IMPLEMENTATION OF ADAPTIVE VITERBI DECODER THROUGH FPGA

Ms. D.R.Laddha, Prof. A.O.Vyas

G.H. Raisoni College of Engineering & Management Amravati, India , ***deepaliladdha29@gmail.com***
G.H. Raisoni College of Engineering & Management Amravati, India.

Abstract

The demand for high speed, low power and low cost for Viterbi decoding especially in wireless communication are always required. Thus the paper presents the design of an adaptive Viterbi decoder that uses survivor path with parameters for wireless communication in an attempt to reduce the power and cost and at the same time increase the speed.

A VHDL description has been adopted to embed the lowpower design. The adopted design were coded in VHDL and implemented on a SPARTAN 3. The results show that speed has been increased since the processing execution time has been reduced that is used to find the correct paths. Furthermore, the survivor path decoder is capable of supporting frequency up to 790 MHz for constraint lengths 7, and 9 , rate 1/3 and long survivor path is 4. Finally, the cost has been reduced since the different constraint length didn't affect of the complexity of the decoder and the processing time of computing the correct path.

Keywords : *Convolutional Encoder, Adaptive Viterbi Decoder,Survivor Path, FPGA Implementation*

---*****---

INTRODUCTION

The Viterbi algorithm is a maximum-likelihood algorithm that can be applied to decoding of convolutional codes. A Viterbi decoder typically consists of three building blocks, as shown in Fig.1

1. Branch Metric Unit (BMU) that calculates the likelihood for the possible transitions in a trellis;
2. Add-Compare-Select Units (ACSUs) that discard suboptimal trellis branches based on current branch metrics and previously accumulated state metrics;
3. Survivor Path Unit (SPU) that works upon the decisions from the ACSUs to produce the decoded bits along the reconstructed state sequence through the trellis.

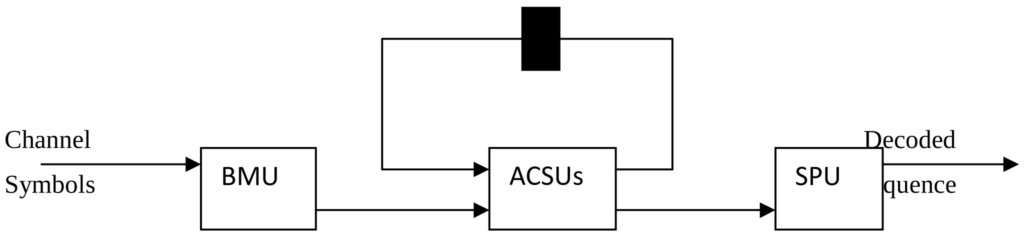

Fig. 1. Principal Block Diagram of a Viterbi Decoder.

The Branch Metric Unit (BMU):

The responsibility of this unit is to compute the Hamming code between the expected code and the receiving code as a frame. Each frame contains four symbols. At each processing, the BMU finds the Hamming code for these four symbols. This will be compared with the expected code represented by the address value that replaced by a counter started with '0000' to '1111'. The followings explain this operation:

Step 1 :In the first frame and starting with ST=0; the BMU computes Hamming code between, Rx0 and EC0 (ST). The variable between these two arches represent the value of the address ROM.

Step 2 :STnew = ST0 (STold).

Step 3 :Next the Hamming code between Rx1 and the value stored in the EC0 (ST) is computed.

Step 4 :STnew = ST0 (STold).

Step 5 :The Hamming code between Rx2 and the value stored in the EC0 (ST) is computed.

Step 6 :STnew = ST0 (STold).

Step 7 :The Hamming code between Rx3 and the value stored in the EC0 (ST) is computed.

Step 8 :STnew = ST0 (STold).

At each step, the Hamming code is computed and the total value is stored in the Hamming code buffer, ST in the ST buffer, and counter value in the address buffer. All the above steps chose ST0 and EC0 because the first value of the counter is '0000'. The selectivity of ST1, and EC1 will depend on counters value at each location bits. The BMU repeated all the steps for counter "0010" to "1110" for even values only while the odd values are rejected because the computation of Hamming code at each bit location is the same with the past even value except the last Hamming code will be (3-computed value), and the STnew=ST0 (STold)+32.

