Design and FPGA Implementation of Low Power Punctured Soft Decision Viterbi Decoder

Habibullah Jamal and Zaheer Ahmed

University of Engineering and Technology Taxila

Pakistan – 47050

drhjamal@uettaxila.edu.pk

Abstract

The paper presents the hardware implementation of basic building blocks of viterbi decoder. Different architectures can be created by using these basic building blocks. A novel design of programmable Viterbi Assist is presented by using these basic building blocks that optimizes both the area and power using a folded ACS unit and a special trace back mechanism based on zig-zag shift registers. The design can be used with any DSP processor for MIMD architecture. The host processor provides different address pointers and parameters through shared memory and configuration registers. The Viterbi Assist reads the symbols from the memory, performs Viterbi algorithm and writes decision back in the memory area specified by the host. The architecture can also be used as a standalone Viterbi decoder for a variety of other applications as well. The FPGA implementation of presented architecture is attractive because of the flexibility and efficient parallel architectures available in FPGAs.

Key Words: Add Compare Select (ACS), Multiple Instruction Multiple Data (MIMD), Branch Metric Unit (BMU), Register Exchange Architecture (REA), Folded ACS.

1 Introduction

Viterbi decoders are widely used as forward error correction devices in many digital communication and multimedia products, including mobile phones, video and audio broadcasting receivers, voice over packet gateways and modems. For low bit rate applications, Viterbi decoding is implemented in software on digital signal processors (DSPs). The bit rate required by modern high quality speech transmission represents the current limitation for Viterbi decoder software implementations due to the high computational requirements imposed by the Viterbi algorithm. A Viterbi Assist is presented which can be used as a standalone Viterbi decoder or can be used with any DSP processor to off-load Viterbi decoding.

The Viterbi assist comprises of a micro-coded algorithmic state machine, an adder/subtractor based folded ACS unit, a shifter, logical unit, a few general-purpose registers, address generation logic and host interface. The state machine comprises of sequencer unit, ROM and configuration registers. The host processor programs the Viterbi decoder through

configuration registers. The constraint length, code rate, encoder polynomials, puncturing patterns are some of the programming parameters provided to the assist.

This paper is arranged in five sections. Convolutional encoder and viterbi decoder algorithm are described in section2. The section 3 describes the low power architectures of basic building blocks of viterbi decoder for their hardware implementations. The Viterbi assist architecture is explained in section 4. Section 5 concludes the paper.

2 CONVOLUTIONAL ENCODER AND VITERBI DECODER

Efficient and reliable data transfer is vital for any communication channel that comprises of encoder, modulator, demodulator and decoder sections. For high bandwidth channel, detection of errors and retrieval of original data requires advance error control codes to be implemented in hardware. The viterbi decoder measures the similarity between the received signal and all trellis paths entering each state at time t_i. Every state selects the optimum path termed as surviving path. The state having minimum path metric is traced back to retrieve information bit. For quick VLSI realization, viterbi encoder and decoder units are divided into following basic building blocks.

2.1 Differential Encoder

The differential encoder/decoder, are employed to resolve inverted data. The differential encoder transforms the input data stream into transitions, the output transitions from 0 to 1 or 1 to 0 depending on previous output. The differential decoder is the inverse of differential encoder; the circuit for 180-degree differential encoder/decoder is shown in *Figure 1*.

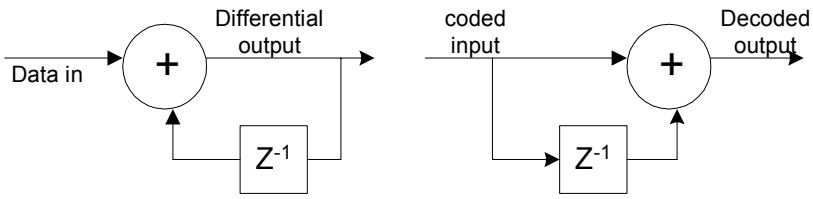

Figure 1: Differential encoder/decoder

2.2 Convolutional Encoder

The convolutional encoder is a linear time-invariant finite state machine, which accepts a semi-infinite stream of information symbols, operates upon them in a sliding block manner and transmits a continuous stream of encoded symbols [1]. Each information symbol remains in the sliding block for a finite amount of time, thus affecting the transmitted symbols during that time span. During encoding, k input bits are mapped to n output bits to give a rate k/n coded bit stream. The ratio k/n is called the code rate. With k=1, only code rates 1/n are possible. If the encoder has a memory of m bits, the code symbols are calculated from K=m+1. K is called the constraint length and the number of states is 2^{K-1}. The generator polynomial g_i defines the tap positions of the delay line to be XORed. A '1' represents a connection and '0' represents no connection. The generator polynomials g_i are of degree m and are usually not written as polynomials, but as numbers in octal notations. The input data bit is encoded by convolution into two or three output bits for rates 1/2 and 1/3 respectively according to given polynomials. Rate 1/2 convolutional encoder with constraint length K=3 is shown in Figure 2.

