Développement d'un décodeur en treillis modulaire sur FPGA

Louis Edmond Pouliot, étudiant 2e cycle

Dr Paul Fortier, directeur de recherche

Dr Marc Tremblay, co-directeur de recherche

Abstract: In this work, we look at a novel approach for the realization of a fully parallel decoder based on the Viterbi algorithm and hypercube architecture using a rapid prototyping method on FPGAs. Our proposed modular/hypercube architecture allows optimization of the surface by connecting modules together in such a way that a minimum of interconnections between modules is needed. Further optimization is possible using temporal multiplexing.

Résumé: Ce travaille présente une nouvelle approche pour la réalisation d'un décodeur parallèle, basé sur l'algorithme de Viterbi et l'architecture hypercube, utilisant une méthode de prototypage rapide sur FPGAs. L'architecture modulaire/ hypercube proposée permet l'optimisation de la surface car les modules peuvent être reliés de façon à ce qu'il y ait un minimum de connexions nécessaires. On peut optimiser encore plus en utilisant un multiplexage temporel.

Introduction

The Viterbi algorithm (VA) [1] is known to be a maximum *a posteriori* (MAP) solution to the problem of estimation of the state sequence that drives a convolutional encoder. A convolutional encoder can be modeled as a finite-state discrete-time Markov process where there is a one-to-one correspondence between the state sequence, $\boldsymbol{x}$, and the input sequence. The VA can be used to decode the estimated input sequence from code words, $\boldsymbol{z}$, transmitted over a communication channel. It looks for the state sequence for which $P(\boldsymbol{x} \mid \boldsymbol{z})$ is maximum.

On the trellis of a convolutional code, the VA finds the shortest path that leads to a particular state. Metrics are associated with each branch and they can be calculated as the Hamming distance of the corresponding code word over the received word for hard decoding or as the Euclidean distance in the case of soft decoding. Many paths can lead to the same state. The VA selects the path whose summation of all metrics, called the cumulative metric, is the lowest. This refers to the add-compare-select (ACS) operation.

One could realize a fully parallel decoder with a processor associated to each state. This direct implementation with register exchanges (REGEX) for cumulative metrics and selected paths between processors implies large data paths. The surface used for these paths is a major

limitation for large 2^n-state REGEX decoder implementation. Furthermore, the REGEX approach is not suitable for a modular implementation because it requires too many interconnections between modules.

Viterbi algorithm on hypercube architecture

The hypercube version of the VA is an effective way to implement a fully parallel decoder [2]. There are 2^n processors located at each corner of an *n*-cube. For example, Figure 1 shows a 3-cube. For processor identification, coordinates of each dimension from the *n*-dimension cube are used. Communication between processors can occur along the edges of the cube.

To eliminate the need for register exchange, two things are done. First, the trellis diagram is modified to fit the connectivity of the FFT algorithm. In Figure 2, the interconnections Λ between processors are shown for the *n* steps (s, $s+1$, $s+2$). The interconnections go along each dimension, one at each step like

$$\Lambda[(ij...k)] = \begin{cases} (ij...\overline{k}) & \text{, for step } s \\ ... & \\ (i\,\overline{j}\,...k) & \text{, for step } s+(n-2) \\ (\overline{i}\,j...k) & \text{, for step } s+(n-1) \end{cases} . \qquad (1)$$

For example, at step $s+1$, the interconnection to the processor (101) is from processor (111). Once step $s+(n-1)$ is done, the process continues with step *s*. This way, decoding on hypercubes goes through *n*-step cycles. Then it should be obvious that each processor goes through states cycles. The initial state is the processor coordinates (*ij...k*) which becomes (*kij...*) in the next step. After *n* step, the state is the processor coordinates once again. Let us define the corresponding state of the processor as

$$\chi[(ij...k)] = \sigma^u[(ij...k)] \quad , \qquad (2)$$

where *u* is the number of steps done and σ is the cyclic shift to the right function. This configuration allows each processor to keep its cumulative metric value for the next step. Thus cumulative metric exchanges are not needed.

