Implementation of Selection Circuit for Successive Cancellation List Decoding

Kamal Baghirli
baghirli@rptu.de

***Abstract*—Successive cancellation list (SCL) algorithm is an efficient way to decode polar codes while achieving excellent error-correction. The metric sorter unit lies on the critical path of the SCL decoder, and therefore has a considerable effect on its speed. In this seminar work, a non-sorting direct selection method is implemented in hardware (VHDL) and software (C++). The simulation result shows that the direct selector has much less stages than the other sorting methods and thus consumes less time as well as hardware resources, especially when the list size is large, whereas its disadvantage is that the selected metrics on the output are not in a sorted order.**

***Index Terms*—Successive cancellation list decoding, metric sorter, selector**

I. Introduction

SCL decoding [1] is a method used in communication systems for decoding error-correcting codes. In simple words, SCL decoding works by considering multiple possible candidates for the original message and gradually eliminating unlikely candidates until the most likely one is found. During SCL decoding, the algorithm constructs a binary tree where each node represents a partial decoding of the received signal. In order to reduce the complexity, at each stage, only L surviving paths out of 2L paths are kept, where L is the list size. Each candidate path is represented as a path metric, therefore choosing L surviving paths is equivalent to choosing L smallest path metrics out of 2L.

Different sorter architectures have been suggested to sort the metrics with low latency. For example, Balatsoukas-Stimming et al. [2] proposed a Pruned Bitonic Sorter (PBS) and Simplified Bubble Sorter (SBS), which apply the ideas of quick sort and bubble sort respectively to the partially ordered structure of the metrics from the previous decoding step, thus reducing the number of stages and CASUs. Besides, Kong et al. [3] developed an odd-even sorter with the same number of stages as PBS using less CASUs.

However, a more recent direct selection algorithm designed by Shi et al. [4] showed that it is not absolutely necessary that the selected metrics be in the sorted order. In fact, just selecting the L smallest metrics by comparing them bitwise in every step can drastically reduce the number of stages, especially if L is large.

The purpose of this work is to implement the direct selection method and compare simulation results with the measured results of the same method as well as of the sorting algorithms. The next section, Section II, briefly summarizes the direct selection algorithm and displays a rough circuit diagram of its hardware implementation, followed by the Section III, which presents the simulation results. Finally, the Section IV draws conclusions.

II. The implemented direct selection approach

The main idea of this method is to compare the corresponding bits of the $2L$ metrics at each stage, starting from MSB to LSB. Obviously, the unlabeled metrics with the s-*th* bit being 0 are smaller than the ones with the corresponding bit being 1. Depending on the number of $0s$ in a column, either smaller or larger metrics are selected at each stage. The algorithm ends when exactly L bits are selected.

The selector unit takes $2L$ M-bit metrics as inputs and generates a bit vector with the size $2L$, where 1 on the n-*th* element indicates that the n-*th* metric is selected among the smallest metrics. Let D_{ij} be the j-*th* bit of the i-*th* metric and F_i be the i-*th* bit of the output. Additionally, let E be another bit vector with the size $2L$, where $E_i = 1$ indicates that the i-*th* metric has already been labelled (decided to belong to either the L smaller or larger metrics) or not. Furthermore, let r_f be the number of metrics that still need to be selected and s be the stage. The 0-*th* bit is assumed to be MSB while $(M-1)$-*th* bit to be LSB.

Initially, $E = 0, F = 0, r_f = L$ and $s = 0$ (stage). The algorithm is described by FSM, having 4 states (Sd, Sa, Sb and Se).

Sd: In the state Sd, the number of unlabelled metrics with the s-*th* bit equal to 0 are computed. In other words, $n0 = sum(D_{is} nor E_i)$. As a result, $n0$ is the number of smaller metrics that can be identified in this stage. The next stage is decided by comparing $n0$ with r_f:

Sa: if $n0 < r_f$, then the FSM will jump to stage Sa. As the number of the smaller metrics is less than r_f, those identified

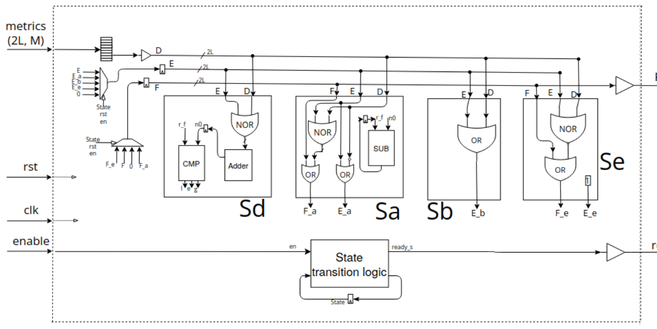

Fig. 1. Architecture of the direct selection circuit

TABLE I
CLK CYCLES FOR $M = 8$

L	SBS	PBS	OES	Direct Selector
2	1	1	1	8.37
4	3	5	5	10.41
8	7	9	9	12.90
16	15	14	14	15.28
32	31	20	20	17.79
64	63	27	27	20.31
128	127	35	35	22.27
256	255	44	44	23.79
512	511	54	54	24.78
1024	1023	65	65	25.26

smaller metrics can be surely labelled as being among the smallest L metrics. Therefore: $E_i = (D_{is} nor E_i)$, and $F_i = (D_{is} nor E_i) or F_i$. Consequently, $n0$ should be subtracted from r_f as $n0$ metrics have been selected in this stage. After Sa, FSM jumps back to Sd for the next stage.