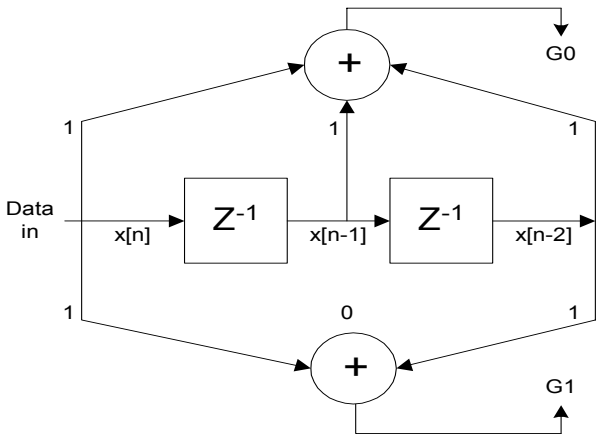

Figure 2: Convolutional Encoder

For every x(n) input bit, the encoder generates two output bits $G_0(n)$ and $G_1(n)$.

$g_0 = 7$ and $g_1 = 5$

$G_0(n) = x(n) + x(n-1) + x(n-2)$

$G_1(n) = x(n) + x(n-2)$

The octal numbers 7 and 5 represent the code generator polynomials, which when read in binary (111_2 and 101_2) correspond to the shift register connections to the upper and lower modulo-two adders, respectively.

2.3 Puncture and De-puncturing Logic Unit

A rate 1/2 convolutional encoder generates two output bits for every input bit; a big portion of channel bandwidth is used to transmit parity bits. A high code rate is achieved by puncturing mode in which input data is encoded with rate 1/2 encoder and certain bits of rate 1/2 coded data are deleted and not transmitted.

The de-puncturing unit inserts alternatively maximum and minimum values at the puncturing position prior to decoding with the rate 1/2 decoder. The null symbols add very little bias to the accumulated errors. In some coding schemes, where null values do not exist, depuncturing is achieved by inserting smallest positive and negative values alternatively [2]. The trace back depth increases as the code rate increases. A trace back memory depth of 35 to 40 is adequate for rate 1/2 decoding. However, the rate 3/4 and 7/8 require memory depth of 70 and 90 respectively [1] [3].

2.4 Branch Metric Unit

The BMU calculates distance of received bits from encoder output bits using hamming distance for hard decision and Euclidian distance for soft decision. The method employed to calculate branch metrics depends on the representation of received data. If the data is represented by a single bit, it is referred to as hard decision. When the data has a precision of multiple bits, it is referred to as soft decision. The soft decision provides an improvement of 2.2 dB in signal to noise ratio (E_b/N_0) at the same bit-error level.

The hamming distance is calculated by summing the individual bit difference between received and expected data. Euclidean distance is employed for calculating local distance for soft-decision. The Euclidean distance for rate 1/n is given by:

Top

$$\text{local\_distance(j)} = \sum_{c=0}^{n-1}[SD_c - G_c(j)]^2$$

Where SD_c are the soft-decision inputs, G_c (j) are the expected inputs for each path state, j is an indicator of the path and n is the inverse of the coding rate.

$$\text{local\_distance(j)} = \sum_{c=0}^{n-1}[SD_c^2 - 2SD_cG_c(j) + G_c^2(j)]$$

For minimization of accumulated errors similar terms of equation can be eliminated, thus reducing the equation in to sum of products of the received and expected values.

$$\text{local\_distance(j)} = -\sum_{c=0}^{n-1} SD_cG_c(j)$$

If the negative sign is removed, maximum values are searched for the metric update. The implementation of BMU in trellis-based decoding architectures can be simplified for a certain class of convolutional codes. For a rate 1/2 code, only two branch metrics have to be calculated instead of four [4].

2.5 ACS (Add Compare Select) Unit

In the ACS unit, path metrics are added to the corresponding branch metrics, resulting new path metrics are compared and the best one is selected as the surviving path metric. The comparator output is the decision of the comparison, which also controls the multiplexer to choose the surviving path metrics. The fully parallel radix-2 ACS architecture uses one ACS unit for each state to update the states in parallel, therefore in each iteration, one level of trellis is updated. Two path metrics and the branch metrics are compared to select the minimum path metric. ACS can be implemented in radix-2, radix-4 and a bit wise pipelining architecture by employing carry save adder. The radix-2 architecture is shown in *Figure 3*.

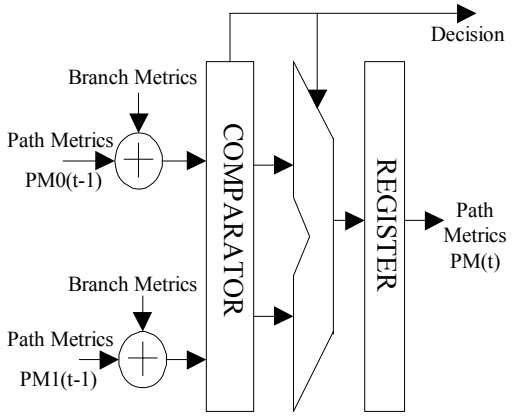

Figure 3: Radix-2 ACS unit

The throughput of the Viterbi decoder is increased by merging two levels of radix-2 trellises into one level of radix-4 trellis, so two levels of ACS operation can effectively be performed in one radix-4 ACS iteration. If the cycle time of the radix-4 operation is less than twice of the cycle time of the radix-2 operation, the throughput of the ACS unit will improve. Each element in radix-4 decoder calculates four path metrics, selects the smallest one, and stores the result of the selection.

