# VLSI IMPLEMENTATION AND COMPLEXITY COMPARISON OF RESIDUE GENERATORS MODULO 3

*Stanisław J. Piestrak*[†,‡] *Fabrice Pedron*[†] *Olivier Sentieys*[†]

[†] ENSSAT — Université de Rennes 1
LASTI, B.P. 447, 6 rue de Kerampont
22305 Lannion CEDEX, France
Tel: +(33) 2 96 46 50 30
e-mail: {name}@enssat.fr

[‡] Wrocław University of Technology
Institute of Engineering Cybernetics
50–370 Wrocław, Poland
Tel: +(48) 71 320 28 73
e-mail: sjp@residue.ict.pwr.wroc.pl

**ABSTRACT**

A generator modulo 3 (mod 3) is a circuit that generates a residue mod 3 from a binary vector. It is an essential circuit used to construct the encoding and checking circuitry for arithmetic error detecting codes, such as residue codes mod 3 and the $3N$ code, as well as some residue number system hardware. In this paper, we compare speed and area of varius VLSI implementations of 16-input generators modulo 3. It is shown that the generator built of full-adders consumes the least area. On the other hand, the generator built as a tree of special 4-input modules is twice as fast, although at the cost of increasing the area by a factor of 1.7.

## 1 INTRODUCTION

A generator modulo 3 (mod 3) is a circuit that generates a residue mod 3 from a binary vector. It is an essential circuit used to construct the encoding and checking circuitry for arithmetic codes with the check base $A = 3$, i.e. residue codes mod 3 and the $3N$ code, as well as some residue number system (RNS) hardware. The arithmetic codes with the check base $A = 3$ are the least redundant codes capable of detecting arithmetic errors, since they only require two extra bits. Several applications of arithmetic codes with $A = 3$ have been reported e.g. in [1]–[5] and [9]. They include various digital blocks protected with the residue codes mod 3, such as: fast carry lookahead adders [1], serial [3] and parallel multipliers [9], as well as parallel multipliers protected with the $3N$ codes [4]. The extensive studies of a complex VLSI processor entirely protected by the mod 3 residue code as well as dedicated digital signal processors with CED (also using the mod 3 residue code) were reported in [9] and [5], respectively. Another important field of application of the residue mod 3 hardware are digital signal processors employing residue number system (RNS) arithmetic, with $A = 3$ as one of the moduli [6]. In the latter case, the residue generator mod 3 constitutes a part of the binary-to-residue number system converter. Also, a $k$-operand adder mod 3 can be seen as nothing else but the $2k$-input generator mod 3.

Several different implementations of the generators mod 3 have been proposed in the literature [2], [7]–[13], but the area-time performance of all these residue generators has been estimated on the gate level, i.e. by the number of gates, gate inputs, and gate levels. However, it has long been recognized that such estimations can be highly inaccurate, as the chip area consumed by the interconnections can reach even 40%. We have not been able to locate results of any investigations reported in the open literature about the performance of any of these generators, when they are actually implemented in VLSI.

In this paper, we will present speed/area evaluation of layouts of four versions of 16-input generators mod 3, designed according to two essentially different approaches, which are currently known as the most efficient 'gate-level' circuits. Amongst them are included the layouts of the generators obtained by using logic-level synthesis tools using VHDL. One general approach, presented in [12] allows for using only full-adders (FAs) to build an entire generator. The other approach, which relies on using multi-output threshold circuits $T^n$ [13], provides a number of alternative designs which allow for optimization of the number of gates, gate inputs, or gate levels. More detailed presentation of these methods will be given in two sections that follow. The comparison and layout details will be given in Section 4.