All the above steps will be repeated for each sub-frame (four symbols). Therefore more execution time will be required to find the suitable path and to select the output code. The parallel operation of these step become more efficiently to reduce the execution processing time. The parallel execution can be divided in two types. The first type involves a parallelism to compute the Hamming code in the 8th probability paths, and this type need four BMU. Each BMU is responsible in the computation of the Hamming code of suitable received symbol with expected code.

Another type for parallel computing Hamming code includes a parallelism of the all probability paths. The four symbols, give a (16) probability path starting from path '0000' to '1111', need an even paths for computing the Hamming codes of all nodes. This model for parallel computing need a memory bank RAM to store the Hamming code of each node. It is clear that this design will be more complicated and need more elements (memory banks, comparator units… act) to represent this model. Therefore, it has not been considered in the proposed design.

The Add Compare Select Unit (ACSU):

This ACSU is the main unit of the survivor path decoder. The function of this unit is to find the addition of the four Hamming code received from BMU's and to compare the total Hamming code stored in the HC_buffer for the even and odd paths. This unit also compares the values and store, the minimum Hamming code, and the counter's value. At the end of the 8th even state, the minimum Hamming code and the counter value are stored in HC_buffer and AD_buffer. The AD_buffer represents the opposite output code. The other value stored in ST_buffer represents the address of the next state number.

For each four symbols selected, the initial value of HC_buffer is set to '1100' and compared to find the minimum value. The 3 bits counter is used in the ACSU and SMU. The ACSU can be used as an address location of the comparator between sixteen different paths and select the suitable counter value that synchronized with minimum value of Hamming code. The same counter will be used in the SMU to lead the address ROM to find the correct state number and expected code for the next state. The block diagram of the ACSU and the 3 bits counter is as shown in Fig. 2.

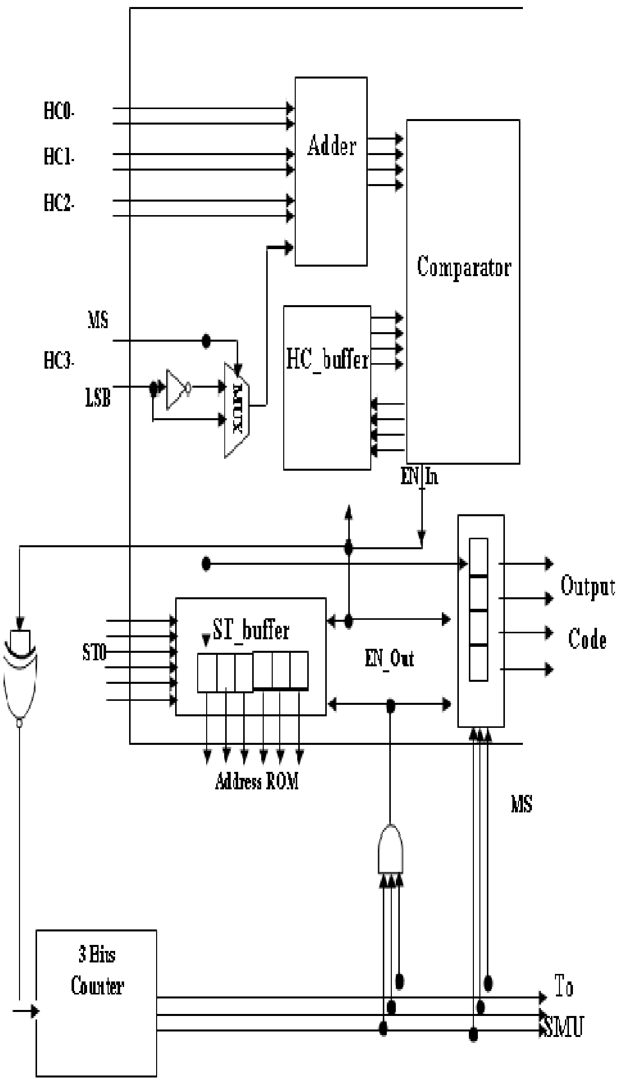

Figure 2 . Block diagram of ACSU

Many processing steps are executed through ACSU, the following processing steps explain the operation of this unit.

