19% compared to the soft-decision architecture using a traditional multiplier and comparator. In addition, compared to a similar soft-decision decoder implementation, the proposed design demonstrates an 18% reduction in cell count and a 12% reduction in delay. The proposed design also demonstrates a 96% reduction in area, though a large portion of that reduction is due to the use of a 32nm technology instead of the 130nm technology used in [14]. These results are summarized in Table 5.1 for RVT-based standard-cells using topographical synthesis with Synopsys® DC™.

Moving forward, this research can be continued by carrying out a full BER analysis of the proposed design in both an AWGN and Rayleigh fading channel. In addition,

RVT Version	Type	#Cells/Area #Total	Area [mm^2]	Delay [ps]	Power [mW] Total
MATLAB™ [15]	hard	17,634	0.0234	721.1	37.79
Open-Source [16]	hard	30,254	0.0376	467.32	54.62
Pipelined Radix 4 [14]	soft	≈ 179,000	2.5160	≈ 1,000.00	–
Proposed - With Mult.	soft	216,706	0.1087	1,086.88	214.24
Proposed - Optimized	soft	146,999	0.0891	880.59	170.56

Table 5.1: 32nm Synopsys® DC™ Summary of Synthesized Topographical Results using GF ARM-based RVT standard-cells.

truncated squaring units have been shown to significantly reduce power and area with only a slight reduction in result accuracy when compared to a non-truncated squaring unit [18]. A truncated squaring unit could be used in the BM module of the proposed design to see whether it could further reduce area and power without causing a significant increase in the BER.

BIBLIOGRAPHY

[1] G. D. Forney, "The Viterbi algorithm," *Proceedings of the IEEE*, vol. 61, pp. 268–278, March 1973.

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

[3] M. Boo, F. Arguello, J. D. Bruguera, R. Doallo, and E. L. Zapata, "High-performance VLSI architecture for the Viterbi algorithm," *IEEE Transactions on Communications*, vol. 45, pp. 168–176, Feb 1997.

[4] M. Kamuf, V. Owall, and J. B. Anderson, "Survivor path processing in viterbi decoders using register exchange and traceforward," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 54, pp. 537–541, June 2007.

[5] "MATLAB™ estimate BER for hard and soft decision Viterbi decoding." `https://www.mathworks.com/help/comm/ug/estimate-ber-for-hard-and-soft-decision-viterbi-decoding.html`.

[6] M. J. Schulte, J. E. Stine, and J. G. Jansen, "Reduced power dissipation through truncated multiplication," in *Proceedings IEEE Alessandro Volta Memorial Workshop on Low-Power Design*, pp. 61–69, Mar 1999.

[7] "ISO/IEC 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 (includes ieee std 802.11, 1999 edition; ieee std

802.11a.-1999; ieee std 802.11b.-1999; ieee std 802.11b.-1999/cor 1-2001; and ieee std 802.11d.-2001)," *ISO/IEC 8802-11 IEEE Std 802.11 Second edition 2005-08-01 ISO/IEC 8802 11:2005(E) IEEE Std 802.11i-2003 Edition*, pp. 1–721, 2005.

[8] C. B. Shung, P. H. Siegel, G. Ungerboeck, and H. K. Thapar, "VLSI architectures for metric normalization in the viterbi algorithm," in *IEEE International Conference on Communications, Including Supercomm Technical Sessions*, pp. 1723–1728 vol.4, Apr 1990.

[9] C. Studer, S. Fateh, C. Benkeser, and Q. Huang, "Implementation trade-offs of soft-input soft-output map decoders for convolutional codes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, pp. 2774–2783, Nov 2012.

[10] J. E. Stine and M. J. Schulte, "A combined two's complement and floating-point comparator," in *2005 IEEE International Symposium on Circuits and Systems*, pp. 89–92 Vol. 1, May 2005.

[11] S. Bui, J. E. Stine, and M. Sadeghian, "Experiments with high speed parallel cubing units," in *2014 IEEE Computer Society Annual Symposium on VLSI*, pp. 48–53, July 2014.

[12] L. Dadda, "Some schemes for parallel multipliers," *Alta Frequenza*, vol. 34, pp. 349–356, 1965.

[13] J. M. Johnson, S. K. Springer, R. Thoma, and J. S. Watts, "Method, system and program storage device for generating accurate performance targets for active semiconductor devices during new technology node development," May 28 2013. US Patent 8,453,101 B1.

[14] W. Yoo, Y. Jung, M. Y. Kim, and S. Lee, "A pipelined 8-bit soft decision Viterbi decoder for IEEE802.11ac WLAN systems," *IEEE Transactions on Consumer Electronics*, vol. 58, pp. 1162–1168, November 2012.

[15] "MATLAB™ Viterbi decoder." https://www.mathworks.com/help/comm/ref/viterbidecoder.html.

[16] "VHCG Viterbi HDL codes generator." http://viterbi-gen.sourceforge.net.

[17] J. Proakis and M. Salehi, *Contemporary Communication Systems Using MATLAB™*. BookWare companion series, Brooks/Cole, 2000.

[18] J. E. Stine and O. M. Duverne, "Variations on truncated multiplication," in *Euromicro Symposium on Digital System Design, 2003. Proceedings.*, pp. 112–119, Sept 2003.

VITA

John Tobola

Candidate for the Degree of

Master of Science

Thesis: OPTIMIZED EFFICIENT SOFT-DECISION VITERBI ARCHITECTURE FOR APPLICATION-SPECIFIC PROCESSORS

Major Field: Electrical Engineering

Biographical:

Personal Data: Born in Bartlesville, Oklahoma, United States of America on November 28, 1994.

Education:

Received the B.S. degree from Oklahoma State University, Stillwater, Oklahoma, United States of America, 2017, in Electrical Engineering

Completed the requirements for the degree of Master of Science with a major in Electrical Engineering from Oklahoma State University in May, 2018.

Experience:

FPGA Engineer - Spectranetix, Inc.
May 2018
Accepted position and will start after graduation

Graduate Research Assistant - VLSI Computer Architecture Research Group
OSU
June 2017 - May 2018

Mixed Signal ASIC/SoC Products Intern - Sandia National Laboratories
June 2016 - April 2017