## 2 GENERATORS MOD 3 BUILT USING FULL-ADDERS

Let $X = \{x_{n-1}, \ldots, x_1, x_0\}$ be an input vector of the generator mod 3, where $x_{n-1}$ is the most significant bit and $x_0$ is the least significant bit. The generator converts an integer $X = \sum_{j=0}^{n-1} 2^j x_j$ into $|X|_3 = \sum_{j=0}^{1} 2^j s_j$, i.e. it computes $|X|_3 = \left|\sum_{j=0}^{n-1} 2^j x_j\right|_3$.

The generators mod 3 from [12] exploit the periodicity of the series of $|2^j|_A$, where $|2^j|_A$ denotes $2^j$ mod $A$. In particular, for $A = 3$ the *half-period*, i.e. the distance between subsequent 1 and $-1$ is $HP(3) = 1$, due to congruence $-1 \equiv 2 \bmod 3$. As a result, an entire generator mod 3 can be built using FAs only

— a sample 16-input generator built on the basis of this concept is shown in Fig. 1. The simplicity of such a generator follows from the observation that it is sufficient to complement every input bit $x_i$ with odd $i$ (whose weight mod 3 is $-1$) as well as all internal carry signals generated by FAs, provided that a suitable correction is added to the final result. For $n$ even no correction is needed, whereas for any odd $n$, the final result must add 2 mod 3. In the latter case, the correction circuit realizes the following functions:

$$\begin{aligned} c_0^* &= c_0 \odot c_1 \\ c_1^* &= c_0 \cdot \overline{c_1} \end{aligned}$$

Note that it allows for only one representation of 0 — (00), to be produced at the output (i.e. it can generate only three output combinations $(c_1^* c_0^*) \in \{(00), (01), (10)\}$). Such a circuit must be added to modify the outputs of any FA-based generator with odd number $n$ of inputs.

Clearly, the advantages of this generator are: (1) basically only one cell is used — a FA, although with some inputs complemented; and (2) highly regular structure, as only adjacent cells are connected and there are no long interconnection lines.

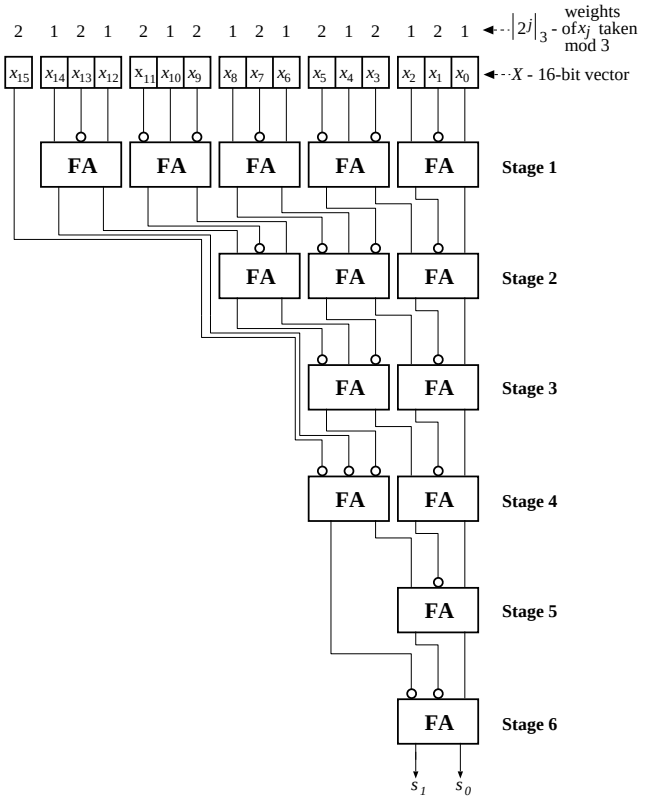


Figure 1: CSA-based 16-input generator mod 3 using half-period.

## 3 GENERATORS MOD 3 BUILT USING THRESHOLD CIRCUITS

The other general approach, which relies on using multi-output threshold circuits $T^k$, was proposed in [13]. For any number of inputs $k$, a family of generators mod 3 can be built of multi-output threshold circuits $T^k$ followed by some merging network. The most cost-efficient threshold circuits can be realized as sorting networks — see [14], [15], [16] — which can be entirely built using a comparator cell composed of a pair of 2-input OR and AND gates. These new generators were shown to be the least complex and the fastest gate-level generators mod 3 currently known.

