

# Programmable Interleaver Design for Analog Iterative Decoders

Vincent C. Gaudet, *Student Member, IEEE*, Rolland J. Gaudet, and P. Glenn Gulak, *Senior Member, IEEE*

**Abstract**—Several programmable analog interleaver architectures for iterative decoders are proposed. The architectures are evaluated in terms of transistor count, path resistance, path capacitance, and programming logic. Interleavers built out of networks consisting of three layers of small crossbars are often deemed the best, reducing both switch count and capacitance by over 70% for an interleaver size of 100, as opposed to full crossbars, while maintaining full programmability.

**Index Terms**—Analog decoders, error control codes, interleavers, iterative decoders, turbo codes.

## I. INTRODUCTION

SEVERAL families of very powerful error-correcting codes, whose performances on additive white Gaussian channels approach the Shannon limit, have recently emerged. Among these are Turbo codes [1], serially concatenated convolutional codes [2], and low-density parity-check codes [3]. Decoding algorithms for such codes follow an iterative procedure to successively improve estimates of decoded symbols. Unfortunately, such iterative procedures lead to long decoding latencies and to high power consumption—undesirable for wireless devices. Hence analog techniques are now being considered as an option for decoding [4]–[6]. To date, most analog decoders take advantage of the low silicon requirements of the associated circuitry in order to parallelize the decoding process. In other words, the trellises or factor graphs [7] on which soft-output algorithms operate are laid out in silicon. This has proven to be quite effective and has led to very-fast and low-power analog maximum *a posteriori* (MAP) decoders [8].

Fig. 1 shows a typical turbo decoder block diagram. Turbo decoders and other such iterative decoders require interleavers—preferably programmable—which permute the decoded symbols from one soft-output decoder for use by another soft-output decoder. In Fig. 1 the interleavers are denoted  $\Pi$  (interleaver) and  $\Pi^{-1}$  (deinterleaver).

In an analog decoder, the interleaver performs a spatial permutation of the soft information from one soft-output decoder,

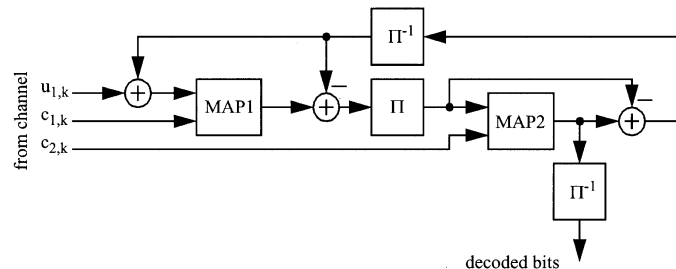


Fig. 1. Typical turbo decoder block diagram.

to be used by the other decoder. The permutation is denoted  $\Pi(k)$ ,  $0 \leq k < L$ , and the permuted bits are determined such that  $u_{2,k} = u_{1,\Pi(k)}$ . One of the challenges of building an analog-iterative decoder involves the design of these analog interleavers capable of performing any desired permutation of its symbols. The concepts involved in the design of analog interleavers are similar in many ways to those encountered in switching, for example in broadband photonic switching [9] and in multiprocessor memory architectures [10].

In this paper, several architectures for programmable analog interleavers are discussed and compared. The paper is organized as follows. Section II describes in general terms some implementation issues specific to analog interleavers. Section III introduces crossbars as a method of generating programmable analog interleavers. Section IV proposes several programmable analog interleaver architectures, based on networks of crossbars. Section V evaluates and compares the architectures based on several metrics. Section VI provides a delay analysis of the interleaver designs. Section VII concludes this paper.

## II. IMPLEMENTATION ISSUES

### A. Temporal Versus Spatial Interleaving

In a digital decoder design, an interleaver is normally implemented using SRAM memories. As the first soft-output decoder in a Turbo decoder produces its extrinsic information in serial order, it writes the data in the same order into an SRAM memory. Then the second soft-output decoder reads the information in a permuted order, the order either being stored in a separate memory block or being generated on-the-fly using algorithmic techniques. Using the former method, different permutations can be realized by storing a different reading order in the separate memory block. A similar procedure is used for the deinterleaving process, using the reverse order of input and output pointers to the memory.

We could extend this notion of interleaving in time to the analog domain by designing an analog memory cell and to

Manuscript received November 15, 2001; revised September 10, 2002. This work was supported by the Natural Sciences and Engineering Research Council of Canada. This paper was recommended by Associate Editor K. Yang.

V. C. Gaudet was with the University of Toronto, Toronto, ON M5B 2K3, Canada. He is now with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB T6G 2V4 Canada (e-mail: vgaudet@ee.ualberta.ca).

R. J. Gaudet is with the Collège Universitaire de Saint-Boniface, Winnipeg, MB R2H 0H7, Canada (e-mail: rgaudet@ustboniface.mb.ca).

P. G. Gulak is with the Edward S. Rogers Sr. Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5B 2K3, Canada (e-mail: gulak@eecg.toronto.edu).

Digital Object Identifier 10.1109/TCSII.2002.805022

read and write from that cell in a way analogous to the digital memory cell. The analog memory cell would likely store a voltage on a capacitor. However, this would discretize the decoding process in time and would, therefore, eliminate many of the benefits of continuous-time analog decoding.

Thus, instead of designing an interleaver which performs permutations in time, we will concentrate on interleavers which perform permutations in space, where symbols at one location on a chip are permuted to another location using switches.

### B. Fixed Versus Programmable Interleaver Design

The current state of the art analog interleaver is a fixed design implemented using wires. The design permutes a subset of the code's information bits in a regular, predetermined fashion [11]. If all  $L$  information bits in a code are permuted this way, then  $L$  wires per interleaver are required ( $2L$  wires are required if differential signaling is used).

In order to function with various standards, an interleaver should, however, be programmable to perform several, if not every single out of  $L!$  possible permutations of its  $L$  inputs. Hence, in this paper, we advance the state of the art by concentrating only on programmable interleaver designs, and specifically those designs on which all  $L!$  combinations of permutations can be programmed.