Top

2.6 Survivor Memory management Unit

The decisions of ACS unit are stored in the path memory, which needs to be long enough to ensure that all paths are merged. The length of memory is defined as trace back depth L. A trace back method is a backward processing algorithm for deriving the survivor path from a starting state and the path decisions. The survivor memory stores the minimum path metrics and the decisions as to which branch should survive. The state sequence is computed based on the path decisions. A block of M symbols are decoded in reversed order during the data trace back phase, thus a LIFO memory is required for reversing the order before the information output. Fast hardware implementations require more memory and exhibit a large latency. Instead of tracing back L steps and decoding one symbol, K*L symbols are decoded where L is the length of the trace back and K is between 1 and 1/L. The memory is organized as a block with different operations executing in different blocks as shown in the *Figure 4*.

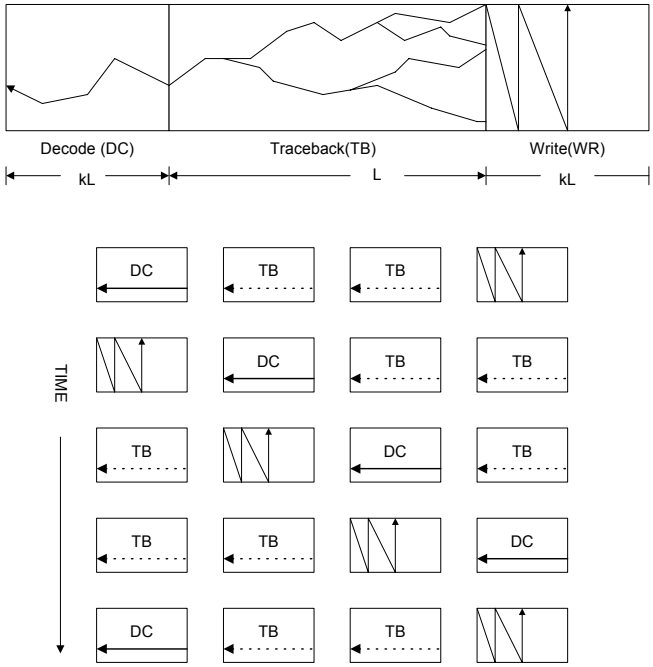

Figure 4: One pointer Trace Back

Following are the steps to retrieve the information:

Trace back read (TB): Read path decisions in current states, combines this information with the current state to find the previous state.

Decode Read (DC): Similar to TB, the pointer value in this operation is the decoded information sent to bit order reversing circuit.

Write New Data (WR): Writes path select bit of every state into memory.

The access bandwidth and computational requirements are greatly reduced as compared to register exchange architecture. The mechanism is more power efficient and consumes lesser area as compared to REA. The memory access is a limiting factor for high-speed implementations.

Top

3 LOW POWER ARCHITECTURES

One of the basic challenges being faced by the ASIC designers is the adoption of methodology incorporating the concept of block-based design for design reuse [5]. The presented design methodology breaks the complex architectures in basic building blocks (bbbs) and different architectures can be generated by manipulating these basic building blocks. This section presents our area and time efficient architectures for implementing viterbi algorithm.

3.1 Puncturing Unit

The puncturing technique holds the key to achieve the high data rates. Under this technique, codes are punctured by deleting some symbols generated by the encoder on the transmit side. Thus if the sequence G1 G2 G1 G2 has the second G2 deleted it becomes G1 G2 G1 (p), where p denotes the punctured symbol and instead of transmitting four symbols to represent two bits of data (R=2/4 or1/2) we now transmit three symbols to represent the same two bits for 2/3 rate. Similarly we can generate a ¾ and higher rates. The straightforward method to implement k/n encoder has k inputs and n outputs, the comparator required for each state is 2k ways, which is difficult to implement. In an alternate technique, the input data is encoded with rate 1/2 encoder and certain bits of the rate 1/2 coded data are deleted and not transmitted. With single input, there will be two branches entering a state and therefore binary comparators will be used. The trellis for this scheme is systematic and helps to simplify the ACS design and reduce the trace back memory.

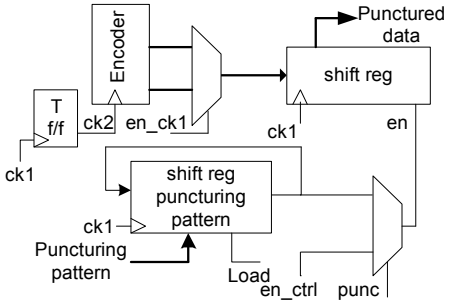

Figure 5: Puncturing Unit

The output of the convolutional encoder is passed through puncturing unit for deleting the bits according to puncturing patterns. The unit comprises of a multiplexer that converts parallel output of the encoder in to serial data, passed through a shift right register, whose shift operation is controlled by puncturing patterns. The puncturing patterns are loaded in the cyclic shift right register, it will transmit encoder bit if the puncturing bit is '1', thus puncturing unit is independent of the puncturing rate. The puncturing will be performed according to the puncturing patterns whatever may be the puncturing rate. The puncturing is controlled by 'punc' bit, set the bit to '0', if puncturing is not required, the en_ctrl will always enable the shift register and encoder data will be directly transmitted without puncturing. The clock ck1 of puncturing unit is two times faster than ck2 as depicted in *Figure 5*.

