Efficient Decompression of Binary Encoded Balanced Ternary Sequences
Olivier Muller, Adrien Prost-Boucle, Alban Bourge, Frédéric Pétrot

To cite this version:
Olivier Muller, Adrien Prost-Boucle, Alban Bourge, Frédéric Pétrot. Efficient Decompression of Binary Encoded Balanced Ternary Sequences. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2019. hal-02103214

HAL Id: hal-02103214
https://hal.archives-ouvertes.fr/hal-02103214
Submitted on 18 Apr 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Efficient Decompression of Binary Encoded Balanced Ternary Sequences

Olivier Muller, Adrien Prost-Boucle, Alban Bourge, Frédéric Pétrot Member, IEEE

Abstract—A balanced ternary digit, known as a trit, takes its values in \{-1, 0, 1\}. It can be encoded in binary as \{11, 00, 01\} for direct use in digital circuits. In this correspondence, we study the decompression of a sequence of bits into a sequence of binary encoded balanced ternary digits. We first show that it is useless in practice to compress sequences of more than 5 ternary values. We then provide two mappings, one to map 5 bits to 3 trits and one to map 8 bits to 5 trits. Both mappings were obtained by human analysis and lead to boolean implementations that compare quite favorably with others obtained by tweaking assignment or encoding optimization tools. However, mappings that lead to better implementations may be feasible.

I. INTRODUCTION

Ternary encoding of data has been shown useful at least in the following contexts: general purpose computing [1], wireless transmission [2], [3], texture representation in images [4], quantum computing [5], optical super computing [6], and artificial neural networks [7][8].

In many applications, the binary value representing a sequence of \{-1, 0, 1\} needs to be stored in memory, so finding an encoding that minimizes both the compressor and decompressor is a legitimate goal. However, our own focus is the VLSI implementation of neural networks making use of ternary weights, in which the weight values are written in a memory only once and read almost continuously [9]. In that case, it is necessary to combinatorially produce a sequence of binary encoded balanced ternary values from an encoded binary string.

Our objective is thus to determine a mapping (i.e. a one-to-one function that maps binary strings to binary encoded balanced ternary values) which, when implemented as a boolean multi-valued one function that maps binary strings to binary encoded balanced ternary string.

In the following sections, we show that it is not useful to compress more than 5 trits on 8 bits, and give the best mappings that we found, i.e. the ones requiring fewer gates, for compressing 3 trits on 5 bits and 5 trits on 8 bits. Please note that we do not propose a general algorithm to solve the problem for sequences of any length.

II. PROBLEM FORMULATION

A ternary digit contains \(\log_3(3) \approx 1.586\) bits of information. We compute the maximum theoretical gain that can be obtained by compressing trits in binary. As a sequence of \(n\) trits \((n \in \mathbb{N})\) represents \(3^n\) values, at least \([\log_2(3^n)] = [n \log_2(3)]\) bits are necessary to encode this sequence in binary. For \(n\) trits the gain compared to the non encoded sequence is given by \(u_n = \frac{2n - n \log_2(3)}{2n}\). By definition \(x \leq \lceil x \rceil < x + 1\), hence

\[
\frac{2n - n \log_2(3)}{2n} \geq u_n > \frac{2n - (n \log_2(3) + 1)}{2n}
\]

Yet \(\forall n \in \mathbb{N}, u_n = 1 - \frac{\log_2(3)}{n}\) and \(\lim_{n \to \infty} u_n = 1 - \frac{\log_2(3)}{n}\). Since \(u_n\) and \(w_n\) increase monotonically, this yields by the squeeze theorem \(\lim_{n \to \infty} u_n = 1 - \frac{\log_2(3)}{n} \approx 0.2075\).

Table I gives the actual gain for small values of \(n\). As can be seen, there is not much interest in encoding sequences of more than 5 trits (actually 10 bits) on 8 bits, since it is at \(\approx 0.75\%) of the maximum achievable gain. The next higher gain, obtained for 17 trits, is given in the table for completeness.