Step 1: Initialization process.
At the beginning, the 3 bits counter is started with '000'. The four Hamming codes of four symbols encoded data with noise, the state number of the next state, and the enable signal of the result of AND gate of the 3 bits counter are entered to ACS. The MSB of the Hamming code of the last symbols is taken to decide if the Hamming code of the last symbols is the even path or the odd path and which one is the minimum value.

If MSB is '1' the Hamming code is '2' or '3' and this give an indication that the Hamming code take the odd path of the last symbol with Hamming code is '1' or '0' and this represent the minimum path of the two paths (even and odd). The NOT gate is used to find the Hamming code of the odd path, while the MSB is used to control the output of the multiplexer, updated the state number, and select the correct output code.

1. If MSB is '0', the output of the multiplexer is set to the value of the LSB whereas if the MSB is '1', the output of multiplexer is NOT the value of LSB.
2. If MSB is '0', the state number ST0 represents the ADDRESS value of the next state otherwise it means that the ADDRESS of the next state is equal to ST0+32 and the 6th bank 0 ST0 to '1' are required.
3. If MSB is '0', the last bank of the output code is '0' and the even counter value is the correct path Otherwise, while the value '1' of MSB is set '1' to the last bank of output code, and indicate that the odd counter value is the correct path.

Step 2: Adding process.
The adder block computes the total sum of the three Hamming codes with the output value of multiplexer. The three 2 bits Hamming codes with 1 bit output of the multiplexer are added and the output of the adder is formed by 4 bits and sent it to the comparator unit. At the beginning of each four symbols, the HC_buffer is set to '1100' which represents the maximum value of the Hamming codes for 4 symbols, because the maximum Hamming code of each symbols is '11' (three bits error).

Step 3: Comparing Process.
The comparator receiving two codes, the first came from the adder which represents the additive of the Hamming code, and the second come from the HC_buffer. The value stored in this buffer is initiated to '1100' and updated throughthe processing. The updated is executed while the value entered to the comparator from the adder is less than the value stored in the buffer. These results perform four processing:

1. Generate an enable signal (EN_In='1') which will be able to replace the old value store in the HC_buffer
with the new small value came from adder.
2. Store the state number in the ST_buffer which perform the ADDRESS value of the Viterbi ROM.
3. Store the value of the counter in the output code buffer.
4. Give a clock signal to the counter to generate the next counters value.

The new value of the counter will allow the ACSU to start with the next values of the Hamming codes and repeated all the above processing. The last processing of ACSU started at the last counter value '111'.

Step 4: The output process.
The output process is started at the counter value equal to '111'. At this value all of the above processing steps are finished and the 16th paths starting from path '0000' and end at path '1111' were computed. The ST_buffer contains the ADDRESS ROM of the next state, and the output code represents the opposite code of the counter that contains the minimum Hamming code of four encoded symbols. The output code represents the decoded bits of these symbols.
The enable signal of the ST_buffer and output code come from the output of AND gate become '1' at the three input bits '111' the end value of the counter. The enable signal (EN_Out='1') can to be able to move the ST_buffer and output code to the execution state of the decoder.

The State Metric Unit (SMU):

The function of this unit is to select the suitable state numbers for the next state of each code of the counter value.

The state number is selected to represent the ADDRESS ROM of the Viterbi ROM. From the Viterbi ROM, (EC0 and ST0) ROM, the selectivity of the suitable state number (ST0 or ST1), and (EC0 or EC1) will depend on the value of the counter code. For example the counter code '0101' indicate that the beginning state is ST0, which is the ADDRESS value of the next state. The next state is ST1, which is used as an ADDRESS for the next state. The next state ST0 is used as anADDRESS to the next. The last step ST1 represents the ADRESS for the next step. For each step, the process is selected the EC0 or EC1, and ST0 or ST1 at the ADDRESS location. The EC0 or EC1 at each step are moved to the BMU to find the Hamming code, while only the state number ST0 or ST1 of the last step will be selected and moved to the ACSU.