3.2 De-puncturing Unit

The hardware implementation of unit comprises of a circular shift register, similar to puncturing unit, which is loaded with the puncturing patterns. A toggle flip/flop generates maximum and minimum values, clocked by the puncturing patterns. A multiplexer, also controlled by the puncturing patterns selects encoded symbol or insert minimum and maximum values at the punctured positions. Two registers x and y hold de-punctured data

with rate '1/2', restored. Two multiplexers are utilized for synchronization, if out of synchronization set bit 'sync' to '1', which shuffles the symbol x and y. The design of de-puncturing unit is shown in *Figure 6*.

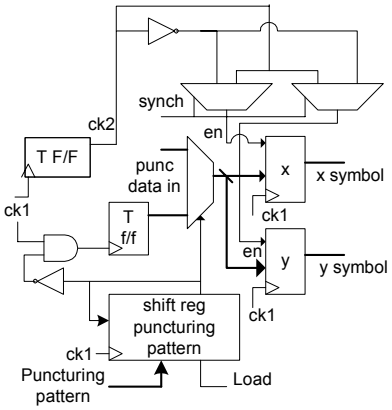

Figure 6: Depuncturing Unit

3.3 Branch Metric Unit

Conventionally multiplier or squarer is employed in Euclidian distance calculation that consumes maximum area in the architecture. In the presented viterbi assist, the approach is to exploit normalized constellations (1,1), (-1,1), (-1,-1), (1,-1) and (2,2), (1,2), (-1,2), (2,-1), (-2,-2),(-1,-2), (1,-2), (2,-2) for four point and eight point constellation respectively. With this approach, branch metrics for all the architectures including V.34 are calculated simply by addition and subtraction. Thus saving a multiplier, which occupies a considerable amount of area in the chip. Branch metrics are calculated by adder/subtractor and a shift register instead of a multiplier. The hardware implementation of the soft decision branch metrics depends on the type of number system used. Design of three-bit soft decision for 2's complement output is depicted in *Figure 7*. The input data to branch metric unit is between 3 and – 4, while output will remain in the range of 6 to – 8.

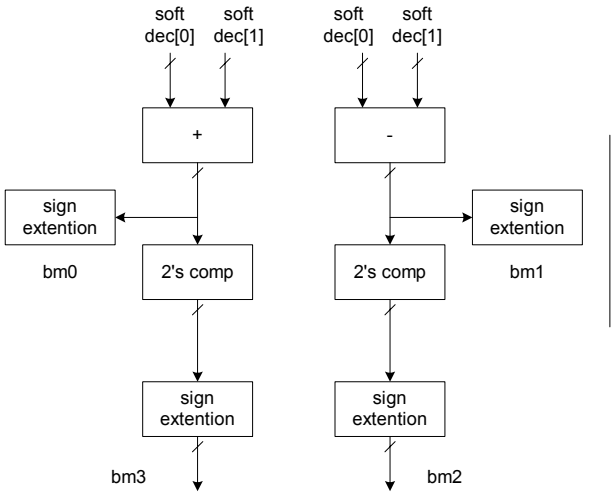

Figure 7: Branch metric unit 2's complement

Top

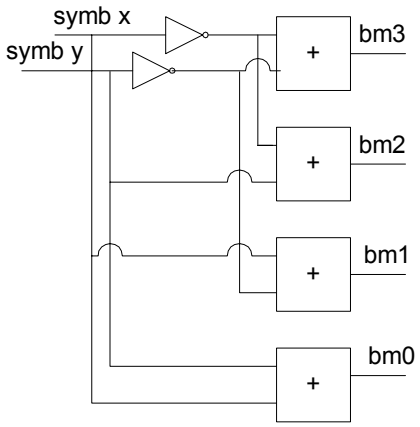

Figure 8: BMU Linear distance

The branch metrics can also be calculated by a linear distance measurement between the received soft decision and the expected symbols [6]. The linear distance has smaller dynamic range and its performance is equivalent to the squared Euclidian distance [1]. The input soft decision is 3 bit binary number lies between 0 and 7, and the output of the branch metrics is a 4 bit binary number between 0 and 14, which is a linear distance. The hardware of the branch metric using linear distance is shown in *Figure 8*.

3.4 Folded ACS Unit

In the viterbi assist one adder/subtractor based folded ACS unit is used for Add, Compare and Select operation for all types of architectures. A folded ACS unit is implemented in the viterbi assist, which calculates minimum path iteratively by using one adder/ subtractor as shown in Figure 9. Thus a single instruction of ACS from Viterbi Assist will add two numbers, store the result in the minimum register (min-reg), add two other numbers and store the result in the feed back register (feed-back), calculate the minimum of two numbers, store the minimum number in the min-reg and store the index of minimum number in the index register (index-reg).