### C. Voltage Mode Versus Current Mode

The analysis of current mode circuitry is quite different from the analysis of voltage mode circuitry. In a current mode circuit, currents which are used by more than one subsequent circuit must be copied, whereas in a voltage mode circuit the voltage can be used as input to several subsequent circuits. In our analysis, we will assume that currents are copied before entry into a purely passive permutation network. Furthermore, we assume that each input to the permutation network is connected to exactly one output, thus making current copying unnecessary within the permutation network.

For this purpose we assume that switches are composed of a pass transistor, which passes either voltages or currents. Since pass transistors are bidirectional in nature, the same programming memories can be used to program an interleaver and its corresponding deinterleaver. More sophisticated switches like transmission gates could of course be used instead of pass transistors with little effect on the analysis herein.

With this type of permutation network, we can implement turbo decoders since they satisfy the one input to one output criterion, as well as other codes such as low-density parity-check codes which do not satisfy the criterion but where inputs being copied to multiple outputs can be copied before entry into the network.

## III. INTERLEAVER DESIGN USING CROSSBARS

### A. Crossbar Design

For the purpose of designing programmable spatial interleavers, we will use networks of complete crossbars with  $X$  inputs and  $X$  outputs, termed of size  $X$ . In a crossbar, a separate switch connects each input to each output. The negative metal-oxide-semiconductor (NMOS) pass transistor

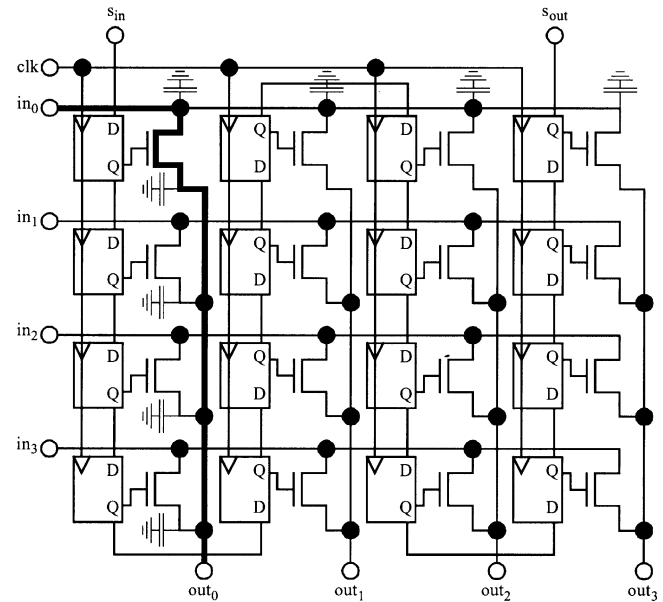


Fig. 2. Crossbar with four inputs and four outputs. A path from input  $in_0$  to output  $out_0$  is selected. The path crosses one pass transistor and is loaded by eight source or drain capacitances.

is chosen as a switch because of its easy programmability, its bidirectional nature, as well as its ability to deal with both voltage-mode and current-mode signals. The state of a switch is stored in a flip-flop, located close to the switch, and is programmed at startup in a serial fashion.

A crossbar of  $X$  inputs and outputs can be built using  $X^2$  switches per channel; for example, a turbo decoder with an interleaver-deinterleaver pair, which uses differential signaling would have four channels. In such a crossbar, each signal passes through exactly one transistor to reach any output from any input. Furthermore, each signal is loaded by the capacitance of  $X$  transistor sources and  $X$  drains, and, thus, the total parasitic capacitance is  $2C_s X$ , where  $C_s$  corresponds to one source or one drain capacitance. Such a crossbar, having four inputs and outputs, is depicted in Fig. 2.

### B. Programmability

Two methods of programming the crossbar switches are available. The first one uses one flip-flop per switch. The other method uses fewer flip-flops, but uses decoders to program the switches. Using the first method, a  $D$  flip-flop is placed near every switch. The output of the flip-flop programs the switch. The flip-flops are programmed serially at power-up, and can be set up as a scan chain for ease of testability. Using this method, a total of  $X^2$  flip-flops and no extra decoding logic are required. This method is the one depicted in the crossbar in Fig. 2.

Since each input in a crossbar is connected to exactly one output (as specified in Section II-C), we can use a one-of- $X$  decoder for each input to program which switch connected to that input is active. A crossbar requires one such decoder for each of the  $X$  inputs, each decoder having itself  $\lceil \log_2 X \rceil$  inputs. Thus,  $X \lceil \log_2 X \rceil$  flip-flops are used. An upper bound on the number of transistors in the decoding logic is  $X(2(2^{\lceil \log_2 X \rceil} - 1) + X \lceil \log_2 X \rceil)$ , assuming a static CMOS-based decoder, an

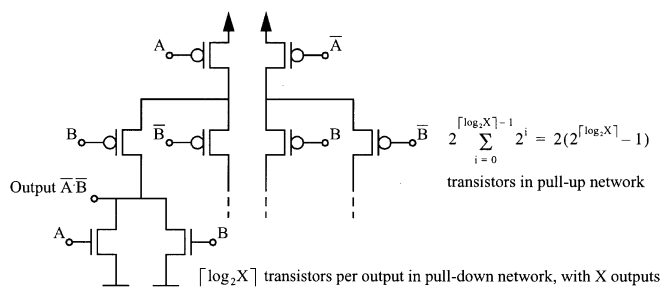


Fig. 3. Static CMOS decoder assumed for decoding logic calculations. A and B are address lines.