Because of the reduction in the computation by used only the even paths, therefore only the ST0 will be moved tothe ACSU while the last expected code of each the last steps EC0 only. The same counter that used in the ACSU is used in this unit to select the state number and the expected code of the all steps in all 16th probability of paths.

The algorithm and operation of the SMU unit can be splits in to two process steps, starting from receiving the ST value from ACSU which represents the ADDRESS Viterbi ROM, and ending to move the suitable expected codes of four symbols to the BMU.

Step 1: The ST process
Initially, the ST value came from the ACSU. This value is the same at each code of the counter for the same four symbols and will be changed after the ending of 16th paths. Another value of ST is reached from the ST0 ROM. The multiplexer selects which one is available at a time. At any value of counter, the ST is still the same but through the execution steps of each counter code the ST value is arrived from the ST0 ROM and changed 4 times in each counter code. The clock signal of the counter is used in multiplexer to select which value of ST to be selected. The '1' selects the ST from ACSU, and the '0' selects ST value from ST0 ROM. The output of multiplexer is shifted one bit each step by shifting the counter value starting from MSB and ending with '0' to represent the even path. The new value of the ST will be used as an ADDRESS to the Viterbi ROM (EC0, and ST0).

Step 2: Expected code process
The expected code is received from EC0 ROM with the suitable ADDRESS. In this process the expected code read from the ROM will be transferred to the 3 bits shift register. The serial bits moved from 3 bits shift register to the 12 bits shift registers, at the end process of the expected code computation with four steps. The 12 bits shift register contains the all expected code of four steps. The first 3 bits represent the expected code for the first step, the second 3 bits is the expected code for the second steps, and so on. The 12 bits (4 code of expected code for the ith path of 16th paths) are moved to the BMU to be compared with the suitable codes.

FPGA IMPLEMENTATION SYSTEM

This part presents the results of the FPGA implementation of the proposed model of Viterbi decoder. The proposed architecture is implemented on a SPARTAN 3 Field Programmable Gate Array (FPGA). The FPGA implementation is performed using version 6.3i of the Xilinx implementation tools which is called (ISE 6.3i). The design architecture is described with Very High speed integrated circuit hardware Description Language (VHDL) using ModelSim (XE II v5.8c).

CONCLUSION

In this paper, a adaptive Viterbi algorithm based on the strongly connected trellis decoding of binary convolutional codes has been presented. The use of error-correcting codes has proven to be an effective way to overcome data corruption in digital communication channels. The adaptive Viterbi decoders are modeled using VHDL, and post synthesized by Xilinx Design Manager FPGA logic. The design simulations have been done based on both the VHDL codes at modelsim and the VHDL codes generated by Xilinx design manager after post synthesis.

REFERENCES

1.Yun-Ching Tang, Do-Chen Hu, Weiyi Wei, Wen-Chung Lin, Hongchin Lin, "A Memory-Efficient Architecture for Low Latency Viterbi Decoders." International Symposium on VLSI design,(IEEE July 2009)
2.Matthias Kamuf, Viktor Owall, John B. Anderson, "Survior Path Processing in Viterbi Decoders Using Register Exchange and Traceforward."(IEEE Jun 2007)
3.Hema S, Sureshbabu V,Ramesh P, "FPGA Implementation of Vitrbi Decoder", Proceedings of the 6th WSEAS Int. Conf. on Electronics,Hardware,Wireless and Optical Communications, Corfu Island, Greece, February 16-19, 2007.
4.S.W. Shaker, S.H. Alramely and K. A. Shehata, "Design and Implementation of Low –Power Viterbi Decoder for Software –Defined WiMAX Receiver", 17th Telecommunication Forum TELFOR, Serbia, Belgrade, 2009.
5.J.S, Reeve., and K. Amarasinghe, "A Parallel Viterbi Decoder for block Cyclic and Convolutional Codes", Journal of Signal Processing Jan 2005.
6.S. Swaminathan, R. Tressier, D. Goeckel., and W. Burleson, "A dynamically reconfigurable adaptive Viterbi decoder", IEEE Transaction on Very Large Scale Integration (VLSI) Systems,Vpl. 13,April 2005.