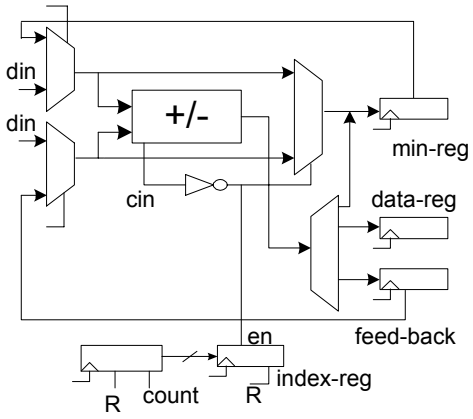

Figure 9: Folded ACS Unit

Top

3.4.1 Survivor Memory Management by Feed Forward Architecture

In this architecture, the starting decode state is determined by employing the register exchange architecture. Normally L levels (Trace back depth) of register exchange units are required, with the state indexes as the register contents and the ACS decisions are used to select the states. After L cycles, the state indexes at the output of the register exchange structure will be merged to a common state index, the starting decode state. The decoding process starts from this state to produce the decoded bits.

The hardware implementation of feed forward architecture involves the calculation of starting decode state, trace back to decode symbol, interface of the memory and LIFO circuitry for data reversal. The starting decode state is calculated by utilizing one stage of register exchange, since only the starting state is needed after every L cycles. The output of the register exchange is feed back to the input through multiplexers for L cycles and initialized to the state indexes after every L cycles. A dual ported RAM, available in FPGA, can be used for storing ACS decisions. Read and write addresses of RAM are generated with one counter by employing a delay of one clock cycle in the write address, so that the read pointer is always ahead of write pointer. The address generation circuitry generates addresses from 0 to 2L, L to 0, 2L to L and the pattern is repeated in a cyclic way. The counter employed is re-loadable up/down counter, loaded with three different addresses for one cycle. The register can be programmed with different values of L, depending on the number of symbols to be decoded. This scheme reduces RAM from 3L to 2L, the complete design of the feed forward architecture is shown in Figure 10.