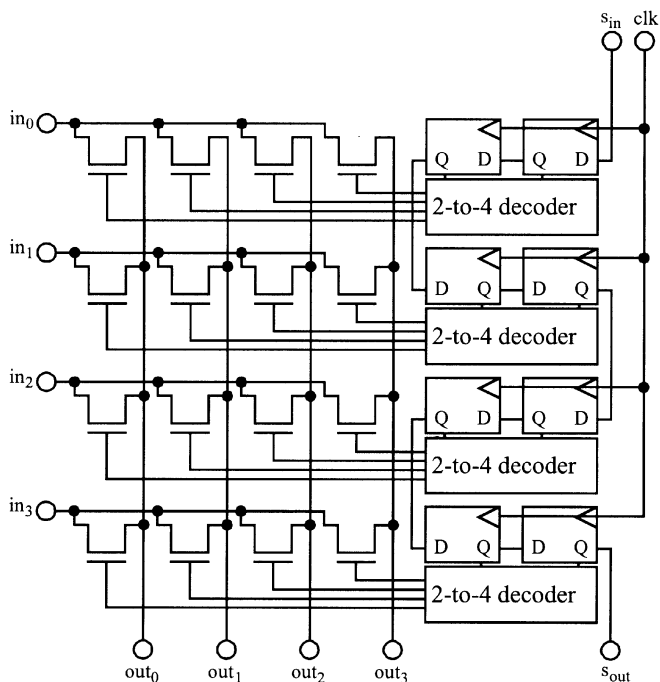


Fig. 4. Crossbar with four inputs and four outputs. Switches are programmed by serially shifting in encoded programming bits. The bits are decoded by  $X = 4$  decoders which determine the state of each switch.

example of which is depicted in Fig. 3. It is an upper bound since only  $X$  signals need to be produced by each decoder, and not necessarily all  $2^{\lceil \log_2 X \rceil}$  combinations. The static CMOS decoder is used since it has a strong pull up that maximizes the voltage range of the pass transistor which it controls.

Fig. 4 depicts a flip-flop/decoder arrangement for the  $X = 4$  crossbar. Note that this method might be impractical in terms of layout for large crossbars since  $X$  wires must run in parallel from each decoder to each row of switches; the multiple crossbar networks of Section IV-B–IV-E should not run into problems, though.

#### IV. INTERLEAVER ARCHITECTURES

In this section, we explore several interleaver architectures composed of one or several crossbars. Designs will be evaluated and compared in terms of five metrics.

The first metric is the total number of switches per channel used. A smaller number of switches usually implies a lower silicon area.

The second metric is the number of switches a signal must go through from any input to any output. A greater number of switches incurs a larger amount of circuit nonidealities such as parasitic resistance or noise.

The third metric is the total parasitic capacitance along any path through the interleaver. This metric is quantified in terms of the number of transistor sources and drains on any path, where the capacitance of one source or one drain is denoted  $C_s$ . For the purposes of this metric, switches are assumed to be NMOS pass transistors, though the analysis is easy to extend to other types of switches. Wiring capacitance is ignored, though it should be emphasized that it could increase total path capacitance for large designs.

The fourth metric is the total number of flip-flops used to program the interleaver. A flip-flop/decoder arrangement, as discussed in Section III-B is assumed. Otherwise, the same number of flip-flops as switches per channel are used. In addition to this fourth metric is a fifth, related metric which counts the total number of transistors in any logic used to decode the programming bits. The decoder logic is assumed to be built in static CMOS, as depicted in Fig. 3.

Most of the designs described below contain restrictions on the number of inputs and outputs. In cases where it would be advantageous (in terms of transistor count, path length, or capacitance), a frame can be buffered with zeros, using the next larger available interleaver size; some known dc voltage or current can be applied to unused inputs.

Five interleaver architectures are described in the next subsections.

##### A. Design Using One Crossbar

An obvious interleaver design uses one single crossbar where  $X = L$ . Of course, it uses  $L^2$  switches. Each signal passes through exactly one transistor, and is loaded by  $2L$  source or drain capacitances. The flip-flop count, assuming decoders, is  $L \lceil \log_2 L \rceil$ . The number of transistors in the decoding logic is bounded by  $L(2^{\lceil \log_2 L \rceil} - 1) + L \lceil \log_2 L \rceil$ .

In the next subsections, we see how to drastically reduce the transistor count and capacitance by allowing an increase in the number of pass transistors along any path.

##### B. Design Using Networks of Butterfly Switches

The second programmable interleaver is based on sorting networks. A sorting network is a combination of parallel processors which can sort a sequence of length  $L$ . If a sorting network is capable of sorting any sequence of length  $L$ , it is obvious that it is also capable of producing any permutation of its  $L$  inputs.

One such sorting network, realized using  $L(\log_2 L)(\log_2 L + 1)/4$  parallel butterfly switches is described in [12]; note that a butterfly switch is simply a crossbar where  $X = 2$ . It is constructed using several levels of butterfly switches, with the wires between subsequent levels performing a known, fixed permutation. Each butterfly switch is programmed using one configuration bit stored in a shift register. All permutations of the  $L$  bit positions are possible by using  $(\log_2 L)(\log_2 L + 1)/2$  levels, each containing  $L/2$  butterfly switches.

An interleaver designed using butterfly switches uses  $L(\log_2 L)(\log_2 L + 1)$  switches per channel. Each path from

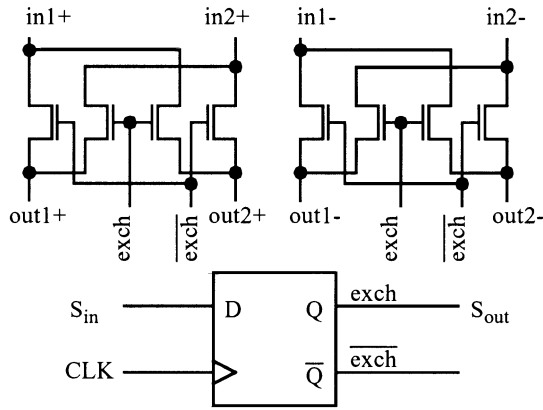


Fig. 5. Butterfly switch programmed using one configuration bit stored in a flip-flop.