The second action taken is to avoid paths exchange. Using the fact that at any time the previous state of a processor is known from eq. 2, each processor generates a trace-back bit [3] which can be used later to go through the selected path. This bit acts like a pointer to the previous processor which is either the same one (local connections) or another one (neighbor connections) (see Figure 2).

Proposed modular/hypercube approach

Fast-prototyping allows development of ASIC circuits in a short time. For a Viterbi decoder, we should be able to determine how many states we want. The choice of a particular code (and the number of states) is guided by the performance we are looking for. Our novel approach consists of a base module of four processors which can implement a 4-state decoder by itself. To create an n-cube decoder with 2^n states, with $n > 2$, it takes 2^{n-2} square decoders (2-cube).

From Figure 2, it seems natural to regroup the upper four processors together in a module. Another module is composed of the four lowest ones. The first module is labelled '0' and the second, '1'. Labelling of processors follows this convention:

$$i...\mathrm{P}jk = [\text{module label}]\mathrm{P}[\text{square coordinates}] \ . \qquad (3)$$

Step s asks for interconnections in dimension $\overrightarrow{k}$ while $s+1$ asks for interconnections in dimension $\overrightarrow{j}$. By analogy to the cartesian system (x,y), we called the interconnections at step s "neighborY" and the ones at step $s+1$ "neighborX". Others steps to $s+(n-1)$, with $n>2$, require outer-module interconnections called "EXT".

The proposed modular/hypercube (MOD/HYP) architecture (see Figure 3) consists of entities which can be classified in two categories: entities that can figure in a module and those that cannot.

In a module, we find four processors and one IO/MUX entities. A processor needs metrics for the local and the neighbor connections. The ACS operation occurs and the corresponding trace-back bit is produced. The IO/MUX takes outer-module information (metrics, type of interconnections) and gives them to the processors. In return, selected trace-back bits and the lowest cumulative metric are available for outer-module communications. IO/MUX also controls its four processors. These operations are independent of others modules operations.

The sequencer/multiplexer presents metrics and control signals to every modules. Metrics should not be included in module for code independence. It also includes the memory paths for trace-back bits and the trace-back procedure to find the decoded bit in a look-up table. Finally it plays the role of multiplexer for outer-module interconnections (EXT.). These global operations cannot be done locally in a module.

Simulations and results

We used the VHDL hardware description language to code the decoders. Performance evaluation is obtained directly from functional simulations using the same VHDL description as the synthesis does.

All results are for a binary symmetric channel (BSC) without memory using antipodal signaling (BPSK). Performance for the REGEX and MOD/HYP 4-state decoders are shown in Figure 4. The optimal code is rate $1/3$ with generators $g_1 = 7$, $g_2 = 7$ and $g_3 = 5$

(generators in octal notation). The path depth is defined as five times the code constraint length, which is 15, and d_{free} is 8. The slightly better performance of the REGEX version can be explained by the use of fixed processors for each state. In fact, in cases where two or more processors are equal, there is an implicit hierarchy between processors which cannot be easily duplicated in the MOD/HYP version because of time varying states on each processor. This slight difference fades out as E_b/N_0 increase. To trace the upper bound estimation, we use [4]

$$P_b \leq \frac{\partial}{\partial I} T(D, I)\Big|_{D = \sqrt{4p(1-p)},\, I = 1}, \qquad (4)$$

where $T(D, I)$ is the generalized transfer function of the state diagram of the encoder with $\beta = 1$ and p is the crossover probability of the BSC.

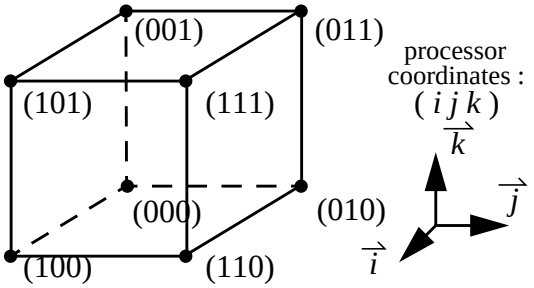

Figure 1 3-cube for 8-state trellis decoder.