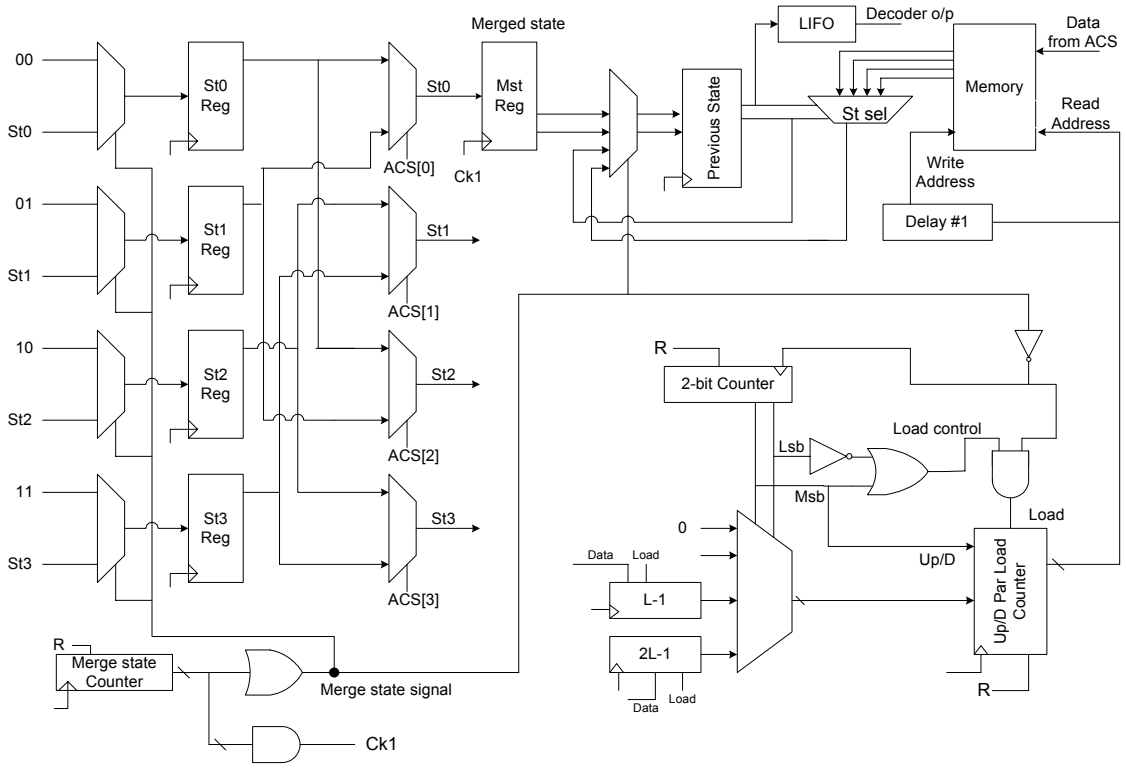

Figure 10: Feed Forward RTL Diagram

Top

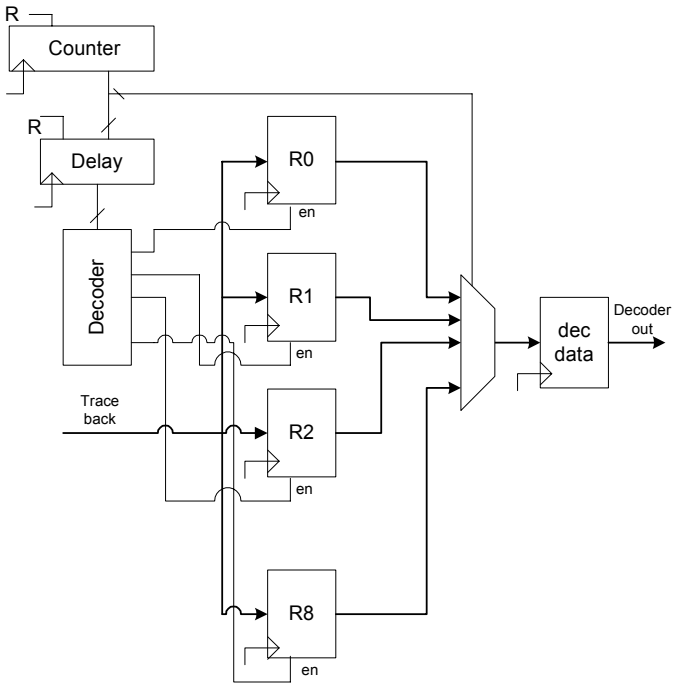

Figure 11: LIFO Architecture

The LIFO can be employed by using RAM or registers. The design of LIFO, using registers is depicted in *Figure 11*. To prevent read and write pointers to access the same location, LIFO is designed with one additional location. The read pointer moves one address ahead of write pointer and is used in a circular way at the output of the trace back unit.

4 Viterbi Assist Architecture

The viterbi assist comprises of a micro coded algorithmic state machine, address generation unit, an adder/subtractor based folded ACS unit, shifter and logical unit, puncturing/depuncturing unit and general purpose registers. The state machine contains sequencer unit configuration registers and micro-code RAM. The transfer of data to and from the viterbi assist is carried out through 64-bit bus connected to the memory; the system level design is shown in *Figure 12*. Viterbi assist comprises of a general purpose register file, which is interfaced with the external as well as internal data bus of viterbi assist. A variable width of data including 16 , 32, 48 and 64 bits can be exchanged between viterbi assist's data registers and memory. A few data registers are internal to the viterbi assist to store temporary data. Immediate data can be moved in any data register by state machine. A logic unit (LU) performs AND, OR, XOR and NOT operations.

4.1 Address Generation Unit

Address generation unit comprises of ALU (Arithmetic logic unit), address registers, address generation logic and bus stealing logic. ALU has the functionality of increment, decrement, offset addressing, indirect addressing and modulo addressing. Address registers serve as address pointers of viterbi assist memory for Viterbi decoder. As depicted in the *Figure 13*, address generation unit calculates the address by adding offset with the fixed bits, which can be programmed by host processor or micro-coded state machine. A data path is also provided to the address generation unit for indirect addressing.

Top

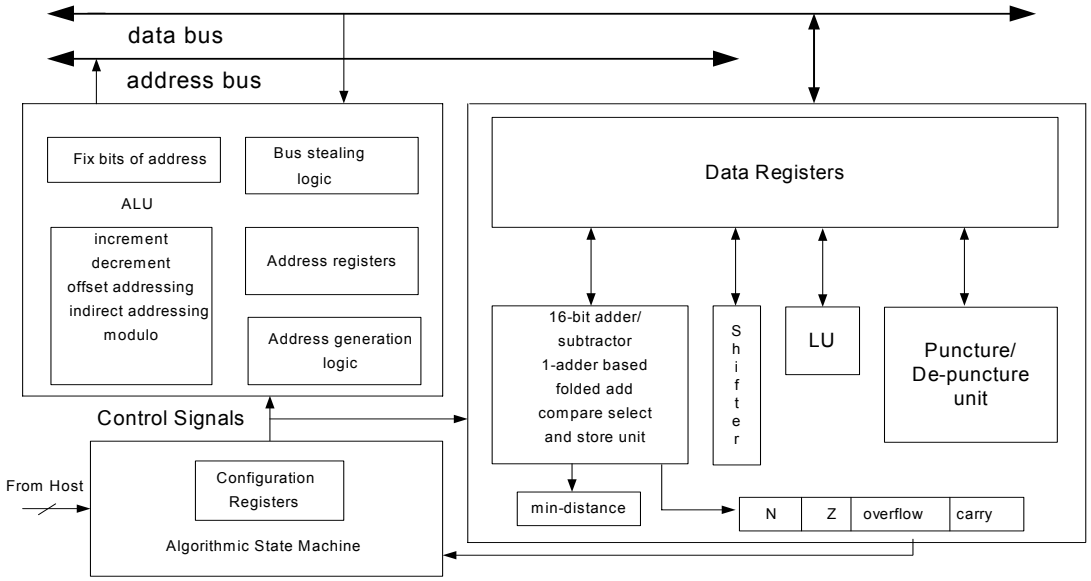

Figure 12: Viterbi Assist Architecture

In the address generation unit initial state counter provides initial state to the address generation logic, and input-bit counter generate state index for v.34 protocol. The previous state is calculated by employing these counters and operating the encoder in reverse order, thus calculating the previous state dynamically instead of storing in a table, reducing the memory size and generalizing the decoder for any encoder polynomial.