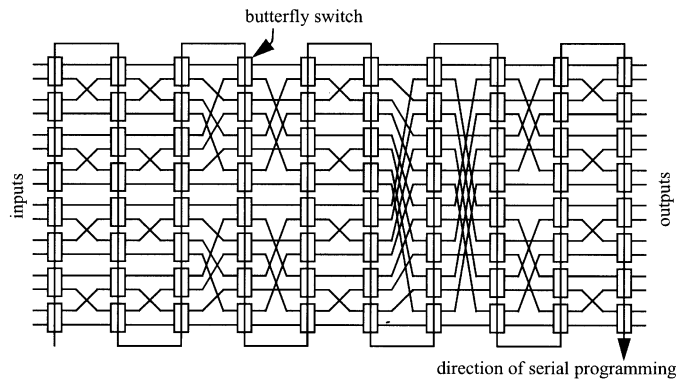


Fig. 6. Interleaver/deinterleaver composed of 10 levels of butterfly switches with direction of serial programming of the switches.

input to output crosses  $(\log_2 L)(\log_2 L + 1)/2$  pass transistors. The total source/drain capacitance along any path is  $2(\log_2 L)(\log_2 L + 1)C_s$ .

Fig. 5 shows a transistor-level implementation of a butterfly switch and how it is programmed. Fig. 6 shows a complete interleaver of size  $L = 16$ .

Though perhaps excessive in terms of path length, this interleaver has some interesting very large-scale integration properties, namely the reuse of a simple building block—the butterfly switch. Also, since butterfly switches only have two states, they can be programmed using one flip-flop each. No extra logic is required if we assume flip-flops which produce both  $Q$  and  $\bar{Q}$  outputs. The total number of flip-flops is given by  $L(\log_2 L)(\log_2 L + 1)/4$ .

### C. Design Using Three Levels of Small Crossbars

The third interleaver architecture is designed using a network of intermediate sized crossbars, and is similar to another sorting network first published during the 1960s and used in telephone switches [13]. If  $L$  is a square number, it is always possible to build a network, consisting of a total of  $3\sqrt{L}$  crossbars of size  $X = \sqrt{L}$ , which can implement all permutations of  $L$ .

The crossbars are organized into three layers of size  $\sqrt{L}$  crossbars. Between successive layers, each crossbar contains exactly one connection to each crossbar on the next level. That is, the  $i$ th crossbars  $j$ th output (where  $0 \leq i < \sqrt{L}$  and

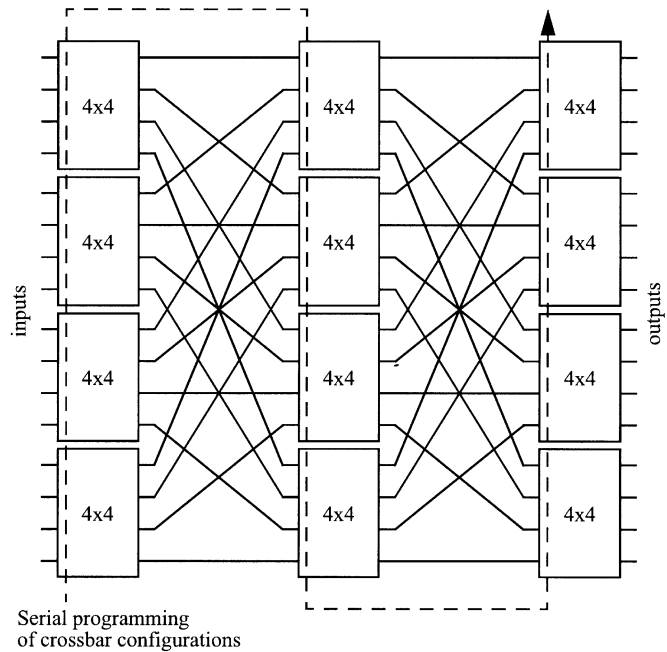


Fig. 7. Interleaver of size  $L = 16$  constructed using 12 crossbars of size  $4 \times 4$ .

$0 \leq j < \sqrt{L}$ ) on the first level is connected to the  $j$ th crossbars  $i$ th input on the second level, with the same interconnection pattern between the second and third levels. Such a network for  $L = 16$  is shown in Fig. 7.

The switch count per channel for this style of interleaver is  $3L^{3/2}$ . Each signal passes through exactly three transistors. The total source/drain capacitance is  $6\sqrt{L}C_s$ . The number of flip-flops is given by  $3L\lceil\log_2 \sqrt{L}\rceil$  and the transistor count in the decoding logic is bounded by  $3L(2(2^{\lceil\log_2 L^{1/4}\rceil} - 1) + \sqrt{L}\lceil\log_2 \sqrt{L}\rceil)$ .

### D. Hierarchical Design

If  $L$  is the square of a square number (that is,  $L = M^4$ ,  $M$  an integer), it is possible to recursively replace each  $\sqrt{L}$ -sized crossbar from Section IV-C with its own network of  $L^{1/4}$ -sized crossbars. Such a network contains  $9L^{5/4}$  switches per channel, distributed over  $9L^{3/4}$  crossbars, has a path length of nine, and has a source–drain capacitance of  $18L^{1/4}C_s$  along any path. Such a simplification may only be useful for large sizes of  $L$ . The flip-flop count is given by  $9L\lceil\log_2 L^{1/4}\rceil$  and the transistor count for the decoding logic is bounded by  $9L(2(2^{\lceil\log_2 L^{1/4}\rceil} - 1) + L^{1/4}\lceil\log_2 L^{1/4}\rceil)$ .

### E. Improvement on the Design Using Three Levels of Small Crossbars

In Section IV-C, a 3-level interleaver architecture composed of crossbars of size  $\sqrt{L}$  was introduced. Each level contained  $\sqrt{L}$  crossbars, with the restriction that  $L$  must be a square number. The concept can also be generalized to the case where  $L = PQ$  is a composite number [14].