The logic structure of one of the basic cells constructed using threshold circuits — $M_4$ with four inputs — is shown in Fig. 2, whereas a tree-structured 16-input generator mod 3 built using only this cell is shown in Fig. 3. It is seen that such a generator also enjoys a modular and highly regular structure composed of gates with low fan-in which makes them attractive for VLSI implementation.

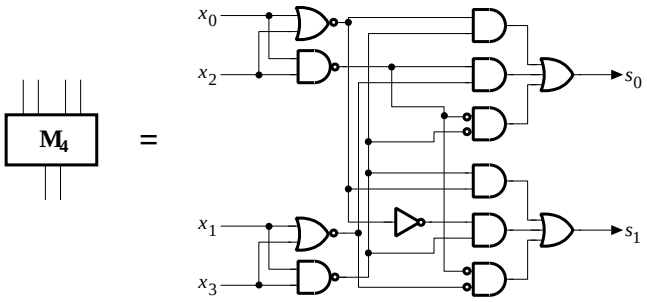


Figure 2: Cell $M_4$ of the 4-input generator mod 3.

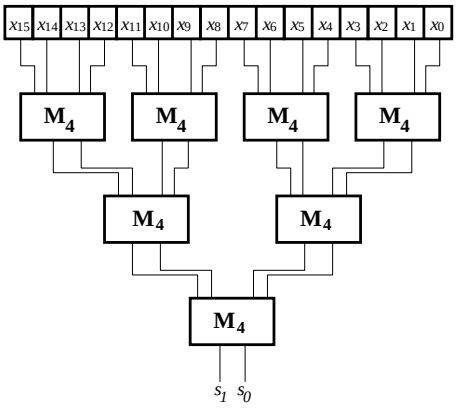


Figure 3: Tree structure of the 16-input generator mod 3 using cell $M_4$.

To convey some idea about relative complexity of a generator built using $M_4$, note that the cell $M_4$ does

the same as two FAs: it reduces the number of bits from four to two. Therefore, the generator from Fig. 3 seems to be potentially faster and perhaps less complex than one from Fig. 1. At present, however, it is difficult to predict which VLSI implementation would actually use less chip area and/or introduce less delay. A more definitive conclusion regarding performance of these as well as many potentially even more efficient alternative generators also presented in [13] cannot be drawn unless some layouts for sample generators mod 3 are generated.

## 4 COMPARISON

As a vehicle for complexity comparison, we have selected four various versions of 16-input generators mod 3, whose layouts are evaluated in Table 1. All layouts were generated using AVANT! tools (formerely COMPASS) and CMOS ES2 0,7$\mu$m technology.

Table 1: Parameter comparison of various versions of 16-input generators mod 3.

| Version | | Area[$0^{-3}$mm$^2$] | Delay[ns] |
|---|---|---|---|
| Stand. cells | Using FAs | 41,70 | 18,12 |
| | Using $M_4$ | 70,11 | 8,78 |
| VHDL | Using $M_4$ | 74,49 | 19,32 |
| | Behavioral | 69,08 | 22,43 |

We have made considered the following specifications:

- Schematic specification using FAs built of standard cells (as in Fig. 1);
- Schematic specification using $M_4$ built of standard cells (as in Figures 2 and 3);
- VHDL specification using $M_4$ as the basic module (the $M_4$ cell is optimized by the logic synthesizer); and
- VHDL behavioral specification using periodicity of $|2^j|_3$ (which requires less than 100 lines of VHDL code).