Sb: if $n0 > r_f$, then the FSM will jump to stage Sb. Contrary to Sa, the number of the smaller metrics is more than what needed to be chosen, therefore, the larger metrics of this stage should be labelled as not being among the smallest L metrics (i.e. $F = 0, E = 1$). Thus: $E_i = D_{is} or E_i$. After Sb, the FSM jumps back to Sd.

Se: if $n0 = r_f$, then the FSM jumps to Se. It is clear that, in this case, all of the remaining smallest metrics can be selected right away, which means the output is ready and the selection procedure ends: $F_i = (D_{is} nor E_i) or F_i$, and $E_i = 1$.

The algorithm is implemented as a sequential FSM machine in VHDL. Fig. 2 shows a rough circuit diagram for the hardware implementation. The adder, comparator and subtractor each takes 1 CLK. Thus, the state Sd takes a total of 2 CLKs while the remaining states are executed in 1 CLK.

III. Evaluation and Results

A. Hardware Implementation

According to [4], the total time taken by the selection unit is:

$$c_{tot} = m(c_d + 2p + 1)$$

, where m is the number of stages the algorithm will take $(1 \leq m \leq M)$, p is the probability of Sd-Sa transition at each stage, and c_d is the CLK required for the adder. As the adder takes 1 CLK in the given implementation, the total CLK is:

$$c_{tot} = m(2p + 2)$$

. It is expected that, when L is increased, m and p will also increase. Indeed, the simulation results confirm that c_{tot} is linearly dependent on L. Table I compares the estimated CLK of different sorters from [2] and [3] to the measured CLK of the direct selector (measured by taking the average of 10000 tests) when $M = 8$. For the sorters, it is assumed that one stage takes one CLK. As seen, the direct selector is slower than the other sorters for a small L, whereas it is much faster when L is greater than 32. The key reason for such efficiency is that the selector mainly consists of simple logic gates which can work parallelly when L is large.

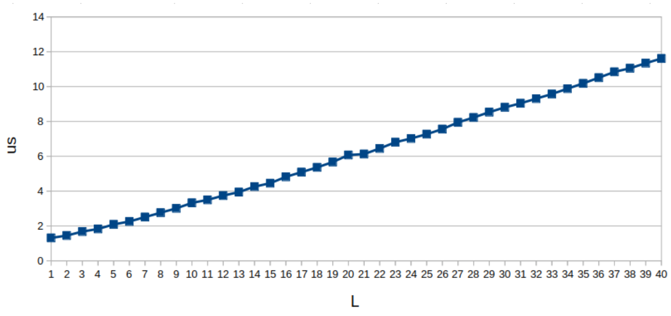

Fig. 2. Time consumption of the software implementation of the direct selection algorithm

B. Software Implementation

As for the software implementation, time complexity is $O(ML)$. Fig. 2 demonstrates a linear increase in the time consumption of the direct selector for $M = 8$, as L is increased from 1 to 40, whereas the corresponding results for SBS and PBS from [2] were quadratic.

C. Resource Consumption

From the circuit diagram in Fig. 1, the used hardware units are 1 adder, 1 comparator, 1 subtractor, $6L$ NOR gates and $8L$ OR gates. Unlike the sorter architectures, the number of comparators is only 1 regardless of the value of L. Therefore, similar to the speed, the resource consumption of the direct selector is also better than that of the sorters.

IV. Conclusion

In summary, a non-sorting direct selection architecture is implemented on both hardware and software. The results turned out to be as estimated by the proposors of the algorithm, showing a significant advantage over the other sorting methods both in terms of speed and hardware usage, especially in case of large L and small M. To be more precise, its time complexity was indeed linear, as opposed to the complexities of the sorters, which are quadratic. On the other hand, the fact that the selected metrics at each stage are not in a sorted order is a downside of this approach.

References

[1] I. Tal and A. Vardy, List decoding of polar codes, IEEE Trans. Inf. Theory, vol. 61, no. 5, pp. 22132226, May 2015.

[2] A. Balatsoukas-Stimming, M. B. Parizi, and A. Burg, On metric sorting for successive cancellation list decoding of polar codes, in Proc. IEEE Int. Symp. Circuits Syst., May 2015, pp. 19931996.

[3] B. Y. Kong, H. Yoo, and I. C. Park, Efficient sorting architecture for successive cancellation list decoding of polar codes, IEEE Trans. Circuits Syst. II, Express Briefs, vol. 63, no. 7, pp. 673677, Jul. 2016.

[4] S. Shi, B. Han, J.L. Gao, and Y.J. Wang, ”Enhanced Successive Cancellation List Decoding of Polar Codes”, IEEE Commun. Lett, vol. 21, no. 6, Jun. 2017