Consider a network containing  $P$  crossbars of size  $Q$  on the first level,  $Q$  crossbars of size  $P$  on the second level, and  $P$  crossbars of size  $Q$  on the third level, where  $L = PQ$ . Between successive levels, each crossbar transmits one of its outputs to

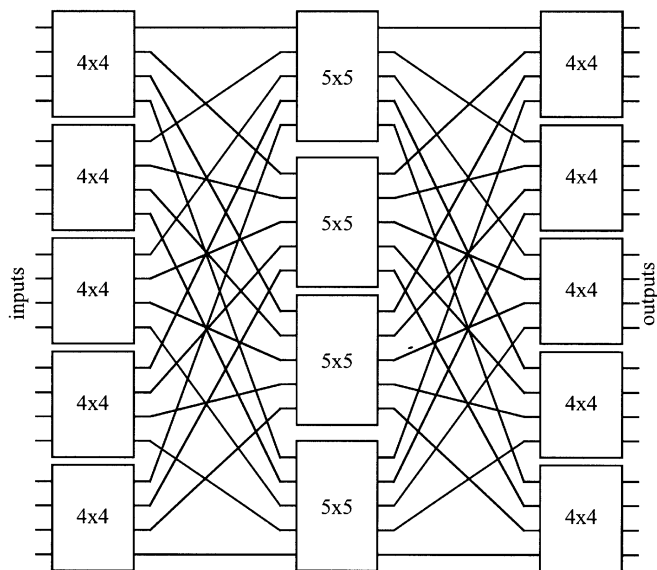


Fig. 8. Interleaver of size  $L = 20$  constructed using  $2P = 10$  crossbars of size  $4 \times 4$  and  $Q = 4$  crossbars of size  $5 \times 5$ .

TABLE I

COMPARISON OF ARCHITECTURES FOR INTERLEAVER SIZE  $L = 16$ . NOTE THAT IN THIS CASE THE HIERARCHICAL NETWORK IS BASED ON BUTTERFLY SWITCHES AND HENCE REQUIRES NO DECODING LOGIC

	Full Crossbar	Butterfly Network	Medium Sized Crossbar Network	Hierarchical Network	P,Q Network (P=4, Q=4)
Switch Count	256	320	192	288	192
Path Length	1	10	3	9	3
Capacitance	32	40	24	36	24
Flip-Flops	64	80	96	144	96
Decoder Logic	1504	0	672	0 <sup>a</sup>	672

each of the crossbars on the next level. That is, the  $i$ th crossbars  $j$ th output (where  $0 \leq i < P$  and  $0 \leq j < Q$ ) on the first level is connected to the  $j$ th crossbars  $i$ th input on the second level; the  $j$ th crossbars  $i$ th output (where  $0 \leq i < P$  and  $0 \leq j < Q$ ) on the second level is connected to the  $i$ th crossbars  $j$ th input on the third level. We call these  $P, Q$  networks. Fig. 8 depicts such a network for  $L = 20$ , with  $P = 5$  and  $Q = 4$ .

This type of architecture uses  $2PQ^2 + QP^2$  switches per channel. The path length is always 3 transistors, and the path capacitance is  $(2P + 4Q)C_s$ . The flip-flop count is given by  $PQ(2\lceil \log_2 Q \rceil + \lceil \log_2 P \rceil)$  and the number of decoding transistors is bounded by  $2PQ(2(2^{\lceil \log_2 Q \rceil} - 1) + Q\lceil \log_2 Q \rceil) + PQ(2(2^{\lceil \log_2 P \rceil} - 1) + P\lceil \log_2 P \rceil)$ .

## V. COMPARISON OF INTERLEAVING ARCHITECTURES: A QUANTITATIVE APPROACH

We now compare the five interleaver architectures presented above, by counting total number of switches, path length, path capacitance, flip-flop count, and decoder logic complexity. We compare the architectures for example interleaver sizes of  $L = 16, 40, 64, 100$ , and  $256$ . Results are given in Tables I–V. Zero padding is used where  $L$  is not an allowed size in the given

TABLE II

COMPARISON OF ARCHITECTURES FOR INTERLEAVER SIZE  $L = 40$  ( $L = 40$  IS THE SMALLEST INTERLEAVER SIZE IN THE UMTS STANDARD). NETWORKS ARE PADDED AS FOLLOWS: BUTTERFLY TO 64,  $L^{1/2}$  TO 49, AND  $L^{1/4}$  TO 81

	Full Crossbar	Butterfly Network (pad to 64)	Medium Sized Crossbar Network (pad to 49)	Hierarchical Network (pad to 81)	P,Q Network (P=8, Q=5)
Switch	1600	2688	1029	2187	720
Path Length	1	21	3	9	3
Capacitance	80	84	42	54	36
Flip-Flops	240	672	441	1458	360
Decoder Logic	14640	0	5145	8748	3840

TABLE III

COMPARISON OF ARCHITECTURES FOR INTERLEAVER SIZE  $L = 64$ . THE  $L^{1/4}$  NETWORK IS PADDED TO 81

	Full Crossbar	Butterfly Network	Medium Sized Crossbar Network	Hierarchical Network (pad to 81)	P,Q Network (P=13, Q=5)
Switch Count	4096	2688	1536	2187	1495
Path Length	1	21	3	9	3
Capacitance	128	84	48	54	46
Flip-Flops	384	672	576	1458	650
Decoder Logic	32640	0	6992	8748	9100

TABLE IV

COMPARISON OF ARCHITECTURES FOR INTERLEAVER SIZE  $L = 100$ . THE BUTTERFLY NETWORK IS PADDED TO 128 AND THE  $L^{1/4}$  NETWORK IS PADDED TO 256

	Full Crossbar	Butterfly Network (pad to 128)	Medium Sized Crossbar Network	Hierarchical Network (pad to 256)	P,Q Network (P=17, Q=6)
Switch Count	10000	7168	3000	9216	2958
Path Length	1	28	3	9	3
Capacitance	200	112	60	72	58
Flip-Flops	700	1792	1200	4608	1122
Decoder Logic	95400	0	21000	32256	21522

TABLE V

COMPARISON OF ARCHITECTURES FOR INTERLEAVER SIZE  $L = 256$

	Full Crossbar	Butterfly Network	Medium Sized Crossbar Network	Hierarchical Network	P,Q Network (P=20, Q=13)
Switch Count	65536	18432	12288	9216	11960
Path Length	1	36	3	9	3
Capacitance	512	144	96	72	92
Flip-Flops	2048	4608	3072	4608	3380
Decoder Logic	654848	0	72192	32256	85800

architecture. The  $P, Q$  networks are frequently the best in terms of switch count and capacitance. For example, with  $L = 100$ , a  $P, Q$  network uses about 70% fewer switches and has about 70% less path capacitance.