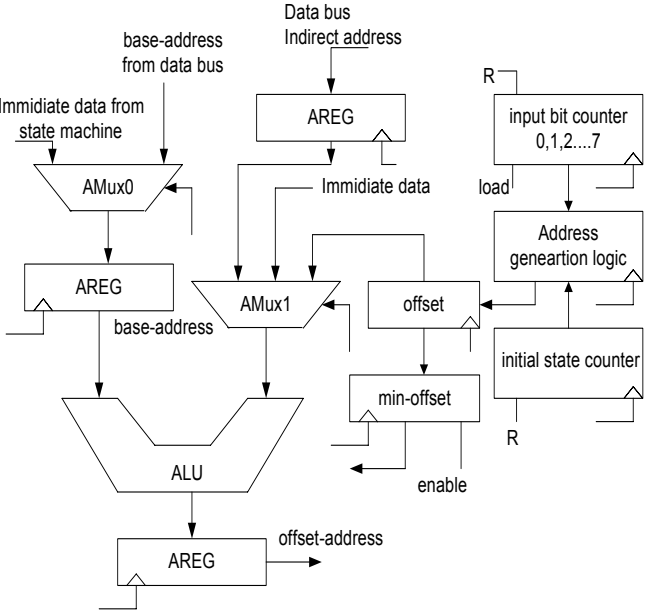

Figure 13: Address Generation Unit

Top

4.2 Algorithmic State Machine

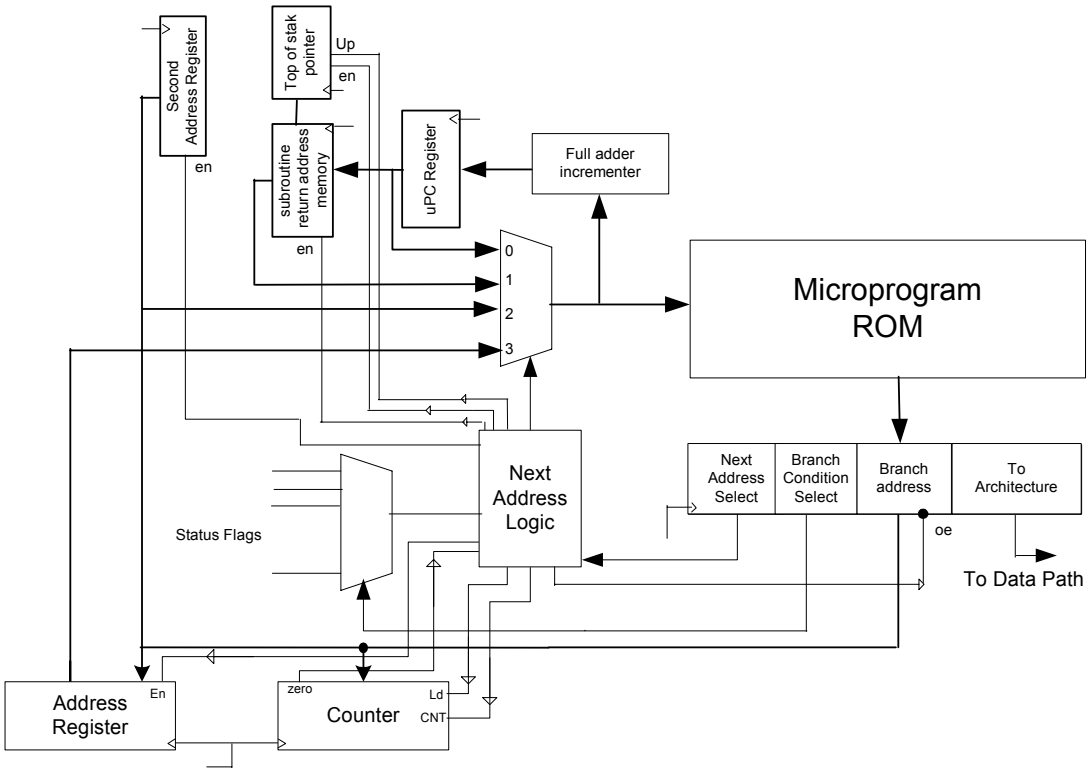

Figure 14: Algorithmic State *Machine*

Algorithmic state machine comprises of a sequencer unit, micro-code RAM and configuration registers. The sequencer unit comprises of a program counter, stack along with stack pointer, counters for nested loops, conditional multiplexer for selecting branch on a particular flag, an address register containing immediate address of the micro-code RAM, which can be used as a starting address or loaded on some event. The state machine depicted in Figure 14 has the functionality of implementing nested for loops, conditional and unconditional jump, and nested subroutines. All the control signals to the Viterbi Assist are generated by the state machine, which can be programmed for variety of Viterbi architectures and other applications as well.

4.3 Interface with Host

Host interface with the Viterbi Assist is provided through configuration registers, which are memory mapped to host for programming the encoder and decoder parameters. Different address pointers for Viterbi decoder are also configured through memory. The parameters of viterbi decoder like code rate, constraint length, encoder polynomial, puncturing patterns are programmed through following configuration registers.

4.3.1 Code Rate k/n :

The input stream of information bits is mapped onto k-bit information symbols. These information symbols are input to convolutional encoder, which codes information symbol generating n>k. The ratio k/n is called code rate. k is the number of input bits and n is the number of output bits. The larger the code rate, the smaller the redundancy introduced by the encoder. The upper eight bits of the configuration register specify the value of k, while lower bits correspond to n.

Top

4.3.2 Constraint Length (K):

Constraint length represents the number of delay units. If the encoder has a memory of m bits, then K=m+1.

4.3.3 Encoder polynomials

Each bit corresponds to the connection with the corresponding delay state to the EXOR gate. In the configuration register a value of '1' indicates connection of memory element with EXOR gate, while '0' indicates no connection.