Given \(b\) bits and \(t\) trits, we have to determine how to map \(b\) values onto \(3^t\) values so that the multi-level logic implementation is minimized, i.e. leads to the use of as few boolean operators as possible with each of these operators having an as low number of inputs as possible. From a combinatorial point of view these mappings are ordered arrangements, so there are \(2^{3^t} = \frac{k!}{\mu^{k} (2^t - 3^t)}\) of them, where \(\mu\) represents the falling factorial. We focus on two particular instances of the problem, the mapping of 3 trits on 5 bits, leading to \(32^{15} \approx 2.2 \times 10^{31}\) possible mappings, and the mapping of 5 trits on 8 bits, leading to \(256^{24} \approx 1.4 \times 10^{47}\) mappings to choose from.

It is quite clear given this analysis that searching for an optimized mapping of 17 trits on 27 bits is totally impractical.

Even for our two cases of relatively small size, given the size of the search space, exhaustive search is not an option, and finding the optimal solutions is statistically unlikely since multi-level optimization is an NP-complete problem [11].

III. RELATED WORK

This problem may seem a fairly well known one, but to our surprise, the work of mapping a bit string representing a subset of its possible values to a subset of bit strings of smaller size containing all permutations does not seem documented in the literature.

The most extensive surveys on logic synthesis and input/output encoding and state assignment [10], [12] target slightly different problems, making the available techniques not easily transposable. Another approach to obtain state assignment, by using symbolic representation of the states as proposed in [13], reaches optimal solutions with more than the minimal number of bits, whereas it is critical in our case to stick to the minimum number of bits. Chapter 7.5 of De Micheli’s book [14] is dedicated to these encoding problems, but again, the problems that are solved are sufficiently different from ours to make the approaches inappropriate.

Output encoding is also the subject of [15]. This paper cites all the relevant works on the topic of encoding targeting logic minimization.

<table>
<thead>
<tr>
<th>trits</th>
<th>bits</th>
<th>gain</th>
<th>free values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>0.00%</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>0.00%</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>16.67%</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>12.50%</td>
<td>47</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>20.00%</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>16.67%</td>
<td>290</td>
</tr>
<tr>
<td>7</td>
<td>12</td>
<td>14.29%</td>
<td>1909</td>
</tr>
<tr>
<td>8</td>
<td>13</td>
<td>18.75%</td>
<td>1631</td>
</tr>
<tr>
<td>9</td>
<td>15</td>
<td>16.67%</td>
<td>13085</td>
</tr>
<tr>
<td>10</td>
<td>16</td>
<td>20.00%</td>
<td>6487</td>
</tr>
<tr>
<td>17</td>
<td>27</td>
<td>20.58%</td>
<td>5977565</td>
</tr>
</tbody>
</table>
We refer the interested reader to its bibliography, to avoid too many citations in this correspondence. This work makes the assumption that the mapping is already chosen, although part of the output uses symbols instead of bits, and it is the bit strings corresponding to these symbols that are searched for. Their proposition is also unfortunately not generalizable to our problem.

Bearing in mind that our goal is not to devise the best assignment for this problem in its generality, but for the two instances that we believe are useful, we focus on these problems only.

IV. Evaluation approach

We take as first approximation of the logic complexity the number of literals after multi-level minimization and technology mapping on a minimal standard cell library using the Alliance VLSI CAD tools [16]. We also tried with Espresso [17] followed by Alliance technology mapping, but the results where not consistent with Synopsys synthesis as Espresso targets multi-level PLA minimization and not standard cells. The library contains exclusively a not and 2, 3 and 4-input and, or, nand, nor, xor, and xnor gates, and uses arbitrary units for area (\\lambda). This step allows first to rank the solutions easily and second to reproduce the results presented here, without the need to access a specific proprietary software and cell library.

As a first step, we generated 10,000 random assignments for the two interesting cases, using the mapping \mu for the ternary values representation. The mapping of 32 values coded on 5 bits to the 27 legal 3 trits coded on 6 bits led to an average number of gates equal to \approx 77.4 with a standard deviation of \approx 6.0 after technology mapping. Regarding the assignment of 256 values coded on 8 bits to the 243 legal 5 trits coded on 10 bits, the average number of gates equals to \approx 886.8 with a standard deviation of \approx 14.8 after technology mapping. We give here the number of gates, but we verified that, given the limited number of gates in the standard cell library, there is a very strong correlation with the area.