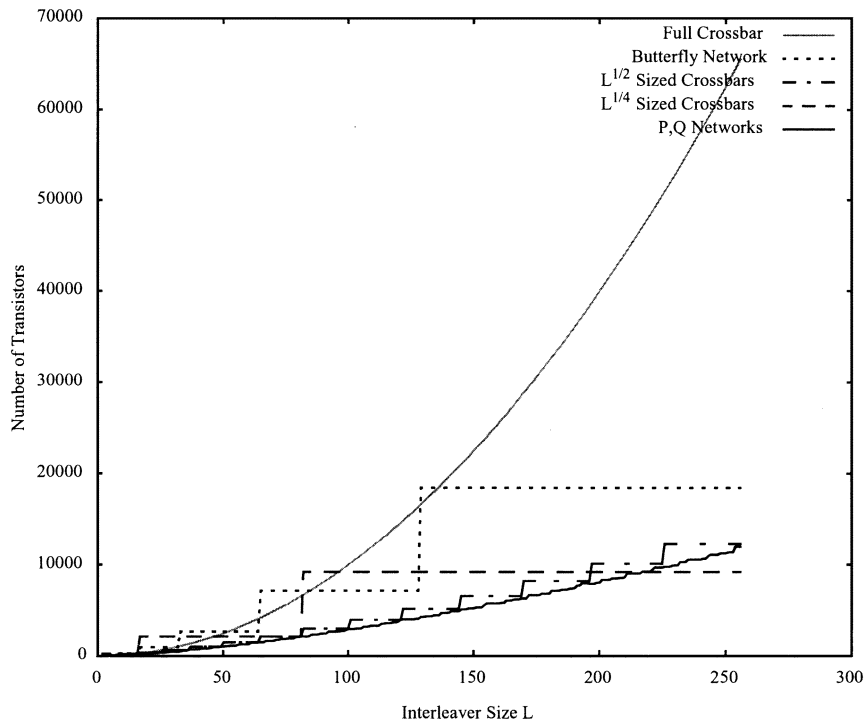


Fig. 9. Transistor counts for interleavers designed using five architectures,  $L = 2$  to 256.

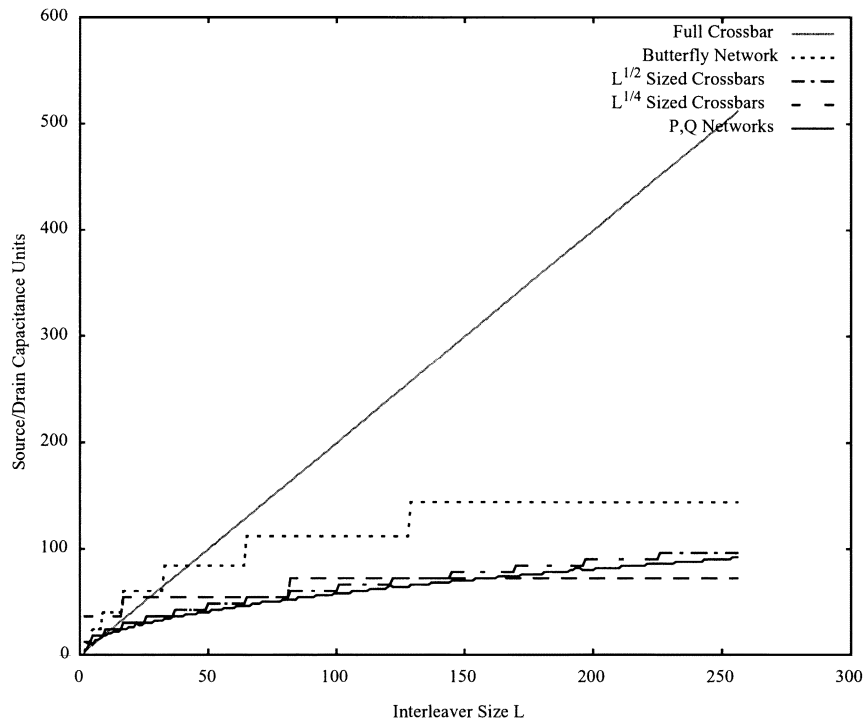


Fig. 10. Source and drain capacitance along signal path for interleavers,  $L = 2$  to 256.

Fig. 9 plots the switch count for all architectures described in Section IV versus interleaver size  $L$  for all interleaver sizes between 2 and 256. Zero buffering is assumed where the interleaver size is not allowed in the given architecture, or if it would lead to a smaller design in terms of switch count. Fig. 10 plots the capacitance per path, in terms of unit capacitance. Again, the  $P, Q$  networks are clearly winners.