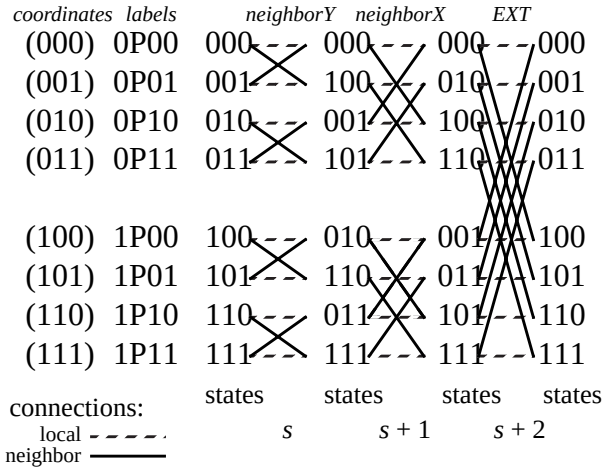

Figure 2 Butterflies interconnections between processors. Steps s and $s + 1$ are inner-module interconnections. Step $s + 2$ calls for outer-module interconnections.

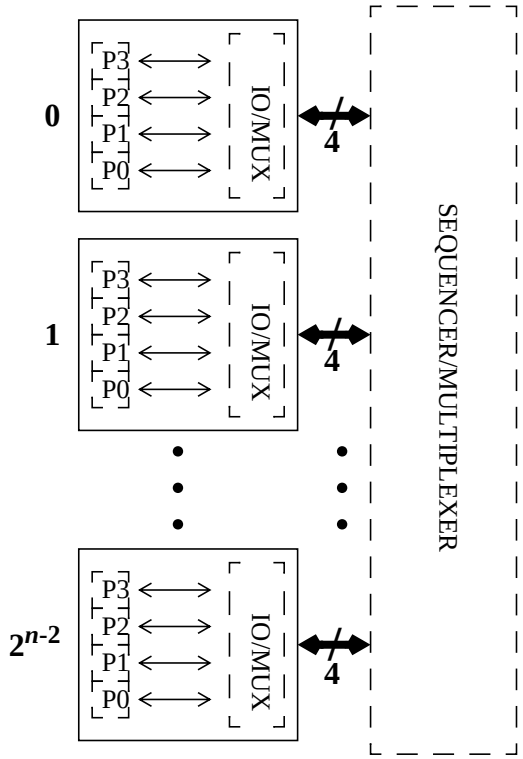

Figure 3 MOD/HYP architecture model with square decoders for an n-cube decoder.

Performance for a 8-state optimal rate $1/3$ code ($g_1 = 13$, $g_2 = 15$ and $g_3 = 17$) with a path depth of five times the code constraint length (15) and $d_{free} = 10$ is shown in Figure 4. This figure also shows the improvement over the 4-state MOD/HYP decoder.

Ways of optimization for *N*-cube

Rapid-prototyping process allows modifications such as path depth, metrics or number of modules (to compare codes) and soft decoding (with surface penalty). One can rapidly simulate a decoder and then make a synthesis in a given technology. For our research, we chose FPGAs and explored their ability to implement DSP algorithms.

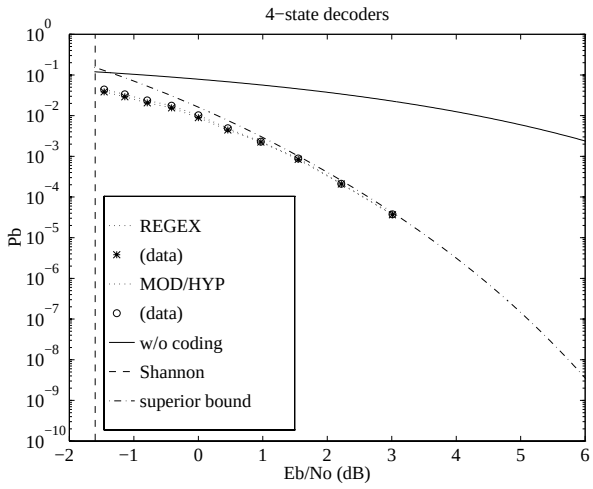

Figure 4 Performances of 4-state decoders (REGEX and MOD/HYP).