The layouts of the first two versions were generated using respectively FAs and $M_4$ as basic modules. The layouts of these two basic modules (each built of standard cells), which were subsequently used to generate layouts of the 16-input generators, are shown in Figures 4 and 5, respectively. The two last specifications have been synthesized with VHDL logic synthesizer of AVANT! suite. The advantage of using such description is that one has a portable version of the generator. In exchange, the result is less efficient, but an area-time trade-off can be obtained by tuning the synthesis tool constraints.

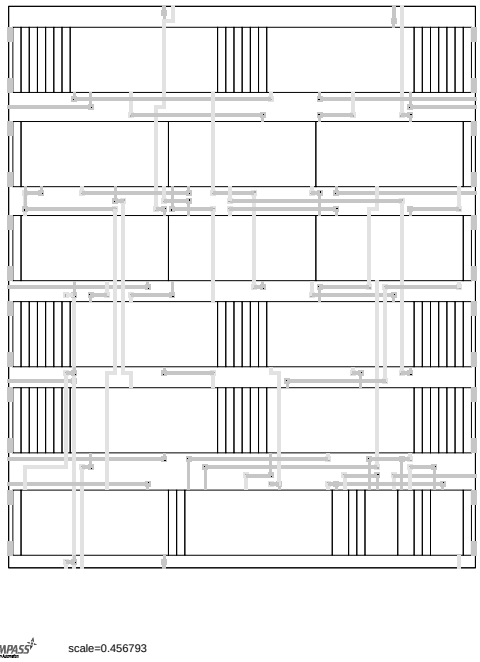


Figure 4: Layout of the FA used to build a generator.

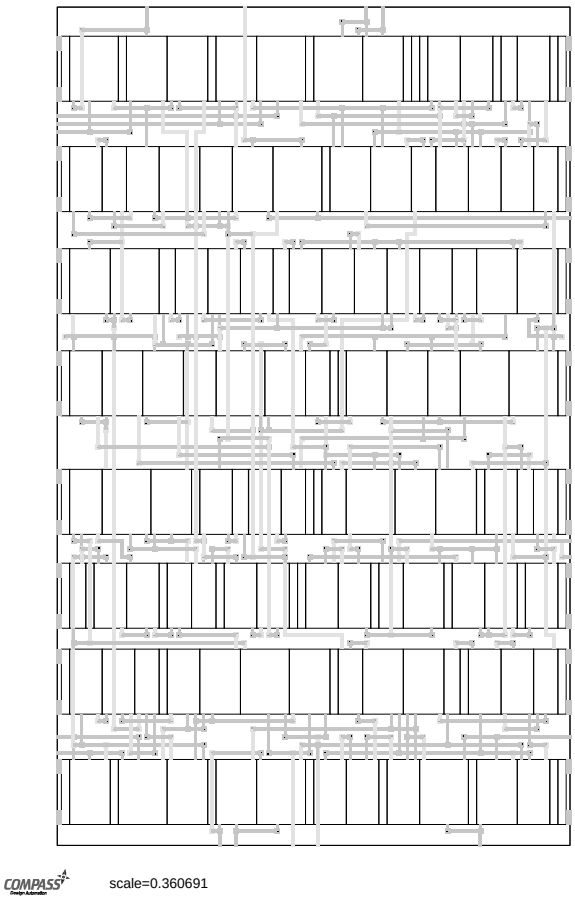


Figure 5: Layout of the module $M_4$.

The VHDL behavioral description is made with three modules: `add8to4`, `add4to3`, and `add3to2`.

```
Entity Add8to4 is
   port ( E : in bit_vector(7 downto 0);
      D : out bit_vector(3 downto 0));
end add8to4;

Architecture behavioral of add8to4 is
begin
      process (E)
         variable TMP : Bit_Vector(3 downto 0);
      begin
         TMP := "0000";
         for i in 0 to 7 loop
            TMP := E(i)+TMP;
         end loop;
      D <<= TMP;
      end process;
end behavioral;
```

Figure 6: VHDL behavioral description of `add8to4` that generates the sum of 8 bits.