## VI. ELMORE DELAY INTERPRETATION

We have now seen that by tolerating an increase in the number of switches a signal must cross on its path through the interleaver, we can significantly decrease interleaver size and path capacitance. By using  $P, Q$  networks, path capacitance is reduced by 70% for an interleaver size of  $L = 100$ , as opposed

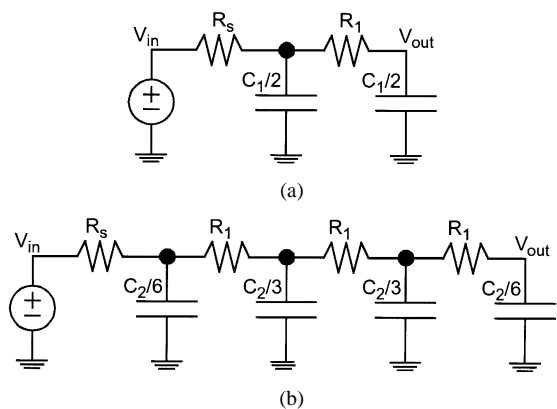


Fig. 11.  $R$ - $C$  networks corresponding to (a) a full crossbar and (b) a  $P$ ,  $Q$  network.

to a full crossbar. However, the advantage of decreasing the capacitance by two-thirds while tripling the resistance is not immediately obvious.

The advantage can be explained in terms of Elmore delay [15]. If the capacitance and resistance in a resistance–capacitance ( $R$ - $C$ ) network are not lumped into one single resistor and one single capacitor, but distributed over a few distinct nodes, then the delay is not as large as might be expected by calculating a simple  $R$ - $C$  delay, and can be calculated more accurately by the Elmore delay.

Let us first examine the  $R$ - $C$  network corresponding to a single crossbar, where the total path resistance is  $R_1$  and the total capacitance due to sources and drains is  $C_1$ . The capacitance is distributed so that  $C_1/2$  are located on either side of the resistor, as illustrated in Fig. 11(a). Also, the network is driven by a voltage source with finite impedance  $R_s$ .

The Elmore delay  $t_{E1}$  of this network is  $t_{E1} = R_s C_1/2 + (R_s + R_1)C_1/2 = (R_s + R_1/2)C_1$ .

We now shift our attention to the  $R$ - $C$  network corresponding to a  $P$ ,  $Q$  network, as illustrated in Fig. 11(b). Each resistor is equal to  $R_1$ . Here the capacitance  $C_2$  is distributed onto four nodes. The first and last node each get  $C_2/6$ , whereas the two internal nodes each get  $C_2/3$ . This is a direct result of the structure of a  $P$ ,  $Q$  network, where twice as many sources and drains are located on the two inner wiring networks as on the inputs and outputs. In this case the Elmore delay  $t_{E2}$  is  $t_{E2} = R_s C_2/6 + (R_s + R_1)C_2/3 + (R_s + 2R_1)C_2/3 + (R_s + 3R_1)C_2/6 = (R_s + 3R_1/2)C_2$ .

In order to compare  $t_{E2}$  to  $t_{E1}$ , the following function is plotted in Fig. 12:

$$\rho_t = t_{E1}/t_{E2} = \frac{(R_s/R_1 + 3/2)C_2}{(R_s/R_1 + 1/2)C_1} = \frac{(\rho_R + 3/2)}{(\rho_R + 1/2)} \rho_C.$$

In the function, the ratio of  $P$ ,  $Q$  network capacitance to crossbar capacitance is  $\rho_C = C_2/C_1$  and the ratio of source to switch resistance is  $\rho_R = R_s/R_1$ . The ratio of Elmore delays is  $\rho_t$ , where  $\rho_t < 1$  implies that the  $P$ ,  $Q$  network is faster. Fig. 12 plots  $\rho_t$  for  $0.1 < \rho_C < 10$  and  $0.1 < \rho_R < 10$ . Values of  $\rho_C$  are obtained from path capacitance, as calculated earlier, and values of  $\rho_R$  are dependent on specific switches and circuits driving the interleaver, though we would expect  $0.1 < \rho_R < 10$ .

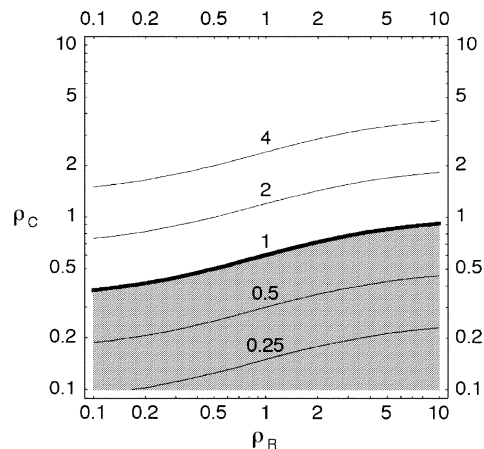


Fig. 12. Plot of  $\rho_t$ , the ratio of Elmore delay for a  $P$ ,  $Q$  network to the Elmore delay for a full crossbar.

The shaded region in Fig. 12 indicates where it is advantageous to use  $P$ ,  $Q$  networks, since the delay through the  $P$ ,  $Q$  interleaver is less than that through a crossbar. It is important to note that the region is greater than that contained by  $\rho_C < 1/3$ .

## VII. CONCLUSION

This paper has explored five novel architectures for programmable analog interleavers.

The first is based on a single crossbar. The other four are based on networks of crossbars. The second consists entirely of butterfly switches; it only requires very simple programming logic, but each signal has to pass through a large number of transistors. The third method, valid for interleavers with a square number of inputs, requires signals to pass through three levels of crossbars, each crossbar having a number of inputs equal to the square root of the total number of inputs for the interleaver. A fourth method proposes to recursively decompose the smaller crossbars from the third method into even smaller crossbars; however, signals have to pass through a greater number of transistors.

The final method, and the one which seems the most promising in terms of several metrics including switch count and path capacitance, involves the decomposition of the interleaver size  $L$  into two factors  $P$  and  $Q$ , and to build a three-level network of crossbars the size of the factors. Significant decreases in total transistor count and in capacitance are achieved.

## ACKNOWLEDGMENT

The authors wish to thank the anonymous reviewers for useful comments.

## REFERENCES

- [1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," in *Proc. IEEE Int. Conf. Communications*, Geneva, Switzerland, May 23–26, 1993, pp. 1064–1070.
- [2] S. Benedetto and G. Montorsi, "Iterative decoding of serially concatenated convolutional codes," *Electron. Lett.*, vol. 32, no. 13, June 1996.
- [3] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inform. Theory*, vol. IT-8, pp. 21–28, Jan. 1962.