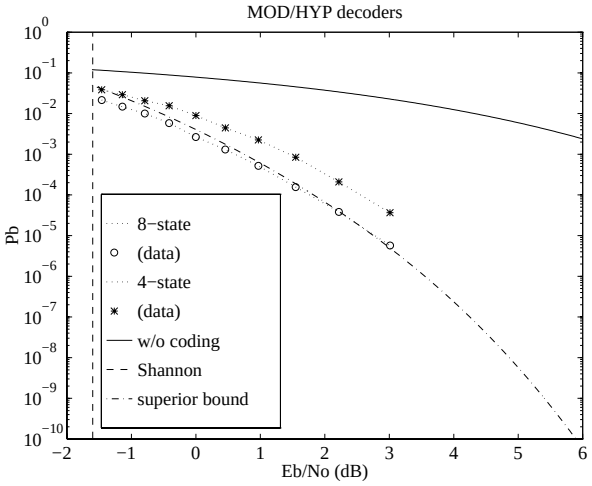

Figure 5 Performance of a MOD/HYP 8-state decoder with 4-state comparaison.

We found another important way of optimization using the spatial representation of modules in an *N*-cube. The idea is to place modules in such a way that a minimum of interconnections occur at each step. An example is illustrated by Figure 6 for a 4-cube decoder (2^4-state MOD/HYP decoder). Decoding goes through 4-step cycles in this case. Of these four steps, one doesn't need outer-module interconnection, two require 4 outer-module interconnections at the same time and only one requires 8 outer-module interconnections.

Using temporal multiplexing, which can be implemented with latches on data bus for the concerned processors, one could split the 8 outer-module interconnections step into two 4 outer-module interconnections steps. This separation is illustrated by the multiplexed interconnections in Figure 6. Without taking into account the fact that this step could be realized faster than the others, this constitutes only a 25% time penalty for a 50% reduction of data path size. This kind of surface reduction can be important for bigger decoders.

For large multichip decoder designs with a large number of states, optimization of the number of processors in each module should be investigated.

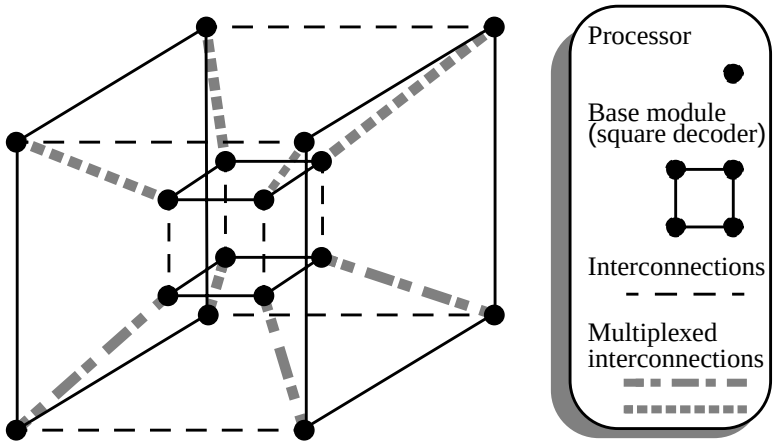

Figure 6 Illustration of temporal multiplexing on a 16-state MOD/HYP decoder.

Conclusion

In this work we presented a novel modular/hypercube architecture for the realization of a fully parallel Viterbi decoder. Performance was found to be comparable to decoders using register exchanges. The modular approach allows rapid prototyping of trellis decoders to fit particular applications. Temporal multiplexing of data with an optimal spatial representation of the modules can reduce up to 50% the space used by data paths with less than 25% of time penalty.

References

[1] G. Forney, “The Viterbi Algorithm”, Proceedings of the IEEE, Vol. 61, pp. 268-278, March 1973.

[2] F. Pollara, “Concurrent Viterbi Algorithm with Trace-back”, SPIE Advanced Algorithms and Architectures for Signal Processing, Vol. 696, pp. 204-209, 1986.

[3] P. J. Black and T. H.-Y. Meng, “Hybrid Survivor Path Architectures for Viterbi Decoders”, Proc. ICASSP 93, Vol. I, pp. 433-436, 1993.

[4] J. B. Anderson and S. Mohan, Source and Channel Coding, Boston: Kluwer Academic Publishers, 1991, ch. 4, pp. 222-225.