4.3.4 Puncturing patterns:

In the puncturing pattern a value of '1' corresponds to transmit and value of '0' represents deletion of the corresponding bit.

4.4 Address Pointers

Different address pointers for Viterbi decoder are configured through memory, which contain the starting address of the different memory blocks for data exchange. Accumulated error buffer pointer, minimum path buffer pointer, 2D and 4D subset error buffer pointers, symbol and decision queue buffer pointers are the address pointers for viterbi decoder, which can be programmed by the host processor or state machine. Sufficient address registers are provided to load the address pointers corresponding to the operation being carried out, to minimize the memory access.

4.5 Development of Micro-code

The viterbi assist is specially designed for v.34 algorithm, it can be programmed for any type of viterbi decoder by downloading the micro-code in the control RAM through host interface. The presented viterbi assist has VLIW architecture, for optimum mapping of the viterbi decoder algorithm, the algorithm needs to be modified. The viterbi algorithm is a load store intensive; the excessive use of load store can be avoided by storing the information of the path in N contiguous registers. Where N depends on the number of information bits per transition, constraint length K and register size. The path history in these N registers is stored in a zig-zag manner [7]. By storing the history of entire path in contiguous registers and then by zig-zag movements of the registers, removes extensive load and store requirements. Travelling back in the trellis is very simple as the entire path is stored in N contiguous registers. The oldest soft decision in the history is always b initial bits in the first surviving path registers. Where b denotes the number of information bits per transition. The load balancing at the algorithmic level is required such that all functional units are optimally used.

4.5.1 Example : Viterbi for 4D Trellis Coded Modulation

4D Trellis Coded Modulation is used in V.34 modem standard. The modem supports two convolutional encoder standards, a 16 state, rate 2/3, and 32 state, rate ¾. The implementation of the algorithm for convolutional encoder is described here

S0: A path in the trellis contains the 4-D subset information of all the transition in the path. Each path in the trellis is stored in two consecutive 32-bit registers in the trellis-src-buffer.

S1: At an instance ti the trellis_src_buffer represents the path information prior to extending the trellis. The trellis_sink_buffer represents the new path information after extending the trellis.

S2: New subset information is appended to the second register from right whereas the oldest subset information, which should already be used for computing the final decision, is thrown out by the same shift left operation.

Top

S3: For each state *d*, algorithm finds the path with the minimum pathmetric. Let this path originates from state *s*.

S4: After finding the path with the minimum distance the algorithm loads the registers corresponding to the surviving path from the trellis_src_buffer.

S5: The second register is appended, from the right, with the subset information and stored at its new location in trellis_snk_buffer.

S6: The first path register associated with the same state is then copied unaltered from trellis_src_buffer to second location in trellis_snk_buffer.

This procedure of extending trellis with zigzag movement of shift-registers from source state *s* to destination state d is shown in Table 1. Two buffers for swapping the accumulated pathmetric and two for shuffling the path information are used. One of the buffers is used as source and the second one is sink buffer.

pathmetricSnkBf

4D Subset 3	4D Subset 5	……	4D Subset new
4D Subset 0	4D Subset 2	……	4D Subset 14

New in

pathmetricSrcBf

Old out

4D Subset 0	4D Subset 2	……	4D Subset 14
4D Subset 1	4D Subset 3	……	4D Subset 15

Table 1: Extending trellis by zigzag shuffling of path registers

5 Conclusions

The architecture of the viterbi assist presented utilizes minimum area by utilizing folded ACS unit implementation, exploiting normalized constellations, taking advantage of the special zig-zag trace back algorithm and minimization of the micro-coded RAM by providing powerful instruction sets. The high speed decoding can be accommodated in the architecture by employing an extra adder and subtractor. This viterbi assist can be interfaced either with a DSP processor for off-loading it or can be used as stand alone viterbi decoder. The presented hardware implementation of basic building blocks of viterbi decoder enables quick implementation of number of architectures that can be mapped easily on FPGA and ASIC.

6 References

[1] Mong-Kai Ku, “Design and Implementation of 100 Mb/s Variable Rate Decoder for Satellite TV Applications” UCLA, 1997

[2] Qualcomm Application Notes AN1401-2a, “Using Punctured Code Techniques with the Q1401 Viterbi Decoder.”

Top

[3] Jason Ming-jen Chang, “Design and Implementation of a forward Error Correction Encoder for Digital Cable Television Applications.” UCLA, 1997.
[4] Matthias Kamuf, John B. Anderson, and Viktor Öwall, “ A Simplified Computational Kernel for Trellis-Based Decoding” IEEE COMMUNICATIONS LETTERS, VOL. 8, NO. 3, MARCH 2004
[5] Zaheer Ahmed, Shoab Khan, and I. Elahi “DEVELOPMENT OF COMPLEX ARCHITECTURES USING ERTL LANGUAGE ” IEEE INMIC 2000
[6] Erik Paake, Jakob Dahl Andersen, “High Speed Viterbi Decoder Architecture”, Dept. of Telecommunication, Technical University of Denmark.
http://www.netaddress.com/tpl/Message/216IIQLVB/Read.
[7] S.A. Khan, M.Saqib, and S.Ahmed, “ Parallel Viterbi algorithm for a VLIW DSP”, ICASSP2000

Top