As a second step, and for actual implementation, we use Synopsys’ design compiler targeting STMicroelectronics 28 nm FDSOI standard cell library at an operating point of 0.9 V. The areas and propagation times of the circuits are given as reported by the synthesis tool.

V. Encoding solutions

We tried several automated solutions that failed to minimize the number of gates. The Hungarian algorithm [18] is optimal for assignment problem, as long as we can provide a cost matrix. We are unable to build a relevant cost matrix. Indeed, costs are interdependent since boolean subexpressions are shared. We attempted unsuccessfully with several cost functions being variation of the Hamming distance between the trits code and the binary values. We also tried to tweak state assignment algorithms to perform this assignment instead of state assignment. Overall, these trials produce encodings that, once synthesized, contain half the number of gates of a random assignment, but are still far from the solution we present below (at least twice as big for the 8 bits to 5 trits case).

In this work, we did not consider classical algorithms used for large NP-complete problems such as simulated annealing, generic algorithms or tabu search. To perform efficiently, these algorithms need a fast and accurate evaluation of the solutions to deal with the huge number of produced solutions during searching. Unfortunately, an ASIC synthesis on the chosen pre-characterized library of logic cells requires at least a minute.

The solutions presented below where derived by hand assuming structural properties. Regarding the notation, ‘...’ represents a “don’t care”, i.e. the value of the signal is irrelevant.

A. 5 bits to 3 trits

Denoting \( t_{5,0} \) the binary encoded ternary values and \( b_{4,0} \) the binary codes, our best assignment solution is:

<table>
<thead>
<tr>
<th>( t_{5,0} )</th>
<th>( b_{4,0} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>00000</td>
</tr>
<tr>
<td>00001</td>
<td>00001</td>
</tr>
<tr>
<td>00011</td>
<td>00110</td>
</tr>
<tr>
<td>00100</td>
<td>10010</td>
</tr>
<tr>
<td>01000</td>
<td>10101</td>
</tr>
<tr>
<td>01100</td>
<td>11011</td>
</tr>
<tr>
<td>10100</td>
<td>11100</td>
</tr>
<tr>
<td>11000</td>
<td>10000</td>
</tr>
<tr>
<td>11100</td>
<td>10100</td>
</tr>
</tbody>
</table>

We now detail how we have obtained this solution. The principle is as follows. First, we generate each trit by coding only its magnitude (\( t_4, t_2, \) or \( t_0 \)), i.e. whether it is null or not. The sign is obtained thanks to one input bit only (\( b_4, b_3, \) or \( b_2 \)). The codes are then \( b_4, b_3, b_4, b_2, b_3, b_4, b_2, t_2 \). We call \( t_4, t_2, t_0 \) the magnitude vector. Second, the idea is to gather codes having similar magnitude vectors in sets. In our proposal, the first set contains the 8 codes associated to the magnitude vector \( (1, 1, 1) \). The second set contains 6 codes, 4 associated to the magnitude vector \( (0, 1, 1) \) and 2 associated to \( (0, 1, 0) \). In this set, the most significant trit is 0. Thus, \( b_4 \) can be reused and the magnitude vector can be \( (0, 1, 1) \). Similarly, the third set contains the 6 codes corresponding to the magnitude vector \( (b_3, 0, 1) \). The last set contains the 7 codes, 4 associated to the magnitude vector \( (1, 1, 0) \), 2 associated to \( (1, 0, 0) \) and one to \( (0, 0, 0) \). The first 6 codes can be efficiently expressed by magnitude vector \( (1, b_2, 0) \). When \( b_2 \) is 0, \( b_3 \) is unused. It is then used to distinguish \( (1, 0, 0) \) from \( (0, 0, 0) \). So we can extend the magnitude vector to \( (b_3+b_2, b_2, 0) \) to cover all cases. These 4 sets are respectively encoded as ‘11’, ‘00’, ‘10’ and ‘01’ using \( b_3 b_4 \).