The principal advantages of the VHDL behavioral description are:

1. It can be easily used as a generic specification (independant of the number of inputs); and

2. It can trade-off speed *vs.* area during synthesis process by putting time constraints.

## 5 CONCLUSIONS

Four VLSI implementations of a 16-input generator modulo 3 were considered. Included were two versions built using standard cells: one entirely built of full-adders while the other built as a tree of special 4-input modules. The remaining two versions were derived on the basis of the VHDL specifications. The comparison of speed and area of varius layouts showed that the generator built of full-adders consumes the least area. On the other hand, the generator built as a tree of special 4-input modules is twice as fast, although at the cost of increasing the area by a factor of 1.7.

### References


[1] G. G. Langdon, Jr. and C. K. Tang, "Concurrent error detection for group look-ahead binary adders," *IBM J. Res. Develop.*, vol. 14, pp. 563–573, Sep. 1970.

[2] A. Avižienis, "Arithmetic codes: Cost and effectiveness studies for applications in digital system design," *IEEE Trans. Comput.*, vol. C-20, pp. 1322–1331, Nov. 1971.

[3] T. J. Brosnan and N. R. Strader II, "Modular error detection for bit-serial multiplication," *IEEE Trans. Comput.*, vol. C-37, pp. 1043–1052, Sep. 1988.

[4] V. Piuri and R. Stefanelli, "Concurrent error detection in parallel multipliers and complex arithmetic structures: Remarks on the use of the $3N$ code," *Microproc. and Microprogr.*, vol. 31, pp. 47–52, 1991.

[5] B. Grimal, "Synthèse d'architectures auto-testables dédiées à des applications de traitement du signal," Ph. D. Thesis, Univ. de Rennes 1 — ENSSAT, Dec. 1994 (in French).

[6] M. A. Soderstrand *et al.*, (Eds), *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*, IEEE Press, NY, 1986.

[7] F. F. Sellers, Jr., M.-Y. Hsiao, and L. W. Bearnson, *Error Detecting Logic for Digital Computers*, McGraw-Hill, New York, 1968.

[8] J. F. Wakerly, *Error Detecting Codes, Self-Checking Circuits and Applications*, North-Holland, New York, 1978.

[9] G. Russell and I. L. Sayers, *Advanced Simulation and Test Methodologies for VLSI Design*, Van Nostrand Reinhold Int., London, 1989, Ch. 9.

[10] S. J. Piestrak, "Design of high-speed and cost-effective self-testing checkers for low-cost arithmetic codes," *IEEE Trans. Comput.*, vol. C-39, pp. 360–374, March 1990.

[11] S. J. Piestrak, "Design of residue generators and multi-operand modular adders using carry-save adders," in *Proc. 10th Symp. Computer Arith.*, Grenoble, France, June 25–27, 1991, pp. 100–109.

[12] S. J. Piestrak, "Design of residue generators and multioperand modular adders using carry-save adders," *IEEE Trans. Comput.*, vol. 43, pp. 68–77, Jan. 1994.

[13] S. J. Piestrak, "Design of residue generators and multioperand adders modulo-3 built of multi-output threshold circuits," *IEE Proc.-Comput. Digit Tech.*, vol. 141, pp. 129–134, March 1994.

[14] D. E. Knuth, *The Art of Computer Programming, Vol. III, Sorting and Searching*, 2nd Ed., Reading, MA, Addison-Wesley, 1973, Ch. 5.

[15] S. J. Piestrak, "The minimal test set for sorting networks and the use of sorting networks in self-testing checkers for unordered codes," in *Dig. Pap. 20th Int. Symp. on FTC*, Newcastle upon Tyne, UK, June 26–28, 1990, pp. 467–474.

[16] S. J. Piestrak, "The minimal test set for multi-output threshold circuits implemented as sorting networks," *IEEE Trans. Comput.*, vol. 42, pp. 700–712, June 1993.