- [4] J. Hagenauer and M. Winklhofer, "The analog decoder," in *Proc. 4th IEEE Int. Symp. Information Theory*, Cambridge, MA, Aug. 16–21, 1998, p. 145.
- [5] H. -A. Loeliger, M. Helfenstein, F. Lustenberger, and F. Tarköy, "Iterative sum-product decoding with analog VLSI," in *Proc. IEEE Int. Symp. Information Theory*, Cambridge, MA, Aug. 16–21, 1998, p. 146.
- [6] C. Winstead, J. Dai, S. Little, C. Myers, C. Schlegel, Y.-B. Kim, and W. J. Kim, "Analog MAP decoder for (8,4) Hamming code in subthreshold CMOS," in *Proc. Int. Symp. Information Theory*, Washington, DC, June 24–29, 2001, p. 330.
- [7] F. R. Kschischang and B. Frey, "Iterative decoding of compound codes by probability propagation in graphical models," *IEEE J. Select. Areas Commun.*, vol. 16, pp. 219–230, Feb. 1998.
- [8] M. Moërz, T. Gabara, R. Yan, and J. Hagenauer, "An analog 0.25  $\mu\text{m}$  BiCMOS tailbiting MAP decoder," in *Proc. IEEE Int. Solid-State Circuits Conf. Dig. Techn. Papers*, 2000, pp. 356–357.
- [9] H. S. Hinton, "Photonics in switching (systems for telecommunications)," *IEEE Mag. Lightwave Commun. Syst.*, vol. 3, pp. 26–35, Aug. 1992.
- [10] D. Rana and C. C. Weems, "The ICAP parallel processor communications switch," in *Proc. IEEE Int. Symp. Circuits and Systems*, vol. 1, Portland, OR, May 9–11, 1989, pp. 126–129.
- [11] F. Lustenberger, "On the design of analog VLSI iterative decoders," Doctor of Technical Sciences dissertation, Swiss Federal Institute of Technology, Zurich, Switzerland, pp. 199, 2000.
- [12] K. E. Batcher, "Sorting networks and their applications," in *Proc. AFIPS Spring Joint Computer Conf.*, vol. 32, Atlantic City, NJ, Apr. 1968, pp. 307–314.
- [13] V. E. Beneš, "Optimal rearrangeable multistage connecting networks," *Bell Syst. Techn. J.*, pp. 1641–1656, 1964.
- [14] D. C. Opferman and N. T. Tsao-Wu, "On a class of rearrangeable switching networks," *Bell Syst. Techn. J.*, vol. 50, pp. 1579–1618, May 1971.
- [15] E. Elmore, "The transient response of damped linear networks with particular regard to wideband amplifiers," *J. Appl. Phys.*, pp. 55–63, Jan. 1948.



**Vincent C. Gaudet** (S'97) received the B.Sc. degree in computer engineering from the University of Manitoba, Winnipeg, MB, Canada, in 1995 and the Master of Applied Science degree from the University of Toronto, Toronto, ON, Canada, in 1997, where he is currently working toward the Ph.D. in the Department of Electrical and Computer Engineering.

From February 2002 to July 2002, he was a Research Associate at the Ecole Nationale Supérieure des Télécommunications de Bretagne, Brest, France.

Since August 2002, he has been an Assistant Professor in the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada. His research interests include the design of integrated circuits—both analog and digital—for high-speed digital communications, the design and use of field-programmable devices, and evolutionary programming techniques.

Dr. Gaudet has been registered as an Engineer-in-Training with the Professional Engineers of Ontario since 1996. He was awarded the University Gold Medal from the University of Manitoba, Winnipeg, MB, Canada in 1995. He has received funding from the Natural Sciences and Engineering Research Council, the Ontario Graduate Scholarship in Science and Technology, and the Walter Sumner Memorial Fund.



**Rolland J. Gaudet** received the B.A. degree from the University of Manitoba, Winnipeg, MB, Canada, in 1966, the Master of Arts degree from the University of Saskatchewan, Saskatoon, SK, Canada, in 1969, and the Ph.D. degree from the University of Alberta, Edmonton, AB, Canada, in 1973.

From 1973 to 1988, he taught at the Université de Sherbrooke, where he also served as Head of the Mathematics and Computer Science Department. From 1987 to 1999, he served on and Chaired examination committees for the Society of Actuaries (Course 162, Construction of mortality tables, and Course 130, Operations research). Since 1988, he has been a Professor at the Collège Universitaire de Saint-Boniface, Winnipeg, MB, Canada. He is coauthor with Dr. Jean-Guy Dion of *Méthodes d'analyse numérique: de la théorie à l'application* (Mont-Royal, QC, Canada: Modulo Editeur, 1996), which was awarded the Adrien-Pouliot prize for excellence in communications.

Dr. Gaudet was named Outstanding Teacher by University Teaching Services at the University of Manitoba, in 1994.



**P. Glenn Gulak** (S'82–M'83–SM'96) received the Ph.D. degree from the University of Manitoba, Winnipeg, MB, Canada, in 1984.

From 1985 to 1988, he was a Research Associate with the Information Systems Laboratory and the Computer Systems Laboratory, Stanford University, Stanford, CA. Currently, he is a Professor with the Department of Electrical and Computer Engineering, at the University of Toronto, Toronto, ON, Canada, where he is the recipient of the L. Lau Chair in Electrical and Computer Engineering. His research

interests are in the areas of memory design, circuits, algorithms, and VLSI architectures for digital communications.

Dr. Gulak is a registered Professional Engineer in the province of Ontario. He received a Natural Sciences and Engineering Research Council of Canada Postgraduate Scholarship and several teaching awards for undergraduate courses taught in both the Department of Computer Science and the Department of Electrical and Computer Engineering at the University of Toronto, Toronto, ON, Canada. He served as the Technical Program Chair for the International Solid State Circuits Conference in 2001.