From classical boolean optimization [19] and factorization techniques, we derive the following equations:

\[
x_0 = b_0 (b_1 + b_2), \quad x_1 = b_0 + b_1 \\
(t_0) = b_1 + b_0 b_4, \quad t_1 = t_0 b_2 \\
(t_2) = x_0 + x_4 t_1, \quad t_3 = t_2 b_3 \\
(t_4) = x_0 + b_3 x_1, \quad t_5 = t_4 b_4
\]

This solution can be synthesized in 17 gates with an area of 17250 \( \lambda^2 \) with Alliance. Compared to the random generated cases, it is about 4.5 times smaller. Synthesis on STMicro 28 nm FDSOI using the entire standard cell library produces an area of 6.52 \( \mu m^2 \), (11 gates instantiated), and a propagation time of 41 ps.

B. 8 bits to 5 trits

Again, denoting \( t_{8,0} \) the binary encoded ternary values and \( b_{7,0} \) the binary codes, our best assignment solution is:

<table>
<thead>
<tr>
<th>( t_{8,0} )</th>
<th>( b_{7,0} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>00000000</td>
</tr>
<tr>
<td>00000001</td>
<td>00000001</td>
</tr>
<tr>
<td>00000011</td>
<td>00000010</td>
</tr>
<tr>
<td>00000101</td>
<td>00000100</td>
</tr>
<tr>
<td>00001001</td>
<td>00001000</td>
</tr>
<tr>
<td>00010001</td>
<td>00010000</td>
</tr>
<tr>
<td>00100001</td>
<td>00100000</td>
</tr>
<tr>
<td>01000001</td>
<td>01000000</td>
</tr>
<tr>
<td>10000001</td>
<td>10000000</td>
</tr>
<tr>
<td>00000010</td>
<td>00000011</td>
</tr>
<tr>
<td>00000100</td>
<td>00000101</td>
</tr>
<tr>
<td>00001000</td>
<td>00001001</td>
</tr>
<tr>
<td>00010000</td>
<td>00010001</td>
</tr>
<tr>
<td>00100000</td>
<td>00100001</td>
</tr>
<tr>
<td>01000000</td>
<td>01000001</td>
</tr>
<tr>
<td>10000000</td>
<td>10000001</td>
</tr>
<tr>
<td>00000000</td>
<td>00000001</td>
</tr>
<tr>
<td>00000010</td>
<td>00000011</td>
</tr>
<tr>
<td>00000100</td>
<td>00000101</td>
</tr>
<tr>
<td>00001000</td>
<td>00001001</td>
</tr>
<tr>
<td>00010000</td>
<td>00010001</td>
</tr>
<tr>
<td>01000000</td>
<td>01000001</td>
</tr>
<tr>
<td>10000000</td>
<td>10000001</td>
</tr>
<tr>
<td>00000000</td>
<td>00000001</td>
</tr>
<tr>
<td>00000010</td>
<td>00000011</td>
</tr>
<tr>
<td>00000100</td>
<td>00000101</td>
</tr>
<tr>
<td>00001000</td>
<td>00001001</td>
</tr>
<tr>
<td>00010000</td>
<td>00010001</td>
</tr>
<tr>
<td>01000000</td>
<td>01000001</td>
</tr>
<tr>
<td>10000000</td>
<td>10000001</td>
</tr>
</tbody>
</table>
This mapping, obtained using a strategy similar to the previous one, can be produced using the following boolean equations:

\[
\begin{align*}
x_0 &= y_1 b_2, \quad x_1 = (b_0 + b_3) b_6 b_1 b_2 + b_7 + x_0 \\
x_2 &= z_2 b_3 b_2, \quad x_3 = (b_0 z_0 + z_2) b_7 y_0 \\
x_4 &= y_2 b_5 + z_1 + y_1, \quad x_5 = y_3 b_6 b_5 + y_6 b_2 b_6 + y_5 y_9 \\
x_6 &= (y_8 + b_1 b_7) b_2 + y_4 b_3 b_7 + y_6 + b_9 z_1 + y_1 \\
x_7 &= b_0 b_5 b_2 (b_1 + b_4) + y_2 b_5, \quad x_8 = (b_7 + y_3) y_1 + y_7 \\
x_9 &= y_6 b_5 + y_4 b_5 + y_2 b_0 b_4 + y_5 + y_3 b_7 b_6 \\
y_0 &= b_1 b_5 + b_7 z_0 + y_9 + b_8 y_0 b_7, \quad y_1 = b_0 b_1 \\
y_2 &= y_7 (b_0 \oplus b_1) b_5 b_2, \quad y_3 = b_0 b_1 b_3 \\
y_4 &= b_0 + b_4, \quad y_5 = b_0 b_1, \quad y_6 = b_0 b_3 \\
y_7 &= x_2 b_6, \quad y_8 = b_0 b_3 b_6, \quad y_9 = b_3 + b_2 \\
z_0 &= b_6 b_7 b_5, \quad z_1 = b_4 b_2, \quad z_2 = b_0 b_1 \\
t_0 &= x_0 + y_0, \quad t_1 = b_4 y_0 + b_3 x_0, \quad t_2 = x_8 + x_9, \quad t_3 = b_5 x_9 + b_4 x_8 \\
t_4 &= x_5 + x_7, \quad t_5 = b_6 x_7 + b_8 x_6, \quad t_6 = x_4 + x_5, \quad t_7 = b_7 x_5 + b_6 x_4 \\
t_8 &= x_1 + x_3, \quad t_9 = b_4 x_3 + b_7 x_1
\end{align*}
\]

These equations lead to a circuit of 85 elementary gates (0.3500 \(\lambda^2\)) using Alliance. Synthesis for STMicro 28 nm FDSOI produces an area of 38.51 \(\mu m^2\) (62 gates instantiated), and a propagation time of 120 ps.

C. Technology mapping optimization

As can be seen in the previous assignment tables, some ternary codes are generated by 2 binary codes because of the “don’t cares” (5 ternary codes for our first decoder 13 for the second one). There are potential boolean simplification and technology mapping optimization opportunities left by assigning only one binary code per ternary code and specifying a “don’t care” output for the unused binary codes. However, there is no general optimization pattern that we could find to select one of these two possibilities for the 5 (respectively 13) cases so as to minimize the number of gates resulting from the implementation. As the number of possible combinations is \(2^5 = 32\) for the first decoder and \(2^{13} = 8192\) for the second one, we decided to use a brute-force approach. Indeed, these numbers are small enough that we can synthesize all theses cases in a few days. For all synthesized circuits, we plot the critical path as a function of the area in Figures 1 and 2. The size of the dot is proportional to the number of cases that match a given (area, time).

1Synthesizing all 5b-to-3t cases on a medium range server actually took a bit more than 4 hours, and all 8b-to-5t cases two and a half days.
As expected, there are better solutions than our original hand-derived one, for area and/or time. However, some of them are worse for both area and time, even though they actually are just subsets of our initial designs. The performance of the optimization process of the synthesis tool (and maybe the targeted technology) is then key for the quality of the solutions and none can be said to be the best in all conditions, as the Pareto front in the figures shows. The search and the choice of the best trade-off has to be done by the users of the decoder with their tool, technology, and target applications.

D. Encoding

For completeness, we also now give the number of gates, area and propagation time for the encoding part of our decoders. If needed, the equations of the encoder can be generated by a boolean minimization program (e.g. Espresso) using the reverse table as entry point. Please note that since we are focused on reading and decompressing the code, the values written into memory may well be computed offline by software instead of requiring dedicated encoding hardware.

1) 3 trits to 5 bits: Using the reversed version of the mapping of subsection V-A using Synopsys on STMicro 28 nm FDSOI gives an area of 18.44 \( \mu m^2 \) (29 gates instantiated), and a propagation time of 92 ps.

2) 5 trits to 8 bits: Identically, for the mapping of subsection V-B, we obtain an area of 102.7 \( \mu m^2 \) (172 gates instantiated), and a propagation time of 178 ps using STMicro 28 nm FDSOI technology.

VI. Area savings

As illustrated in Figure 3, the area overhead brought by our decoders depends only on the memory width (in bits) and not on the memory depth (number of words). So no matter how large the decoder, there will always exist a minimum number of words above which the memory area savings are higher than the decoder overhead.

In [20], the authors report an SRAM bit cell area of \( A_B = 0.12 \mu m^2 \) in the technology we use, also for an ANN application. Given this information, we can derive rough but credible estimates of the size of a memory cut of \( W \) words of \( B \) bits each, and decide when it is interesting to use our encoding approach. We note \( D \) the number of decoders and \( A_D \) the area of one decoder. For the 3-trit case, we have \( D = \frac{B}{6} \). The area overhead of the decoders is \( A_D \times D = A_D \times \frac{B}{6} \). The memory area spared is \( D \times W \times A_B = \frac{B}{6} \times W \times A_B \). Hence, to obtain a saving of \( R \) as ratio of the original memory size, the condition is:

\[
\frac{B}{6} \times W \times A_B - A_D \times \frac{B}{6} > R \times W \times A_B
\]

which simplifies to:

\[
W > \frac{A_D}{A_B} \times \frac{1}{1 - R \times 6}
\]

A similar argument gives in the 5-trit case:

\[
W > \frac{A_D}{A_B} \times \frac{1}{2 - R \times 10}
\]

Figure 4 gives the value of \( W \) such that Equation 2 (3-trit case) and 3 (5-trit case) hold for continuous values of \( R \). It shows that the 3-trit approach is more interesting for small memories, while the highest area

2Slide 25 of their oral ISSCC presentation.
savings are obtained with the 5-trit approach. At the intersection point, both approaches bring overall savings of 15.4% with 691 memory words. The need for memories of such size is very common in the context of neural network architectures. For example in the AlexNet network [21], the largest layers feature 4096 neurons with 4096 weight values per neuron. The corresponding memories of weights, both deep and wide, are perfect candidates to ternary compression. The proposed compression approaches, completely devoid of control, are also among the few—if not the only ones—suitable for such wide memories under a sustained throughput measured in TB/s.

Clearly, any better trits-to-bits mapping associated to an optimized logic synthesis process would enable lowering these thresholds, hence the interest in any approach that could address this class of problems.

VII. Power Considerations

SRAM memories are known to consume orders of magnitude more than logic gates [22], even with relatively small memories. This is still the case with the technology we use, although the raw data is not publicly accessible. The power—and area—of memory cells depends on a lot on technological features and architectural-level parameters anyway, so we provide power results as a general trend only.

We did power simulations with the STMicro 28 nm FD-SOI technology, assuming a toggle rate of 50% on the inputs (standard assumption, but high in the context of ternary ANN where zero weights dominate). We observed that the 5-trit decoder consumes roughly 8× more than the 3-trit decoder. However, due to the much higher consumption of the SRAM, the 5-trit approach brings overall better power savings than the 3-trit one thanks to its better compression ratio (~20% versus ~16.7% in memory width). Both decoding solutions bring around 15% power savings for small memories, e.g. 512 words, with a slight advantage for the 5-trit approach. Higher savings, closer to the 20% limit, can be obtained with the 5-trit approach for deeper and/or wider memories, e.g. around 18% overall power savings are observed with 4k words.

In the case of external DRAM access, the power consumption of decoding (and even encoding) is so insignificant compared to DRAM operations that the proposed approaches would bring a solid 16.6% power savings for the 3-trit approach and 20% savings for the 5-trit, along with similar reduction in memory size requirements.

VIII. Summary

In this correspondence, we address the problem of efficiently decompressing a vector of bits into binary encoded trits. We first show that it is neither necessary nor practical to compress more than 5 trits into 8 bits, and then give two optimized mappings and their corresponding multivalued and multilevel boolean function. These mappings were obtained by human reasoning, and no automatic method we could think of gave better or even approaching results. It is left as an open problem to know if better mappings exist.

In conclusion, the proposed approaches bring noticeable savings both on area and power, which makes them essential in all classes of applications where ternary values are stored in memory and read frequently.

Acknowledgments

This work is funded in part by Grenoble Alpes Métropole through the Nano2017 Esprit project. We want to acknowledge CMP for the provision of products and services that facilitated this research, including access to STMicroelectronics 28 nm design kits and macro-block generators data. Finally, we would like to thank Olivier Menut (STMicroelectronics, Crolles) for insightful discussions during the project and Mario Diaz-Nava (ST Microelectronics, Grenoble) for his long-term encouragement and support.

References